

实验三：汇编和 c 语言开发独立内核

实验者：张海涛，张晗宇，张浩然，张镓伟
学号：15352405, 15352406, 15352407, 15352408
院系：数据科学与计算机学院
专业：15 级软件工程（移动信息工程）
指导老师：凌应标

【实验题目】

用汇编和 c 语言开发独立内核

【实验目的】

1. 用 C 和汇编实现操作系统内核。
2. 扩展内核汇编代码，增加一些有用的输入输出函数
3. 提供用户程序返回内核的一种解决方案
4. 增加内核的 C 模块中批处理功能

【实验要求】

1. 将实验二的原型操作系统分离为引导程序和 MYOS 内核，由引导程序加载内核，用 C 和汇编实现操作系统内核。
2. 扩展内核汇编代码，增加一些有用的输入输出函数，供 C 模块中调用
3. 提供用户程序返回内核的一种解决方案
4. 在内核的 C 模块中实现增加批处理能力
 - (1) 在磁盘上建立一个表，记录用户程序的存储安排
 - (2) 可以在控制台命令查到用户程序的信息，如程序名、字节数、在磁盘映像文件中的位置等
 - (3) 设计一种命令，命令中可加载多个用户程序，依次执行，并能在控制台发出命令
 - (4) 在引导系统前，将一组命令存放入磁盘映像，系统可以解释执行

【实验方案】

1. 虚拟机配置方法

无操作系统，10M硬盘，4MB内存，启动时连接软盘

2. 软件工具和作用

Notepad++: 用于生成. 汇编语言文件

Nasm: 用于编译. asm 类型的汇编语言文件，生成 bin 文件

Vmware Workstation 12Player: 用于创建虚拟机，模拟裸机环境

TCC: 用于由. c 文件生成. obj 文件

TNASM: 用于由. asm 文件生成. obj 文件。

Tlink: 用于链接 obj 文件生产. com 可执行文件

3. 方案思想

1) 在 os.asm 文件中实现设置光标位置, 输出一个字符(串), 键盘输入一个字符, 清屏, 读软盘到内存, 跳转到用户程序等基础功能, 而在.c 文件中实现用户交互, 批处理功能等上层功能。二者混合编译, 形成操作系统内核。

2) 引导程序中载入四个子程序和系统内核, 虚拟机运行引导程序之后, 控制权交给系统内核, 然后按照用户指令执行程序。



















4. 实验原理

(1) 汇编和 C 语言混合编程

首先要将写好的 c 文件 和 汇编文件, 编译生产 obj 文件, 前者用 TCC 编译, 后者用 Tasm 编译, 然后用 tlink 混合编译形成.com 可执行文件。

说起来简单, 执行起来难啊! 单个编译器的要求配置各式各样, 要和自己电脑兼容就变得复杂很多。多方求教之后, 我们终于找到一条可行之路, 使用 DOSBOX 调用各个编译器来实现编译。

将所有文件放到同一目录下, 然后再将 exe 格式的 tcc, tlink, tasm 编译器放到同一个目录。

 c.c	2016/3/31 20:16	C Source file	7 KB
 C.OBJ	2017/4/19 23:57	3D 对象	6 KB
 loader.asm	2017/4/19 21:10	ASM 文件	2 KB
 orange1.asm	2017/4/19 23:48	ASM 文件	4 KB
 orange1.bin	2017/4/19 23:59	BIN 文件	1 KB
 orange2.asm	2017/4/19 23:52	ASM 文件	3 KB
 orange2.bin	2017/4/19 23:59	BIN 文件	1 KB
 orange3.asm	2017/4/19 23:53	ASM 文件	3 KB
 orange3.bin	2017/4/20 0:00	BIN 文件	1 KB
 orange4.asm	2017/4/19 23:53	ASM 文件	2 KB
 orange4.bin	2017/4/20 0:00	BIN 文件	1 KB
 os.asm	2016/3/31 15:00	ASM 文件	5 KB
 OS.COM	2017/4/19 23:58	MS-DOS 应用程序	4 KB
 OS.MAP	2017/4/19 23:58	MAP 文件	1 KB
 OS.OBJ	2017/4/19 23:03	3D 对象	1 KB
 TASM.EXE	1991/11/10 10:30	应用程序	113 KB
 TCC.EXE	2017/4/17 20:42	应用程序	177 KB
 TLINK.EXE	1997/9/28 14:43	应用程序	22 KB

