

实验三 开发独立内核的操作系统 实验报告

数据科学与计算机学院 2017 级计算机科学与技术 17341146 王程钊

1 实验题目

开发独立内核的操作系统

2 实验目的

配置 C 与汇编联合编译的环境
掌握 C 与汇编联合编译的方法
熟悉引导程序和内核的调用
熟悉 C 与汇编的编程方法

3 实验要求

3.1 用 C 和汇编实现操作系统内核

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。

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。

提供用户程序返回内核的一种解决方案

3.2 在内核的 C 模块中实现批处理

在磁盘上建立一个表，记录用户的存储安排

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
设计一种命令，命令中可以加载多个用户程序，依次执行，并能在控制台发出命令。

在引导系统前，将一组命令存在磁盘映像中，系统可以解释执行。

4 实验方案

4.1 实验环境

编程环境：Dosbox+TCC+TASM+Tlink,NASM

16 进制编辑器：Hex Editor

虚拟机：VMware Workstation

虚拟机环境

- 操作系统 MS-DOS
- 内存 1MB
- 硬盘 102MB
- 处理器 1 核心

编译命令

```
nasm -f bin guide.asm -o guide.com  
tasm myos.asm myos.obj
```

```
tcc -mt -c -omain.obj main.c
tlink /3 /t myos.obj main.obj,test.com,
```

4.2 引导扇区程序引导内核

引导扇区的大小只有 512b, 很难放下操作系统的内核。为了实现更大的操作系统内核, 我们需要通过存放在引导扇区的引导程序, 通过 int13h 中断加载操作系统内核。

4.3 C 与汇编联合编译

C 语言作为高级语言编写起来较为方便, 汇编语言作为底层语言适合与 CPU 进行交互, 比如调用 BIOS 中断。因此, 系统内核一般由 C 语言与汇编联合编写。我们需要通过连接器实现 C 与汇编联合编译, 生成可执行文件。

4.4 批处理功能的实现

为了执行相关操作, 需要设计一种命令行来运行相关程序。在设计命令行的基础上, 还需要实现批处理功能, 即在一条指令中执行多个用户程序。另外, 为了丰富用户程序的运行方式, 我还开发了脚本功能, 通过运行脚本程序执行多个指令。

4.5 程序存储设计

功能	程序名	存储扇区
引导程序	Guide.com	1
内核	Test.com	2-11
文件夹	File.txt	12
用户程序 1	Stone1.com	13
用户程序 2	Stone3.com	14
用户程序 3	Stone3.com	15
用户程序 4	Stone4.com	16
脚本程序 1	Shell1.bat	17
脚本程序 2	Shell2.bat	18

5 实验过程

首先简单介绍一下本次实验主要实现的内容。在完成内核编写, 文件表显示, 批处理命令, 脚本运行的基础上, 我还实现了一些个性化的功能。

- 1) 我以 windows 的 cmd 和 linux 的终端为参考, 设计了一个基于命令行控制的操作系统。我的控制台可以实现字符回显, 屏幕上下卷, 上下键切换前序命令, 左右键修改命令, 退格键删除等功能。
- 2) 我设计了一个操作系统的开始动画, 在每次开机后会先运行动画。
- 3) 我实现了文件读写的功能, 并依此实现了一个简单的文件系统和批处理程序运行功能。
- 4) 我实现了输出系统时间的功能, 输入相关命令即可显示系统时间。

5.1 引导扇区程序引导内核

我是用 NASM 设计引导程序, 通过引导程序加载内核。

我使用老师给的 showstr.asm 和 upper.c 当操作系统内核使用。我将操作系统内核

加载到 a100h, 并将 showstr.asm 的 org 100h 改为 org a100h。我将其放入 TASM 编译, 却显示编译错误, 程序入口非法。

我尝试使用 org 100h 直接装入虚拟机编译运行。结果发现这段程序并不能正常运行, 屏幕会显示一堆乱码。我回忆了一下, 实模式下的地址有 20 位, 格式是(段地址<<4+偏移量)。于是我怀疑段寄存器没有设好。我将 ds,es 都改为 a00h, 再装入虚拟机运行, 结果屏幕上成功显示了字符串。

以下为引导扇区程序

```
org 7c00h
OffsetOfUserPrg1 equ 0a100h
start:
    mov ax,cs
    mov ds,ax
    mov ax,ds
    mov es,ax

    mov bx,OffsetOfUserPrg1
    mov cx,bx
    mov ah,2
    mov al,10
    mov dl,0
    mov dh,0
    mov ch,0
    mov cl,2
    int 13H ;
    jmp OffsetOfUserPrg1

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

5.2 内核的汇编部分

通过查询 TCC 的帮助文件, 我发现在 TCC 环境下可以直接在 C 语言中调用汇编, 只需要在之前加入 asm 关键字。这就给编程带来了很大便利。通过 TCC 内嵌汇编, 内核的纯汇编部分就变得很简单, 只需要调用 TCC 中的 main 函数即可。

以下为内核汇编部分的代码

