武汉大学国家网络安全学院教学实验报告

课程名称	操作系统设计与实践	实验日期	2023.11.6
实验名称	由盘上结构实现程序加载	实验周次	第五周
姓名	学号	专业	班级
王卓	2021302191791	网络空间安全	9 班
程子洋	2021301051114	网络空间安全	9 班
聂森	2021302191536	网络空间安全	9 班
刘虓	2021302121234	网络空间安全	9 班

一、实验目的及实验内容
（本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析）

本次实验内容

1．学习 FAT12 文件结构

2．向软盘镜像文件写入一个你自己任意创建的文件，手工方式在软盘中找到指定的文件，读取其扇区信息，记录步骤

3．学习将指定的可执行文件装入指定内存区的方法，并调试执行，记录原理与步骤

4．学会使用 xxd 读取二进制信息，读取步骤 2 中写入的文件

思考问题

1．FAT12 格式是怎样的？

2．如何读取一张软盘的信息？

3．如何在软盘中找到指定的文件？

4．如何在系统引导过程读取并加载一个可执行文件到内存，转交控制权？

5．为什么需要这个 Loader 程序不包含 dos 系统调用？

6．为什么前面几个章节中的 a.img、pm.img 等文件不能直接 mount，在本章代码里面却可以？

7．扩展提高：调研在硬盘上，文件系统格式为 FAT32 或者 NTFS，应该怎么来实现类似功能呢？

二、实验环境及实验步骤
（本次实验所使用的器件、仪器设备等的情况；具体的实验步骤）

2.1. 实验环境

- 虚拟机：VMware Workstation Pro/VMware Workstation 16 player
- 操作系统：Ubuntu 16.04
- 模拟系统软件：Bochs 2.7

2.2. 具体实验步骤

2.2.1. 学习 FAT12 文件结构

FAT12 是 DOS 时代就开始使用的文件系统（File System），直到现在仍然在软盘上使用。几乎所有的文件系统都会把磁盘划分为若干层次以方便组织和管理，这些层次包括：

- 扇区（Sector）：磁盘上的最小数据单元。
- 簇（Cluster）：一个或多个扇区。
- 分区（Partition）：通常指整个文件系统。

FAT12 软盘的被格式化后为：有两个磁头，每个磁头 80 个柱面（磁道），每个柱面有 18 个扇区，每个扇区 512 个字节空间。

所以标准软盘的总空间为：2 * 80 *18 * 512=1474560B=1440K=1.44M。

图 1 软盘（1.44MB，FAT12）

如上图所示，一个 1.44M 的软盘，可以划分为 2879 个扇区，共分为图上所示的五个区域。

在将软盘格式化成 FAT12 文件系统的过程中，FAT 类文件系统会对软盘里

的扇区进行结构化处理，进而把软盘扇区划分成：引导扇区、FAT 表、根目录区和数据区这 4 部分。

（1）引导扇区

FAT12 文件系统的引导扇区不仅包含有引导程序，还有 FAT12 文件系统的整个组成结构信息。MBR (Main Boot Record) 主引导记录占用大小为 1 个扇区，即 512 B。在这个扇区里记录了整个文件系统的组织结构信息和引导程序两部分内容。

下表描述了 FAT12 文件系统的引导扇区结构：

名称	偏移	长度	内容	Orange'S的值
BS_jmpBoot	0	3	一个短跳转指令	jmp LABEL_START nop
BS_OEMName	3	8	厂商名	'ForrestY'
BPB_BytsPerSec	11	2	每扇区字节数	0x200
BPB_SecPerClus	13	1	每簇扇区数	0x1
BPB_RsvdSecCnt	14	2	Boot记录占用多少扇区	0x1
BPB_NumFATs	16	1	共有多少 FAT 表	0x2
BPB_RootEntCnt	17	2	根目录文件数最大值	0xE0
BPB_TotSec16	19	2	扇区总数	0xB40
BPB_Media	21	1	介质描述符	0xF0
BPB_FATSz16	22	2	每 FAT 扇区数	0x9
BPB_SecPerTrk	24	2	每磁道扇区数	0x12
BPB_NumHeads	26	2	磁头数（面数）	0x2
BPB_HiddSec	28	4	隐藏扇区数	0
BPB_TotSec32	32	4	如果BPB_TotSec16是0，由这个值记录扇区数	0
BS_DrvNum	36	1	中断 13 的驱动器号	0
BS_Reserved1	37	1	未使用	0
BS_BootSig	38	1	扩展引导标记 (29h)	0x29
BS_VolID	39	4	卷序列号	0
BS_VolLab	43	11	卷标	'OrangeS0.02'
BS_FileSysType	54	8	文件系统类型	'FAT12'
引导代码及其他	62	448	引导代码、数据及其他填充字符等	引导代码（剩余空间被0填充）
结束标志	510	2	0xAA55	0xAA55

图 2 软盘（1.44MB，FAT12）

（2）FAT 表——文件分配表

FAT 表又叫“文件分配表”，FAT12 具有两个 9 扇区大小的 FAT 表。 FAT2 通常是 FAT1 的备份，两者可以认为是一样的。在 FAT 表中，每 12 位被称为一个 FAT 项，第 0 个和第 1 个 FAT 项始终不使用，从第 2 个 FAT 项开始，每个 FAT 项对应数据区的一个簇，数据区首个簇号为 2，FAT Entry N 正好对应数据区簇号为 N 的簇。 每个 FAT 项中存储的是当前文件的当前簇的下一个簇的簇号，如

果值大于等于 0xFF8，那么就表示这已经是文件的最后一个簇，0xFF7 则表示这对应了一个坏簇。

（3）根目录区

根目录区存储了若干条目录条目，每个目录条目长 32 字节，最多存储 BPB_RootEntCnt 个条目。因此可以得到公式：根目录区扇区数=(BPB_RootEntCnt * 32)/BPB_BytsPerSec。BPB_RootEntCnt 和 BPB_BytsPerSec 就是上文中起始扇区中定义的相应字段。

名称	偏移	长度	描述
DIR_Name	0	0xB	文件名 8 字节，扩展名 3 字节
DIR_Attr	0xB	1	文件属性
保留位	0xC	10	保留位
DIR_WrtTime	0x16	2	最后一次写入时间
DIR_WrtDate	0x18	2	最后一次写入日期
DIR_FstClus	0x1A	2	此条目对应的开始簇号
DIR_FileSize	0x1C	4	文件大小

图 3 根目录区中的条目格式

（4）数据区

数据区存储的就是文件的实际内容。如果这个文件实际是一个目录，那么这个簇实际存储的就是这个目录下文件构成的条目列表，具体信息与根目录区中的条目格式相同。

经过上述 FAT12 分区的介绍，我们就已经可以清楚的知道如何在一个 FAT12 类型的磁盘上寻找一个文件了：

1、获取文件系统基本信息——读取位于第 0 个扇区的起始扇区；

2、计算数据区首个扇区——根据起始扇区中的 BPB_RootEntCnt 字段和 BPB_BytsPerSec 字段计算根目录区大小，从而计算出数据区对应的扇区号；

