操作系统实验报告

实验三：引导程序+操作系统内核的设计

学院：　　数据科学与计算机学院

班级：　　16级计算机科学与技术 教务2班

姓名：　　郑映雪

学号：　　16337327

完成时间：　　2018.4.2

一、实验目的

1、把原来在引导扇区中实现的监控程序(内核)分离成一个独立的执行体，存放在其它扇区中，为后续实验扩展内核提供发展空间。

2、学习汇编与 c 混合编程技术，改写实验二的监控程序，扩展其命令处理能力，增加实现实验要求 2 中的部分或全部功能。

二、实验要求

1、规定时间内单独完成实验。

2、实验三必须在实验二基础上进行，保留或扩展原有功能，实现部分新增功能。

3、监控程序以独立的可执行程序实现，并由引导程序加载进内存适当位星，内核获得控制权后开始显示必要的操作提示信息，实现若干命令，方便使用者(测试者)操作。

4、制作包含引导程序，监控程序和若干可加载并执行的用户程序组成的 1.44M 软盘映像。

5、在指定时间内，提交所有相关源程序文件和软盘映像文件，操作使用说明和实验报告。

6、实验报告格式不变，实验方案、实验过程或心得体会中主要描述个人工作，必须有展示技术性的过程细节截图和说明。

三、实验方案

1、相关原理

a.操作系统内核的实现

操作系统内核是操作系统的核心，在本实验中，操作系统内核主要为了实现使用命令执行操作和加载用户程序的功能，类似于我们平时常用的 shell。在本实验中，老师介绍了使用 tasm+tcc 编译的方法，再用 tlink 链接，可以实现汇编语言和 C 语言程序联合编译成为可执行程序。这种方法的优势在于一个允许输入指令的操作系统内核，如果单用汇编语言写会冗长和费事，但是大致架构使

用我们熟悉的 C 语言，关键模块再用汇编语言写的话，就会方便很多。但是 tasm+tcc 是 16 位的，与我 64 位的电脑不兼容，我为了方便特地装了一个 32 位的 win7 虚拟机。在 C 和汇编互相调用中，我采取的是在 C 中调用汇编语言。这个还得感谢老师提供的 kliba.asm 文件，虽然我对其中大部分代码都进行了改写，还删减了部分代码，但是这个文件给了我 C 语言调用汇编语言的语法格式，让我之后的编写比较得心应手。在 C 中调用汇编语言，使用的参数会从右往左压栈，可以对栈进行读取和操作来达到对参数进行操作。

b.为什么要用引导程序

之前的实验里，我们直接把程序放进了第一个扇区，但是总要受到限制，好在之前的程序并不是很大。但本次实验中所要求实现的内容，编译完成后在第一扇区肯定是放不下的。好在上一个实验中，我们已经掌握了利用中断从引导程序跳转的功能，所以在这个实验中，第一个扇区可以放置一个简单的引导程序，根据生成的操作系统大小，我们可以直接在引导程序中设置读取该操作系统的扇区数，这样就可以跳脱大小的限制了。

c.软盘扇区分配

我将首扇区分配给引导程序，从第二扇区后的累计 10 个扇区分配给系统内核。其实原本不用这么多，但是对代码的添加和修改会使.com 文件大小改变，所以留多一些备用。接下来的八个扇区，我分别分配给上一个实验中的四个用户程序，以实现在操作系统内核中调用用户程序的功能。

d.如何实现 shell

利用一系列 BIOS 中断可以实现裸机中的输入输出。在 C 语言的代码中，通过对汇编语言包含输入中断的函数的调用来获得输入的字符串。再用 C 语言编写一个 strcmp 函数，将获得的字符串与自行规定的命令做比较，若相符则跳转至相应功能。如此便可实现 shell 功能。