```
extrn _main:near ;声明一个 c 程序函数 main
.8086
_TEXT segment byte public 'CODE'
DGROUP group _TEXT,_DATA,_BSS
    assume cs:_TEXT
org 0100h
start:
    mov ax, 0a00h
```

```

;mov ax, cs
mov ds, ax      ; DS = CS
mov es, ax      ; ES = CS
mov ax, 0c00h
mov ss, ax      ; SS = cs
mov sp, 0ffffh
call near ptr _main
jmp $

_TEXT ends
;*****DATA segment*****
_DATA segment word public 'DATA'
_DATA ends
;*****BSS segment*****
_BSS segment word public 'BSS'
_BSS ends
;*****end of file*****
end start
```

5.3 内核的 C 语言部分

由于不能调用标准库，所以我自己实现了 4 个标准库供 main 函数调用。下表为这四个库的名称及功能。

库文件	stdio.h	string.h	algo.h	clock.h
功能	输入输出	字符串处理	算法库	时钟操作

5.3.1 输出部分

输出部分需要实现以下功能：输出可见字符，输出带颜色字符，移动光标，回车键换行，删除键退格，滚屏，字符串与数字的输出。

1) 输出可见字符

ASCII 码大于等于 32 的字符为可见字符，若输出函数传入这种字符则调用 BIOS 的 10h 号中断的 0ah 号功能输出字符。

2) 输出带颜色字符

BIOS 的 10 号中断的 0ah 号功能的颜色输出有点奇怪，改不了颜色。所以对于这部分我使用 10 号中断的 09h 号功能。

3) 回车键换行与移动光标

每次输出一个字符后光标都需要移动。我使用两个全局变量 x,y 存储当前光标的位置，每次需要移动光标时，调用 BIOS 的 10 号中断的 02h 号功能将光标移动到相应位置。

若输出为回车键(ASCII 码 13)，则需要换行，将光标移动到下一行的第 0 个位置。

4) 删除键退格

若输入为删除键(ASCII 码 8)，则需要退格。退格需要将光标向左移一个位置，并删除掉当前位置的字符。对于删除字符，通过在屏幕上对应位置显示空格实现。

5) 滚屏

当屏幕的最后一行已经输满后若还要输出则需要将屏幕上滚。如果有用户程序运行的话，我希望运行的那块区间不会上滚。我在全屏上滚前先将其下滚。我使用 BIOS 的 10 号中断的 06h 号功能实现屏幕的上滚，使用 07h 号功能实现屏幕的下滚。

6) 字符串与数字输出

字符串输出比较简单，通过 `string.h` 库的相关函数获取字符串长度，一个一个字符输出即可。输出数字时，先将数字转换为字符串，后输出字符串即可。

以下为输出部分的相关代码。

输出字符：

```
void putchar(char c)
{
    int pos;
    if(c>=32)
    {
        asm mov ah,0ah
        asm mov al,c
        asm mov bh,0
        asm mov dh,x
        asm mov dl,y
        asm mov cx,1
        asm int 10h
    }
    //输出有效字符
    if(c==13)pos=(x+1)*80;
    else if(c==8)pos=x*80+max(y-1,2);
    else pos=x*80+y+1;
    if(pos>=MAX_SIZE)
    {
        if(have_run)rool_down(0,40,15,79);
        rool_up(0,0,24,79);
        pos-=80;
    }
    //滚屏调用
    x=pos/80;y=pos%80;
    move(x,y);
    //移动光标位置
    if(c==8)
    {
        asm mov ah,0ah
        asm mov al,' '
        asm mov bh,0
        asm mov dh,x
        asm mov dl,y
        asm mov cx,1
```

```

    asm int 10h
}
//删除键退格
}
(带颜色字符输出与无颜色字符输出类似, 略。)

```

屏幕上下滚:

```

void rool_up(char a,char b,char c,char d)
{
    asm mov ah,06h
    asm mov al,1
    asm mov bh,07h
    asm mov ch,a
    asm mov cl,b
    asm mov dh,c
    asm mov dl,d
    asm int 10h
}
//上滚
void rool_down(char a,char b,char c,char d)
{
    asm mov ah,07h
    asm mov al,1
    asm mov bh,07h
    asm mov ch,a
    asm mov cl,b
    asm mov dh,c
    asm mov dl,d
    asm int 10h
}
//下滚

```

移动光标到(a,b):