3、获取根目录中的文件——从 19 号扇区开始读取根目录区条目，找到 DIR_NAME 保存的相同文件名的文件或目录，读取对应的簇号 DIR_FstClus；

4、获取文件内容——通过 DIR_FstClus 存储的簇号找到对应的 FAT 项，同时读取数据区中对应的簇号的文件内容，并根据 FAT 项获取下一簇号递归进行读取，直到 FAT 项标识文件内容损坏或文件读取完成。

2.2.2. 向软盘镜像文件写入一个你自己任意创建的文件，手工方式在软盘中找

到指定的文件，读取其扇区信息，记录你的步骤

1．引导扇区需要有 BPB 等头信息才能被识别，故首先加上该信息。修改 boot.asm，如下图所示。

boot.asm (~/Desktop/chapter4/a) - gedit

Open

```
;%define        _BOOT_DEBUG_    ; 做 Boot Sector 时一定将此行注释掉!将此行打开后用 nasm Boot.asm -o Boot.com 做成一个.COM文
易于调试

%ifdef  _BOOT_DEBUG_
        org  0100h                      ; 调试状态, 做成 .COM 文件, 可调试
%else
        org  07c00h                     ; Boot 状态, Bios 将把 Boot Sector 加载到 0:7C00 处并开始执行
%endif

        jmp short LABEL_START           ; Start to boot.
        nop                             ; 这个 nop 不可少

        ; 下面是 FAT12 磁盘的头
        BS_OEMName      DB 'ForrestY'   ; OEM String, 必须 8 个字节
        BPB_BytsPerSec  DW 512          ; 每扇区字节数
        BPB_SecPerClus  DB 1            ; 每簇多少扇区
        BPB_RsvdSecCnt  DW 1            ; Boot 记录占用多少扇区
        BPB_NumFATs     DB 2            ; 共有多少 FAT 表
        BPB_RootEntCnt  DW 224          ; 根目录文件数最大值
        BPB_TotSec16    DW 2880         ; 逻辑扇区总数
        BPB_Media       DB 0xF0         ; 媒体描述符
        BPB_FATSz16     DW 9            ; 每FAT扇区数
        BPB_SecPerTrk   DW 18           ; 每磁道扇区数
        BPB_NumHeads    DW 2            ; 磁头数(面数)
        BPB_HiddSec     DD 0            ; 隐藏扇区数
        BPB_TotSec32    DD 0            ; wTotalSectorCount为0时这个值记录扇区数
        BS_DrvNum       DB 0            ; 中断 13 的驱动器号
        BS_Reserved1    DB 0            ; 未使用
        BS_BootSig      DB 29h          ; 扩展引导标记 (29h)
        BS_VolID        DD 0            ; 卷序列号
        BS_VolLab       DB 'OrangeS0.02'; 卷标, 必须 11 个字节
        BS_FileSysType  DB 'FAT12   '   ; 文件系统类型, 必须 8个字节

LABEL_START:
        mov     ax, cs
```

图 4 修改 boot.asm

2．修改 bochsrc，接下来依次执行

nasm boot.asm -o boot.bin

dd if=boot.bin of=x.img bs=512 count=1 conv=notrunc

sudo bochs -f bochsrc

生成 boot.bin，将之写入引导扇区。运行效果如下：

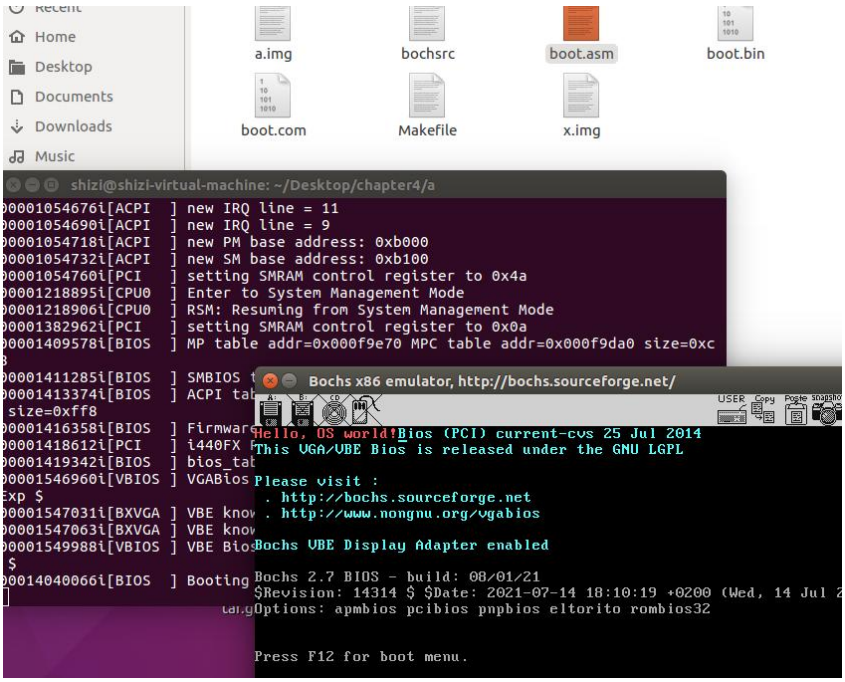


图 5 运行展示

3．一个简单的 loader

Loader.asm。编译为 loader.bin

```
org     0100h

        mov     ax, 0B800h
        mov     gs, ax
        mov     ah, 0Fh
        mov     al, 'L'
        mov     [gs:((80 * 0 + 39) * 2)], ax

        jmp     $
```

图 6 编译为 loader.bin

4．为加载 loader.bin 到软盘，需要读软盘。核心思想为修改 boot.asm，引导扇区，使其功能改为读软盘，寻找 loader.bin

用 bios 中断 int 13h 读软盘，其用法如下所示：

表 4.4 BIOS 中断 int 13h 的用法

中断号	寄存器		作用
13h	ah=00h	dl=驱动器号（0表示A盘）	复位软驱
	ah=02h	al=要读扇区数	从磁盘将数据读
	ch=柱面（磁道）号	cl=起始扇区号	入es:bx指向的
	dh=磁头号	dl=驱动器号（0表示A盘）	缓冲区中
	es:bx→数据缓冲区		

图 7 bios 中断 int13h 用法

boot.asm 中增加的读软盘扇区的函数如下：

```
;
ReadSector:

        push  bp
        mov   bp, sp
        sub   esp, 2

        mov   byte[bp-2], cl
        push  bx
        mov   bl, [BPB_SecPerTrk]
        div   bl
        inc   ah
        mov   cl, ah
        mov   dh, al
        shr   al, 1
        mov   ch, al
        and   dh, 1
        pop   bx
        mov   dl, [BS_DrvNum]
.GoOnRednding:
        mov   ah, 2
        mov   al, byte [bp-2]
        int   13h
        jc  .GoOnReading
        add   esp, 2
        pop   bp
        ret
```

图 8 读软盘扇区的函数

由于上述代码用到堆栈，故需要在程序开头初始化堆栈，初始化 ss 和 esp

```
LABEL_START:
        mov     ax, cs
        mov     ds, ax
        mov     es, ax
        mov     ss, ax
        mov     sp, BaseOfStack

        xor     ah, ah  ; `.
        xor     dl, dl  ;  |  软驱复位
        int     13h     ; /
```