打开 DOSBOX, 将这个文件夹创建为一个虚拟盘, 在这个虚拟盘下便于操作。

```
DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
Welcome to DOSBox v0.74
For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c d:\os
Drive C is mounted as local directory d:\os\

Z:\>c
Illegal command: c.

Z:\>c:

C:\>
```

TCC 编译器编译 c 文件为 obj 文件。

```
DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c d:\os
Drive C is mounted as local directory d:\os\

Z:\>c
Illegal command: c.

Z:\>c:

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

    Available memory 438286

C:\>
```

Tasm 编译器便宜 asm 文件为 obj 文件

```
DOS FOR DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
Available memory 438286

C:\>tasm os.asm os.obj
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:  os.asm
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 474k
**Fatal** Error writing object file

C:\>tasm os.asm os.obj
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:  os.asm
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 474k

C:\>_
```

Tlink 编译器将 两个 obj 链接生成 com 文件

```
DOS FOR DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
C:\>tasm os.asm os.obj
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:  os.asm
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 474k
**Fatal** Error writing object file

C:\>tasm os.asm os.obj
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:  os.asm
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 474k

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

C:\>_
```

（在这一步我们遇到了一个小细节问题,交换了 c. obj 和 os. obj, 导致编译失败）

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

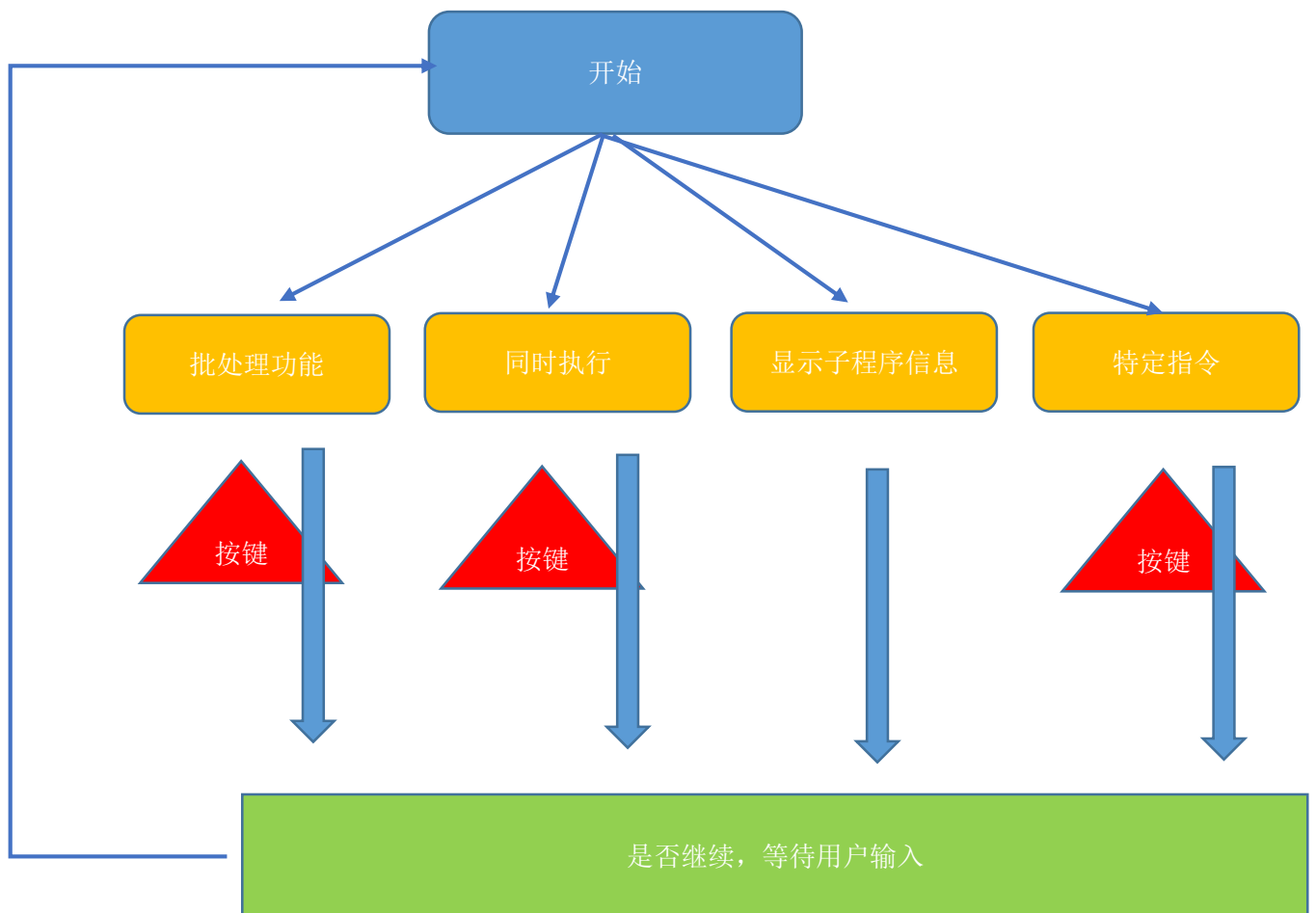
C:\>tlink/3/t c.obj os.obj,os.com,,
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
Cannot generate COM file : data below initial CS:IP defined
```

Data below initial CS: IP defined, 这个报错信息我们一直不懂, 还以为是配置环境出了问题, 结果只是交换了参数顺序, 就编译成功了, 令人费解。

最后使用 nasm 编译用户程序和引导盘。