e.功能设计

这次实验实现的基本功能有：

① dir 功能。类似于我们平常控制台中的 dir 命令，由于不知道怎么实现自动读取，又因为只有四个用户程序，所以在这个实验里简化成对四个程序的信息的输出。

②help 功能。输出所有可用的指令以便使用者查询。

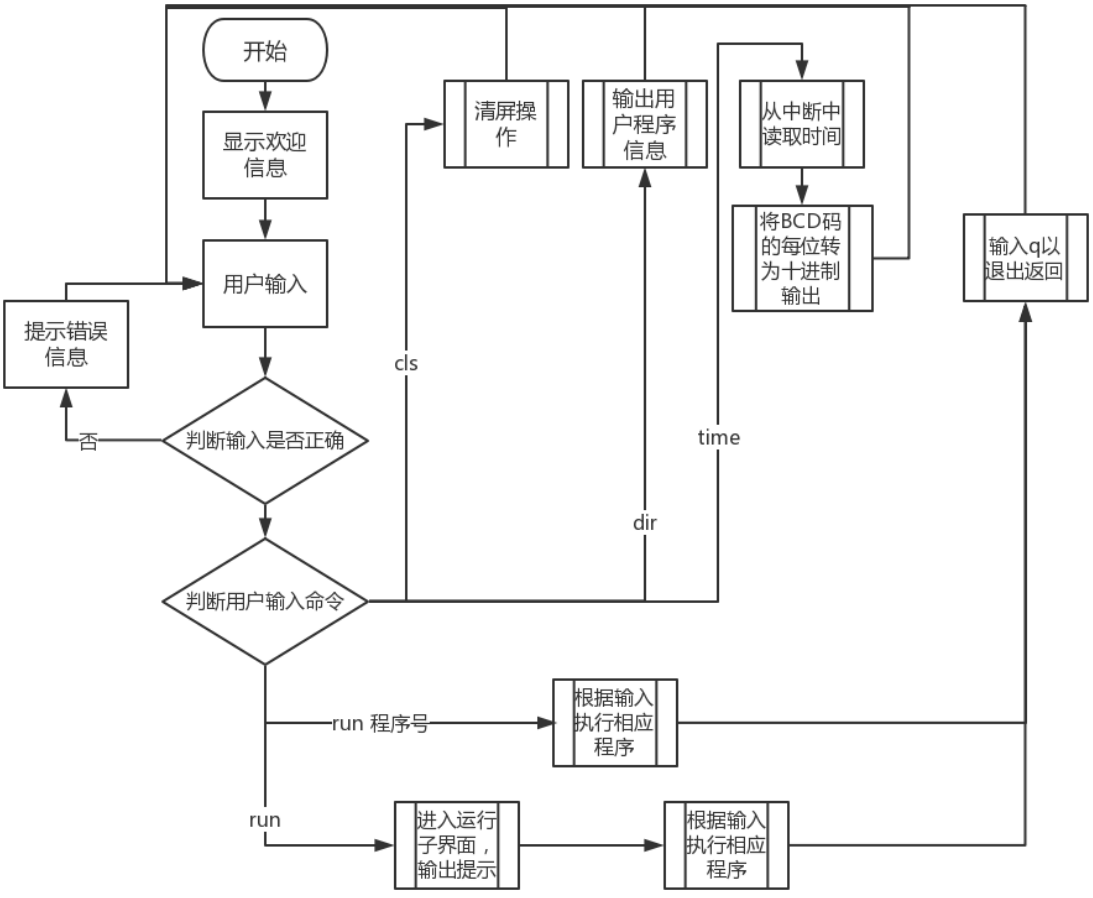
③run 功能。调用用户程序的功能是本次实验中最重要的功能。

④time 功能。实现这个功能差不多是机缘巧合，在查询 BIOS 中断时我偶然发现了功能号为 2 的 1Ah 号中断，更特别的是它返回的是 BCD 码而不是普通的二进制码。不过这也不难，在 C 程序里写一个 BCD 码的转化函数就可以显示了。同时我对它进行了一些修改，使它看起来更像日常生活中的时间显示。具体请在内核页面输入“time”查看。

⑤cls 功能。这是常见的清屏的功能，理论上调用老师的 kliba 文件里的 cls 模块就可以实现，但是有一些小 bug，所以我对它进行了修改，具体见后文。

2、程序流程图

本次实验的流程图如下所示：



3、程序关键模块

（代码相关解释见代码块中的注释）

a. C 程序主程序部分代码 主要是实现操作系统内核的框架。

```
……
n = strlen(tmp); /*读取当前输入字符串的长度，以在下面实现 strcmp 功能*/
printf("\r\n");
if(strcmp(tmp,"help",n,4)) flag = 1; /*进入帮助界面*/
else if(strcmp(tmp,"time",n,4)) flag = 2;   /*显示当前时间*/
else if(strcmp(tmp,"run",n,3) && !strcmp(tmp,"run ",n,4)) flag = 3; /*进入运行子界面时运行程序*/
else if(strcmp(tmp,"run ",n,4)) flag = 4; /*直接运行某程序*/
else if (strcmp(tmp,"dir",n,3))flag = 5; /*显示用户程序信息*/
else if (strcmp(tmp,"cls",n,3)) flag = 6; /*清屏*/
else flag = 7;
……   /*接下来根据 flag 的值来运行相关的函数，这些函数有的在 C 中实现，有的在汇编语言中实现*/
```

b. C 程序部分函数代码 主要是更简便地实现一些功能，具体如下：

```
/*printf 函数我没有采用老师 kliba 里的函数，因为我觉得只要遍历字符串数组，再逐字调用 printchar 函数使用中断显示该字母，也可以达到显示字符串的效果*/

void printf(char *ch){
	while (*ch != '\0'){
		printchar(*ch);
		ch++; /*遍历数组*/
	}
}



/*strcmp 起名源于 C 语言中原本就有的函数，在大一的时候就写过实现 cstring 中函数的作业，所以这个还是比较简单的*/

int strcmp(char str1[], char str2[], int len1, int len2) /*实现两个字符串的比较*/
{
	if(len1 < len2) return 0;
	for(i = 0; i < len2; i++)
		if(str1[i] != str2[i]) return 0;
	return 1;
}
```