图 9 初始化 ss 和 esp

现在开始完成关于在软盘中寻找 Loader.bin 的函数：

```
; 下面在 A 盘的根目录寻找 LOADER.BIN
        mov     word [wSectorNo], SectorNoOfRootDirectory
LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
        cmp     word [wRootDirSizeForLoop], 0   ;  `. 判断根目录区是不是已经读完
        jz      LABEL_NO_LOADERBIN              ;  /  如果读完表示没有找到 LOADER.BIN
        dec     word [wRootDirSizeForLoop]      ; /
        mov     ax, BaseOfLoader
        mov     es, ax                  ; es <- BaseOfLoader
        mov     bx, OffsetOfLoader      ; bx <- OffsetOfLoader
        mov     ax, [wSectorNo]         ; ax <- Root Directory 中的某 Sector 号
        mov     cl, 1
        call    ReadSector

        mov     si, LoaderFileName      ; ds:si -> "LOADER  BIN"
        mov     di, OffsetOfLoader      ; es:di -> BaseOfLoader:0100
        cld
        mov     dx, 10h
LABEL_SEARCH_FOR_LOADERBIN:
        cmp     dx, 0                                   ; `. 循环次数控制,
        jz      LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR ;  / 如果已经读完了一个 Sector,
        dec     dx                                      ; /  就跳到下一个 Sector
        mov     cx, 11
LABEL_CMP_FILENAME:
        cmp     cx, 0
        jz      LABEL_FILENAME_FOUND    ; 如果比较了 11 个字符都相等, 表示找到
        dec     cx
        lodsb                           ; ds:si -> al
        cmp     al, byte [es:di]
        jz      LABEL_GO_ON
        jmp     LABEL_DIFFERENT         ; 只要发现不一样的字符就表明本 DirectoryEntry
                                        ; 不是我们要找的 LOADER.BIN
LABEL_GO_ON:
        inc     di
        jmp     LABEL_CMP_FILENAME      ; 继续循环
```

```
LABEL_GO_ON:
        inc     di
        jmp     LABEL_CMP_FILENAME      ; 继续循环

LABEL_DIFFERENT:
        and     di, 0FFE0h              ; else `. di &= E0 为了让它指向本条目开头
        add     di, 20h                 ;       |
        mov     si, LoaderFileName      ;       | di += 20h  下一个目录条目
        jmp     LABEL_SEARCH_FOR_LOADERBIN;    /

LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
        add     word [wSectorNo], 1
        jmp     LABEL_SEARCH_IN_ROOT_DIR_BEGIN

LABEL_NO_LOADERBIN:
        mov     dh, 2                   ; "No LOADER."
        call    DispStr                 ; 显示字符串
%ifdef  _BOOT_DEBUG_
        mov     ax, 4c00h               ; `.
        int     21h                     ; /  没有找到 LOADER.BIN, 回到 DOS
%else
        jmp     $                       ; 没有找到 LOADER.BIN, 死循环在这里
%endif

LABEL_FILENAME_FOUND:                   ; 找到 LOADER.BIN 后便来到这里继续
        jmp     $                       ; 代码暂时停在这里
```

图 10 寻找 loader.bin 的函数

5．接下来写入 boot.bin，loader.bin 到软盘，执行：

nasm boot.asm -o boot.bin

nasm loader.asm -o loader.bin

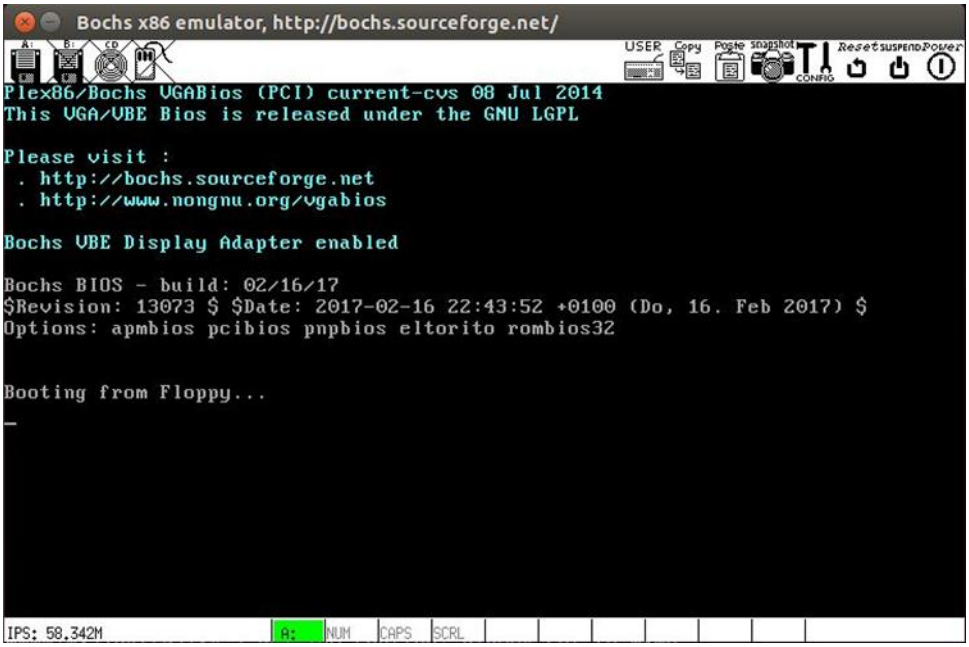
sudo mount -o loop a.img /mnt/floppy/

sudo cp boot.bin /mnt/floppy/

sudo cp loader.bin /mnt/floppy/

sudo umount /mnt/floppy/

sudo bochs -f bochsrc



因为此时 boot.bin 只是找到了 loader.bin，所以运行不会有效果。

2.2.3. 学习将指定的可执行文件装入指定内存区的方法，并调试执行，记录原理与步骤

1. 加载 Loader 进入内存

要将一个文件加载进入内存的话，需要读取软盘，那么就会用到 BIOS 的 13h 号中断，具体来说如下：

AH 取值	功能
00h	复位磁盘驱动器
01h	检查磁盘驱动器状态
02h	读扇区
03h	写扇区
04h	校验扇区
05h	格式化磁道
08h	获取驱动器参数
09h	初始化硬盘驱动器参数
0Ch	寻道
0Dh	复位磁盘控制器
15h	获取驱动器类型

其他的参数为：

参数	功能
AL	处理对象扇区数（连续的扇区）
CH	柱面号
CL	扇区号
DH	磁头号
DL	驱动器号
ES:BX	缓冲地址（校验及寻道时不用）
CF	判断是否校验成功

可以看到从 DL 号驱动器的 DH 磁头 CH 柱面 CL 扇区开始连续读取 AL 个扇区并存入 ES:BX 指向的缓冲区中。软盘有 2 个磁头、80 个柱面，每个柱面有 18 个扇区，所以对于模拟软盘可以通过如下方式求得各项参数：