```
c:\TCC-TASM>nasm orange1.asm -o orange1.bin  
c:\TCC-TASM>nasm orange2.asm -o orange2.bin  
c:\TCC-TASM>nasm orange3.asm -o orange3.bin  
c:\TCC-TASM>nasm orange4.asm -o orange4.bin  
c:\TCC-TASM>nasm loader.asm -o test5.img
```

5. 程序流程



6. 程序关键代码及解释

【相互声明外部函数和变量】

汇编语言中，声明外部函数与变量

```
;声明外部函数
extrn _cmain:near
extrn _cal_pos:near
;声明外部变量
extrn _pos:near
extrn _ch:near
extrn _x:near
extrn _y:near
extrn _offset_user:near
extrn _d:near
```

C语言中，声明外部函数与变量

```
//声明外部函数
extern void clear();
extern void inputchar();
extern void printchar();
extern void printstring();
extern void setcursor();
extern void load();
extern void jmp();
extern void filldata();
extern void readdata();
```

【引导程序中加载内核和四个子程序】

org 7c00h ; BIOS将把引导扇区加载到0:7c00h处，并开始执行

OffsetOfKernal equ 8100h

Start:

```
;读软盘或硬盘上的若干物理扇区到内存的ES:BX处:
mov ax,cs ;段地址 ; 存放数据的内存基地址
mov es,ax ;设置段地址（不能直接mov es,段地址）
mov bx,OffsetOfKernal ;偏移地址; 存放数据的内存偏移地址
mov ah,2 ;功能号
mov al,8 ;扇区数
mov dl,0 ;驱动器号 ; 软盘为0，硬盘和U盘为80H
mov dh,0 ;磁头号 ; 起始编号为0
mov ch,0 ;柱面号 ; 起始编号为0
mov cl,10 ;起始扇区号 ; 起始编号为1
int 13h ; 调用读磁盘BIOS的13h功能
jmp OffsetOfKernal
```

```
incbin 'orange1.bin'
incbin 'orange2.bin'
incbin 'orange3.bin'
incbin 'orange4.bin'
incbin 'os.com'
```

（此处我们沿用了参考代码，我们的设计思路是先将自己的子程序移植进来，通过运行来加深对内核调度的实现过程的理解，然后再对内核进行自己的修改。但光是移植过程，就消耗了我们一周的时间，问题从加载地址错误，导致毫无显示，再到分时显示失败，最后同时运行指令的错误执行，我们不断尝试，查询相关资料，在这个艰苦的过程中的确学到了不少知识，最后在对内核修改时，错误实在太多，我们选择充分理解这个内核，然后仿照实现）

【汇编程序和 c 程序的主函数】