（接上文）

```
/*在调用时间模块之后要把 BCD 码转为十进制码才能输出，BCD 码转为十进制码在计组课上曾经说过，网上也有教学相关内容，所以不难实现*/

void todec(int BCD){
	int n1 = 4096;
	int n2 = 0;
	char num1[6] = {'\0'};
	int index = 0;
	if(BCD == 0){
		printf("00");
		return;
	}
	while(BCD < n1) n1/=16;
	while(n1 > 0){ /*BCD 码转为十进制*/
		n2 = BCD/n1;
		num1[index] = n2 + '0';
		index++;
		BCD %= n1;
		n1/=16;
	}
	if (index == 1) {   /*当某个单位为个位数时，在前面补上一个 0，使它看上去更像真实的时钟*/
		num1[1] = num1[0];
		num1[0] = '0';
	}
	printf(num1);
}

……C 程序中其余函数便不放在实验报告中了，具体请看代码附件
```

c. 汇编语言部分代码

注：由于 boot 引导程序和 os 调用和老师的模板相差不大，故本模块只展示部分我修改过后的汇编库里的被 C 语言调用的模块。

首先是清屏模块。在这里我将老师发给我们的清屏模块代码进行了增加，增加了每次清屏后光标回到第一行第一列的操作。其实只要调用功能号为 02 的 10h 中断就可以对光标进行修改。具体修改的原因我会在实验总结里说明。

```
public _cls
_cls proc
        push ax
        push bx
        push cx
        push dx
        mov ax, 600h  ; AH = 6,   AL = 0
        mov bx, 700h  ;  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 dx,0 ;调用功能号为 2 的 10h 号中断，每次清屏后重新设置光标位置为
（0,0）
        mov bx,0
        mov ah,2
        int 10h

        pop dx
        pop cx
        pop bx
        pop ax
        mov word ptr [_disp_pos],0
        ret
_cls endp
```

然后是 strlen 模块。此模块较长，此处我在解释一些重要部分，完整部分见代码的附件。

```
    ……
    push bp
    push cx
    push dx
    push bx
    mov bp, sp
    xor dx,dx
    mov ax, offset [bp+10] ;传参中，参数的地址存入堆栈中，但前面 push 了 4 个寄存
器，故使 bp+10，下同。
```

（接上文）

```
    ……
    mov ah,0
    int 16h     ;调用 16h 中断，输入字符

    cmp al,13
    je exit     ;回车即结束长度统计
    cmp al,8
    jne save    ;退格就不会计入长度

    mov ax, 0
    cmp dx, ax
    je first
    jmp back
    ……
    ;显示字符和退出部分在此实验报告中略，具体请参见代码附件
_strlen endp
;退格时存储空格字符在光标处
back proc
    sub bx,1
    sub dx,1 ;不存储退格字符
    push bx
    push dx
    mov bh,0
    mov ah,3
    int 10h
    sub dl,1
    mov bh,0
    mov ah,2
    int 10h
    mov bh,0
    mov al,' '
    mov bl,7
    mov cx,1
    mov ah,9    ;9 号功能的 10h 号中断作用是在当前光标处显示字符，此处显示空格
    int 10h
    pop dx
    pop bx
    jmp next
back endp
```