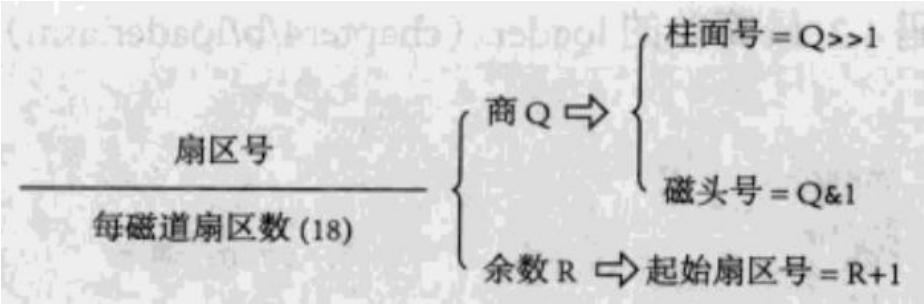


图 11 参数的计算方法

我们可以基于该计算方法写出读扇区函数

```
;-------------------------------------------------------------------------
; 函数名: ReadSector
;-------------------------------------------------------------------------
; 作用:
; 从第 ax 个 Sector 开始, 将 cl 个 Sector 读入 es:bx 中
ReadSector:
    ; -----------------------------------------------------------------------
    ; 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
    ;                           ┌ 柱面号 = y >> 1
    ;       x           ┌ 商 y ┤
    ; -------------- => ┤      └ 磁头号 = y & 1
    ;  每磁道扇区数     │
    ;                   └ 余 z => 起始扇区号 = z + 1
    push bp
    mov bp, sp
    sub esp, 2 ; 辟出两个字节的堆栈区域保存要读的扇区数: byte [bp-2]

    mov byte [bp-2], cl
    push bx     ; 保存 bx
    mov bl, [BPB_SecPerTrk] ; bl: 除数
    div bl      ; y 在 al 中, z 在 ah 中
    inc ah      ; z ++
    mov cl, ah  ; cl <- 起始扇区号
    mov dh, al  ; dh <- y
    shr al, 1   ; y >> 1 (y/BPB_NumHeads)
    mov ch, al  ; ch <- 柱面号
    and dh, 1   ; dh & 1 = 磁头号
    pop bx      ; 恢复 bx
    ; 至此, "柱面号, 起始扇区, 磁头号" 全部得到
    mov dl, [BS_DrvNum]  ; 驱动器号 (0 表示 A 盘)
.GoOnReading:
```

```
    mov ah, 2   ; 读
    mov al, byte [bp-2]  ; 读 al 个扇区
    int 13h
    jc .GoOnReading  ; 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
    add esp, 2
    pop bp

    ret

```

为了将 Loader 读取到内存中，我们需要知道 Loader 的起始簇号（在根目录中）和簇链（在文件分区表中），所以我们要遍历根目录来找到 Loader 的目录项来确定起始簇号，再带着起始簇号在 FAT 中得到簇链，从而将 Loader 读取进入内存，软盘中寻找 Loader.bin 的具体代码如下：

```
    xor ah, ah ; `.
    xor dl, dl ;  |  软驱复位
    int 13h ; /

; 下面在 A 盘的根目录寻找 LOADER.BIN
    mov word [wSectorNo], SectorNoOfRootDirectory
LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
    cmp word [wRootDirSizeForLoop], 0 ;  `. 判断根目录区是不是已经读完
    jz LABEL_NO_LOADERBIN  ;  /  如果读完表示没有找到 LOADER.BIN
    dec word [wRootDirSizeForLoop] ; /
    mov ax, BaseOfLoader
    mov es, ax   ; es <- BaseOfLoader
    mov bx, OffsetOfLoader ; bx <- OffsetOfLoader
    mov ax, [wSectorNo]  ; ax <- Root Directory 中的某 Sector 号
    mov cl, 1
    call ReadSector

    mov si, LoaderFileName ; ds:si -> "LOADER  BIN"
    mov di, OffsetOfLoader ; es:di -> BaseOfLoader:0100
    cld
    mov dx, 10h
LABEL_SEARCH_FOR_LOADERBIN:
    cmp dx, 0       ; `. 循环次数控制,
    jz LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR ;  / 如果已经读完了一个 Sector,
    dec dx       ; /  就跳到下一个 Sector
    mov cx, 11
LABEL_CMP_FILENAME:
    cmp cx, 0
```

```
29.     jz LABEL_FILENAME_FOUND ; 如果比较了 11 个字符都相等，表示找到
30.     dec cx
31.     lodsb    ; ds:si -> al
32.     cmp al, byte [es:di]
33.     jz LABEL_GO_ON
34.     jmp LABEL_DIFFERENT  ; 只要发现不一样的字符就表明本 DirectoryEntry
35.         ; 不是我们要找的 LOADER.BIN
36.   LABEL_GO_ON:
37.     inc di
38.     jmp LABEL_CMP_FILENAME ; 继续循环
39.
40.   LABEL_DIFFERENT:
41.     and di, 0FFE0h  ; else `. di &= E0 为了让它指向本条目开头
42.     add di, 20h   ;       |
43.     mov si, LoaderFileName ;       | di += 20h  下一个目录条目
44.     jmp LABEL_SEARCH_FOR_LOADERBIN;    /
45.
46.   LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
47.     add word [wSectorNo], 1
48.     jmp LABEL_SEARCH_IN_ROOT_DIR_BEGIN
49.
50.   LABEL_NO_LOADERBIN:
51.     mov dh, 2   ; "No LOADER."
52.     call DispStr   ; 显示字符串
53.   %ifdef _BOOT_DEBUG_
54.     mov ax, 4c00h  ; `.
55.     int 21h   ; /  没有找到 LOADER.BIN，回到 DOS
56.   %else
57.     jmp $   ; 没有找到 LOADER.BIN，死循环在这里
58.   %endif
59.
60.   LABEL_FILENAME_FOUND:   ; 找到 LOADER.BIN 后便来到这里继续
61.     jmp $   ; 代码暂时停在这里
```

接下来写入 boot.bin，loader.bin 到软盘，执行以下命令：

- nasm boot.asm -o boot.bin
- nasm loader.asm -o loader.bin
- sudo mount -o loop a.img /mnt/floppy/
- sudo cp boot.bin /mnt/floppy/
- sudo cp loader.bin /mnt/floppy/
- sudo umount /mnt/floppy/