```
_TEXT segment byte public 'CODE'      ;定义代码段
DGROUP group _TEXT, _DATA, _BSS       ;将 TEXT, DATA, BSS合成一组
    assume cs:_TEXT                   ;关联段名与段寄存器
    org 100h                          ;定义全局基地址
start:
    mov ax, cs
    add ax, 800h                      ;800h*10h+100h=8100h
    mov ds, ax
    mov es, ax
    mov ss, ax
    mov sp, 100h
    ;设置段地址之后，访问c代码中的_cmain
    call near ptr _cmain
    jmp $                             ;访问当前指令，即死循环
```

执行之后，一直访问 c 文件中的主函数（cmain）。

```

cmain() {
    while(1) {
        clear();
        printstr(message1);
        printstr(message2);
        printstr(message3);
        printstr(message4);
        printstr(message5);
        printstr(message6);
        printstr(message7);
        printstr(message8);
        inputcmd();
        if(str[0]=='A' || str[0]=='a') {
            printstr(info1);
            printstr(info2);
            printstr(info3);
            printstr(info4);
            printstr(back1);
            inputcmd();
        }
        else if(str[0]=='B' || str[0]=='b') {
            b();
            printstr(back1);
            inputcmd();
            if(str[0]!='13') continue;
        }
        else if(str[0]=='C' || str[0]=='c') {
            c();
            printstr(back1);
            inputcmd();
            if(str[0]!='13') continue;
        }

        else if(str[0]=='D' || str[0]=='d') {
            printstr(defaultm);
            for(i=0;i<80;i++) {
                str[i]=defaultinst[i];
                if(defaultinst[i]=='\0') {
                    length=i;
                    break;
                }
            }
            b();
            printstr(back1);
            inputcmd();
            if(str[0]!='13') continue;
        }
        else {
            printstr(error1);
            printstr(back1);
            inputcmd();
            if(str[0]!='13') continue;
        }
    }
}

```

进入主函数，显示用户界面（这些信息用 80 字节的字符串分别存储，包含使用说明，功能提示，这里需要汇编实现的输出字符串功能支持），然后用户输入指令（inputcmd()函数中调用了汇编实现的输入字符串功能），在对用户的输入进行判断，跳转到相应的函数中去执行。

这里有批处理指令，同时运行指令，显示子程序信息的指令，**default** 默认指令（**b 1 2 3 4**）。执行用户过程中，子程序会一直监视键盘，如果键入相应序列号，即停止执行该子程序，若全部终止，则由用户输入任意字符，回到主页面。

【批处理指令和同时运行的实现】

```
b() {
    clear();
    offset_begin=offset_user1;
    num_shanqu=8;
    pos_shanqu=2;
    load(offset_begin,num_shanqu,pos_shanqu);
    filldata(hex600h,0);
    filldata(hex600h+2,0);
    filldata(hex600h+4,0);
    filldata(hex600h+6,0);
    for(i=0;i<length;i++){
        if(str[i]=='1'){
            offset_user=offset_user1;
            jmp();
        }
        else if(str[i]=='2'){
            offset_user=offset_user2;
            jmp();
        }
        else if(str[i]=='3'){
            offset_user=offset_user3;
            jmp();
        }
        else if(str[i]=='4'){
            offset_user=offset_user4;
            jmp();
        }
    }
}
```

首先将设定初始的偏移、扇区等信息。然后将 0 传入用户程序内存，表示这是批处理指令在运行，要先跑完整个程序再到下一个。然后根据字符串内容判断依次该执行什么用户程序。

```

c() {
    clear();
    offset_begin=offset_user1;
    num_shanqu=8;
    pos_shanqu=2;
    load(offset_begin,num_shanqu,pos_shanqu);
    filldata(hex600h,0);
    filldata(hex600h+2,0);
    filldata(hex600h+4,0);
    filldata(hex600h+6,0);
    for(i=0;i<length;i++){
        if(str[i]=='1') filldata(hex600h,1);
        if(str[i]=='2') filldata(hex600h+2,1);
        if(str[i]=='3') filldata(hex600h+4,1);
        if(str[i]=='4') filldata(hex600h+6,1);
    }
    while(1){
        s=0;
        readdata(hex600h);
        s=s+d;
        if(d)
        {
            offset_user=offset_user1;
            jmp();
        }
        readdata(hex600h+2);
        s=s+d;
        if(d)
        {
            offset_user=offset_user2;
            jmp();
        }
        readdata(hex600h+4);
        s=s+d;
        if(d)
        {
            offset_user=offset_user2;
            jmp();
        }
        readdata(hex600h+4);
        s=s+d;
        if(d)
        {
            offset_user=offset_user3;
            jmp();
        }
        readdata(hex600h+6);
        s=s+d;
        if(d)
        {
            offset_user=offset_user4;
            jmp();
        }
        if(s==0)
            break;
    }
}

```

同样首先对偏移地址和扇区等信息进行初始化，然后把 1 传入用户程序内存位置，表示这是同时运行用户程序，程序内部每循环一次都要返回内核。接着根据字符串内容进行相应的判断该运行哪几个用户程序。

【汇编中的输入输出函数】