获取当前时间模块，采用了 BIOS 里功能号为 2 的 1Ah 号中断，具体解释请看下面的注释：

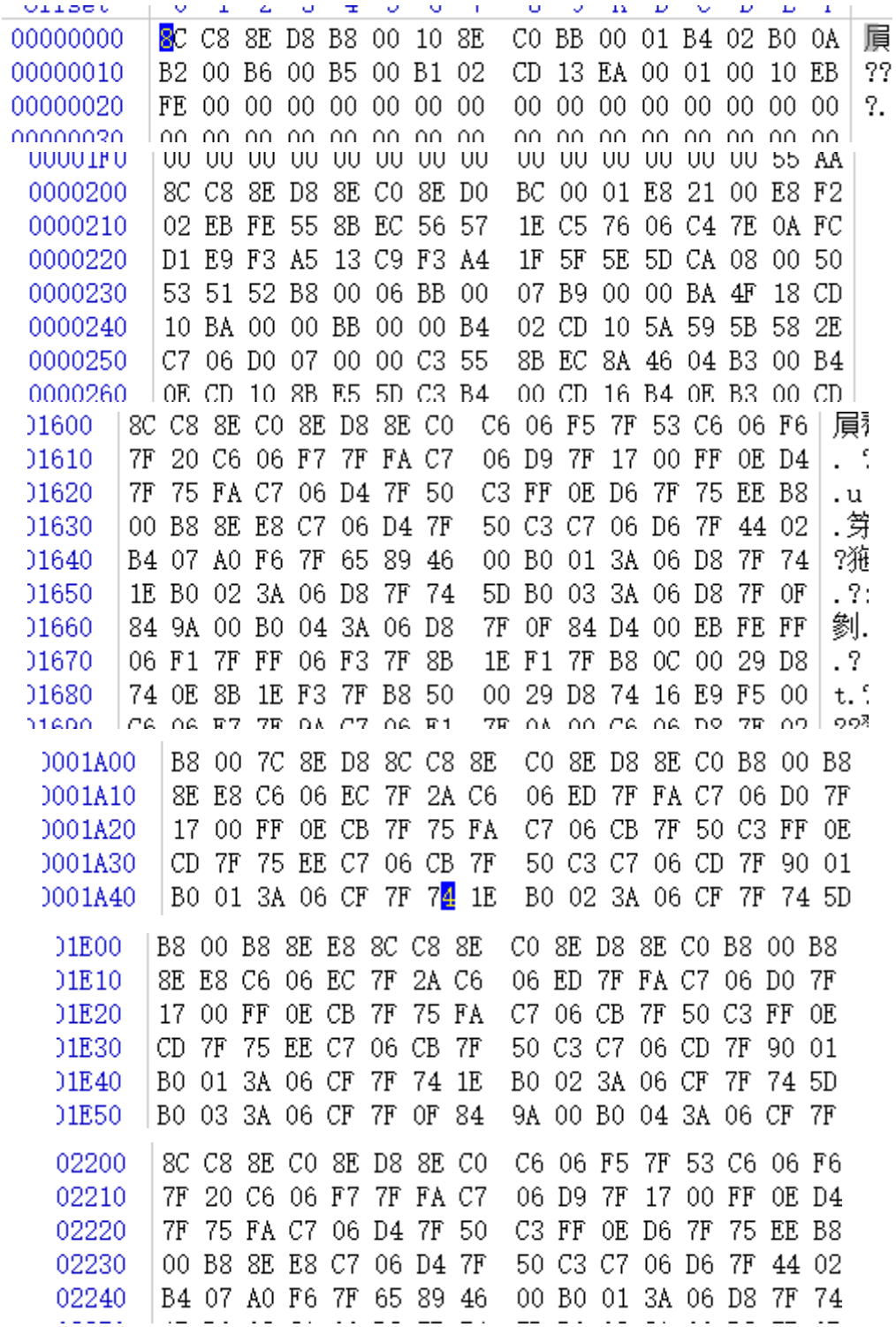
```
public _gettime
_gettime proc
	push bp
	push bx
	push cx
	push dx
	push ax
	push di
	mov bp,sp
	mov ah, 2h
	int 1ah    ;采用 BIOS 里的 1ah 中断
	mov ax, [bp + 14] ;传参中，参数的地址按序存入堆栈中，但是由于之前 push 的寄存器占了 12byte，故要使 bp+14，以下同理
	mov bx, ax
	mov [bx], ch    ;ch 为 BCD 码的小时
	mov ax, [bp + 16]
	mov bx, ax
	mov [bx], cl    ;cl 为 BCD 码的分钟
	mov ax, [bp + 18]
	mov bx, ax
	mov [bx], dh    ;dh 为 BCD 码的秒
	mov sp, bp
	pop di
	pop ax
	pop dx
	pop cx
	pop bx
	pop bp
	ret
_gettime endp
```