```

void move(char a,char b)
{
    asm mov ah,02h
    asm mov bh,0
    asm mov dh,a
    asm mov dl,b
    asm int 10h
    x=a;y=b;
}

```

5.3.2 输入部分

输入部分我参照了 windows 自带的 cmd 终端的输入格式进行实现, 需要实现以下功能: 输入字符, 输入字符回显, 删除退格, 字符串输入, 左右键移动光标, 上下键切换指

令。

1) 输入字符

输入字符通过调用 BIOS 中断的 16h 的 0 号功能实现。因为左右键移动光标的缘故，在输入字符时，如果不是在串尾输入，则需要在输入后，将当前位置及后续位置的字符向右移动一位，并且在屏幕上的显示也要向右移动一位。

2) 输入字符回显

在完成输入后，将刚输入的字符调入 putchar 函数输出，即可完成回显。

3) 删除字符

和输出时的简单删除不同，输入过程删除字符时，如果不是在串尾删除，则需要在删除后，将当前位置及后续位置的字符向左移动一位，并且在屏幕上的显示也要向左移动一位，右边补上空格。

4) 字符串输入

字符串输入模仿 C++ 库函数 gets 的输入方式，持续输入直到在输入换行时会停止。输入得到的字符存在输入缓冲区中，遇到退格键时要在输入缓冲区删除相应字符串。

5) 左右键移动光标

当输入为上下左右键时，调用 int10h 中断后 al 寄存器的值为 0，但是 ah 寄存器的值会发生改变，可以依此判断输入的符号。当输入左右键后，除非当前位置已经时字符串头或字符串尾，否则光标要向左或向右移动。

6) 上下键切换指令

当按下上键后，屏幕上当前行的指令会变成上一次输入的指令，若在按上键，则输入指令会变成再上一行。这需要我不仅修改指令字符串的内容，还要修改屏幕上显示的内容。我先将当前行的字符全部用空格清空，后在屏幕上显示上一行的字符。

我开了一个缓冲区数组，用于存放当前输入前的 30 条命令。因此上下键切换的最大限度为 30 条指令。

以下为输入部分相关代码。

```
char getchar()
{
    char c,d;
    asm mov ah,0;
    asm int 16h;
    asm mov c,al;
    asm mov d,ah;
    if(!c)return -d;
    //输入上下左右键返回负值以区分
    putchar(c);
    return c;
}
char mem[SIZE][BUFLen];
int sum_cmd;
void gets(char *s)
{
```

```

int i, len=0, p=0, po=sum_cmd; char c=getchar();
for(; c!=13; c=getchar())
{
    if(c==72){
        if(!po)continue;
        move(x,2);
        for(i=0; i<len; i++) putchar(' ');
        move(x,2);
        printf(mem[--po]);
        strcpy(s, mem[po]);
        len=y-2; p=len;
    }
    //上
    else if(c==80){
        if(po>=sum_cmd-1)continue;
        move(x,2);
        for(i=0; i<len; i++) putchar(' ');
        move(x,2);
        printf(mem[++po]);
        strcpy(s, mem[po]);
        len=y-2; p=len;
    }
    //下
    else if(c==75){
        if(p-->0, --p, move(x,y));
    }
    //左
    else if(c==77){
        if(p<len) ++y, ++p, move(x,y);
    }
    //右
    else if(c==81){
        len=max(len-1, 0); p=max(p-1, 0);
        for(i=p; i<len; i++)
            s[i]=s[i+1], putchar(s[i]);
        putchar(' ');
        move(x, p+2);
    }
    //删除
    else{
        for(i=p; i<len; i++) putchar(s[i]);
        for(i=len; i>p; i--) s[i]=s[i-1];
        ++len; s[p++]=c;
        move(x, p+2);
    }
}

```



```
    }  
}  
s[len]='\000';strcpy(mem[sum_cmd],s);  
if(++sum_cmd>=SIZE)  
{  
    for(i=0;i<SIZE-1;i++)strcpy(mem[i],mem[i+1]);  
    sum_cmd--;  
}  
}
```

5.3.3 字符串处理部分

字符串处理部分我实现了几个常用的函数。相关函数名称及功能见下表。

函数名	功能
strlen	求字符串长度
strcmp	字符串比较
strcmp_prefix	字符串前缀比较
strcpy	字符串复制
string_to_int	字符串转为数字

以下为相关代码。

