

实验报告——实验四

姓名：任文頔

学号：14322181

院系：数据科学与计算机专业

专业、年级：14级计算机科学与技术

指导教师：凌应标

【实验题目】

在FAT12盘中的引导操作系统

【实验目的】

- 1、认识并掌握FAT12文件系统的结构
- 2、学会用程序创建完整的FAT12盘结构，通过编写新的引导扇区，在根目录中搜索文件并加载，从而不再通过扇区储存文件。

【实验要求】

- 1、用C语言+汇编语言编写一个程序，编译产生一个COM格式的可执行程序，用于显示一个FAT12的BPB和EBPB内容。这个程序作为一个用户程序，保存在你的映像文件中的FAT12文件系统中。
- 2、用C语言+汇编语言编写一个程序，编译产生一个COM格式的可执行程序，用于列出一个FAT12的根目录中所有文件信息，如文件名、文件大小、文件创建日期等等。这个程序作为一个用户程序，保存在你的映像文件中的FAT12文件系统中。
- 3、用C语言+汇编语言编写一个程序，编译产生一个COM格式的可执行程序，用于列出一个FAT12的根目录中一个文件分配到的所有簇号和对应的逻辑扇区号等等，如果文件太大，只要列出前5个簇即可。这个程序作为一个用户程序，保存在你的映像文件中的FAT12文件系统中。
- 4、修改你的操作系统原型3，将内核执行体用一个文件形式存放在映像盘中，同时按FAT12盘格式的要求，修改引导程序，从FAT12根目录中加载内核
- 5、修改内核程序，实现用命令行从FAT12中加载任意一个用户程序。

【实验方案】

1 硬件或虚拟机配置方法

系统环境：Linux Ubuntu 14.04

虚拟机配置方法：在Linux下VMware Player是收费软件，因此选择免费的VirtualBox软件。

配置方法：操作系统选择其他，选择从软盘启动，添加自己的软盘，虚拟机名称为14322181renwendi

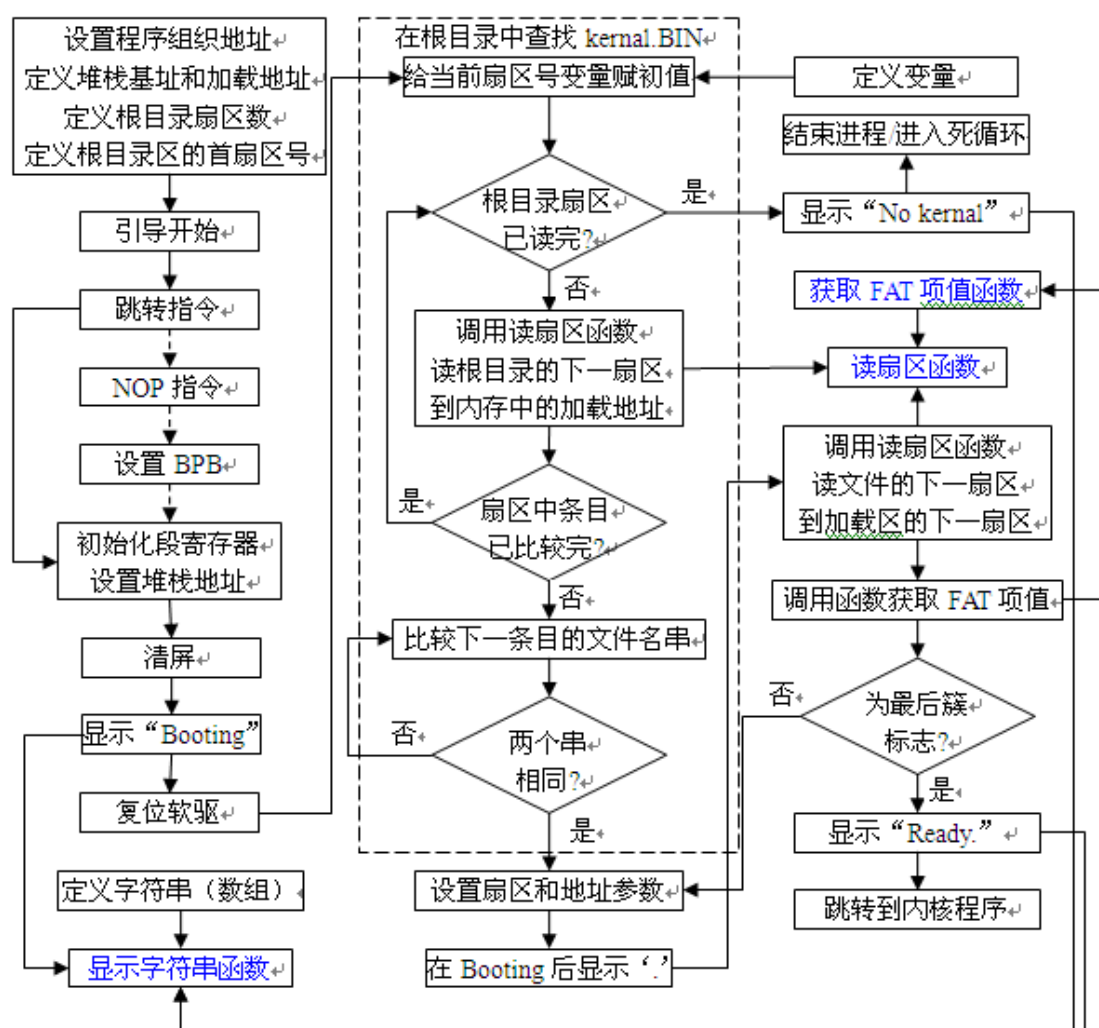
2 软件工具与作用

- (1) 汇编语言编译器NASM：针对Intel x86架构的汇编与反汇编程序
- (2) C语言编译器GCC：由 GNU 开发的编程语言编译器。
- (3) 编辑器Vim：功能强大、高度可定制的文本编辑器。
- (4) 虚拟机软件Bochs：自己编写的操作系统的测试环境
- (5) 软盘创建工具:dd 软盘写入工具:Linux 自带挂载命令

3 方案的思想

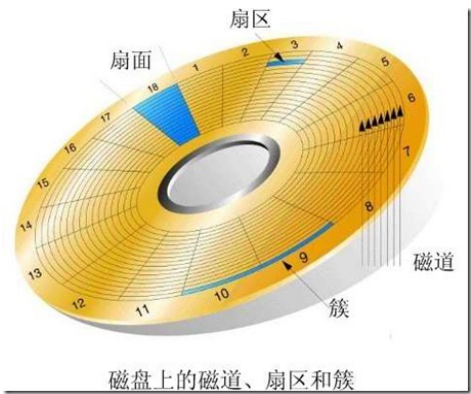
在引导扇区中增加汇编代码，实现搜索软盘根目录中的操作系统内核文件 kernel.bin / kernel.com，并将其加载到内存，最后将控制权交给内核程序。文件加载程序的逻辑结构图：

4 相关知识原理



(1) FAT12软盘结构

1.44MB软盘的格式：2个磁头（head） / 盘面（side）（编号0~1）、每盘面 / 磁头80个柱面（cylinder） / 磁道（track）（编号0~79）、每个柱面有18个扇区（sector）（编号1~18）、每个扇区有512（=200h）个字节（byte），所以软盘的容量为：2 磁头 * 80 柱面 * 18 扇区 * 512 B = 2880 扇区 * 512B = 1474560B = 1440KB = 1.44MB



对1.44MB软盘，可采用FAT12文件系统，其引导扇区中没有分区信息，但在引导扇区和FAT区之后有根目录区（224条目*32B = 7168B = 14扇区）。

FAT12的每个FAT项占12位（1.5B），每个簇只有一个扇区。整个软盘2880个扇区，全部FAT项需要2880*1.5B = 4320B = 8.44个扇区，所以每个FAT表的大小设为9个扇区。

1.44MB软盘的FAT12文件系统结构

起始扇区号	占用扇区数	内容	CHS参数
0	1(512B)	引导程序	起：0/0/1、止：0/0/1
1	9(4608B)	FAT1	起：0/0/2、止：0/0/10
10	9(4608B)	FAT2	起：0/0/11、止：1/0/1
19	14(9728B)	根目录	起：1/0/2、止：1/0/15
33	2847(1457664B)	文件数据区	起：1/0/16、止：79/1/18

(2) FAT12分区引导扇区的结构

偏移	长度	内容
0x0000	3B	跳转指令（jmp short 地址 nop = EB ??[一般为3C] 90）
0x0003	8B	OEM名串（如“MyOS 1.0”）
0x000B	25B	BPB

0x0024	26B	扩展BPB
0x003E	384B	引导程序代码（对软盘，大小为448B）
0x01BE	64B	磁盘分区表（软盘无，可用作代码续区）
0x01FE	2B	有效结束标志（55 AA）

FAT12的BPB结构（25B）

偏移	长度	内容	1.44M软盘值	10M硬盘值
0x0B	2B	每扇区字节数（一般为512）	200H=512	200H=512
0x0D	1B	每簇扇区数	1	8
0x0E	2B	保留扇区数[含本扇区]（≥1）	1	1
0x10	1B	FAT数（一般为2）	2	2
0x11	2B	最大根目录项数	E0H=224	200H=512
0x13	2B	总扇区数（<64K，容量<32MB）	B40H=2880	4EC0H=20160
0x15	1B	介质描述符（盘类型，高密度3.5寸软盘为0xF0、硬盘和U盘为0xF8）	F0H	F8H
0x16	2B	FAT占扇区数	9	8
0x18	2B	每道扇区数	12H=18	3FH=63
0x1A	2B	磁头数	2	10H=16（4）
0x1C	4B	隐藏扇区数（对无分区的介质必须为0）	0	0
0x20	4B	总扇区数（≥64K，容量≥32MB）	0	0

FAT12的EBPB结构（26B）

FAT12/16 偏移	长度	内容	1.44M软盘值	10M硬盘值
0x24	1B	驱动器号（软0/硬0x80）	0	80H
0x25	1B	保留（一般为0）	0	0
0x26	1B	扩展引导标签（一般为0x29）	29H	29H
0x27	4B	卷ID（序列号，随机数）	如12345678	如23456789
0x2B	11B	卷标（不足补空格）	如“NO NAME ”	如全空格符

0x36	8B	文件系统类型（不足补空格）	“FAT12 ”	“FAT12 ”
------	----	---------------	----------	----------

(3) FAT12根目录

目录表由若干描述文件或子目录的条目（entry，项）组成，目录表中的每个条目占32个字节，保存着文件或子目录的名字、属性、时间、首簇号、大小等信息。

FAT12软盘的根目录位于第二个FAT之后，每个条目占用32B，其格式如下：

文件目录条目的格式

偏移(H)	长度(B)	内容
00	11	文件名8B（不足补空格）、扩展名3B（不足补空格），尾部空格被忽略
0B	1	文件属性
0C	10(1)	保留（WinNT使用其中的第3和4位）
0D	1	创建时间的最小时间分辨率，以10ms为单位，取值0~199 从DOS 7.0的VFAT起使用
0E	2	创建时间，从DOS 7.0的VFAT起使用
10	2	创建日期，从DOS 7.0的VFAT起使用
12	2	最后访问日期，从DOS 7.0起使用（需在CONFIG.SYS中激活对应驱动器的ACCDATE）
14	2	开始簇号的两个高位字节（FAT32）
16	2	最后写入时间（0~4位：秒/2[0~29]、5~10位：分[0~59]、11~15位：时[0~23]）
18	2	最后写入日期（0~4位：日[1~31]、5~8位：月[1~12]、9~15位：年[0=1980~127=2107]）
1A	2	开始簇号（的两个低位字节，FAT32）（卷标、指向FAT12/16根目录的子目录“..”、空文件条目的开始簇号必须为0）
1C	4	文件大小（字节数）

其中：

- 灰色文字部分是高版本（FAT32和DOS 7.0）新增的，现在可暂不考虑
- 合法的DOS文件名和扩展名字符（ASCII字符）：
 - 大写英文字母：A-Z（用户输入的小写字母必须转换成大写字母后再存储）
 - 数字：0-9

- 空格（文件名和扩展名尾部的填充空格，不算作文件名和扩展名的组成部分）
- 标点字符：!#\$%&()-@^_`{|}~'
- 字节高位（第7位）为1的字符（字符值为128-255）

非法的字符包括：

- ◆ 控制字符（值0-32、127[DEL]）
 - ◆ 标点字符：”*+.,/;<=>?[\\|
 - ◆ 小写英文字母（转换成大写字母后存储，长文件名中允许使用小写字母）
- VFAT（Virtual FAT，虚拟FAT）使用传统的FAT结构来保存长文件名（Long File Names, LFN），从Windows 95/NT 3.5起支持。