运行用户程序模块的代码上个实验已经展示过，这里就不重复展示了，与之前不同的是它的参数变成了 C 程序传来的参数，存在堆栈中，如下图：

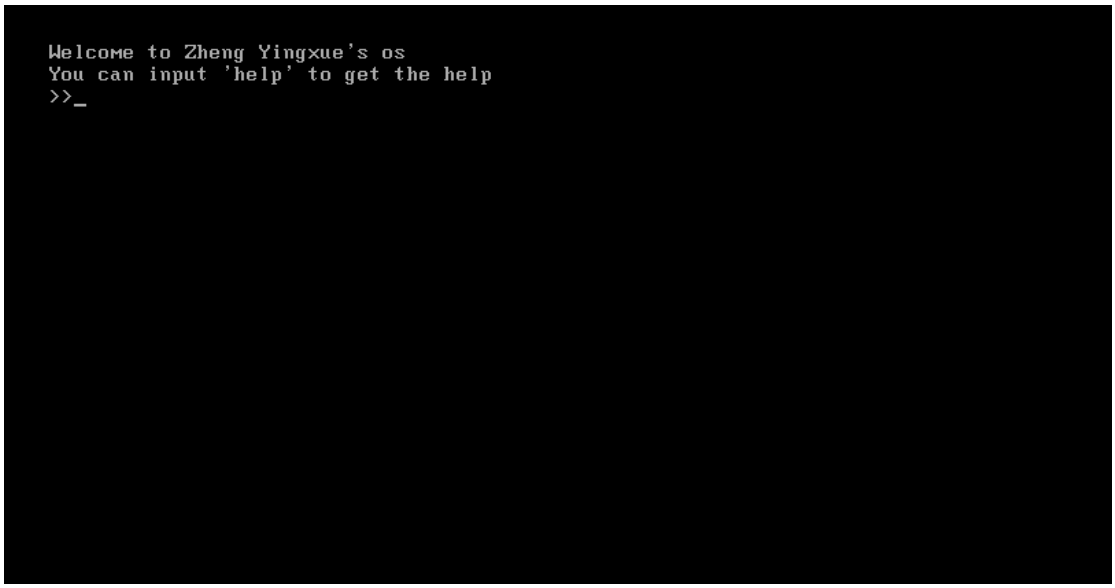
```
	mov ah,2                    ;扇区数和用户程序存放位置以传参的形式存在堆栈中
	mov al,[bp+12]
	mov dl,0
	mov dh,0
	mov ch,0
	mov cl,[bp+14]
	int 13H
```

三、实验过程

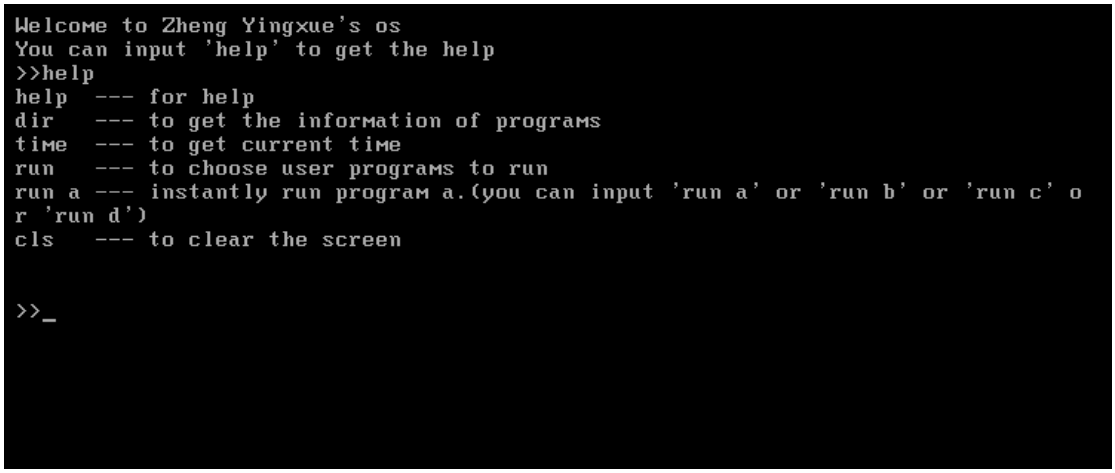
1、在第一扇区放入引导程序的.com 文件，在第二扇区起放入操作系统内核的.com 文件，从第十二扇区起每两扇区存放用户程序的.com 文件