```
int strlen(char *s)  
{  
    int len=0;  
    while(s[len]!='\000')len++;  
    return len;  
}  
  
int strcmp(char *a,char *b)  
{  
    int n=strlen(a),m=strlen(b),i;  
    if(n!=m)return 0;  
    for(i=0;i<n;i++)  
        if(a[i]!=b[i])return 0;  
    return 1;  
}  
  
char strcpy(char *a,char *b)  
{  
    int i,len=strlen(b);  
    for(i=0;i<len;i++)a[i]=b[i];  
    a[len]='\000';  
}  
  
int strcmp_prefix(char *a,char *b)  
{
```

```

    int n=strlen(a),m=strlen(b),i;
    if(n<m)return 0;
    for(i=0;i<m;i++)
        if(a[i]!=b[i])return 0;
    return 1;
}

int string_to_int(char *s)
{
    int i,len=strlen(s),res=0;
    for(i=0;i<len;i++)res=res*10+s[i]-48;
    return res;
}

```

5.3.4 算法库

算法库实现了 min(最小值)和 max(最大值)两个函数，代码如下。

```

int max(int a,int b){return a>b?a:b;}
int min(int a,int b){return a<b?a:b;}

```

5.3.5 时钟操作

时钟操作目前只实现了 get_time 函数用于获取当前的系统时间。

时间的获取需要使用 BIOS 的 1ah 号中断的 04h 和 02h 号功能。04h 号功能用于获取年月日，年存在 ch 和 cl 寄存器中，月存在 dh 寄存器中，日存在 dl 寄存器中。02h 号功能用于获取时分秒。时存在 ch 中，分存在 cl 中，秒存在 dh 中。

中断返回的年月日是 BCD 码格式的，需要转为 10 进制输出

以下为相关代码。

```

int hex_dec(int x)
{
    int i=0,res=0;char ch[5];
    for(;x;x/=16)ch[i++]=x%16;
    for(i--;i>=0;i--)res=res*10+ch[i];
    return res;
}
//BCD 码转十进制

void get_time()
{
    char Yh,Yl,M,D,h,m,s;
    asm mov ah,04h
    asm int 1ah
    asm mov Yh,ch
    asm mov Yl,cl
    asm mov M,dh
    asm mov D,dl

```

```

    putint(hex_dec(Yh));
    putint(hex_dec(Yl));
    putchar(':');
    putint(hex_dec(M));
    putchar(':');
    putint(hex_dec(D));
    putchar(' ');
    asm mov ah,02h
    asm int 1ah
    asm mov h,ch
    asm mov m,cl
    asm mov s,dh
    putint(hex_dec(h));
    putchar(':');
    putint(hex_dec(m));
    putchar(':');
    putint(hex_dec(s));
    puts(13);
}
//获取系统时间

```

5.4 命令行设计

我设计了如下的几条命令行。

```

‘help’  获取帮助信息
‘ls’    获取用户程序文件表
‘cls’   清屏
‘res’   重新启动
‘time’  获取系统时间
‘exit’  退出操作系统
‘./<name>’ 运行用户程序
‘./<name1>,<name2>’ 依次运行多个用程序
‘shell <name>’ 运行脚本程序
‘shell <name1>,<name2>’ 运行多个脚本程序

```

5.5 文件表与文件系统

实验要求要实现一个简单的文件表。我的用户表会输出用户程序的程序名，程序大小，程序类型和存储扇区位置。我将用户程序的程序名，程序类型和存储位置转为二进制码，中间用‘\000’连接，转入软盘的第 11 扇区。对于用户程序大小，我调用 int13h 中断，通过计算对应用户程序字符数来计算用户程序大小。

对于文件系统的使用，我调用 int13h 中断进入 11 号扇区，从扇区读取字符串。这其实是类似 C 语言中的文件读写的功能。因此我写了 fopen 函数用于将调用制定扇区，还写了 fscanf 函数用于从扇区中读入字符串。

相关代码如下。

Ls 操作

```
void ls()
```

```

{
    int i,size,spos,fsize,addr=OffsetOfUserName;
    puts("=====");
    print_len("filename",10);
    putchar('|');
    print_len("type",8);
    putchar('|');
    print_len("sector",8);
    putchar('|');
    print_len("filesize",8);
    putchar(13);

    fopen(addr,11);
    addr=fscanf(addr,str,'\000');
    size=string_to_int(str);

    for(i=0;i<size;i++)
    {
        puts("-----");
        addr=fscanf(addr,str,'\000');
        print_len(str,10);
        putchar('|');
        addr=fscanf(addr,str,'\000');
        print_len(str,8);
        putchar('|');
        addr=fscanf(addr,str,'\000');
        spos=string_to_int(str);
        putint_len(spos,8);
        putchar('|');
        fsize=get_size(spos);
        putint_len(fsize,8);
        putchar(13);
    }
    puts("=====");
}

```

获取文件大小