条目第一个字节特殊值的含义

值 (H)	00	05	2E	E5
含义	条目未用且可用	值实际为E5H	点条目（.或..）	条目被删且不可用

文件属性的标志位

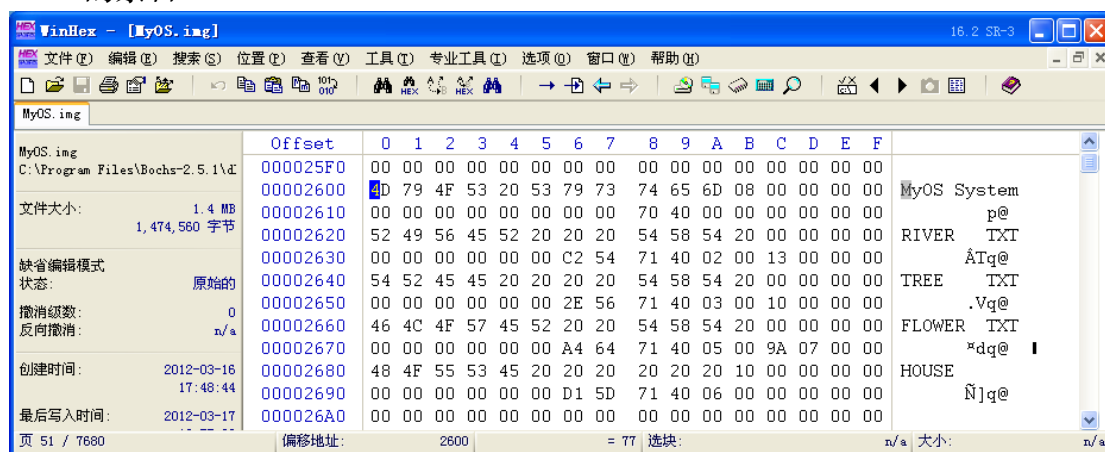
位	7	6	5	4	3	2	1	0
掩码(H)	80	40	20	10	08	04	02	01
含义	未用	设备	档案	子目录	卷标	系统	隐藏	只读

普通文件的属性标志值=0或20h。

属性值0x0F（1111）表示长文件名条目（占用20个条目空间[640B]，其中的每个条目[32B]用26B存13个UTF-16字符，20个条目可存255[260]个字符）。

具体计算实例：

在偏移量2600h处（根目录区的开始地址），可以看到在原来的卷标条目后增加了4个新条目，分别是描述文本文件river.txt、tree.txt和flower.txt，子目录house的条目。



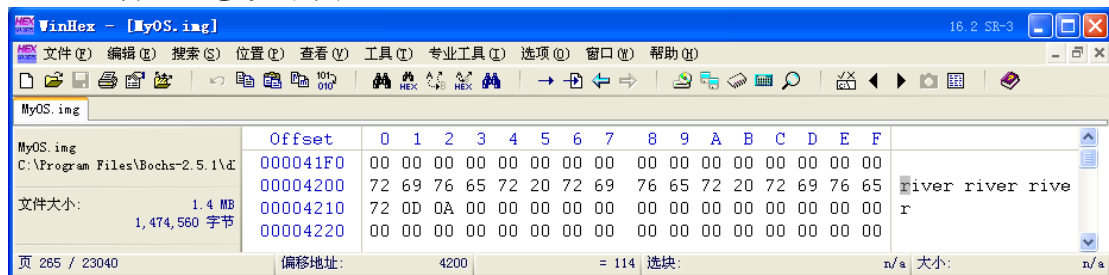
根目录区中的文件和子目录条目

下面是这些条目各字段的取值（其中“□”表示空格符20h）：

文件目录条目取值

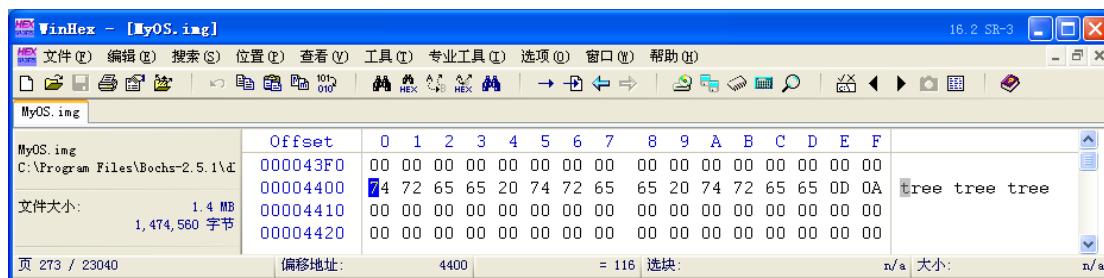
偏移(H)	长度(B)	内容	river.txt	tree.txt	flower.txt	house
00	11	文件名 扩展名	RIVER□□ □ TXT	TREE□□□□ □ TXT	FLOWER□□ □ TXT	HOUSE□□□ □□□
0B	1	文件属性	20h	20h	20h	10h
0C	10	保留	全0	全0	全0	全0
16	2	最后写入时间	54C2h	562Eh	64A4h	5DD1h
18	2	最后写入日期	4071h	4071h	4071h	4071h
1A	2	开始簇号	0002h	0003h	0005h	0006h
1C	4	文件大小	00000013h	00000010h	0000079Ah	00000000h

- 文件大小：river.txt的为13h=19B、tree.txt的为10h=16B、flower.txt的为79Ah=1946B, 子目录house的文件大小为0
- 开始簇号
 - river.txt文件的开始簇号为2，因为FAT中的0号和1号簇已经被保留的介质描述符和文件结束符占用，所以首个可用簇号就是2，它对应于数据区的首个扇区（地址为4200h，计算方法参见3.3的第4小节最后部分），参见下图：



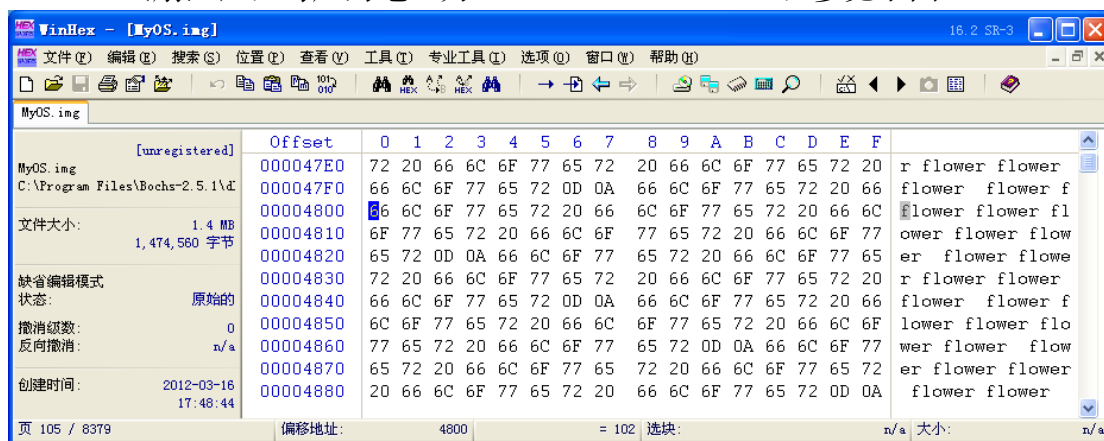
river.txt文件的数据区

- tree.txt文件的开始簇号为3，对应于磁盘数据区的第 $(3-2=)$ 1个簇，因为软盘B中一个簇中只有一个扇区（每个扇区512B），所以tree.txt文件的数据区应该从磁盘数据区的第1个扇区开始，它对应的地址为 $4200h + 1 \times 512 = 4400h$ ，参见下图：



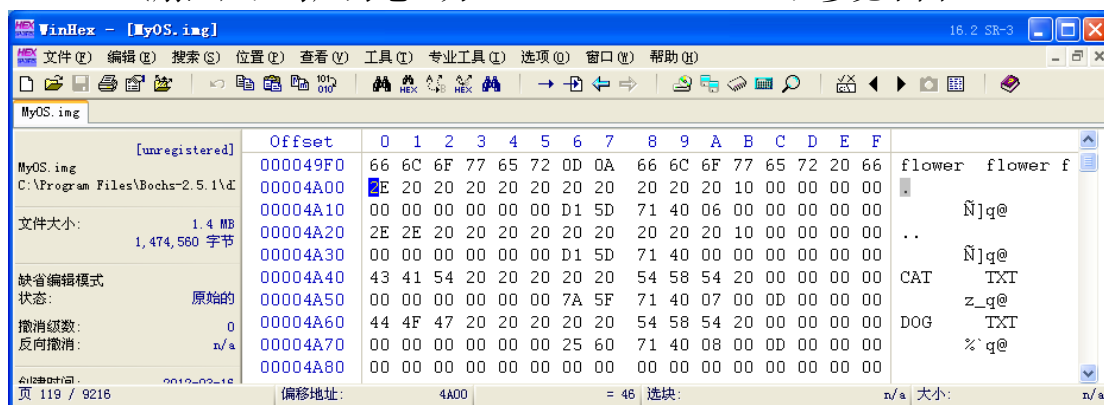
tree.txt文件的数据区

- flower.txt文件的开始簇号为5，对应于磁盘数据区的第（5-2=）3个簇（扇区），对应的地址为4200h + 3*512 = 4800h，参见下图：



flower.txt文件数据的开始扇区

- house子目录的开始簇号为6，对应于磁盘数据区的第（6-2=）4个簇（扇区），对应的地址为4200h + 4*512 = 4A00h，参见下图：



子目录house的数据区

可见house的数据区包含另一个目录表结构，里面有代表当前目录的“.”点文件、代表父目录的“..”点文件、两个文本文件cat.txt（大小为0Dh=13，首簇号为7）和dog.txt（大小也为0Dh=13，但首簇号为8）。

(4) FAT表

FAT (File Allocation Table, 文件分配表) 是映射到分区中每个簇 (cluster) 的项 (entry, 条目) 列表, FAT12/16/32的每个项占12/16/32位, FAT项取值的含义参见下表:

FAT项值 (H)			
FAT12	FAT16	FAT32	含义
000	0000	?0000000	空闲簇
001	0001	?0000001	保留簇
002~FEF	0002~FFEF	?0000002~?FFFFFFEF	被占用簇, 值为下一簇的地址
FF0~FF6	FFF0~FFF6	?FFFFFFF0~?FFFFFFF6	保留值
FF7	FFF7	?FFFFFFF7	坏簇
FF8~FFF	FFF8~FFFF	?FFFFFFF8~?FFFFFFF	文件最后簇

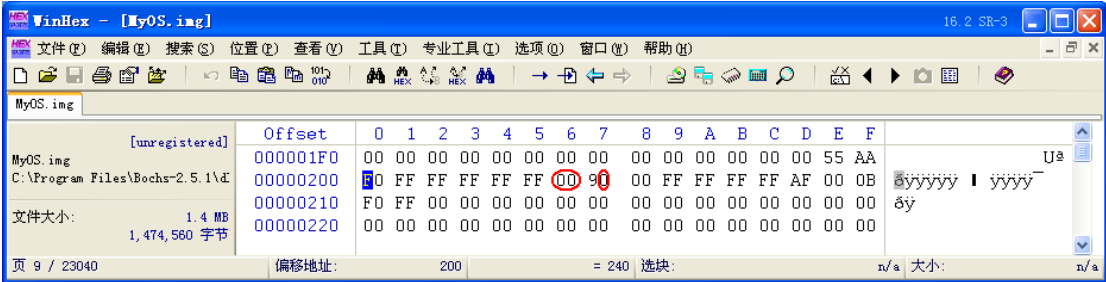
其中, FAT32只使用了32位中的低28位, 高4位是保留位 (在表中用?表示), 一般置为0。

FAT表的前两个项保存特殊的值:

- 项0: 低8位 (首字节) 为介质描述符 (软盘为F0h、硬盘和U盘为F8h)、其余位 (FAT32的高4位除外) 全为1 (FAT12: Fh、FAT16: FFh、FAT32: 0FFFFFFh)
- 项1: 结束簇标记 (通常取值为——FAT12: FFFh、FAT16: FFFFh、FAT32: 0FFFFFFFh)

具体计算实例:

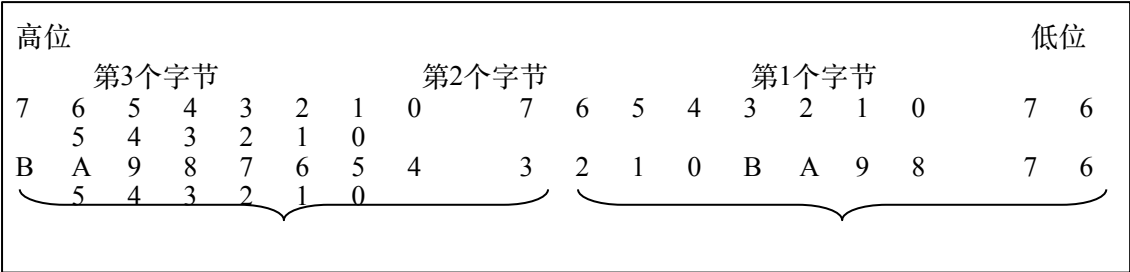
flower.txt文件的首个簇 (扇区) 的前后都是flower字符串 (前面的扇区为 [编号为04的] 空闲簇扇区, 对应的FAT项值为000h [参见下图中的红圈]。里面的flower串, 是该文件被加长前遗留下来的)。因为flower.txt文件大于512B (实际为79Ah=1946B), 一个扇区装不下, 需要多个扇区 (实际需4个)。在 (起始地址为200h的) FAT#1中, 这些扇区 (簇) 不一定是连续分布的。