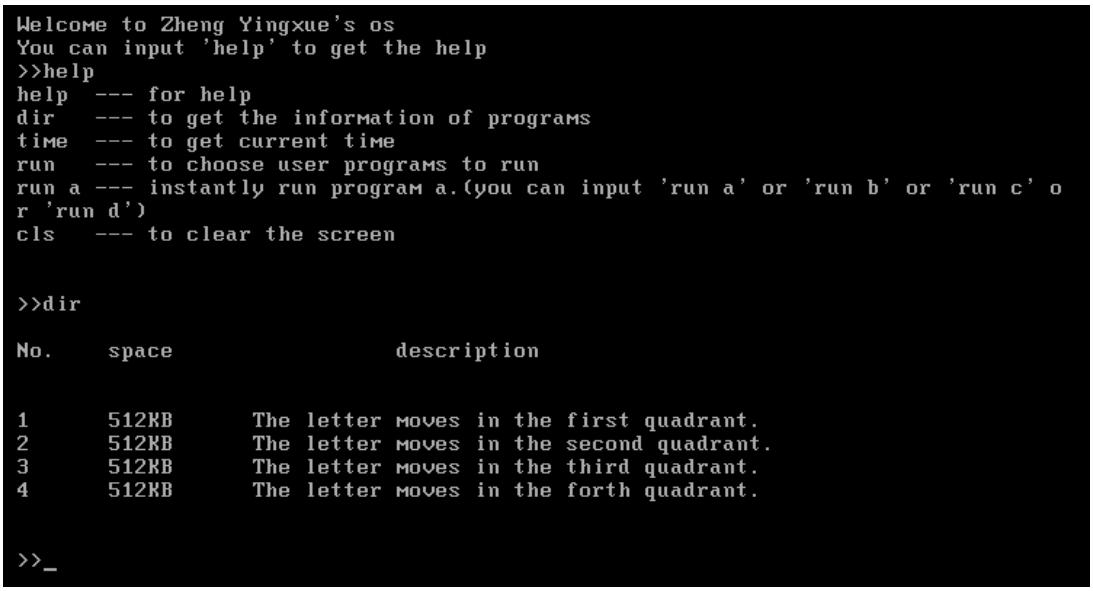
2、将软盘读入裸机，可以看到开机显示主页面和提示：



3、输入 help 查看相关的口令和功能。



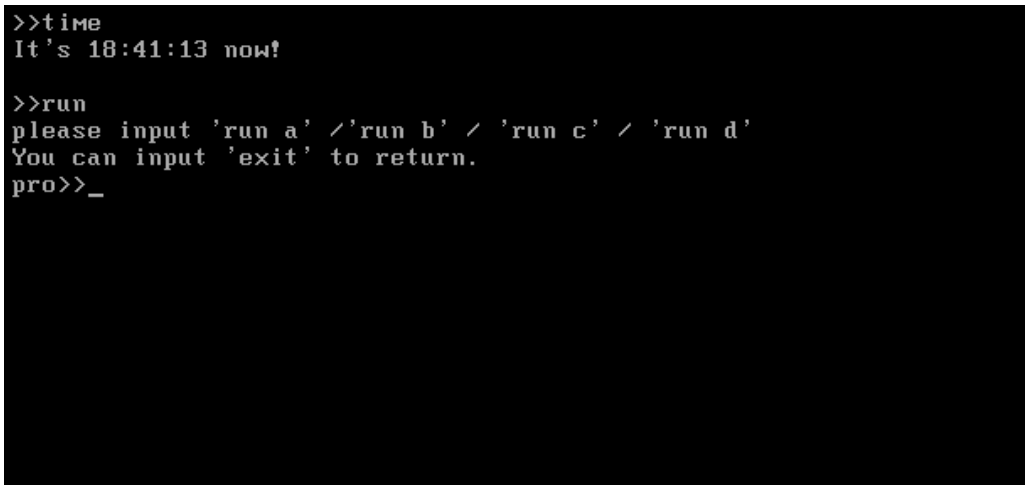
4、根据 help 界面的提示，先输入 dir 命令，看到四个用户程序的信息。



5、输入 cls 指令，可以看到已经清屏，光标在第一行第一列，可以输入命令。

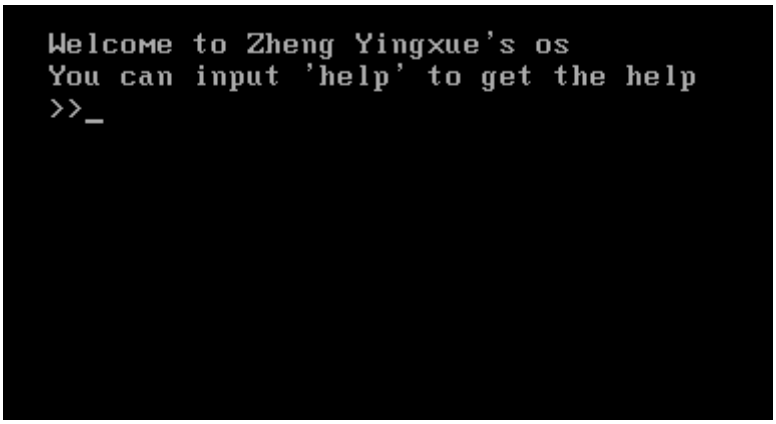
6、输入 time 指令，可以看到当前的时间。

7、最后我们运行用户程序，先在主界面上输入 run，可以看到进入了运行的子界面并有提示如何输入。在此子界面输入 exit 可以返回到主界面。

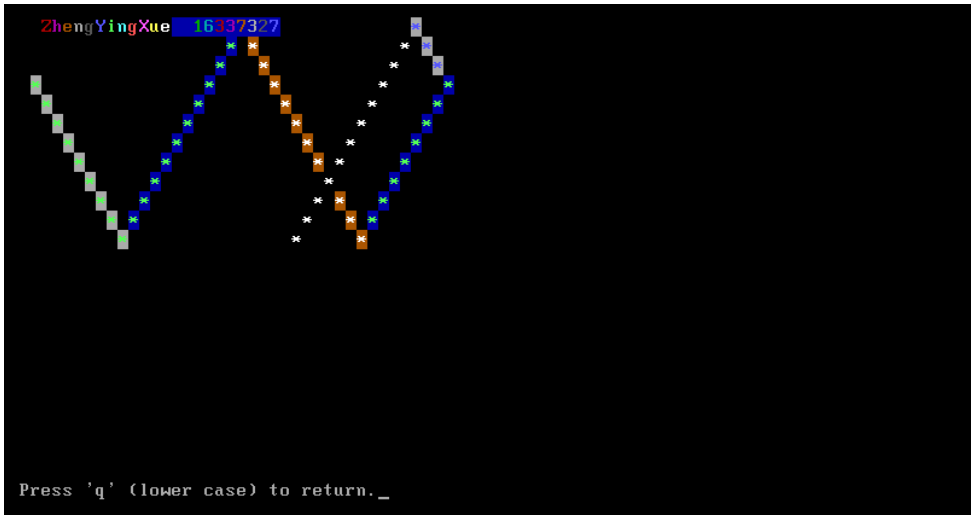


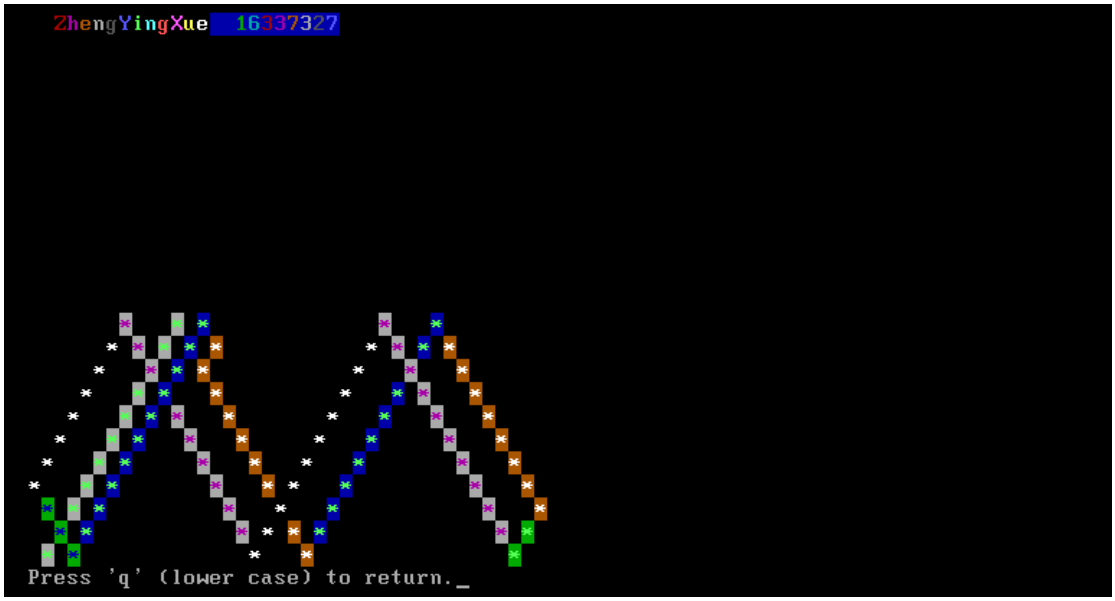
8、输入 run a，可以看到第一象限飞翔的字符和个人信息。左下角有提示按下 q 键即可退出。

9、按下 q 键后，退回到主界面。



10、同理可以运行用户程序 b\c\d。





四、实验创新点

因为能力有限，本次实验除了老师要求的功能以外，我只增加了一个获取时间的功能。主要是通过 1Ah 号中断获取时间。值得注意的是在此中断中时分秒以 BCD 码的形式存入，如果要正确显示在屏幕上，还需要进行 BCD 码和十进制的转换。与此类似的还有日期的获取，但是跟时间获取没有什么差别，就不重复实现了。