```

int get_size(char pos)
{
    char tmp;
    int i,res=0,addr=OffsetOfSize;
    fopen(OffsetOfSize,pos);
    for(i=0;i<512;i++)
    {
        asm mov bx,addr
    }
}

```

```

    asm mov al,[bx]
    asm mov tmp,al
    asm inc bx
    asm mov addr,bx
    if(tmp!='\000')res=i;
}
return res+1;
}

```

从扇区中读取字符串

```

int fscanf(int addr,char *s,char end)
{
    int i,len=0,flag=0;char tmp;
    while(1)
    {
        asm mov bx,addr
        asm mov al,[bx]
        asm mov tmp,al
        asm inc bx
        asm mov addr,bx
        if(tmp==end)break;
        if(tmp<=32&&!flag)continue;
        flag=1;s[len++]=tmp;
    }
    s[len]='\000';
    return addr;
}

```

打开特定扇区

```

void fopen(int addr,char pos)
{
    asm mov bx,addr
    asm mov ah,2
    asm mov al,1
    asm mov dl,0
    asm mov dh,0
    asm mov ch,0
    asm mov cl,pos
    asm int 13h
}

```

5.6 加载用户程序

每次输出‘./’命令后，程序会在文件表中寻找对应的用户程序，如果找到则执行。同时加多个用户程序与加载一个用户程序类似，两个用户程序之间用逗号隔开以区分。

加载用户程序部分在实验 2 已经完成，我直接使用了实验 2 的这部分代码。

在实验的时候，我发现了一些问题。在调用用户程序返回后操作系统会死机。我并不

知道为什么，试了各种办法都差不出错。我尝试在 `run_com` 函数中加入参数，结果却可以运行了。我百思不得其解。后来我发现，可能是寄存器被修改导致的问题。于是我在调用用户程序前把所有的寄存器都 `push` 到栈里，在调用返回后再 `pop`，就解决了这个问题。

为了保持界面的美观，四个用户程序的反弹区间都被固定在了界面的右上方，并且在屏幕上滚时这块区间不会上滚。

以下为相关代码。

运行用户程序

```
void run_com(char pos,int offset)
{
    int i;char p;
    if(!pos)return;
    p=(pos-1)/18;
    pos%=18;if(!pos)pos=18;
    have_run=1;
    asm push es
    asm push ds
    asm push si
    asm push di
    asm push bp

    asm mov ax,0h
    asm mov es,ax
    asm mov bx,offset
    asm mov cl,pos
    asm mov ah,2
    asm mov al,1
    asm mov dl,0
    asm mov dh,p
    asm mov ch,0
    asm int 13h

    asm mov ax,0
    asm mov es,ax
    asm mov ax,offset
    asm call es:ax

    asm pop bp
    asm pop di
    asm pop si
    asm pop ds
    asm pop es

    puts("successfully run!");
}
```

寻找用户程序

```
int find(char *sec,int tp)
{
    int i,pos=0,size;
    int addr=OffsetOfUserName;
    char sz[20],name[20],type[20],po[20],tw[20];
    if(!tp)strcpy(tw,"com");
    else strcpy(tw,"bat");
    fopen(addr,11);
    addr=fscanf(addr,sz,'\000');
    size=string_to_int(sz);
    for(i=0;i<size;i++)
    {
        addr=fscanf(addr,name,'\000');
        addr=fscanf(addr,type,'\000');
        addr=fscanf(addr,po,'\000');
        pos=string_to_int(po);
        if(strcmp(sec,name)&&strcmp(tw,type))
        {
            printf("program found in sector ");
            putint(pos);puts("!");
            return pos+11;
        }
    }
    puts("program not found");
    return 0;
}
```

5.7 清屏

当从控制台输入 cls 指令后，界面会清屏。清屏部分调用 BIOS 的 10 号中断的 06h 号功能，相当于将整个屏幕上卷。相关代码如下。

```
void clear()
{
    asm mov ah,06h
    asm mov al,0
    asm mov ch,0
    asm mov cl,0
    asm mov dh,24
    asm mov dl,79
    asm mov bh,07h
    asm int 10h
    move(0,0);
    have_run=0;
}
```

5.7 脚本程序的运行

我希望能实现脚本程序的运行，将写入相关命令的脚本程序放入扇区中，输入 shell 指令以运行。脚本程序的运行需要从扇区中读入字符串，这可以使用上面文件系统中实现的 fscanf 和 fopen 函数执行。

不过这个脚本程序系统有一点缺点，泛化性不够好。如果没有 end 语句，或两句话间没有分号，或指令中有空格，后果不堪设想。由于时间紧迫，所以我没有再修改这些问题。

以下为脚本程序例程和相关代码。

脚本程序例程