• sudo bochs -f bochsrc

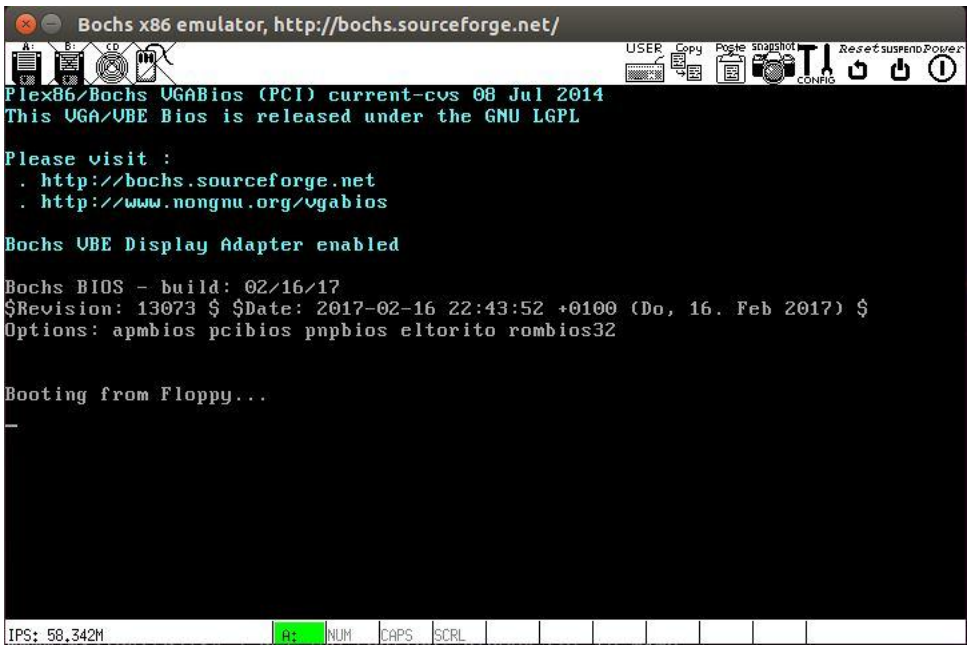


图 12 启动 bochs

发现直接启动没有什么现象，因为我们仅仅找到 Loader.bin 就让程序停止了，这里我们通过加断点进行调试一下：

1. bochs1：b 0x7c00　在开始处设置断点，因为 bios 将 boot sector 加载到了 0x7c00 处
2. bochs2：c　　执行到断点
3. bochs3：n　　跳过 BPB
4. bochs4：u /45　　反汇编
5. bochs5：b 0x7cb4　在 jmp $ 设置断点
6. bochs6：c　　执行到断点
7. bochs7：x /32xb es:di -16
8. 查看 es:di 前后的内存
9. bochs8：x /13xcb es:di -11 发现 es:di 前为我们要找的文件名
10. bochs9：sreg　　查看 es
11. bochs10：r　　查看 di

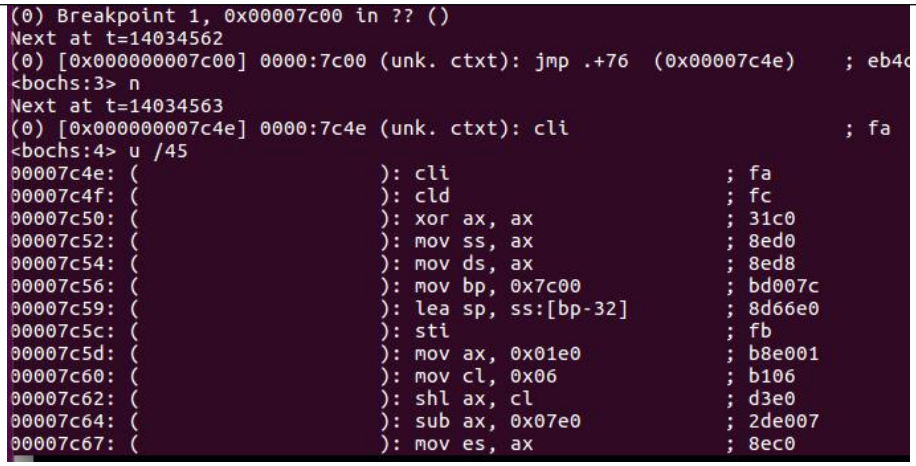


图 13 调试 1

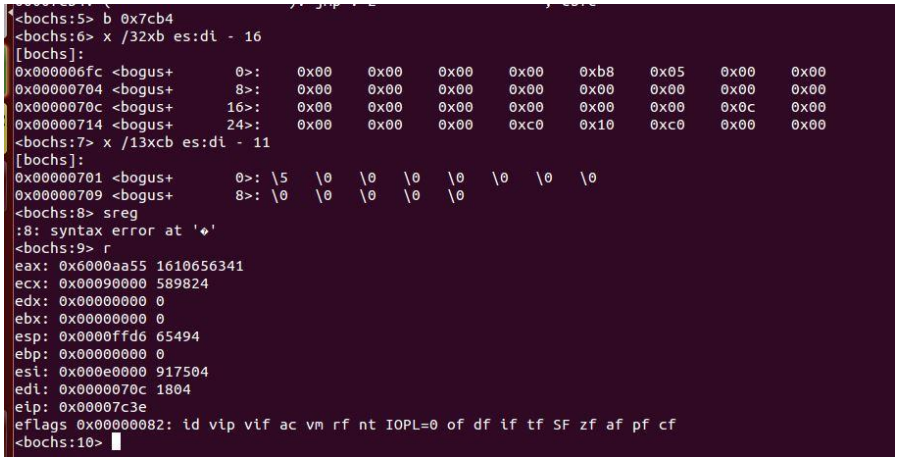


图 14 调试 2

这里我们就可以确定成功找到了文件，接下来我们完成将文件加入内存。

在根目录中找到 loader 之后，可以看到上面的代码中保存了簇号，然后调用了过程 GetFATEntry，其作用是查找当前簇号在 FAT 表中对应的值（来判断当前簇是不是最后一簇，如果不是的话可以得到下一个簇号），从而可以得到簇链：

```
1.  ;-------------------------------------------------------------------------
2.  ; 函数名: GetFATEntry
3.  ;-------------------------------------------------------------------------
4.  ; 作用:
5.  ; 找到序号为 ax 的 Sector 在 FAT 中的条目, 结果放在 ax 中
6.  ; 需要注意的是, 中间需要读 FAT 的扇区到 es:bx 处, 所以函数一开始保存了 es 和 bx
7.  GetFATEntry:
8.   push es
9.   push bx
10.  push ax
11.  mov ax, BaseOfLoader; `.
12.  sub ax, 0100h ;  | 在 BaseOfLoader 后面留出 4K 空间用于存放 FAT
13.  mov es, ax  ; /
```

```
    pop ax
    mov byte [bOdd], 0
    mov bx, 3
    mul bx   ; dx:ax = ax * 3
    mov bx, 2
    div bx   ; dx:ax / 2  ==>  ax <- 商, dx <- 余数
    cmp dx, 0
    jz LABEL_EVEN
    mov byte [bOdd], 1
LABEL_EVEN:;偶数
    ; 现在 ax 中是 FATEntry 在 FAT 中的偏移量,下面来
    ; 计算 FATEntry 在哪个扇区中(FAT 占用不止一个扇区)
    xor dx, dx
    mov bx, [BPB_BytsPerSec]
    div bx ; dx:ax / BPB_BytsPerSec
        ;  ax <- 商 (FATEntry 所在的扇区相对于 FAT 的扇区号)
        ;  dx <- 余数 (FATEntry 在扇区内的偏移)
    push dx
    mov bx, 0 ; bx <- 0 于是, es:bx = (BaseOfLoader - 100):00
    add ax, SectorNoOfFAT1 ; 此句之后的 ax 就是 FATEntry 所在的扇区号
    mov cl, 2
    call ReadSector ; 读取 FATEntry 所在的扇区, 一次读两个, 避免在边界
         ; 发生错误, 因为一个 FATEntry 可能跨越两个扇区
    pop dx
    add bx, dx
    mov ax, [es:bx]
    cmp byte [bOdd], 1
    jnz LABEL_EVEN_2
    shr ax, 4
LABEL_EVEN_2:
    and ax, 0FFFh

LABEL_GET_FAT_ENRY_OK:

    pop bx
    pop es
    ret
```