不过在查询中断大全时，除了这种“中规中矩”的中断，还有一些很有意思

的中断，比如 01 功能号的 1Ah 中断可以获取时钟的滴答声。真的是很有意思，我想这样的话可以写一个拟态时钟。但是由于时间问题并没有来得及实现。因此我只实现了一个时间获取的功能。

五、实验总结

这次实验可以说是一波三折，好多次遇到迷之错误，导致我在这个实验上花费的时间非常多。

在一开始写 C 程序部分时，我光修改语法错误就花了一个小时——最近一段时间 python 写得比较多，习惯使然，我大部分的语句都没有加分号……（好丢脸）我用老师给的 tcc 指令显示的错误信息不像写 C 的 IDE 那样高级，少了分号也不会具体提示，于是我走上了“靠眼找错”的漫漫之路。

tasm 不像 nasm 可以直接在我们的电脑上执行，它是 16 位的，与我们现在 64 位电脑不兼容。于是我下载了一个 win7 的 32 位虚拟机，不仅可以直接编译，还可以直接执行.com 文件。要知道在前两个实验里，因为 64 位的电脑不能直接运行.com 文件，我每次测试都要把它放到扇区里再开裸机。前两个实验比较简单，所以这么做还不会浪费太多时间，可是这个实验需要调整的地方太多了，于是装个 32 位的虚拟机对我来说可以说是最好的选择。所以本次实验，我都是在这个 win7 32 位的虚拟机上编译完成，只要在扇区分配好后从虚拟机拖动到电脑，就可以在裸机上直接运行，出错就继续返回 win7 虚拟机修改。如果扇区有更新的操作仍然可以将新文件拖动到电脑上，毕竟新文件会覆盖旧文件嘛！