```
//声明外部函数
extern void clear();
extern void inputchar();
extern void printchar();
extern void printstring();
extern void setcursor();
extern void load();
extern void jmp();
extern void filldata();
extern void readdata();
```

具体包含：清屏，键入字符，打印字符，打印字符串，设置光标，加载子程序，跳转，写数据和读数据。取一个函数为例：

```
;清屏
public _clear
_clear proc
    push ax
    push bx
    push cx
    push dx
    mov ax, 0600h    ; AH = 6,  AL = 0
    mov bx, 0700h    ; 黑底白字 (BL = 7)
    mov cx, 0        ; 左上角: (0, 0)
    mov dx, 184fh    ; 右下角: (24, 79)
    int 10h          ; 显示中断
    mov word ptr [_x], 0
    mov word ptr [_y], 0
    mov word ptr [_pos], 0
    call _setcursor
    pop dx
    pop cx
    pop bx
    pop ax
ret
clear endp
```

具体可分为三步：保存现场，实现功能，恢复现场。因为调用者也会用到为数不多的寄存器，所以要想将其保存，才能用寄存器来实现功能。这里是通过 BIOS 的 10h 中断，填充整个屏幕，以此来实现清屏效果。

【可执行子程序】

四个用户都是在上一次的实验内容基础上进行修改而来的。第一个是显示字符的动态变化和学号，第二个是显示一个变色的爱心，第三个是显示一个变色的爱心和变色的学号，第四个是一个日历。

四个程序修改地方主要在于用户程序开始位置的调整、对里面的延时函数进行了统一和整理、加入一些变量和端口参数判断以什么样的方式返回内核，等等。

【实验过程】

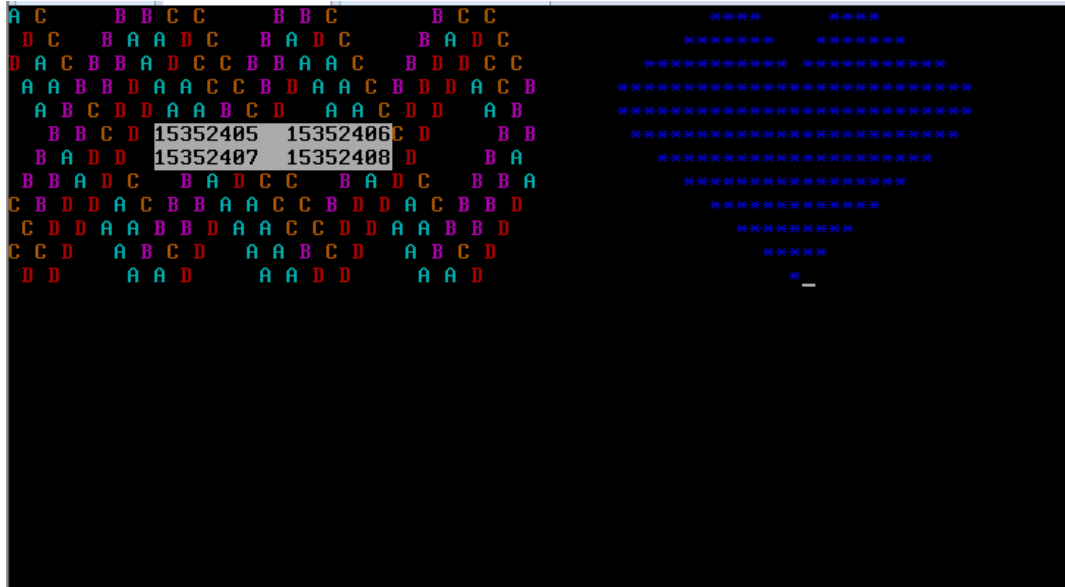
进入系统，根据提示输入命令。