可以看到在上面的函数中区分了扇区号的奇偶性，原因是 FAT12 是一个簇号占 12 位，两扇区才对齐一次。也就是说扇区号为奇数的时候，该扇区第一个字节为一个新的簇号的开始；而当扇区号为偶数的时候，该扇区的第一个字节和上一个字节的高四位共同组成一个簇号，第二字节才是新的簇号，所以要进行区分，这同时也是代码中一次读两个扇区的原因。

在完成在根目录中找到首簇，在FAT找到簇链的功能后就可以加载loader：

```
LABEL_FILENAME_FOUND:   ; 找到 LOADER.BIN 后便来到这里继续
 mov ax, RootDirSectors
 and di, 0FFE0h  ; di -> 当前条目的开始
 add di, 01Ah  ; di -> 首 Sector
 mov cx, word [es:di]
 push cx   ; 保存此 Sector 在 FAT 中的序号
 add cx, ax
 add cx, DeltaSectorNo ; cl <- LOADER.BIN 的起始扇区号(0-based)
 mov ax, BaseOfLoader
 mov es, ax   ; es <- BaseOfLoader
 mov bx, OffsetOfLoader ; bx <- OffsetOfLoader
 mov ax, cx   ; ax <- Sector 号

LABEL_GOON_LOADING_FILE:
 push ax   ; `.
 push bx   ;  |
 mov ah, 0Eh   ;  | 每读一个扇区就在 "Booting  " 后面
 mov al, '.'   ;  | 打一个点，形成这样的效果:
 mov bl, 0Fh   ;  | Booting ......
 int 10h   ;  |
 pop bx   ;  |
 pop ax   ; /

 mov cl, 1
 call ReadSector
 pop ax   ; 取出此 Sector 在 FAT 中的序号
 call GetFATEntry
 cmp ax, 0FFFh
 jz LABEL_FILE_LOADED
 push ax   ; 保存 Sector 在 FAT 中的序号
 mov dx, RootDirSectors
 add ax, dx
 add ax, DeltaSectorNo
 add bx, [BPB_BytsPerSec]
 jmp LABEL_GOON_LOADING_FILE
LABEL_FILE_LOADED:

 mov dh, 1   ; "Ready."
 call DispStr   ; 显示字符串
```

这个时候loader被加载到内存中，接着移交控制权开始执行loader：

```
jmp BaseOfLoader:OffsetOfLoader ; 这一句正式跳转到已加载到内
      ; 存中的 LOADER.BIN 的开始处，
```

```
3.          ; 开始执行 LOADER.BIN 的代码。
4.          ; Boot Sector 的使命到此结束
```

2．调试运行

loader 的功能为打印字符 L，调试结果如下：

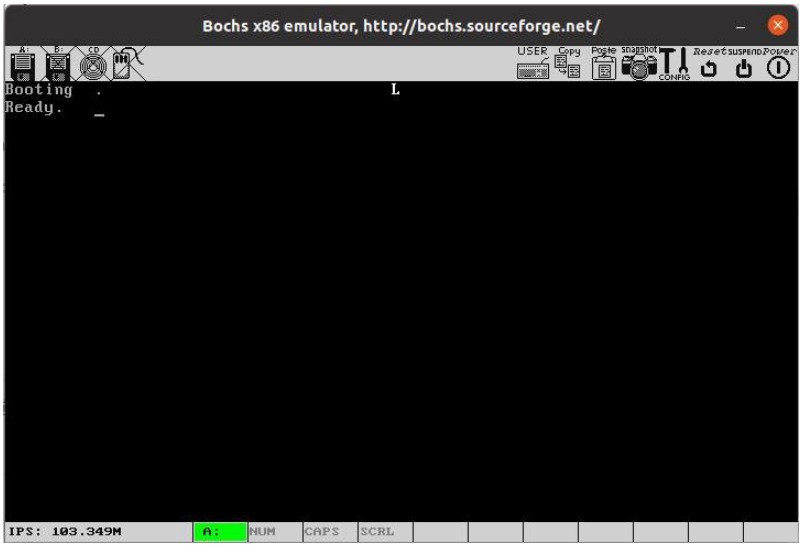


图 15 调试结果

运行调试结果发现第一行的“Booting”后有一个点，loader 确实占用一个扇区，并且打印出了字符 L，实验结果符合预期，loader 被正确加载并执行。

2.2.4. 学会使用 xxd 读取二进制信息，读取步骤 2 中写入的文件

接着实验步骤第 2 部分往下：

用 xxd 查看引导扇区内容

```
0000000: EB 3C 90 46 6F 72 72 65 73 74 59 00 02 01 01 00  .<.ForrestY.....
0000010: 02 E0 00 40 0B F0 09 00 12 00 02 00 00 00 00 00  ...@............
0000020: 00 00 00 00 00 00 29 00 00 00 00 4F 72 61 6E 67  ......)....Orang
0000030: 65 53 30 2E 30 32 46 41 54 31 32 20 20 20 8C C8  eS0.02FAT12   ..
0000040: 8E D8 8E C0 8E D0 BC 00 7C 30 E4 30 D2 CD 13 C7  ........|0.0....
0000050: 06 B8 7C 13 00 81 3E B6 7C 00 00 74 50 FF 0E B6  ..|...>.|..tP...
0000060: 7C B8 00 90 8E C0 BB 00 01 A1 B8 7C B1 01 E8 8D  |..........|....
0000070: 00 BE BB 7C BF 00 01 FC BA 10 00 81 FA 00 00 74  ...|...........t
0000080: 24 4A B9 0B 00 81 F9 00 00 74 29 49 AC 26 3A 05  $J.......t)I.&:.
0000090: 74 03 E9 03 00 47 EB ED 81 E7 E0 FF 81 C7 20 00  t....G........ .
00000a0: BE BB 7C EB D6 81 06 B8 7C 01 00 EB A8 B6 02 E8  ..|.....|.......
00000b0: 30 00 EB FE EB FE 0E 00 00 00 00 4C 4F 41 44 45  0..........LOADE
00000c0: 52 20 20 42 49 4E 00 42 6F 6F 74 69 6E 67 20 20  R  BIN.Booting
00000d0: 52 65 61 64 79 2E 20 20 20 4E 6F 20 4C 4F 41 44  Ready.    No LOAD
00000e0: 45 52 B8 09 00 F6 E6 05 C7 7C 89 C5 8C D8 8E C0  ER.......|......
00000f0: B9 09 00 B8 01 13 BB 07 00 B2 00 CD 10 C3 55 89  ..............U.
0000100: E5 66 81 EC 02 00 00 00 88 4E FE 53 8A 1E 18 7C  .f.......N.S...|
0000110: F6 F3 FE C4 88 E1 88 C6 D0 E8 88 C5 80 E6 01 5B  ...............[
0000120: 8A 16 24 7C B4 02 8A 46 FE CD 13 72 F7 66 81 C4  ..$|...F...r.f..
0000130: 02 00 00 00 5D C3 00 00 00 00 00 00 00 00 00 00  ....]...........
0000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
*
00001f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  ..............U.
```