```
time;
./stone1,stone2;
time;
./stone4,stone,stone3;
time;
end;
```

脚本运行代码

```
int shell_work(char *command)
{
    if(strcmp_prefix(command, "./"))work(command);
    else if(strcmp(command, "ls"))ls();
    else if(strcmp(command, "cls"))clear();
    else if(strcmp(command, "time"))get_time();
    else if(strcmp(command, "end"))return 0;
    else puts("invalid command!");
    return 1;
}

void run_shell(int pos)
{
    int i,addr=OffsetOfUserSrc;
    char command[BUFLen];
    fopen(addr,pos);
    while(1)
    {
        addr=fscanf(addr,command,'%s');
        if(!shell_work(command))break;
    }
    puts("successfully run a script program");
}

void shell(char *str)
{

```



```

int i,curr_pos;
char sec[BUFLen],temp[BUFLen];
strcpy(temp,str);
puts("");
puts("-----");
puts("running in shell mod");
for(i=6;i<strlen(temp);i=curr_pos+1)
{
    curr_pos=get_name(i,temp,sec);
    if(curr_pos<0)break;
    run_shell(find(sec,1));
    puts("");
}
puts("shell mod end");
puts("-----");
puts("");
}

```

5.8 初始界面设计

操作系统刚开机时会显示一个初始界面。我设计了一个简单的初始界面，字符会在屏幕上不断地划 w 型。每划一次字符的颜色会发生变化，每到转角显示的字符会发生变化。

我希望在屏幕中央显示一些带颜色的提示词，这个利用之前提过的带颜色输出和光标移动即可实现。

同时我希望在字符划动过程中，如果键盘键入了任何字符，则初始界面会停止滑动，清屏，并进入控制台界面。因此我实现了 kbhit 函数，利用 BIOS 的 16h 中断的 1 号功能，输入阻塞字符。

在有了 C 语言这个强大的工具后，界面设计变得简单很多。通过 C 语言短短的是十几而是行代码就可以解决汇编近百行才能解决的问题。相关代码如下。

字符滑动

```

int s[]={20,30,40,50,60,80};
void show()
{
    char x=5,y=0,col=0x03,c='A',st,pre=-1;
    int p=0,xx=1;
    while(1)
    {
        move(x,y);st=kbhit();
        if(st==pre)break;pre=st;
        putchar_color(c,col);
        if(++y==s[p])p++,c++,xx=-xx;
        if(c>'Z')c='A';x+=xx;
        if(y==80)
        {
            x=5;y=0;p=0;clear();putword();

```

```

        if(++col==0x0c)col=0x03;
    }
    sleep(12000);
}
}

```

键盘阻塞输入

```

void putword()
{
    move(10,25);
    put_col("Welcome to wcy1222's OS",0x0a);
    move(11,25);
    put_col("Hello World!",0x0a);
    move(12,25);
    put_col("Press any key to continue.",0x0a);
}

```

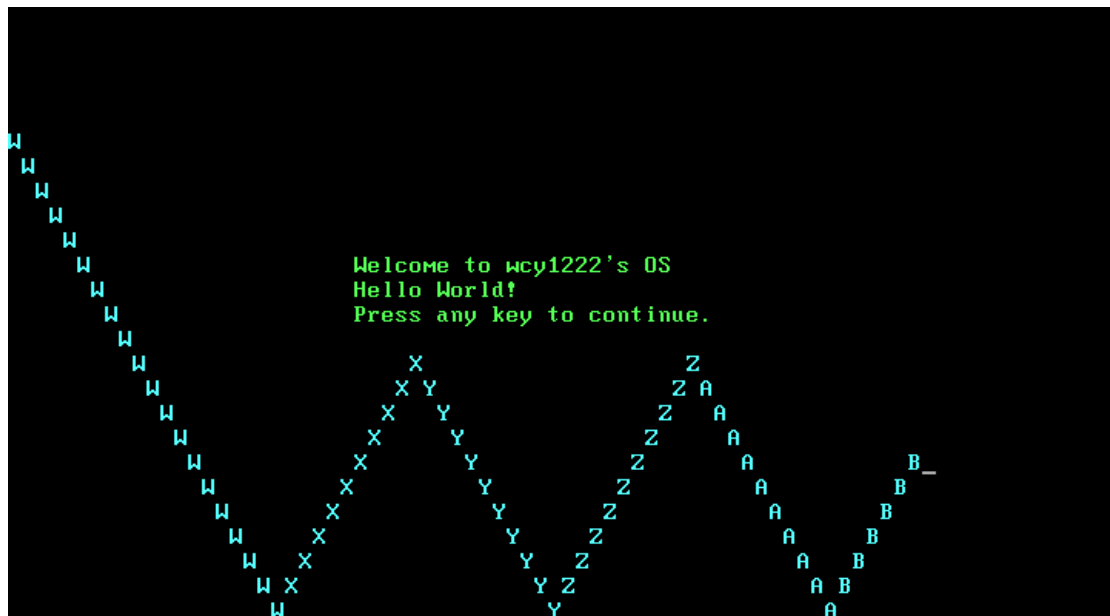
提示词输出