```

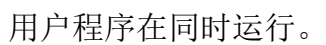
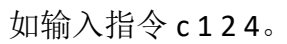
Welcome to MYOS
Please input your instructions
A.Check the information of the OS
B.Run the programs in sequence
e.g B 1 2 3 4 (Then programs will be run in such a order)
C.Run the programs simultaneously
e.g C 1 2 3 (Then program 1,2,3 will be run at the same time)
D.Run as default
MYOS> _
```

如输入指令 b 1 2 3 4,便会一次执行用户程序 1、2、3、4。





执行完指令的用户程序，系统会提醒你返回内核，继续操作。



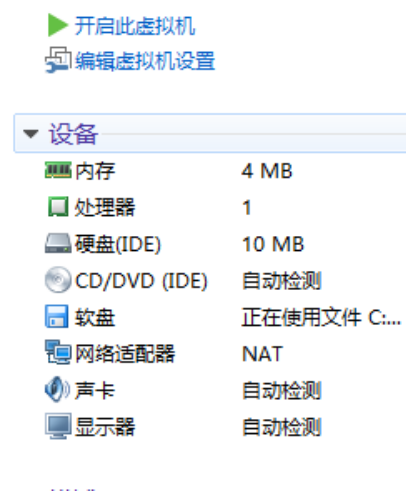
输入 a，显示用户程序信息。

```

Welcome to MYOS
Please input your instructions
A.Check the information of the OS
B.Run the programs in sequence
e.g B 1 2 3 4 (Then programs will be run in such a order)
C.Run the programs simultaneously
e.g C 1 2 3 (Then program 1,2,3 will be run at the same time)
D.Run as default
MYOS> a
USER PROGRAM1-Name:function1 Size:1024byte Position:section 2
USER PROGRAM2-Name:function2 Size:1024byte Position:section 4
USER PROGRAM3-Name:function3 Size:1024byte Position:section 6
USER PROGRAM4-Name:function4 Size:1024byte Position:section 8
Input anything and return
MYOS> _
```

输入 d，执行默认指令 b1234.

虚拟机的设置：



【实验总结】

张浩然的实验总结：

这次的实验遇到的困难是比较多的。首先是需要搭建很多的新环境:tcc、tasm、tlink。为了研究如何在我的电脑上运行花了很多工夫，最后使用 DOSBOX 来运行这些程序。然后这次的要求比较的多，刚看要求不太有头绪哪一个要求具体该用什么样的方式去实现。再者就是这次的实验又提到了内核、磁盘映像之类的新的

内容，怎么编写它们也是一个需要解决的困惑。

这次实验中遇到最大的问题就是我们的同时显示多个用户程序出了问题，它依旧是执行完一个用户程序才往下面走。经过很长时间的研究和尝试后，我们才发现原来问题出在用户程序中。用户程序缺少判断运行模式（依次还是同时）来区别操作的部分，所以无法实现预期的效果。

由于缺少很多汇编语言的基础，以及汇编语言的习惯了解还不够，每次的实验都是在一边模仿，一边学习，一边尝试的。感觉这样的确有很多地方的感受会很深，但也有一些地方迷迷糊糊就过去了，要是下次依旧遇到了相同的问题或者用法，自己未必能准确而又理解透彻地去运用。可能还是不能忙于单纯的研究每周的实验内容和实验要求，需要抽出更多的时间研读相关的课外参考教材，多向老师咨询。

张海涛的实验总结：

这次实验做的事用 C 语言和汇编语言混合编译操作系统，越来越具有操作系统的雏形了。仔细阅读了参考代码之后，我发现内核中汇编编写的部分是较为底层的功能实现，而 c 代码主要是负责各用户程序之间的通信，提供上层交互服务。

我们继承上一次写好的用户程序时，遇到一个致命的问题：段地址的设置。子程序的段地址，都要一致设置为磁盘中的存储地址，而有一个子程序需要用到显存，需要设置为 0b800h，由于我们忽略了这个问题，导致没有任何显示。

在阅读参考代码的过程中，对于混合编程多了一点认识。

还有关于 ptr 的用法，查询之后了解到：ptr 是临时的类型转换，相当于 C 语言中的强制类型转换。在没有寄存器名存在的情况下，既都是在内存，得用操作符 X ptr 指明内存单元的长度，X 在汇编指令中可以为 byte，word 或者 DWORD。要不然内存是片连续的区域，操作就乱了。因为 ptr 指向的都是外部变量或函数，是存储在内存中，使用的时候需要指明格式大小。

本次实验中最困难的部分就是开头第一步：编译。首先是不同编译器的配置，然后是编译流程，最后是编译指令。

配置好编译器，就开始向其他同学了解编译流程，因为不懂得编译原理，所以常常是一头雾水。在一次 link 时，踩到了一个匪夷所思的坑。tlink 编译时是有文件顺序要求的，一直编译失败，就交换了参数顺序，就成功了。这个探索过程还真是挺艰苦。

张晗宇的实验总结：

这次的实验主要是学习生成内核，生成 com 文件，c 文件与汇编文件相结合。一开始在按步骤进行 TLINK 操作生成 com 文件时，遇到了 CS: IP defined 问题，因为我们没有意识到两个文件的输入顺序也有影响，需要先输入汇编文件再输入 c 文件。这次实验我主要负责各子程序的修改，一开始模仿王师兄的修改方法，由于一开始没注意王师兄的结尾已经不是