图 16 查看引导扇区内容

加断点反汇编调试，使用 xxd 命令查询二进制信息。

首先执行 b 0x7c00，在 0x7c00 处加断点，因为 bios 把 boot sector 加载到

0x7c00 处。

使用 c 到达断点处，再使用 n 单步执行并跳过函数，跳过了 BPB。

```
(0) Breakpoint 1, 0x00007c00 in ?? ()
Next at t=2068273
(0) [0x000000007c00] 0000:7c00 (unk. ctxt): jmp .+60 (0x00007c3e)     ; eb3c
<bochs:3> n
Next at t=2068274
(0) [0x000000007c3e] 0000:7c3e (unk. ctxt): mov ax, cs                ; 8cc8
<bochs:4> u
00007c3e: (                   ): mov ax, cs                ; 8cc8
<bochs:5> u /45
00007c3e: (                   ): mov ax, cs                ; 8cc8
00007c40: (                   ): mov ds, ax                ; 8ed8
00007c42: (                   ): mov es, ax                ; 8ec0
00007c44: (                   ): mov ss, ax                ; 8ed0
00007c46: (                   ): mov sp, 0x7c00            ; bc007c
00007c49: (                   ): xor ah, ah                ; 30e4
00007c4b: (                   ): xor dl, dl                ; 30d2
00007c4d: (                   ): int 0x13                  ; cd13
00007c4f: (                   ): mov word ptr ds:0x7cb8, 0x0013 ; c706b87c1300
00007c55: (                   ): cmp word ptr ds:0x7cb6, 0x0000 ; 813eb67c0000
00007c5b: (                   ): jz .+80                   ; 7450
00007c5d: (                   ): dec word ptr ds:0x7cb6    ; ff0eb67c
00007c61: (                   ): mov ax, 0x9000            ; b80090
00007c64: (                   ): mov es, ax                ; 8ec0
00007c66: (                   ): mov bx, 0x0100            ; bb0001
00007c69: (                   ): mov ax, word ptr ds:0x7cb8 ; a1b87c
00007c6c: (                   ): mov cl, 0x01              ; b101
00007c6e: (                   ): call .+141                ; e88d00
00007c71: (                   ): mov si, 0x7cbb            ; bebb7c
00007c74: (                   ): mov di, 0x0100            ; bf0001
00007c77: (                   ): cld                       ; fc
00007c78: (                   ): mov dx, 0x0010            ; ba1000
00007c7b: (                   ): cmp dx, 0x0000            ; 81fa0000
00007c7f: (                   ): jz .+36                   ; 7424
00007c81: (                   ): dec dx                    ; 4a
00007c82: (                   ): mov cx, 0x000b            ; b90b00
00007c85: (                   ): cmp cx, 0x0000            ; 81f90000
00007c89: (                   ): jz .+41                   ; 7429
00007c8b: (                   ): dec cx                    ; 49
00007c8c: (                   ): lodsb al, byte ptr ds:[si] ; ac
00007c8d: (                   ): cmp al, byte ptr es:[di]  ; 263a05
00007c90: (                   ): jz .+3                    ; 7403
00007c92: (                   ): jmp .+3                   ; e90300
00007c95: (                   ): inc di                    ; 47
00007c96: (                   ): jmp .-19                  ; ebed
00007c98: (                   ): and di, 0xffe0            ; 81e7e0ff
00007c9c: (                   ): add di, 0x0020            ; 81c72000
00007ca0: (                   ): mov si, 0x7cbb            ; bebb7c
00007ca3: (                   ): jmp .-42                  ; ebd6
00007ca5: (                   ): add word ptr ds:0x7cb8, 0x0001 ; 8106b87c0100
```

图 17 设置断点查看

使用 b 0x7cb4 在 boot.bin 的 jmp $处下断点，使用 c 跳到断点，然后：

x /32xb es:di - 16 ←查看 es:di 前后的内存

x /13xcb es:di - 11 ←容易发现 es:di 前乃我们要找的文件名

sreg ←查看 es

r 查看 di

```
<bochs:13> r
eax: 0x0000004e 78
ecx: 0x00090000 589824
edx: 0x0000000e 14
ebx: 0x00000100 256
esp: 0x00007c00 31744
ebp: 0x00000000 0
esi: 0x000e7cbf 949439
edi: 0x0000012b 299
eip: 0x00007cad
```

拷贝成功

三、实验结果总结

（对实验结果进行分析，完成思考题目，并提出实验的改进意见）

4.1. 思考题

4.1.1. FAT12 格式是怎样的？

详见具体实验步骤中的“学习 FAT12 文件结构”

4.1.2. 如何读取一张软盘的信息？

1. 打开软盘镜像文件：首先，需要以二进制模式打开软盘的镜像文件。

2. 读取扇区数据：软盘是以扇区为单位进行存储和读取的。可以使用文件操作函数从软盘镜像文件中读取特定扇区的数据。

3. 解析扇区数据：读取扇区数据后，需要根据软盘的文件系统结构来解析数据。对于 FAT12 文件系统，需要理解 FAT 表、根目录和文件数据区等结构。

4. 提取所需信息：根据需求，可以从扇区数据中提取所需的信息，如文件名、文件大小、文件起始簇号等。

4.1.3. 如何在软盘中找到指定的文件？

1. 读取根目录：FAT12 文件系统中，根目录通常位于第一个扇区（扇区号为 19）。可以读取这个扇区的数据来获取根目录中的条目信息。

2. 解析根目录：根目录中的每个条目通常占用 32 字节，包括文件名、扩展名、文件属性、文件大小等信息。需要遍历根目录中的所有条目，并根据要查找的文件名或其他标识来识别所需文件的条目。

3. 获取文件簇号：一旦找到了所需文件的条目，可以从条目中获取文件的起始簇号。这个簇号将帮助在 FAT 表中找到文件的簇链。

4. 遍历 FAT 表：使用文件的起始簇号，可以遍历 FAT 表来找到文件的所有簇号，这些簇号构成了文件的数据区。需要查找 FAT 表中的簇号条目，直到找到文件的结束标记。

5. 读取文件数据：一旦知道了文件的簇号链，可以使用这些簇号来读取文件的数据。需要根据簇号找到相应的扇区并读取数据，然后将它们合并成完整的文件内容。

4.1.4. 如何在系统引导过程读取并加载一个可执行文件到内存，转交控制权？

本问题在实验内容部分已经有所涉及，简单的讲就是读 BPB 找到根目录和 FAT 表的位置，然后读根目录找到首簇簇号，将首簇簇号对应的扇区的数据读入

内存指定地址，再带入 FAT 表得到簇链的下一项，然后重复读入内存、带入 FAT 表，直至簇链结束，就可以将文件加载进入内存，然后直接跳转到加载的位置就可以移交控制权。