```

void putword()
{
    move(10,25);
    put_col("Welcome to wcy1222's OS",0x0a);
    move(11,25);
    put_col("Hello World!",0x0a);
    move(12,25);
    put_col("Press any key to continue.",0x0a);
}

```

5.9 运行结果



开始界面

```
wcy1122's OS v1.0 by shell  
Input 'help' if you need  
>>
```

进入控制台

```
wcy1122's OS v1.0 by shell  
Input 'help' if you need  
>>help  
  'time'    --- get server time  
  'ls'      --- show information of user program  
  'cls'     --- clear the screen  
  'res'     --- restart wcy1122's OS  
  'time'    --- get server time  
  'exit'    --- exit  
  './<name>' --- run a user program  
  './<f1>,<f2>' --- run multiple program  
  'shell <name>' --- run a script program  
  'shell <f1>,<f2>' --- run multiple script program  
>>_
```

输入 help

```

'cls'      --- clear the screen
'res'      --- restart wcy1122's OS
'time'     --- get server time
'exit'     --- exit
'./<name>'  --- run a user program
'./<f1>,<f2>' --- run multiple program
'shell <name>' --- run a script program
'shell <f1>,<f2>' --- run multiple script program
>>ls
=====
filename : type : sector : filesize
-----
stone1   : com  : 13    : 512
-----
stone2   : com  : 14    : 512
-----
stone3   : com  : 15    : 512
-----
stone4   : com  : 16    : 512
-----
shell1   : bat  : 17    : 71
-----
shell2   : bat  : 18    : 72
=====
>>

```

Ls 指令，查看文件表

```

'shell <name>'  --- run a script program
'shell <f1>,<f2>' --- run multiple script program
>>ls
=====
filename : type : sector : filesize
-----
stone1   : com  : 13    : 512
-----
stone2   : com  : 14    : 512
-----
stone3   : com  : 15    : 512
-----
stone4   : com  : 16    : 512
-----
shell1   : bat  : 17    : 71
-----
shell2   : bat  : 18    : 72
=====
>>time
2019:3:27 15:24:51
>>time
2019:3:27 15:24:59
>>time
2019:3:27 15:25:3
>>

```

Time 指令，查看系统时间

```

filename | type | sector | filesize
-----
stone1 | com | 13 | 512
-----
stone2 | com | 14 | 512
-----
stone3 | com | 15 | 512
-----
stone4 | com | 16 | 512
-----
shell1 | bat | 17 | 71
-----
shell2 | bat | 18 | 72
=====
>>time
2019:3:27 15:24:51
>>time
2019:3:27 15:24:59
>>time
2019:3:27 15:25:3
>>invalid
invalid command!
>>aaabbbccc
invalid command!
>>

```

输入非法指令

```

stone1 | com | 13 | 512
-----
stone2 | com | 14 | 512
-----
stone3 | com | 15 | 512
-----
stone4 | com | 16 | 512
-----
shell1 | bat | 17 | 71
-----
shell2 | bat | 18 | 72
=====
>>time
2019:3:27 15:24:51
>>time
2019:3:27 15:24:59
>>time
2019:3:27 15:25:3
>>invalid
invalid command!
>>aaabbbccc
invalid command!
>>./stone1
program found in sector 13!

```


[illegible]

运行 stone1

```

-----
stone3 : com : 15 : 512
-----
stone4 : com : 16 : 512
-----
shell1 : bat : 17 : 71
-----
shell2 : bat : 18 : 72
=====
>>time
2019:3:27 15:24:51
>>time
2019:3:27 15:24:59
>>time
2019:3:27 15:25:3
>>invalid
invalid command!
>>aaabbbccc
invalid command!
>>./stone1
program found in sector 13!
successfully run!
>>./stone1,stone2
program found in sector 13!

```




批处理指令，运行 stone1 和 stone2，如图运行 stone1