软盘MyOS.img的FAT#1开始处

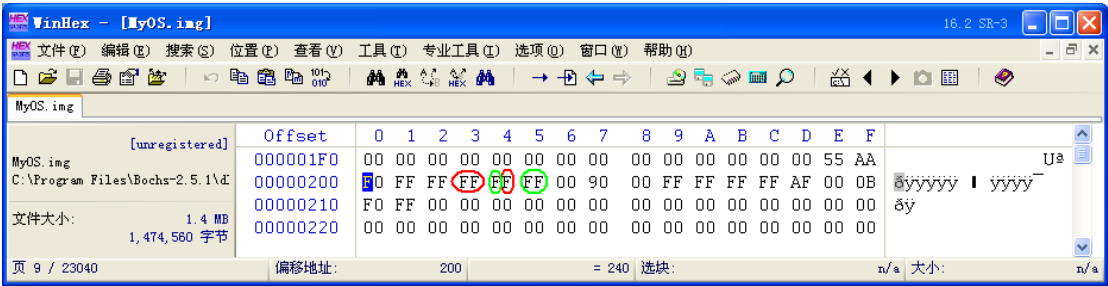
FAT12分区的每个FAT项占12位（=1个半字节），不是字节的整数倍，奇偶项的位置不同，计算起来非常麻烦。

奇偶数FAT项



从上面的FAT#1图中可见，保留的头两个FAT项中（F0 FF FF），第1个FAT项（0号项，偶数）的值为FF0h，其低位字节的值为F0h，是磁盘的介质描述符；高4位为Fh，是规定的填充值。第2个FAT项（1号项，奇数）的值为FFFh，属于文件结束簇的标志之一，也是规定的取值。其实这两个FAT项的值是我们自己用汇编程序填写的。

river.txt文件的开始簇号为2（对应于第三个FAT项），对应FAT项的起始偏移地址为2*1.5=3，取值为FFFh（参见下图中的红圈，注意Intel CPU的多字节整数的低位字节在前，而一个字节里的低4位在后），为文件结束标记，即river.txt只占据1个簇（扇区）。



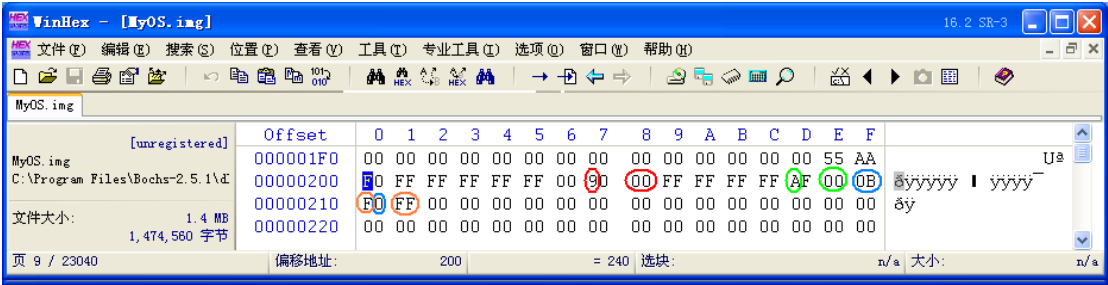
第3个和第4个FAT项的值

虽然river.txt文件的大小只有19B，但是因为文件存储的最小单位是簇，所以它也要占一整个簇（软盘一般一个簇只有1个扇区）。因此，采用大簇的分区（如NTFS的默认簇大小是4KB=8个扇区，最大可以是64KB=128个扇区），对小型文件是很大浪费，但对大型文件，可以加快读写速度。

tree.txt文件的开始簇号为3，对应的起始偏移地址为3*1.5=4.5=4处的高4位，（参见上图中的绿圈），项值为FFFh，也是文件结束标记。

flower.txt文件的开始簇号为5，对应的起始偏移地址为5*1.5=7.5=7处的高4位（参见下图中的红圈），项值为009h（即下一个簇号是9），簇的序号并不连续。这是因为flower.txt文件被我们加长过（原来的簇号为4和5）。而在此之前，第6号簇所对应的FAT项（值为FFFh），已经被被house子目录占据；第7和8号簇分别被house目录中的cat.txt和dog.txt占据。所以只好安排第9号簇给flower.txt文件了。

而第9簇中的项值（起始偏移地址为 $9 \times 1.5 = 13.5 = 13 = 0Dh$ 处的高4位，参见下图中的绿圈）为00Ah（即下一个簇号是10）、第10簇中的项值为00Bh（即下一个簇号是11）、第11簇中的项（参见下图中的橙圈）值为FFFh，表示文件到此结束。



flower.txt文件FAT项的值

5 程序流程

加载内核文件到内存的主要步骤：

1. 设当前扇区号为根目录的起始扇区号19
2. 调用读扇区函数（int 13h的02h功能）读根目录的当前扇区到内存中的加载地址
3. 依次比较当前扇区中各个文件条目（共16个）中的文件名串与“KERNEL.COM”
4. 若无相等的文件条目，前扇区号加1，转到2
5. 若找到相等的文件条目，从该条目获取起始簇号N
6. 计算出簇号N所对应的装载文件的扇区号（ $=N+31$ ）
7. 调用读扇区函数将装载文件的当前扇区读到内存中加载地址的当前扇区
8. 由当前簇号值N计算其对应FAT项在FAT表中的偏移地址D（ $=N \times 1.5B$ ），由D值计算出FAT项所在扇区的序号K（ $=1+D/512$ ）和偏移值O（ $=D \% 512$ ）
9. 调用读扇区函数将磁盘的（FAT表中的）K号和K+1号两个扇区（因为一个FAT项可能跨越两个扇区）读入内存缓冲区（8F000h）
10. 利用偏移值O获取FAT项值N（=文件下一个簇的序号）
11. 若 $N < FF8h$ ，则转到6
12. 若 $N \geq FF8h$ ，则文件已经读完，跳转到内核

6 程序关键模块

(1) 引导扇区boot.asm

直接使用了老师所给的参考代码LoadKnl.asm，修改了其中的三处。分别是：

(a) 程序开始的第一行：

%define _BOOT_DEBUG_ ;用于生成.COM文件易于调试

(修改: 此行需要注释掉, 这里是为了生成.com文件来调试, 而现在需要的是编译生成引导.bin文件)

(b) 程序第189行文件名处:

```
kernelFileName db " kernal BIN", 0 ; kernal.BIN之文件名
```

(修改: 删除文件名前面的空格, 并且将文件名改成自己内核的名字, 一共11字节。

```
kernelFileName db "KERNEL COM")
```

(c) 修改磁盘头, 加入个人信息

```
BS_OEMName DB 'MyOS-REN' ; OEM串, 必须8个字节, 不足补空格
```

```
BS_VolID DD 14322181h ; 卷序列号
```

```
BS_VolLab DB 'MyOS System'; 卷标, 必须11个字节, 不足补空格
```

```
BS_FileSysType DB 'FAT12 ' ; 文件系统类型, 必须8个字节, 不足补空格
```

(2) 内核程序

(a) 命令行部分, 汇编和c语言共同实现, 与上一次实验三的命令行基本一致, 区别是此次没有再加入时间和日期的显示, 内部指令只保留了cls (清屏), 新添加的内部指令为h (for help) 。

```
if( strcmp( key, "cls\0")){
    cls();
    return i;
}
if( strcmp( key, "h\0")){
    h();
    return i;
}
```

(b) 内部命令汇编实现

```
; -----
cls:  mov ah, 6
      mov al, 0          ; 滚动的文本行数 (0=整个窗口)
      mov bh, 0fh        ; 设置插入空行的字符颜色为黑底亮白字
      mov cx, 0          ; 窗口左上角的行号=CH、列号=CL
      mov dh, 24         ; 窗口右下角的行号
      mov dl, 79         ; 窗口右下角的列号
      int 10h
      ; 设置光标位置
      mov ah, 2          ; 功能号
      mov bh, 0          ; 第0页
```



```

        mov word [wRootDirSizeForLoop], RootDirSectors ; 根目录区剩余扇区数
                                           ; 初始化为14, 在循环中会递减至零
LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
        cmp word [wRootDirSizeForLoop], 0 ; 判断根目录区是否已读完
        jz LABEL_NOT_FOUND ; 若读完则表示未找到COM文件
        dec word [wRootDirSizeForLoop] ; 递减变量wRootDirSizeForLoop的值
        ; 调用读扇区函数读入一个根目录扇区到装载区
        mov ax, BaseOfLoader
        mov es, ax ; ES <- BaseOfLoader (4000h)
        mov bx, OffsetOfLoader ; BX <- OffsetOfLoader (100h)
        mov ax, [wSectorNo] ; AX <- 根目录中的当前扇区号
        mov cl, 1 ; 只读一个扇区
        call ReadSec ; 调用读扇区函数

        mov si, fnbuf ; DS:SI -> COM文件
        mov di, OffsetOfLoader ; ES:DI -> BaseOfLoader:0100
        cld ; 清除DF标志位
        ; 置比较字符串时的方向为左/上[索引增加]
        mov dx, 10h ; 循环次数=16 (每个扇区有16个文件条目)
LABEL_SEARCH_FOR_COM_FILE:
        cmp dx, 0 ; 循环次数控制
        jz LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR ; 若已读完一扇区 就跳到
        下一扇区
        dec dx ; 递减循环次数值
        mov cx, 11 ; 初始循环次数为11
LABEL_CMP_FILENAME:
        repe cmpsb ; 重复比较字符串中的字符, CX--, 直到不相等
        或CX=0
        cmp cx, 0
        jz LABEL_FILENAME_FOUND ; 如果比较了11个字符都相等, 表示找到
LABEL_DIFFERENT:
        and di, 0FFE0h ; DI &= E0为了让它指向本条目开头 (低5位清
        零)
        ; FFE0h = 1111111111100000 (低5位=32=目录
        条目大小)
        add di, 20h ; DI += 20h 下一个目录条目
        mov si, fnbuf ; SI指向装载文件名串的起始地址
        jmp LABEL_SEARCH_FOR_COM_FILE; 转到循环开始处

LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
        add word [wSectorNo], 1 ; 递增当前扇区号
        jmp LABEL_SEARCH_IN_ROOT_DIR_BEGIN

LABEL_NOT_FOUND:
        pop es ; 恢复ES
        call showwrong ; 显示字符串

```

ret

; 下面将COM文件加载到内存

LABEL_FILENAME_FOUND:: 找到 COM文件后便来到这里继续

; 计算文件的起始扇区号

mov ax, RootDirSectors ; AX=根目录占用的扇区数

and di, 0FFE0h ; DI -> 当前条目的开始地址

add di, 1Ah ; DI -> 文件的首扇区号在条目中的偏移地址

mov cx, word [es:di] ; CX=文件的首扇区号

push cx ; 保存此扇区在FAT中的序号

add cx, ax ; CX=文件的相对起始扇区号+根目录占用的扇区

数

add cx, DeltaSectorNo ; CL <- COM文件的起始扇区号(0-based)

mov ax, BaseOfLoader

mov es, ax ; ES <- BaseOfLoader (COM程序基址
=4000h)

mov bx, OffsetOfLoader ; BX <- OffsetOfLoader (COM程序偏移地址
=100h)

mov ax, cx ; AX <- 起始扇区号

LABEL_GOON_LOADING_FILE:

push bx ; 保存COM程序偏移地址

mov cl, 1 ; 1个扇区

call ReadSec ; 读扇区

; 计算文件的下一扇区号

pop bx ; 取出COM程序偏移地址

pop ax ; 取出此扇区在FAT中的序号

call GetFATEntry ; 获取FAT项中的下一簇号

cmp ax, 0FF8h ; 是否是文件最后簇

jae LABEL_FILE_LOADED ; ≥FF8h时跳转, 否则读下一个簇

push ax ; 保存扇区在FAT中的序号

mov dx, RootDirSectors ; DX = 根目录扇区数 = 14

add ax, dx ; 扇区序号 + 根目录扇区数

add ax, DeltaSectorNo ; AX = 要读的数据扇区地址

add bx, [BPB_BytsPerSec] ; BX+512指向COM程序区的下一个扇区地址

jmp LABEL_GOON_LOADING_FILE

; 下面跳转执行COM程序

LABEL_FILE_LOADED:

add sp, 4 ; 弹出call指令压栈的返回地址和保存的ES

jmp BaseOfLoader:OffsetOfLoader ; 这一句跳转到已加载到内存中的
; COM文件的开始处, 开始执行 COM文件的代码。

(d) 读扇区函数

读磁盘是本实验在汇编中需要实现的非常重要的一个函数，并且被多次使用，在用户程序中也要用到。

其中最麻烦的是将（起始的）逻辑扇区号LBA（Logical Block Addressing，逻辑块寻址）转换为传统的磁盘寻址方式CHS（Cylinder/Head/Sector，柱面/磁头/扇区）。

下面是LBA与CHS的转换公式：

CHS→LBA：

$$LBA = ((C * HPC) + H) * SPT + S - 1$$

LBA→CHS：

$$\begin{cases} C = (LBA / SPT) / HPC \\ H = (LBA / SPT) \% HPC \\ S = (LBA \% SPT) + 1 \end{cases}$$