4.1.5. 为什么需要这个 Loader 程序不包含 dos 系统调用？

Loader 程序不包含 DOS 系统调用的原因是为了保持 Loader 的简单性和独立性。Loader 的主要任务是加载操作系统内核或其他程序，并将控制权转交给它们。如果 Loader 包含 DOS 系统调用或其他高级功能，它将变得复杂，可能需要依赖操作系统的支持。但在 Loader 启动之前，操作系统还没有加载，因此无法使用 DOS 系统调用。因此，Loader 通常只包含最基本的功能，例如加载内核和设置硬件环境，以便操作系统能够正常运行。

4.1.6. 为什么前面几个章节中的 a.img、pm.img 等文件不能直接 mount，在本章代码里面却可以？

在前几个章节中，a.img、pm.img 等文件不能直接 mount 的是因为它们的只是磁盘的二进制映像，而没有文件系统结构。

在本章代码中，我们已经学习了 FAT12 文件结构，并手工向软盘镜像文件写入了一个文件，这意味着我们已经为这些文件创建了有效的文件系统结构，包括分区表和引导记录。因此，在本章代码中，这些文件可以被正确地 mount，因为操作系统能够识别并理解它们的文件系统结构，从而正确加载文件。

4.1.7. 扩展提高：调研在硬盘上，文件系统格式为 FAT32 或者 NTFS，应该怎么来实现类似功能呢？

1．准备引导设备：与 FAT12 不同，FAT32 和 NTFS 文件系统通常存储在硬盘上而不是软盘上。首先，你需要准备一个引导设备，可以是硬盘、SSD、U 盘等，然后将 Loader 程序复制到该设备上的一个合适位置。

2．BIOS 或 UEFI 引导：在 BIOS 或 UEFI 系统中，你需要配置引导顺序，以便在启动时选择引导设备，就像之前提到的软盘一样。UEFI 系统还提供了更现代和灵活的引导机制。

3．读取文件系统：一旦引导设备被选为引导源，引导程序需要了解文件系统的结构，并定位 Loader 程序。对于 FAT32 文件系统，你需要解析 FAT 表来查找 Loader 程序的位置。对于 NTFS 文件系统，你需要了解 NTFS 的数据结构和

元数据来定位 Loader。

4．加载 Loader 程序：一旦找到 Loader 程序的位置，引导程序需要将 Loader 程序加载到内存中。这通常涉及到读取文件数据并将其加载到适当的内存地址。

5．转交控制权：引导程序需要将控制权转交给 Loader 程序。这通常是通过跳转指令或函数调用来实现的，将控制权传递给 Loader 程序的入口点。

6．Loader 程序执行：一旦控制权被传递给 Loader 程序，它可以继续执行自己的任务，如加载操作系统内核、初始化系统环境等。

4.2. 对实验的改进意见

4.2.1. 王卓

1.提供更详细的实验指导：在每个实验步骤中，提供更详细的指导和说明，比如预期结果等，这样能帮助我们更好地完成实验。

2.介绍实验目的和背景：实验开始之前提供实验的目的和背景，解释一下为什么需要进行该实验以及其与操作系统的关系，这能帮助我们更好理解实验的意义和重要性。

4.2.2. 程子洋

错误排除指南：提供学生在遇到常见问题时进行自我排除的指南。这可以包括常见错误消息的解释以及如何解决这些问题的步骤。

4.2.3. 聂森

1.可以先让同学阅读教材资料后再介绍实验内容和实验相关的知识，可以帮助同学更快更好的了解实验的目标和实验的原理及任务

2.给出具体的实验指导获给出更加丰富的实验参考资料，可以帮助同学更快上手实验并在有问题时找到解决方式

4.2.4. 刘虓

如果能有详细的操作教学可以帮助学生更快地上手，并且希望老师能讲解汇编代码的一些关键部分，学生自己阅读汇编码容易忽略一些问题。

四、各人实验贡献与体会（每人各自撰写）

5.1. 王卓

5.1.1. 分工

在本次实验中，本人完成全部实验内容，负责实验内容第 3 题、基础题第 4 题实验报告的撰写与实验报告的最终整合。

5.1.2. 心得体会

在手工定位文件中，我对 FAT12 系统各部分的结构有了一个系统的梳理，能够理清其中的逻辑，并总结出定位文件的一般步骤。同时，我进一步了解了 bios 硬盘中断的使用功能方法、使用汇编代码读取 FAT12 中文件的方法，成功将 Loader 程序读入了内存，为加载操作系统打下了坚实的基础。通过本次实验，我们熟悉了各种二进制编辑工具的使用，尤其是在 linux 下使用 xxd 进行二进制读取与编辑的方法。收货颇满！

5.2. 程子洋

5.2.1. 分工

此次实验为本人独立完成大部分实验内容，并主要负责实验内容第 1 题，思考题第 1 题，第 7 题实验报告的撰写。

5.2.2. 心得体会

这次实验我有很大的收获，总体上来说，本周的实验不算很复杂，但仍有很多值得慢慢分析和理解的地方，首先有很多新的概念要去熟悉；同时，阅读汇编代码、熟悉语句和段落的功能也是较为困难的任务。

通过本次实验，我深入地分析、理解并掌握了以下的内容：FAT12 文件系统格式和工作方式、向软盘镜像文件写入文件的方法、读取文件在磁盘中的信息的方法、学会使用 xxd 读取二进制信息、学会通过 int 13h 读磁盘、读取其扇区信息、学会将指定文件装入指定内存区并执行……对其各种相关知识都有了一定程度的理解和自我的掌握；同时，对汇编代码的阅读和分析过程，也对我自己汇编的语法和代码编写的知识和能力带来的极大的锻炼和提升。

5.3. 聂森

5.3.1. 分工

在本次实验中，本人完成大部分题目，并主要负责实验内容第四小问和实验思考第六七小问实验报告的撰写。

5.3.2. 心得体会

在本次实验课中，我通过学习 FAT12 文件结构，我理解了文件系统的基本概念和结构，通过手工向软盘镜像文件写入文件的过程让我更加深入地了解了磁盘操作和数据存储的细节。同时，学会使用 xxd 工具读取二进制信息，有助于我分析和调试二进制数据，提高了我的实验效率。通过创建 Loader 程序，我学会了如何将一个可执行文件加载到内存中，并将控制权转交给它，这是操作系统启动的关键步骤。这个过程使我深刻体会到了操作系统的核心功能和启动流程，以及 Loader 的简洁性和独立性的重要性。希望自己能从实验中学会更多知识，掌握更多技能。

5.4. 刘虓

5.4.1. 分工

在本次实验中，本人完成大部分题目，并主要负责实验内容 2 和思考题 2，3。

5.4.2. 心得体会

通过学习向软盘镜像文件写入文件的方法、读取文件在磁盘中的信息的方法、学会使用 xxd 读取二进制信息、学会通过 int 13h 读磁盘、读取其扇区信息。我对各种相关知识都有了一定程度的理解和自我的掌握；同时，对汇编代码的阅读和分析过程，也对我自己汇编的语法和代码编写的知识和能力带来的极大的锻炼和提升。

五、教师评语

教师评分（请填写好姓名、学号）		
姓名	学号	分数
王卓	2021302191791	
程子洋	2021301051114	
聂森	2021301051114	
刘虓	2021302121234	
教师签名： 年　　月　　日		