```

-----
stone4 : com : 16 : 512
-----
shell1 : bat : 17 : 71
-----
shell2 : bat : 18 : 72
=====
>>time
2019:3:27 15:24:51
>>time
2019:3:27 15:24:59
>>time
2019:3:27 15:25:3
>>invalid
invalid command!
>>aaabbbccc
invalid command!
>>./stone1
program found in sector 13!
successfully run!
>>./stone1,stone2
program found in sector 13!
successfully run!
program found in sector 14!

```



如图，批处理中运行 stone2

```

2019:3:27 15:24:59
>>time
2019:3:27 15:25:3
>>invalid
invalid command!
>>aaabbbccc
invalid command!
>>./stone1
program found in sector 13!
successfully run!
>>./stone1,stone2
program found in sector 13!
successfully run!
program found in sector 14!
successfully run!
>>time
2019:3:27 15:26:5
>>shell shell1,shell1

-----
running in shell mod
program found in sector 17!
2019:3:27 15:26:16
program found in sector 13!

```



Runing on program 1!
Press any key to stop:_

Shell 命令，运行两次 shell1,如图为第一次(shell1 的代码见 5.7 的脚本例程)

```

>>time
2019:3:27 15:26:5
>>shell shell1,shell1

-----
running in shell mod
program found in sector 17!
2019:3:27 15:26:16
program found in sector 13!
successfully run!
program found in sector 14!
successfully run!
2019:3:27 15:26:28
program found in sector 16!
successfully run!
program not found
program found in sector 15!
successfully run!
2019:3:27 15:26:32
successfully run a script program

program found in sector 17!
2019:3:27 15:26:32
program found in sector 13!

```



Runing on program 1!
Press any key to stop:_

Shell 模式下第二次运行 shell1

```
program not found
program found in sector 15!
successfully run!
2019:3:27 15:26:32
successfully run a script program

program found in sector 17!
2019:3:27 15:26:32
program found in sector 13!
successfully run!
program found in sector 14!
successfully run!
2019:3:27 15:26:40
program found in sector 16!
successfully run!
program not found
program found in sector 15!
successfully run!
2019:3:27 15:26:43
successfully run a script program

shell mod end
-----
>>
```

脚本运行结束

```
>>_

```

清屏

题。保护模式下又没有中断向量可以使用，因此还要学习如何在保护模式下写中断，这又是一个很大的问题。

所以在学习了三天毫无进展后，我开始有点害怕交不上作业。于是我决定放弃 GCC+NASM+LD 的环境，使用老师给的 TCC+TASM+TLINK 环境。这套环境比较古老，但是也有它的好处。首先它可以编译成 16 位文件，不用进入保护模式。其次它支持很方便的内嵌汇编的功能，不像 GCC 的内嵌汇编比较麻烦。于是即使后来我发现 gcc 也可以变异成 16 位，我还是决定使用 TCC 这套环境。

在搭好了环境后，我还遇到了很多其他的问题。比如在最开始使用老师的程序尝试 C+汇编联合编译，一直显示不出字符，这耗费了我很多时间。比如调用用户程序后死机了无法从 run_com 函数返回，这样让我思考了很久，才发现可能是寄存器被修改的问题，才成功将代码改好。很多内容都是我通过自己瞎试才找到的解决方案。

造成这种情况的一个重要原因是上学期计算机组成原理没有学好，虽然分数还不错，但很多知识都没有学得很深刻。上学期计组学习的主要内容是 CPU 的原理，内存、栈空间等问题其实都没有深究。而且上学期汇编学的是 MIPS，这学期则直接上 x86，这之间有很多的不同。缺乏理论基础的情况下，想要做出实验也就只能靠瞎试了。

只算内核，本次实验就写了 10 几 k 的代码，还有 5000 多字的实验报告。很幸运的是我发现了 TCC 内嵌汇编的写法，这减少了我的代码量和编程难度。本次实验的工程量可以说是非常大的，不过能够完成也是很有意义的呢。

其实还有一些想实现的功能还没有实现，比如输出时间的功能，我本来是希望它能够像 windows 一样在屏幕右下角实时更新。这需要用到时钟中断，我打算在实验四继续实现它。

这次实验花费了我很多的时间，不过我认为这是值得的。我完成了一个操作系统的内核，设计了一种指令集，还实现了一些附加功能。虽然有一些遗憾，但我总体上对于本次试验的结果还是很满意的。通过本次实验，我学到了很多知识，了解了 x86 的寄存器和栈的相关知识，了解了段寄存器的相关知识，也锻炼了工程能力和调试能力。

参考文献

[1]TCC 内嵌汇编 <https://www.docin.com/p-74057905.html>

[2]BIOS 中断大全 <https://blog.csdn.net/piaopiaopiaopiaopiao/article/details/9735633>