其中：HPC = Heads Per Cylinder，每柱面磁头数（fat12中是2）；SPT = Sectors Per Track，每磁道扇区数（fat12中是18）。

```
-----
; 作用：从第 AX 个扇区开始，将 CL 个扇区读入 ES:BX 中
ReadSec:
; -----
; 怎样由扇区号求扇区在磁盘中的位置 (扇区号->柱面号、起始扇区、磁头号)
; -----
; 设扇区号为 x (= AX)
;          ┌ 柱面号 C = y / 磁头数
;   x      └ 商 y ┘
; ----- => ┘ ┌ 磁头号 H = y % 磁头数
; 每磁道扇区数  |
;          └ 余 z => 起始扇区号 S = z + 1
push cx          ; 保存要读的扇区数 CL
push bx          ; 保存 BX
mov  bl, [secspt]; BL(= 磁道扇区数) 为除数
div  bl          ; AX/BL, 商 y 在 AL 中、余数 z 在 AH 中
inc  ah          ; z ++ (因磁盘的起始扇区号为 1), AH = 起始
扇区号
mov  cl, ah      ; CL <- 起始扇区号 S
mov  ah, 0       ; AX <- y
mov  bl, [heads] ; BL(= 磁头数) 为除数
div  bl          ; AX/BL, 商在 AL 中、余数在 AH 中
mov  ch, al      ; CH <- 柱面号 C
mov  dh, ah      ; DH <- 磁头号 H
; 至此, "柱面号、起始扇区、磁头号"已全部得到
pop  bx          ; 恢复 BX
pop  ax          ; AL = 恢复的要读的扇区数 CL
```



```

        mov    dl, [drvno]    ; 驱动器号
.1: ; 使用磁盘中断读入扇区
        mov    ah, 2          ; 功能号 (读扇区)
        int     13h           ; 磁盘中断
        jc .1    ; 如果读取错误, CF会被置为1, 这时就不停地读, 直到正确为止
        ret

```

(e) 获取FAT项中的下一簇号GetFATEntry

在加载用户程序中用到了这个比较重要的函数, 为了得到下一个的 12 位簇号, 需要先读取 16 位数据, 然后对 16 位的数据进行处理。

有如下计算公式:

当簇号为偶数时: 下一个簇号 = (当前簇号/2 * 3), 取低 12 位

当簇号为奇数时: 下一个簇号 = (当前簇号/2 * 3)+1, 取高 12 位

取高低位通过移位和屏蔽:

取低12位: 簇号 = 簇号 & 0x0fff

取高12位: 簇号 = (簇号 & 0xfff0) >>4

(3) 用户程序list.com

该程序用户显示用户可选择运行的文件, 在之前的h (for help) 中的提示串所说的就是You can enter list to choose a program, list就是此处的用户程序。

考虑该程序可以在c语言中设置一个512b大小的数组, 来存储每次调用读扇区函数之后读到的数据, 然后对数据进行处理。也可以在汇编中, 设置一个512b大小的缓冲区, 对数据进行处理。个人感觉需要c语言进行循环和判断的复杂程度与在汇编中差不多, 并且要多次调用汇编中显示字符串的函数, 因此就直接在汇编中实现了这个用户程序。之前也有考虑将其直接设置为内部命令, 但是觉得这样内核看起来太繁杂, 就单独拿出来做成了.com程序。

核心代码是两层循环, 三次判断。

两层循环是循环读入一个根目录扇区到缓冲区和在当前扇区中循环16次逐个检查文件目录项, 第一层循环也可不要, 直接一次读入根目录共占用的14个扇区。在现有内存较为充足的情况下也是可以实现的, 但是在真正的操作系统中这样占用内存是不太优化的。

三次判断是判断是否为文件条目, 若不是则指向下一个目录条目开始处, 继续循环。

值 (H)	00	05	2E	E5
含义	条目未用且可用	值实际为E5H	点条目 (.或..)	条目被删且不可用

(0开始的为空项、E5h开始的为已删项、属性低4位全1的为长文件名项或系统占用项、卷标项的属性3号位为1)

begin:

```

; 下面在磁盘根目录中寻找文件目录条目
searchrdir: ; 搜索根目录循环 (逐个读入根目录扇区)
    cmp     word [nsec], 0          ; 判断根目录区是否已读完
    jz      exit                   ; 若读完则退出
    dec     word [nsec]             ; nsec--
    ; 调用读扇区函数读入一个根目录扇区到缓冲区
    mov     bx, Sector              ; BX = Sector
    mov     ax, [isec]              ; AX <- 根目录中的当前扇区号
    mov     cl, 1                   ; 读一个扇区到缓冲区
    call    ReadSec
    mov     di, Sector              ; ES:DI -> Sector
    mov     word [i], 10h           ; 循环次数=16
searchfi: ; 搜索文件项循环 (在当前扇区中逐个检查文件目录项)
    cmp     word [i], 0             ; 循环次数控制
    jz      nextsec                 ; 若已读完一扇区, 跳到下一扇区
    dec     word [i]
    ; 判断是否为文件条目
    cmp     byte [di], 0
    jz      notfind
    cmp     byte [di], 0E5h
    jz      notfind
    cmp     byte [di + 11], 0Fh
    jz      notfind

    ; 显示文件名串
    push    dx
    mov     dh, [Ins]
    mov     dl, 0
    mov     cx, 11
    ; 显示文件名串
    mov     bp, di                  ; BP=文件名字符串的起始地址
    call    DispStr                  ; 调用显示字符串例程 (较为简单, 不再列出)
    inc     word [Ins]               ; 当前屏幕上的文件条目数Ins++
    pop     dx

notfind:
    add     di, 20h                 ; DI += 20h 指向下一个目录条目开始处
    jmp     searchfi
nextsec:
    inc     word [isec]              ; 递增当前扇区号
    jmp     searchrdir              ; 继续搜索根目录循环

exit: ; 终止程序, 返回
    mov     ax, 4c00h
    int     21h
Sector: ; 定义缓冲区, 用于存放从磁盘读入的扇区
    resb    512
(4) 用户程序的返回

```

在之前的实验中，用户程序使用扇区储存，系统内核的基地址和用户程序的基地址相同,所以直接使用 call和ret,不会出现问题。

这次由于内核基地址是 0x9000,用户程序是 0x4000,使用近调用,会出现问题。所以需要使用远调用。

远调用有如下几个方法：在 nasm 里,远调用可以通过 call far ‘基地址’:‘偏移地址’实现。

另外一个方法是只是用 retf。根据 retf 的返回原理,可以这样实现远调用。

本实验中我使用的是21h号中断来返回（此中断在DOS中，是系统调用的入口，其中的AH=4Ch功能是：结束程序的运行，返回DOS，返回值为AL一般为0）。

```
mov ax,4c00h
int 21h
```

(5) 用户程序bpb.com

该用户程序用于显示软盘的BPB和EBPB信息。程序的难点在于获取变量数据并显示。我采取的方法是，先通过读扇区函数将引导扇区的数据读入大小为512b的缓冲区，然后根据偏移量读取对应的数据。值得注意的地方是有符号数和无符号数要进行区分,不然可能在显示数字的时候会出现问题。再需要考虑的是32位的数据,可以用 2 个 16 位的数据分别存高和低两个部分,然后对其进行操作。

相应偏移量的取值在上文中BPB和EBPB的图片中已有说明。

对每一项内容的操作基本差不多，选取其中的几个不同类型的代笔进行说明。

;-----以下显示 OEM名串-----

```
    mov dh,[Ins]
    mov dl,0
    ;显示OEM提示串"OEM:"
    mov bp,str1
    mov cx,str1len
    call DispStr    ; 调用显示字符串例程
    ;显示偏移量为0x0003处的字符串内容（我的是MyOS-REN）
    mov dh,[Ins]
    mov dl,[In]
    mov cx,8
    mov bp,BootSector+3 ;字符串起始位置
    call DispStr    ; 调用显示字符串例程
    inc word [Ins]
str1 db "OEM:"
str1len equ $ - str1
```

;-----总扇区数（2880）-----

```
    mov dh,[Ins]
    mov dl,0
    mov bp,str2
    mov cx,str2len
```

```

call DispStr

mov dh,[Ins]
mov dl,[In]
mov ax, [BootSector + 13h] ; AX=总扇区数
call GetDigStr ; 以AX为传递参数, BP(串地址)和CX(字符个数)为返回值
call DispStr
inc word [Ins]
str2 db "BPB_TotSec16: "
str2len equ $ - str2
; 获取字数据值十进制串例程
dn equ 5 ; 最大位数
GetDigStr: ; 以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
    mov cx, 1 ; 位数=1 (初值)
    mov bp, sbuf ; BP = sbuf + dn - 1 = sbuf的当前位置
    add bp, dn - 1
    mov bx, 10 ; 除数=10
DLoop: ; 循环开始处
    mov dx, 0 ; DX=0, DX:AX / BX -> 商AX、余DX
    div bx
    add dl, 30h ; 余数 + 30h = 对应的数字ASCII码
    mov [bp], dl ; sbuf[BP] = DL
    cmp ax, 0 ; 商AX = 0 ?
    je OutLoop ; = 0 跳出循环
    inc cx ; 位数CX++
    dec bp ; 数字的当前位置BP--
    jmp DLoop ; 继续循环
OutLoop: ; 退出循环
    ret ; 从例程返回
; ----- FAT数 (一般为2) -----
    mov dh,[Ins]
    mov dl,0
    mov bp,str3
    mov cx,str3len
    call DispStr

    mov ax,[BootSector+10h] ; 取偏移为10h地址处的内容, 送入寄存器ax
    call ShowChar ; 已知fat数一般为2, 可调用显示单个字符的函数
    call newline
    inc word [Ins]
ShowChar: ; 显示一个十六进制数字: 0~9、A~F (以AL为传递参数)
    cmp al, 10 ; AL < 10 ?
    jl .1 ; AL < 10: 跳转到.1
    add al, 7 ; AL >= 10: 显示字母 ( = 数值 += 37h)
.1: ; 数字
    add al, 30h ; 数字字符 = 数值+=30h
    mov ah, 0Eh ; 功能号 (以电传方式显示单个字符)

```

```

        mov bl, 0                ; 对文本方式置0
        int 10h                 ; 调用10H号中断
        ret                     ; 从例程返回
str3 db "BPB_NumFATs: "
str3len equ $ - str3

```

BootSector:
resb 512

(6) 用户程序dir.com

显示文件名和文件大小的操作与上两个用户程序一样，这里主要说明显示时间的过程。

为了得到时间和日期,需要先获取一个 16 位变量,然后通过与或等逻辑运算,实现屏蔽部分位,然后通过移位得到需要的数据。

; - - - - - 显示时间 (年月日时分秒) - - - - -

```

        ; 显示日期 (年.月.日)
        mov ax, [di + 18h]      ; AX = 日期 (低5位为日、中4位为月、高7位为年-
1980)
        push ax                 ; 保存AX进栈
        ; 显示年 (高7位为年-1980)
        shr ax, 9               ; AX >> 9, AX = 年 - 1980
        add ax, 1980            ; AX + 1980 = 年
        call GetDigStr          ; 以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
        call DispStr            ; 显示年字符串
        ; 显示月 (中4位为月)
        pop ax
        push ax
        shr ax, 5               ; AX >> 5
        and ax, 0Fh             ; AX & 1111 b = 月
        call GetDigStr          ; 以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
        call DispStr            ; 显示月字符串
        ; 显示日 (低5位为日)
        pop ax                  ; 弹出AX = 日期
        and ax, 1Fh             ; AX & 1 1111 b = 日
        call GetDigStr          ; 以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
        call DispStr            ; 显示日字符串
        ; 显示时间 (时:分:秒)
        mov ax, [di + 16h]      ; AX = 时间 (低5位为秒/2、中6位为分、高5位为时)
        push ax                 ; 保存AX进栈
        ; 显示时 (高5位为时)
        shr ax, 11              ; AX >> 11, AX = 时
        call GetDigStr          ; 以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
        call DispStr            ; 显示时字符串
        ; 显示分 (中6位为分)

```

```

pop ax                ;弹出AX = 时间
push ax               ;保存AX进栈
shr ax, 5             ;AX >> 5
and ax, 3Fh          ;AX & 11 1111 b = 分
call GetDigStr        ;以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
call DispStr          ;显示月字符串
;显示秒 (低5位为秒/2)
pop ax                ;弹出AX = 时间
and ax, 1Fh          ;AX & 1 1111 b = 秒/2
shl ax, 1             ;AX << 1, AX*2 = 秒
call GetDigStr        ;以AX为传递参数, [串地址]BP和[字符个数]CX为返回值
call DispStr          ;显示日字符串

```

(7) 之前的4个用户程序use1.com, use2.com, use3.com, use4.com

稍微进行了修改, 之前是直接加载到1000h处, 然后从命令行直接通过call和ret加载, 现在是加载到100h处, 用之前所说从根目录中寻找的方式加载到指定内存40100处。

修改部分:

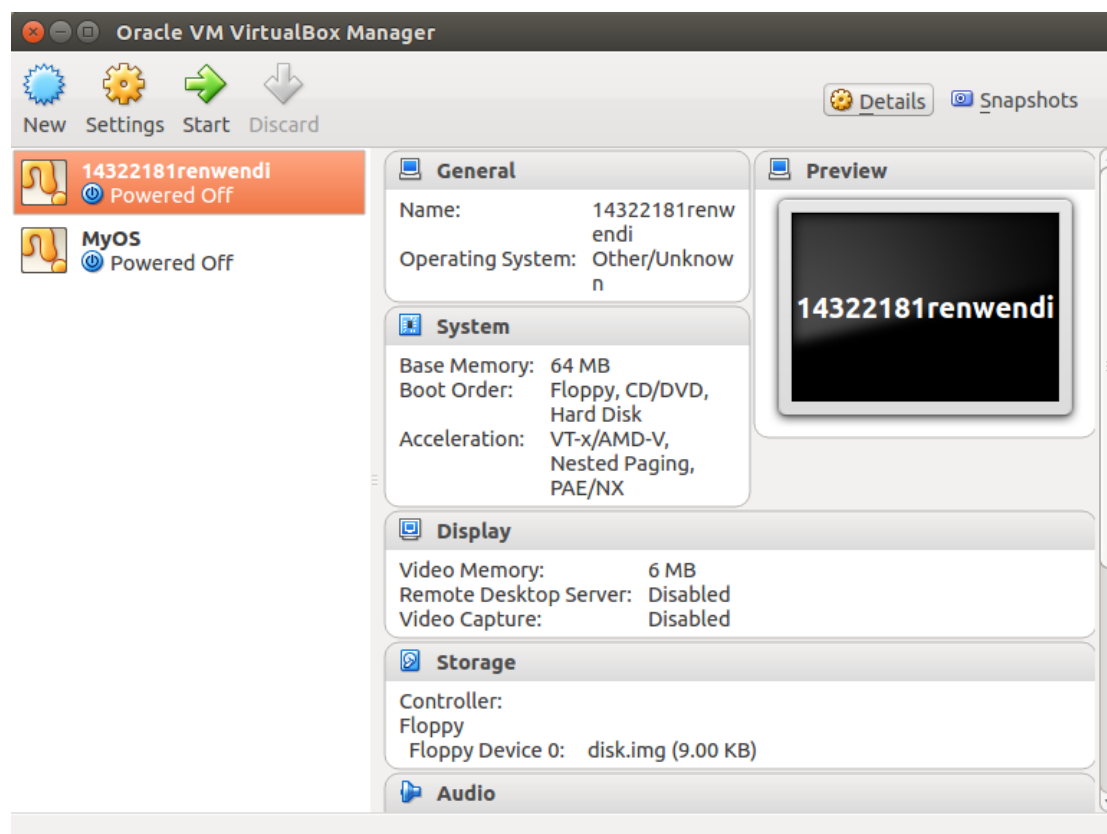
```

org 100h
call clean_s
...
mov ah, 4ch
int 21h

```

【实验过程】

1 主要工具安装使用过程及截图结果: 虚拟机的配置



2 程序过程中的操作步骤

使用linux平台上的dd指令创建软盘，并将引导扇区写入软盘中。

使用mount挂载指令将文件写入软盘。

创建空软盘:

```
$dd if=/dev/zero of=fat.img bs=512 count=2880
```

将引导扇区写入软盘:

```
$dd if=boot.bin of=fat.img conv=notrunc
```

将文件写入软盘:

挂载:

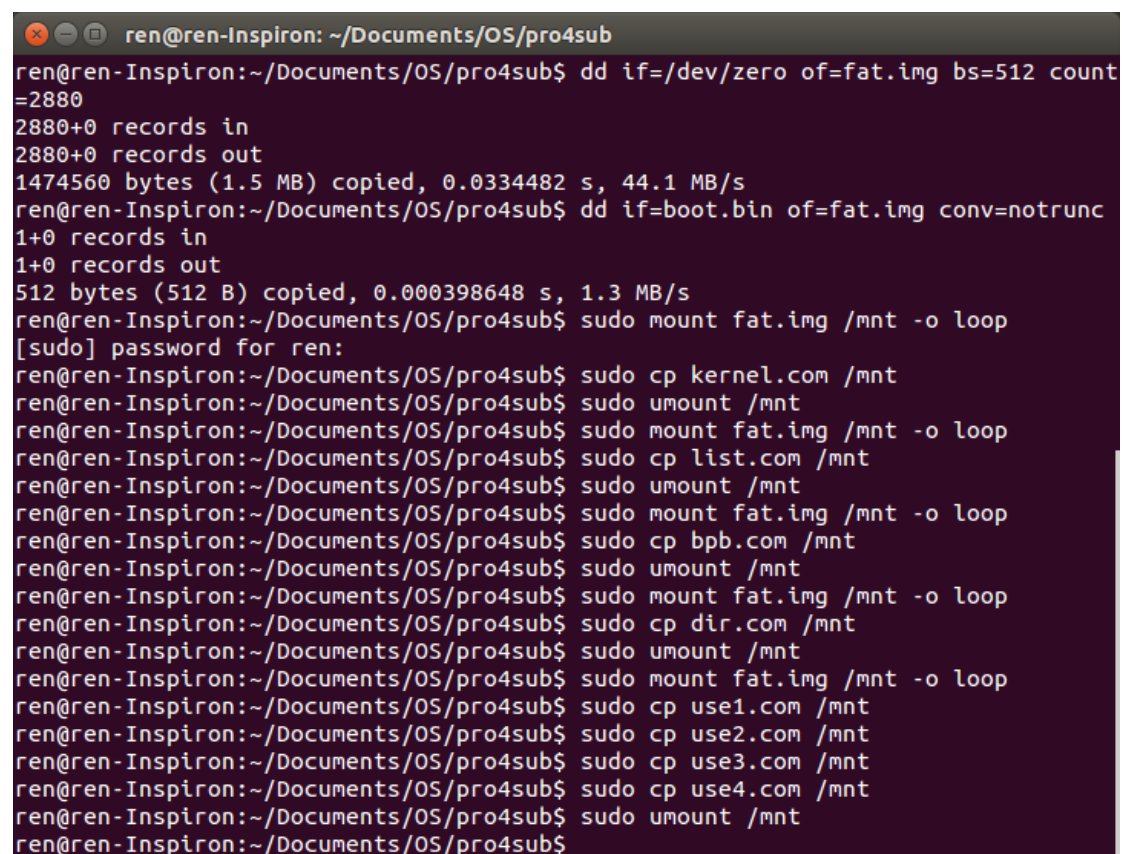
```
$sudo mount SeanOS.img /mnt -o loop
```

将文件复制到软盘中:

```
$sudo cp xxx.com /mnt
```

卸载:

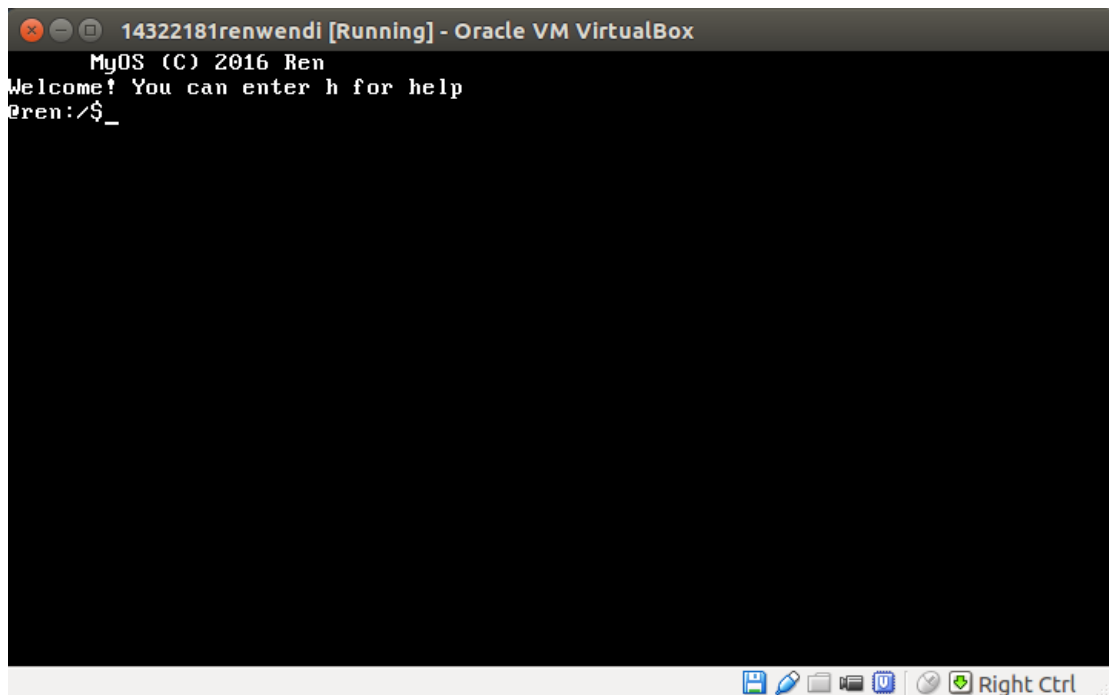
```
$sudo umount /mnt
```



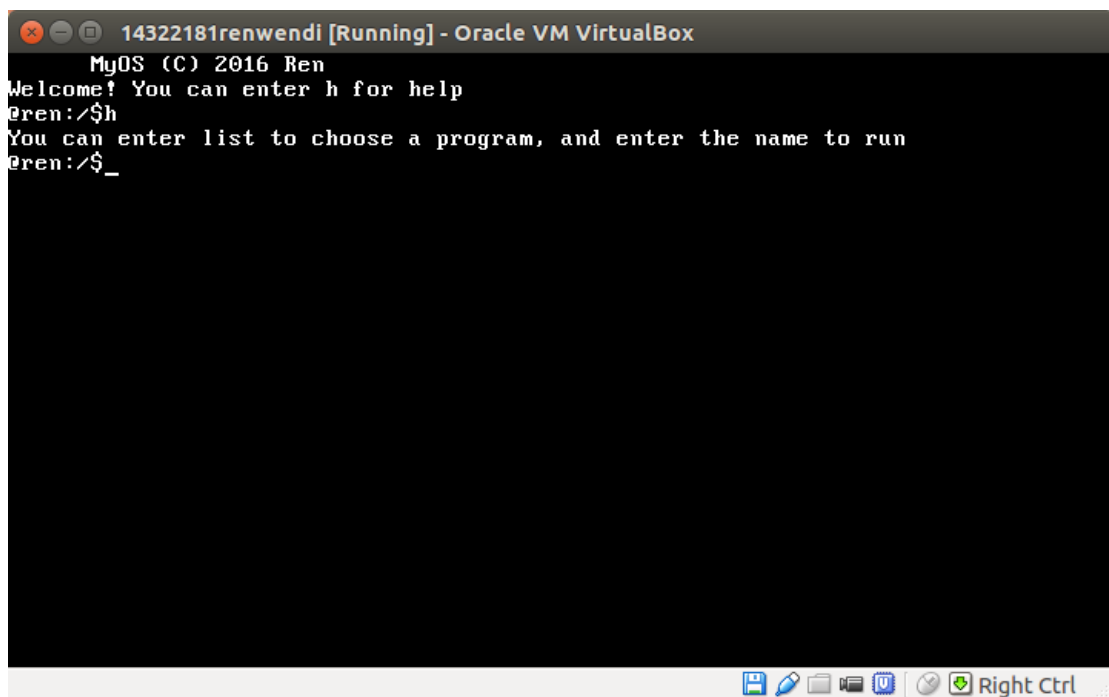
```
ren@ren-Inspiron: ~/Documents/OS/pro4sub
ren@ren-Inspiron:~/Documents/OS/pro4sub$ dd if=/dev/zero of=fat.img bs=512 count
=2880
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB) copied, 0.0334482 s, 44.1 MB/s
ren@ren-Inspiron:~/Documents/OS/pro4sub$ dd if=boot.bin of=fat.img conv=notrunc
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000398648 s, 1.3 MB/s
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo mount fat.img /mnt -o loop
[sudo] password for ren:
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp kernel.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo umount /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo mount fat.img /mnt -o loop
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp list.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo umount /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo mount fat.img /mnt -o loop
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp bpb.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo umount /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo mount fat.img /mnt -o loop
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp dir.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo umount /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo mount fat.img /mnt -o loop
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp use1.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp use2.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp use3.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo cp use4.com /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ sudo umount /mnt
ren@ren-Inspiron:~/Documents/OS/pro4sub$ =
```