后来老师发给我们 nasm 的相关指令时，我考虑到自身能力不足的问题，在试了两天 nasm 发现很多自己没法解决的问题后我还是选择了老师教授的 tasm 和 tcc 方法。

老师的模板中，org 100h 引起了我的注意。在查阅资料后，我了解到.com 文件是没有可执行文件头的，.com 文件载入内存后的起始偏址就是 100h，前面的 100h 字节是该程序的 PSP 部分。了解到这个原理后，我一开始便用一个简单的.com 程序实验了一下，在运行成功后，我便没有再动引导程序和调用了

kliba 库的 myos 主程序，至此我不再因为这个问题出过错。

老师提供了 kliba 的 asm 文件，里面展示了模块的语法格式。当我在 C 程序里调用老师提供的代码时会出错（果然是不能偷懒的）。中间还出了一个关于 CS：IP 低于初始值的错误，但是把原码给 RA 姐姐和老师看都没法解决，老师说可能是我动了汇编的模板。于是我开始把一切推倒重来，仔细阅读该文件里的每一块代码，先放弃 C 程序的编写，先简单地在 C 程序里调用每一个函数慢慢调试，通过一个函数再注释掉源代码单独测试下一个模块的代码，果然这样效率高了很多（由此可见我大概之前的错误真的是不小心动了汇编模板的某个地方）。我发现了 kliba 文件中大部分代码都有删改的空间。比如该文件的 printf 部分，输上一行两行字符还可以，但是多了之后就会乱码蹭蹭蹭冒出来。我又注意到该文件里有 printchar 的代码，索性放弃了修改 printf 代码，而是直接删掉它，在 C 程序里遍历字符串数组，遍历时每次单个字符打印即可。

有了上一个实验的铺垫，在运行用户程序的时候，一切都比较顺利，但是当我退出回到主界面时，显示字符跑到了中间。原来在返回时调用的清屏代码直接调用了 kliba 文件中的 cls 代码，可是它只起到了清屏的作用，光标还是在原来的位置。我查询 BISO 中断大全，得知功能号为 02h 的 10h 中断可以实现光标位置的设置。于是我在 cls 的代码中添加了此中断，并设置在每次清屏后将光标移动到第一行第一列。这个问题解决了。

这个实验中我还学会了用批处理来执行命令。前两个实验的编译命令很短，每次编译时我也就直接输入命令。但是这次实验中，ta 和 tc 倒是还好，tlink \3 \t ……这一条命令实在是太长了。在老师给的 nasm 和 gcc 文件中，我发现老师给了一个 1.bat 的批处理文件，在 cmd 下运行后发现是老师把三个指令存入了这个文件（nasm 环境下那些编译指令更长），我也有样学样地建立了一个批处理文件，里面输入 ta、tc 和 tlink 指令，每次编译只需要运行这个批处理文件就行了，这为我的实验节省了大量的时间。

这个实验里我最有感触的就是了解了 C 程序传递的参数在汇编语言里怎么使用的原理。原来参数传递时是要从右往左压栈（因为顺序问题我也吃过亏），如果汇编程序中需要对其进行操作时，使用 bp 指针即可读取到正确的参数地

址，再进行相关的操作。但是需要注意的是，在这一段程序运行以前把相关的寄存器已经压入了栈中，所以 bp 指针的加量需要考虑到这些在参数压栈后又压栈的寄存器，若还是 bp+2 则只能读到最后一个压栈的寄存器中的值了。这一点一定要注意。

本次实验中还存在的不足是，我在用户程序跳转回主页面时，其实是想实现跳转回运行子页面的，但由于子页面只是 C 程序里 if 的一个分支，我还不知道怎么确定这段代码执行的位置，所以我只好让用户程序返回时直接返回 1000h：100h 处存放的主程序。

从本次实验中我的做法可以看出，本次实验我是采取自底向上的方法的。先一个功能、一个功能地实现，出现问题就一个问题、一个问题地解决，当所有问题都解决了之后，再编写大的框架，如果此时出错，便只能出错在框架中。其实我在一开始，是把 C 程序写好，再把汇编程序写好，此时联合编译，问题一大堆，我一个一个去解决非常麻烦，因为这中间可能存在着因为某两个错误有联系而出现新的错误的情况。所以，这种先局部再整体的方法，对能力不是很强的我来说是很有效的，以后的实验我将继续采取这种做法，我相信它会使我的做实验的效率提高不少。