```
times 510-($-$$) db 0
db 0x55,0xaa
```

也就是已经不是 512kb 的汇编程序而是 1024kb 的文件，也使得程序一开始就停止运行了，直到一遍一遍对比代码差异才发现问题所在。第一个子程序也就是显示学号，改过之后发现字符移动的特别缓慢，将 DELAY 从 0ffffh 降低到 001ffh 才使字符的显示流畅，但是跳转时间又过于快，再次调整后才正常显示。第二子程序（心形的显示以及闪烁）也遇到了问题，一开始心形的显示字符是乱码，试了几次我们都以为不能拿中断显示，差点要放弃，在一次偶然的尝试去掉

了 `mov ax, 0b800h`, 发现正常显示了, 后来才知道中断调用的段地址和偏移地址要相同。之后发现心形并不闪烁, 调整延迟之类的发现也没什么变化, 直到再增加了两个 `DELAY` 之后才发现之前也是有闪烁的, 只是变化太快看不清, 最后通过改变跳转时的 `byte[count]` 改成了 `word` 才得以正常显示, 之后的子程序 3, 4 也是同样修改。在观察王师兄的程序时, 发现他的子程序执行完的输入即使不是 ‘yes’ 也在进行相同跳转, 也算是一点不足吧。

张镓伟的实验总结:

这次实验第一个挑战是搭建新环境, `tcc`, `tlink` 和 `tasm`。然后还要使用 `DOSBOX` 去运行。在各种百度和向同学请教之后最终弄清了如何使用。这次实验的程序实在王师兄的代码的基础上再结合我们上一次实验的代码修改而成。总体来说在引导程序和 C 代码的编写上还算比较顺利。C 和汇编的结合很强大, 在汇编中写好合适的接口函数, 就可以在 C 中调用, 让我们的开发变得方便, 毕竟相对于没什么基础的 x86 汇编, 我们对 C 语言比较熟悉。另外, 我有在 linux 下写过 `fortran+c` 混合编程的经历, 对于混合编程的接口函数命名有点心得。我们发现在汇编程序中可以被 C 调用的函数其函数名前都有一条下划线。这其实跟编译后产生的 `.o` 文件有关。`.o` 文件中有代码中各个函数的符号表, C 要调用汇编的接口函数, 则 C 文件编译产生的 `obj` 文件中该函数的符号 要在 汇编程序编译后产生的 `obj` 文件中找到的对应符号才能成功将两者链接。而 C 的函数符号表一般就是函数名前+一个下划线。所以要这样命名。

本次实验在于 4 个子程序用分时功能执行时, 出现了无法同时执行而是顺序执行的情况。经过与王师兄代码的不断对比, 我们发现在子程序代码循环的最后缺少判断当前如果是分时功能执行状态, 就将当前状态压入栈中, 然后返回内核, 留待下一次进入该子程序继续执行。修改了这里之后又出现了心形图案闪烁过快(我们有两个子程序是显示变色心形)导致看上去颜色没有变化。后面我们尝试增加 `delay`, 这样的话会看到子程序 1 的执行变慢了, 这是因为两者的延时不同步导致的。最后我们通过不同调整 `delay` 的时间, 还有循环闪烁的次数终于将这个问题解决。

总的来说, 这次实验又学到了不少东西。