3 输出说明

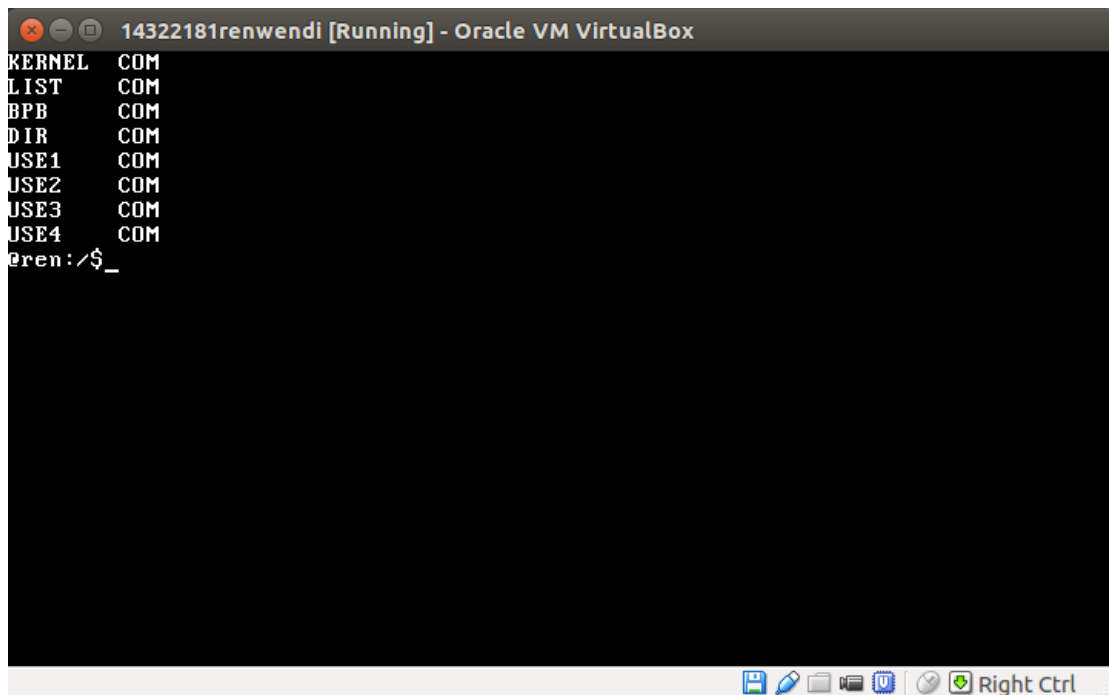
(1) 打开时的欢迎界面



(2) 输入h后的帮助界面

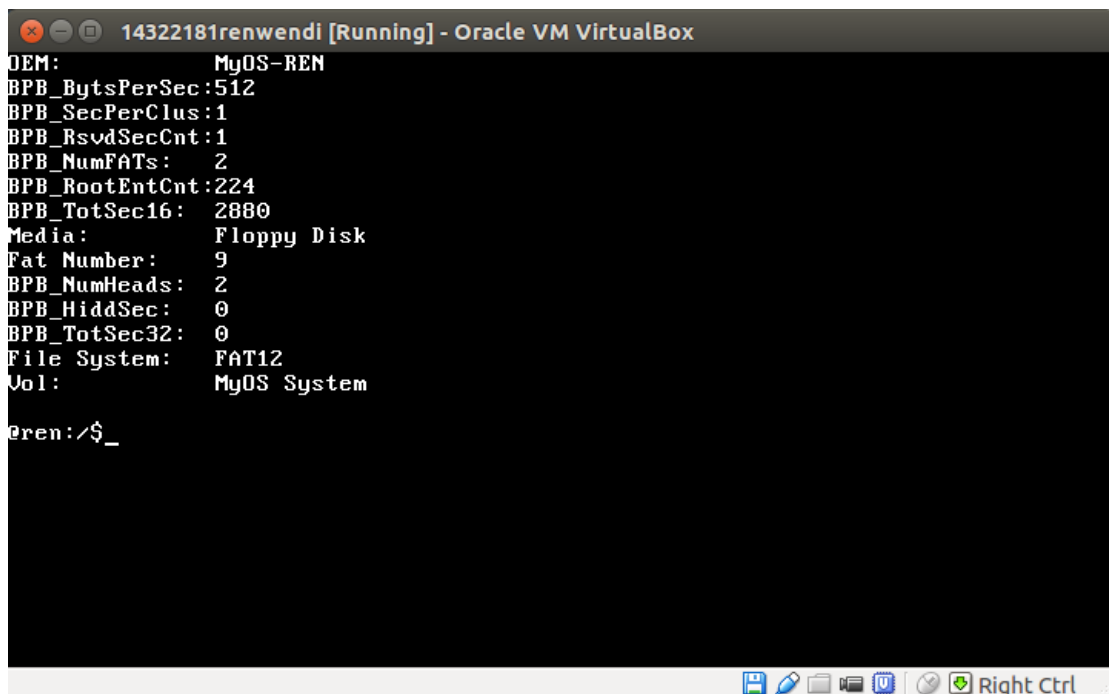


(3) 输入list后的目录名列表界面



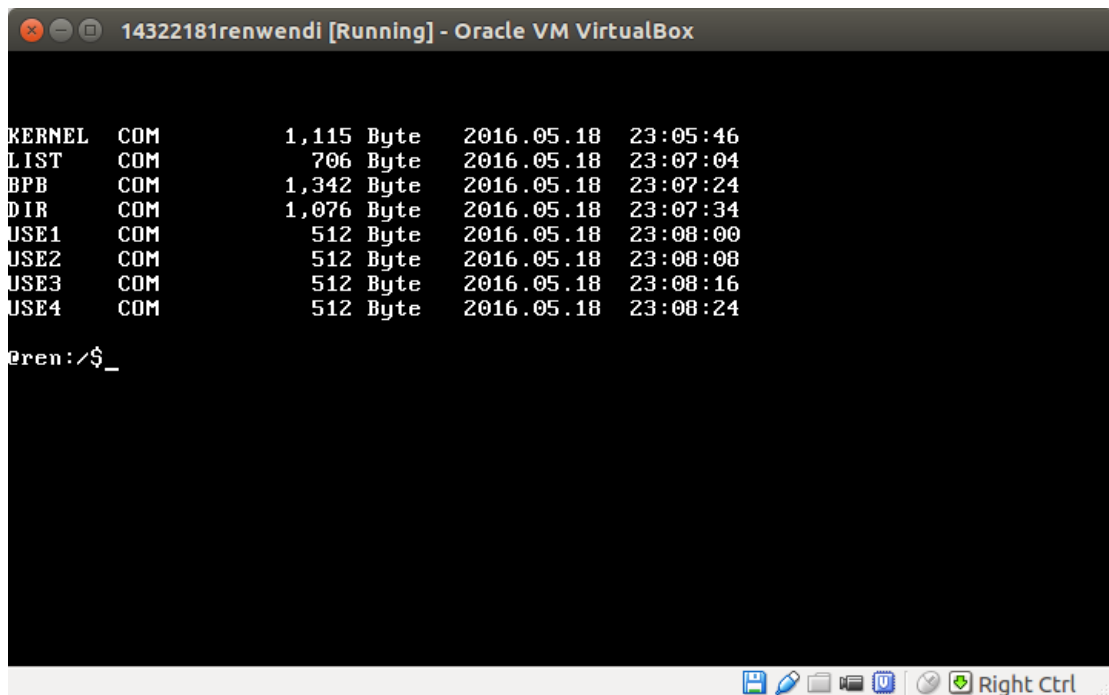
```
14322181renwendi [Running] - Oracle VM VirtualBox
KERNEL  0
LIST    0
BPB     0
DIR     0
USE1    0
USE2    0
USE3    0
USE4    0
ren:/$_
```

(4) 用户程序bpb.com运行界面



```
14322181renwendi [Running] - Oracle VM VirtualBox
MyOS-REN
BPB_BytsPerSec:512
BPB_SecPerClus:1
BPB_RsvdSecCnt:1
BPB_NumFATs: 2
BPB_RootEntCnt:224
BPB_TotSec16: 2880
Media: Floppy Disk
Fat Number: 9
BPB_NumHeads: 2
BPB_HiddSec: 0
BPB_TotSec32: 0
File System: FAT12
Vol: MyOS System
ren:/$_
```

(5) 用户程序dir.com运行界面



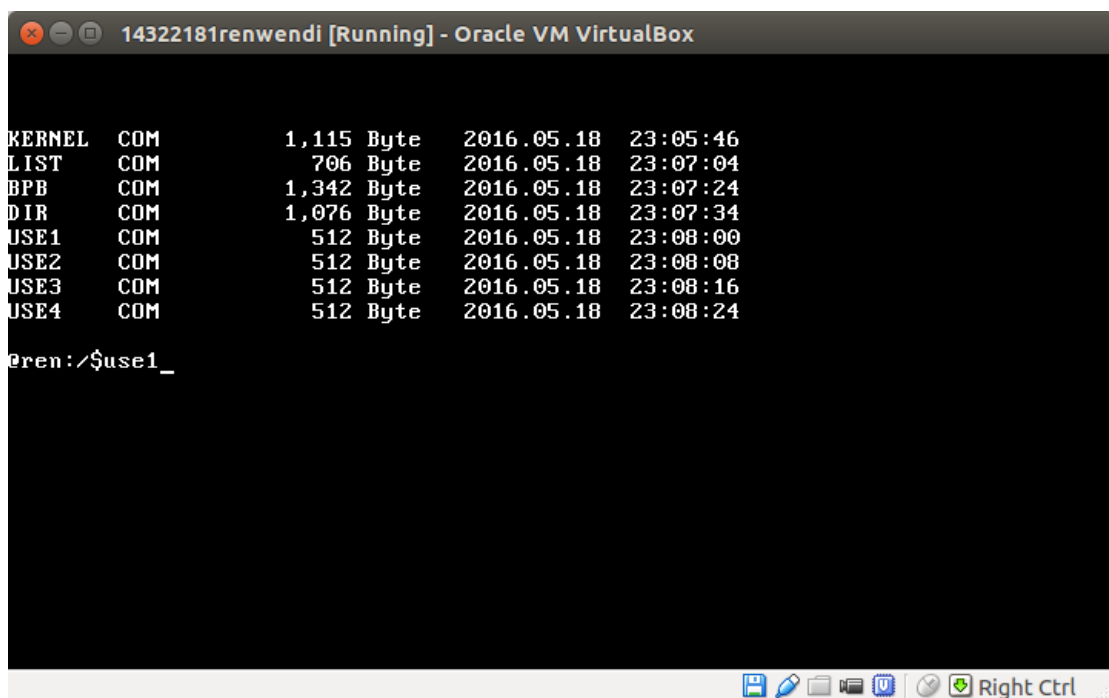
```
14322181renwendi [Running] - Oracle VM VirtualBox

KERNEL.COM      1,115 Byte    2016.05.18    23:05:46
LIST.COM        706 Byte     2016.05.18    23:07:04
BPB.COM         1,342 Byte    2016.05.18    23:07:24
DIR.COM         1,076 Byte    2016.05.18    23:07:34
USE1.COM        512 Byte     2016.05.18    23:08:00
USE2.COM        512 Byte     2016.05.18    23:08:08
USE3.COM        512 Byte     2016.05.18    23:08:16
USE4.COM        512 Byte     2016.05.18    23:08:24

ren:/$_
```

(6) 用户程序use1.com运行界面

首先输入use1

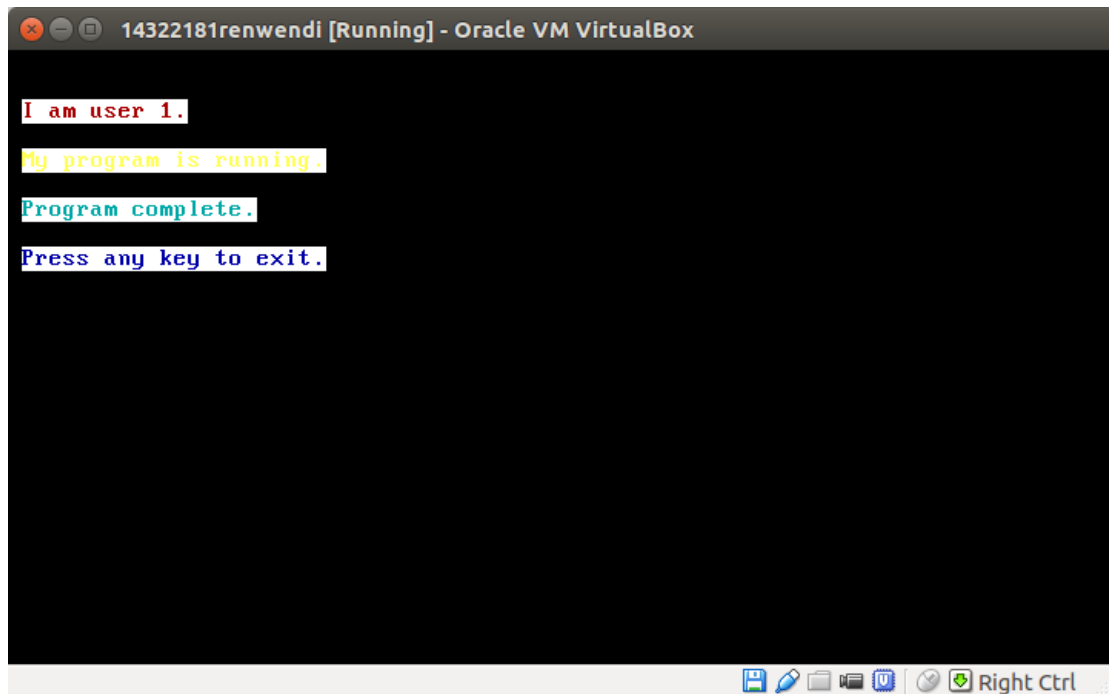


```
14322181renwendi [Running] - Oracle VM VirtualBox

KERNEL.COM      1,115 Byte    2016.05.18    23:05:46
LIST.COM        706 Byte     2016.05.18    23:07:04
BPB.COM         1,342 Byte    2016.05.18    23:07:24
DIR.COM         1,076 Byte    2016.05.18    23:07:34
USE1.COM        512 Byte     2016.05.18    23:08:00
USE2.COM        512 Byte     2016.05.18    23:08:08
USE3.COM        512 Byte     2016.05.18    23:08:16
USE4.COM        512 Byte     2016.05.18    23:08:24

ren:/$_use1_
```

运行



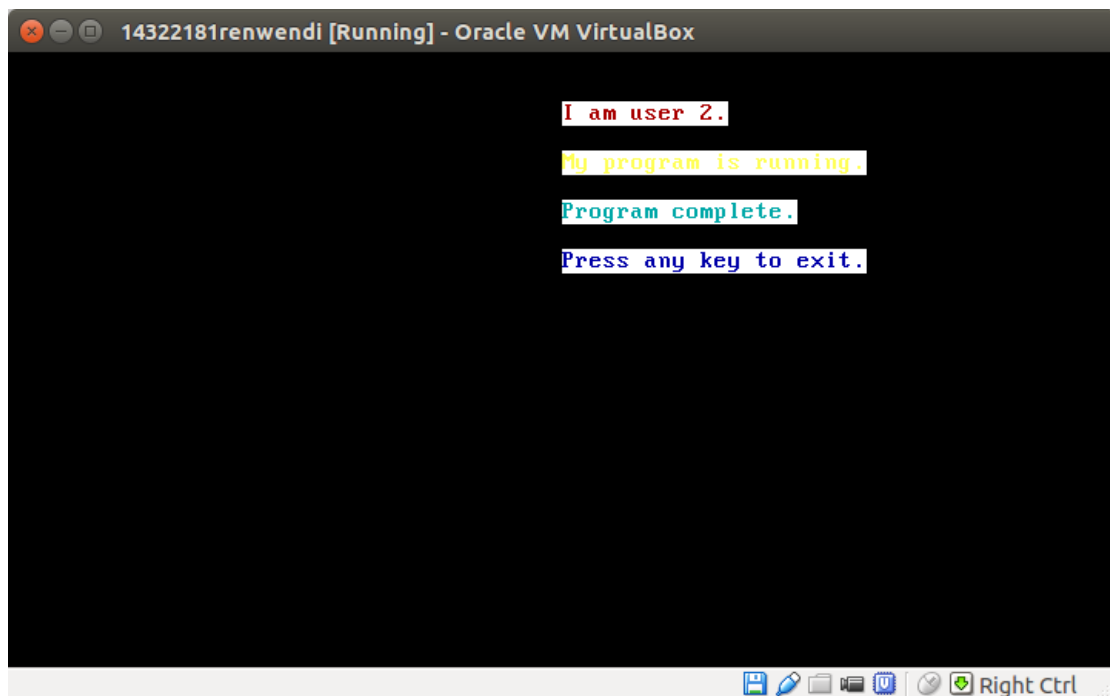
14322181renwendi [Running] - Oracle VM VirtualBox

```
I am user 1.  
My program is running.  
Program complete.  
Press any key to exit.
```

Right Ctrl

This screenshot shows a terminal window within an Oracle VM VirtualBox. The window title is "14322181renwendi [Running] - Oracle VM VirtualBox". The terminal displays four lines of text: "I am user 1." in red, "My program is running." in yellow, "Program complete." in cyan, and "Press any key to exit." in blue. At the bottom right of the window, there is a toolbar with icons for saving, editing, and other functions, along with the text "Right Ctrl".

(7) use2.com ,use3.com, use4.com同理




14322181renwendi [Running] - Oracle VM VirtualBox

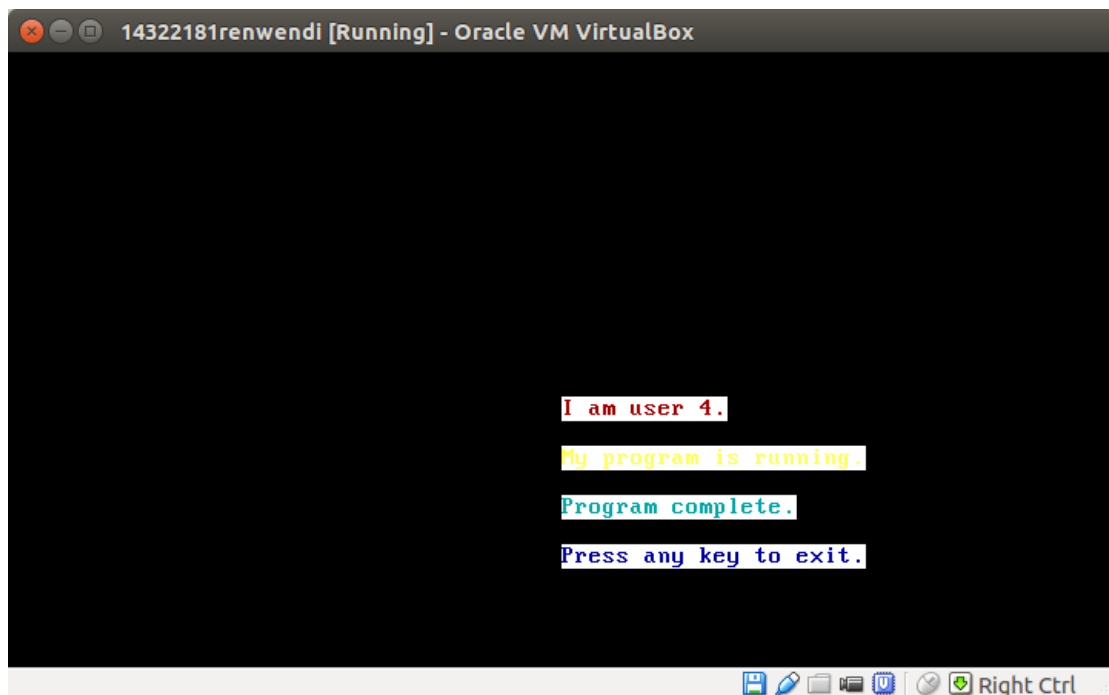
```
I am user 2.  
My program is running.  
Program complete.  
Press any key to exit.
```

Right Ctrl

This screenshot shows a terminal window within an Oracle VM VirtualBox, similar to the one above. The window title is "14322181renwendi [Running] - Oracle VM VirtualBox". The terminal displays four lines of text: "I am user 2." in red, "My program is running." in yellow, "Program complete." in cyan, and "Press any key to exit." in blue. At the bottom right of the window, there is a toolbar with icons for saving, editing, and other functions, along with the text "Right Ctrl".



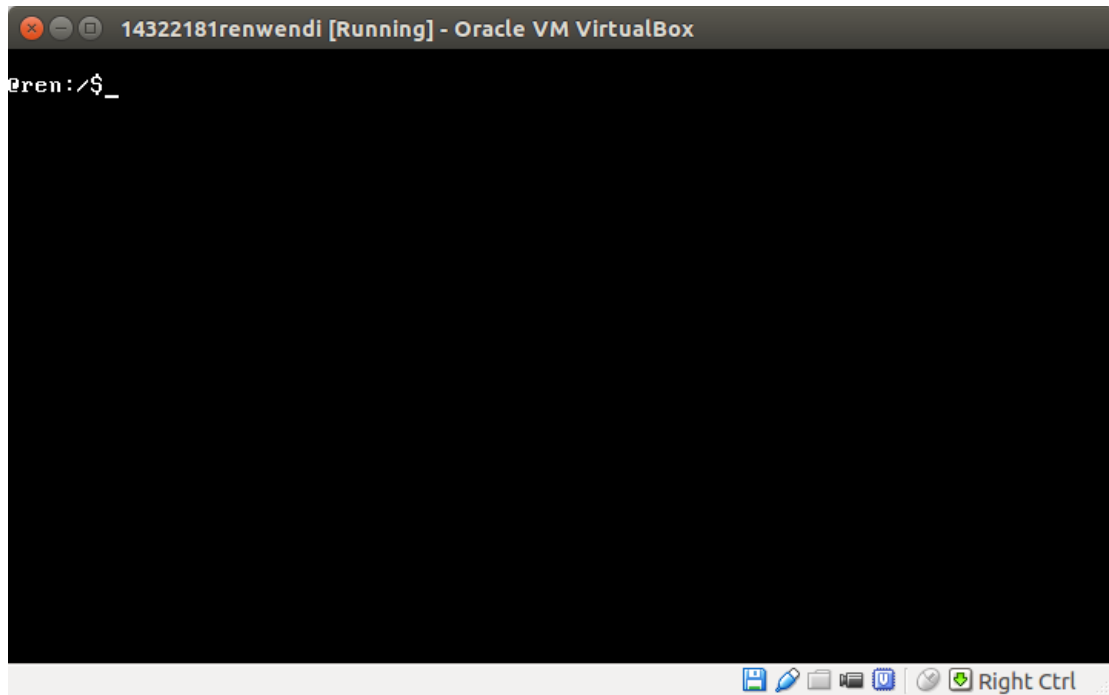
```
I am user 3.  
My program is running.  
Program complete.  
Press any key to exit.
```



```
I am user 4.  
My program is running.  
Program complete.  
Press any key to exit.
```

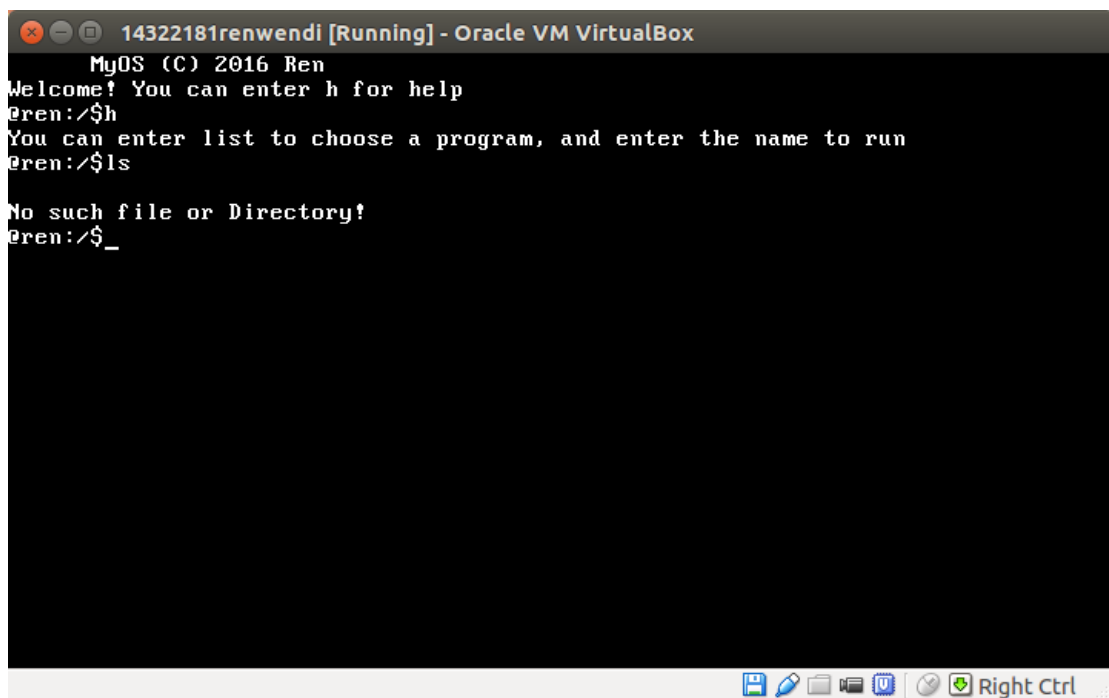
(8) 清屏指令cls

输入之后清屏结果如下



(9) 若输入有误，输出错误信息提示

No such file or Directory!



4 遇到的问题及解决情况

(1) 最先遇到的问题是在linux平台下无法成功挂载文件到软盘上。

参考于渊的书和网上搜索相关mount命令之后，还是无法解决。

会出现以下图中各种奇怪的问题，为了解决这个问题耗了很长的时间，两三天
的时间才最终通过询问同样在linux平台下做的同学才得以解决。

（未找到挂载地址）

```
ren@ren-Inspiron: ~  
[sudo] password for ren:  
mount: mount point /mnt/floppy/ does not exist  
ren@ren-Inspiron:~/Documents/OS/4$ sudo mount -o loop a.img /mnt/floppy/  
mount: mount point /mnt/floppy/ does not exist  
ren@ren-Inspiron:~/Documents/OS/4$ sudo cp loader.bin /mnt/floppy/ -v  
'loader.bin' -> '/mnt/floppy/'  
cp: cannot create regular file '/mnt/floppy/': Not a directory  
ren@ren-Inspiron:~/Documents/OS/4$ sudo umount /mnt/floppy  
umount: /mnt/floppy: not found  
ren@ren-Inspiron:~/Documents/OS/4$
```

（不能创建挂载地址）

```
ren@ren-Inspiron:~/Documents/OS/4$ mkdir /mnt/share  
mkdir: cannot create directory '/mnt/share': Permission denied  
ren@ren-Inspiron:~/Documents/OS/4$ mkdir /mnt/share  
bash: mkdir/mnt/share: No such file or directory  
ren@ren-Inspiron:~/Documents/OS/4$ mkdir /mnt/floppy  
mkdir: cannot create directory '/mnt/floppy': Permission denied  
ren@ren-Inspiron:~/Documents/OS/4$ mkdir /mnt/floppy/  
mkdir: cannot create directory '/mnt/floppy/': Permission denied  
ren@ren-Inspiron:~/Documents/OS/4$ cd  
ren@ren-Inspiron:~$ mkdir /mnt/floppy/  
mkdir: cannot create directory '/mnt/floppy/': Permission denied  
ren@ren-Inspiron:~$
```

（挂载位置无空间）

```
ren@ren-Inspiron: ~/Documents/OS/4  
ren@ren-Inspiron:~/Documents/OS/4$ cd /mnt/  
ren@ren-Inspiron:/mnt$ ls  
fd floppy loader.bin  
ren@ren-Inspiron:/mnt$ sudo mount -o loop a.img /mnt/floppy/  
a.img: No such file or directory  
ren@ren-Inspiron:/mnt$ sudo cp loader.bin /mnt/floppy/ -v  
'loader.bin' -> '/mnt/floppy/loader.bin'  
cp: cannot create regular file '/mnt/floppy/loader.bin': No space left on device  
ren@ren-Inspiron:/mnt$ cd floppy/  
ren@ren-Inspiron:/mnt/floppy$ cd ..  
ren@ren-Inspiron:/mnt$ sudo cp loader.bin /mnt/floppy/  
cp: cannot create regular file '/mnt/floppy/loader.bin': No space left on device  
ren@ren-Inspiron:/mnt$ sudo cp -fv loader.bin /mnt/floppy/  
'loader.bin' -> '/mnt/floppy/loader.bin'  
cp: cannot create regular file '/mnt/floppy/loader.bin': No space left on device  
ren@ren-Inspiron:/mnt$ cd  
ren@ren-Inspiron:~$ cd Documents/OS/4  
ren@ren-Inspiron:~/Documents/OS/4$ ls  
a.img boot2.bin bootF12.asm loader.asm loader.bin  
boot2.asm boot.asm bootF12.bin loader.bi LoadKnL.asm  
ren@ren-Inspiron:~/Documents/OS/4$ sudo cp -fv loader.bin /mnt/floppy/  
'loader.bin' -> '/mnt/floppy/loader.bin'  
cp: cannot create regular file '/mnt/floppy/loader.bin': No space left on device  
ren@ren-Inspiron:~/Documents/OS/4$
```

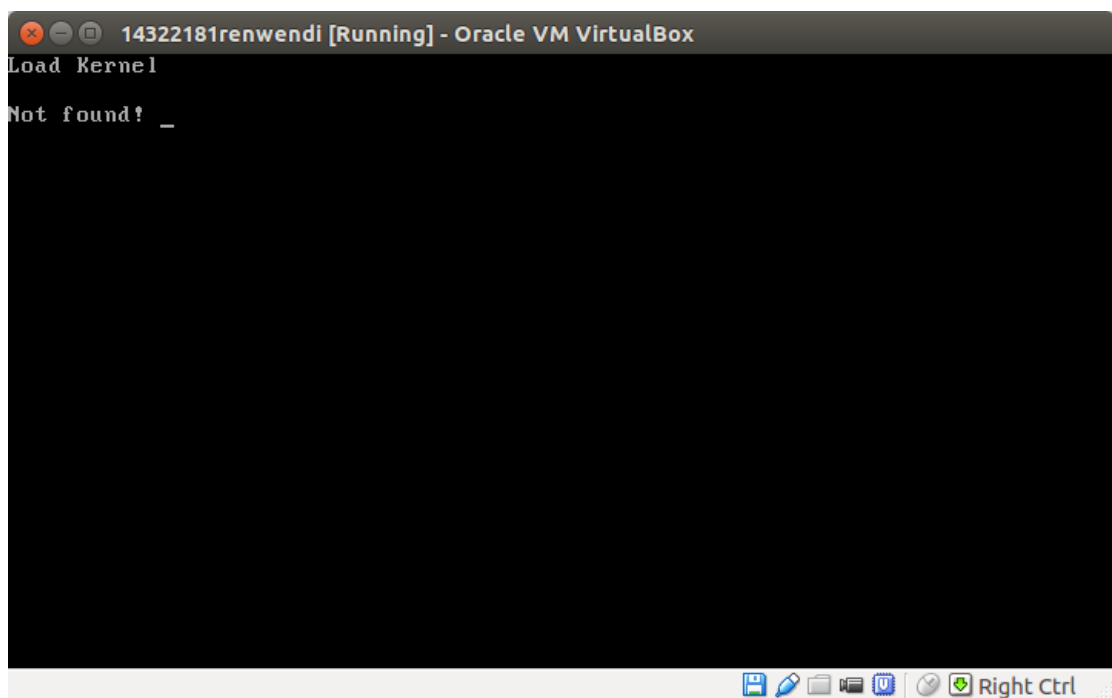
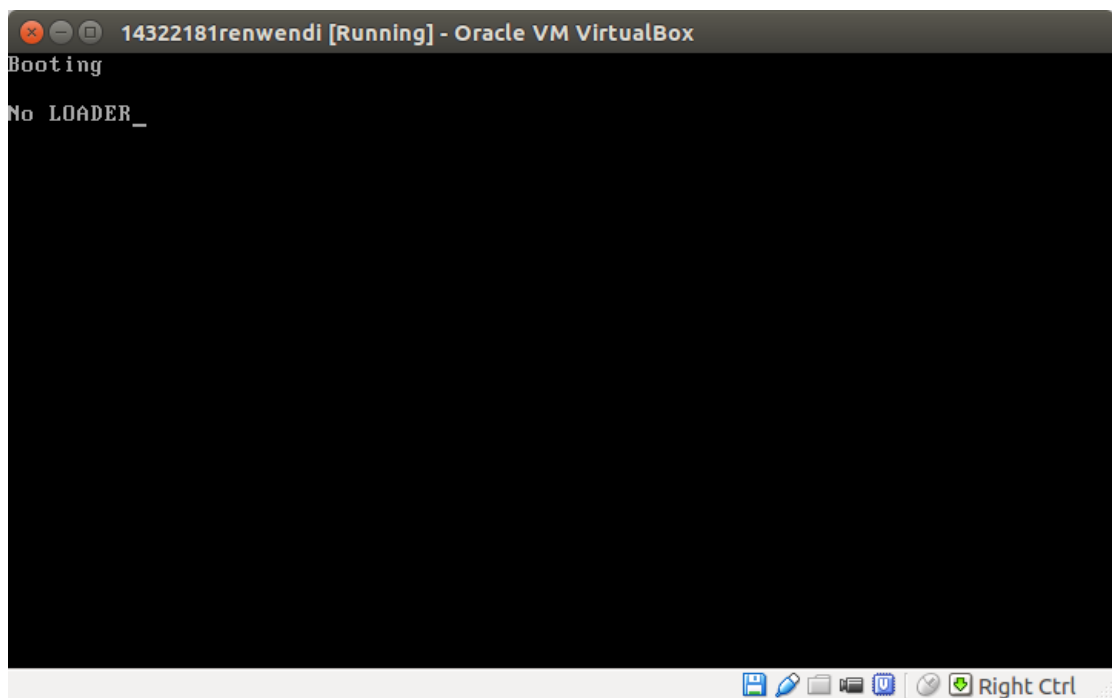
将之前错误的指令与最后正确的指令相比较，感觉问题一个是挂载地址不正确，
直接用mnt目录即可，因为挂载点必须是一个目录。

还有复制文件的指令直接cp就可以，不需要其他多余的指令。

最后成功挂载后查看mnt目录截图如下。（cd /mnt进入目录）

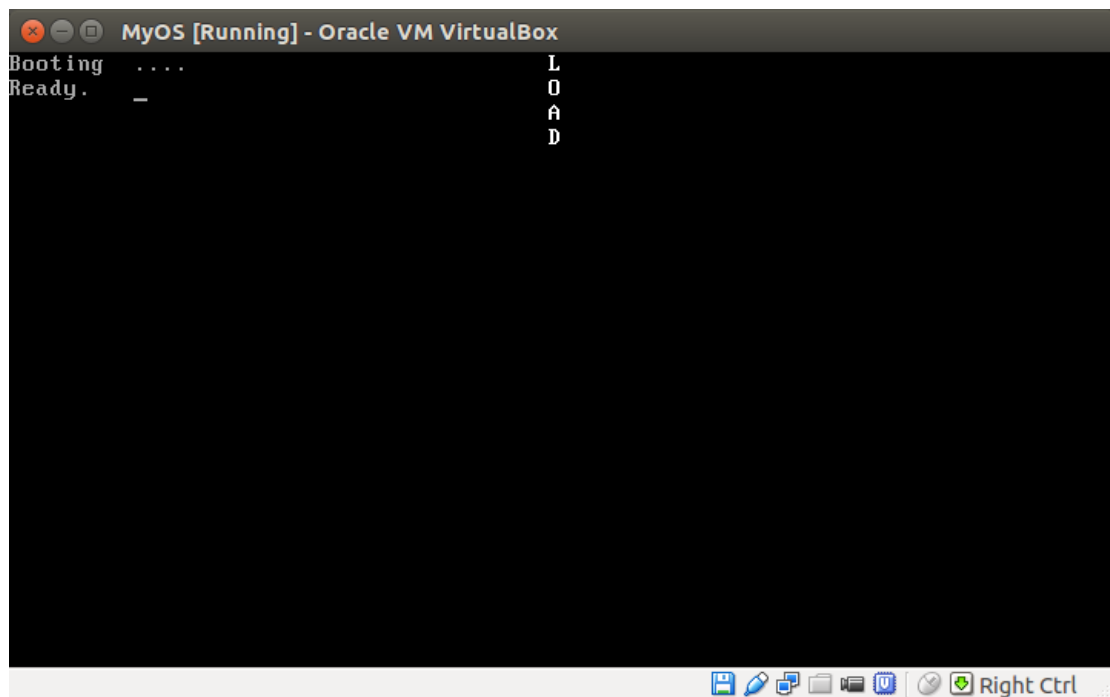
```
ren@ren-Inspiron:~/Documents/OS$ cd /mnt
ren@ren-Inspiron:/mnt$ ls
bpb.com  fl  fol  kernel.com  name.com  use3.com
cmd.bin  flo kernel.bin loader.bin  use2.com  use4.com
ren@ren-Inspiron:/mnt$ _
```

(2) 引导扇区也无法成功找到内核，不断出现no kernel和no loader（之前没有改名字时）



然后就先用较为简单的代码进行了测试，使用只显示字符的内核。

测试成功，说明引导扇区应该没有问题，是内核的地址或者名字不正确，未找到。



后来仔细一看，发现是名字那里的空格没有修改。此外，原来的内核地址也不正确。

(3) 显示数字时会出现奇怪的符号

这是因为刚开始时考虑问题过于简单，只使用了int 10h的0Eh功能号，通过电传方式显示单个字符。当数字超过一位时，就无法正确显示。如在bpb.com中，显示fat数（一般为2）可以正确显示，而总扇区数（2880）就无法显示。后来使用了新的函数，以偏移量AX为传递参数，BP(串地址)和CX(字符个数)为返回值，然后再调用int 10h的13h显示字符串，最后得以正确显示。

(4) 无法从用户程序回到内核

这个问题在之前也有所说明，在之前的实验中，用户程序使用扇区储存，系统内核的基地址和用户程序的基地址相同，所以直接使用 call和ret,不会出现问题。

这次由于内核基地址是 0x9000,用户程序是 0x4000,使用近调用,会出现问题。所以需要使用远调用。

远调用有如下几个方法：在 nasm 里,远调用可以通过 call far ‘基地址’:‘偏移地址’实现。

另外一个方法是只是用 retf。根据 retf 的返回原理,可以这样实现远调用。

本实验中我使用的是21h号中断来返回（此中断在DOS中，是系统调用的入口，其中的AH=4Ch功能是：结束程序的运行，返回DOS，返回值为AL一般为0）。

【实验总结】

1、本次实验不足之处

尽管本次实验用了很多的时间，但最后的效果还是不尽如人意。最大的问题在于还是没有实现用c读fat12软盘，最后的效果应该是只使用汇编里的读扇区函数，然后其他操作在c语言里面实现。我尝试了在c语言里面开一个512b的数组，将读入的扇区的内容存储在其中，然后进行进一步的操作，但是显示总会出错，所以最后还是用汇编来操作主要内容了。

此外，修改后的新的内核加载用户程序的方法是在老师所给的引导扇区的代码上修改而来的，没有在合适的地方成功添加循环和判断，因此没有实现批处理。

2、实验心得

本次实验进行的较为困难的原因个人觉得是理论知识没有了解清楚，对fat的数据结构刚开始其实并没有完全理解，将老师所给的参考引导扇区和于渊书中对fat的讲解阅读了多遍才彻底理解。此外，尽管理解概念之后，但是实施起来还是有很多困难，因为涉及到大量fat的读写操作。刚开始也没有想到通过传递基址和偏移量来达到获取变量以及读取软盘，因此不知道该如何实现要求显示的dir的内容。

感觉这次实验耗时太久，在中途一度陷入停滞不知道该怎么办，其实真正行动起来解决问题反而就不会那么痛苦了。因为我是转专业的学生，在上大二的课程同时还要补大一的课程，作业和课程压力很大，做实验四的同时还有大一的一个c++项目大作业，所以耽误了比较久的时间，还望老师谅解。

【参考文献】

于渊 《Orange's——一个操作系统的实现》

linux的mount（挂载）命令详解：<http://www.jb51.net/os/RedHat/1109.html>