|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |
| 1. 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？ | | | | |
| 1. 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） | | | | |
| * 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   * 1. **具体实验步骤**      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：    图2 软盘（1.44MB, FAT12）   1. 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。   1. 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。    图3 根目录区中的条目格式   1. 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。    * 1. **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** 5. 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。     图4 修改boot.asm   1. 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   1. 一个简单的loader   Loader.asm。编译为loader.bin    图6 编译为loader.bin   1. 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：    图7 bios中断int13h用法  boot.asm中增加的读软盘扇区的函数如下：    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    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：      图10 寻找loader.bin的函数   1. 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。   * + 1. **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：  20231107092142  **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   1. ;---------------------------------------------------------------------------- 2. ; 函数名: ReadSector 3. ;---------------------------------------------------------------------------- 4. ; 作用: 5. ; 从第 ax 个 Sector 开始, 将 cl 个 Sector 读入 es:bx 中 6. ReadSector: 7. ; ----------------------------------------------------------------------- 8. ; 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) 9. ; ----------------------------------------------------------------------- 10. ; 设扇区号为 x 11. ;                           ┌ 柱面号 = y >> 1 12. ;       x           ┌ 商 y ┤ 13. ; -------------- => ┤      └ 磁头号 = y & 1 14. ;  每磁道扇区数     │ 15. ;                   └ 余 z => 起始扇区号 = z + 1 16. push bp 17. mov bp, sp 18. 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] 19. mov byte [bp-2], cl 20. push bx   ; 保存 bx 21. mov bl, [BPB\_SecPerTrk] ; bl: 除数 22. div bl   ; y 在 al 中, z 在 ah 中 23. inc ah   ; z ++ 24. mov cl, ah   ; cl <- 起始扇区号 25. mov dh, al   ; dh <- y 26. shr al, 1   ; y >> 1 (y/BPB\_NumHeads) 27. mov ch, al   ; ch <- 柱面号 28. and dh, 1   ; dh & 1 = 磁头号 29. pop bx   ; 恢复 bx 30. ; 至此, "柱面号, 起始扇区, 磁头号" 全部得到 31. mov dl, [BS\_DrvNum]  ; 驱动器号 (0 表示 A 盘) 32. .GoOnReading: 33. mov ah, 2   ; 读 34. mov al, byte [bp-2]  ; 读 al 个扇区 35. int 13h 36. jc .GoOnReading  ; 如果读取错误 CF 会被置为 1, 37. ; 这时就不停地读, 直到正确为止 38. add esp, 2 39. pop bp 40. 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：   1. xor ah, ah ; `. 2. xor dl, dl ;  |  软驱复位 3. int 13h ; / 5. ; 下面在 A 盘的根目录寻找 LOADER.BIN 6. mov word [wSectorNo], SectorNoOfRootDirectory 7. LABEL\_SEARCH\_IN\_ROOT\_DIR\_BEGIN: 8. cmp word [wRootDirSizeForLoop], 0 ;  `. 判断根目录区是不是已经读完 9. jz LABEL\_NO\_LOADERBIN  ;  /  如果读完表示没有找到 LOADER.BIN 10. dec word [wRootDirSizeForLoop] ; / 11. mov ax, BaseOfLoader 12. mov es, ax   ; es <- BaseOfLoader 13. mov bx, OffsetOfLoader ; bx <- OffsetOfLoader 14. mov ax, [wSectorNo]  ; ax <- Root Directory 中的某 Sector 号 15. mov cl, 1 16. call ReadSector 17. mov si, LoaderFileName ; ds:si -> "LOADER  BIN" 18. mov di, OffsetOfLoader ; es:di -> BaseOfLoader:0100 19. cld 20. mov dx, 10h 21. LABEL\_SEARCH\_FOR\_LOADERBIN: 22. cmp dx, 0       ; `. 循环次数控制, 23. jz LABEL\_GOTO\_NEXT\_SECTOR\_IN\_ROOT\_DIR ;  / 如果已经读完了一个 Sector, 24. dec dx       ; /  就跳到下一个 Sector 25. mov cx, 11 26. LABEL\_CMP\_FILENAME: 27. cmp cx, 0 28. jz LABEL\_FILENAME\_FOUND ; 如果比较了 11 个字符都相等, 表示找到 29. dec cx 30. lodsb    ; ds:si -> al 31. cmp al, byte [es:di] 32. jz LABEL\_GO\_ON 33. jmp LABEL\_DIFFERENT  ; 只要发现不一样的字符就表明本 DirectoryEntry 34. ; 不是我们要找的 LOADER.BIN 35. LABEL\_GO\_ON: 36. inc di 37. jmp LABEL\_CMP\_FILENAME ; 继续循环 38. LABEL\_DIFFERENT: 39. and di, 0FFE0h  ; else `. di &= E0 为了让它指向本条目开头 40. add di, 20h   ;       | 41. mov si, LoaderFileName ;       | di += 20h  下一个目录条目 42. jmp LABEL\_SEARCH\_FOR\_LOADERBIN;    / 43. LABEL\_GOTO\_NEXT\_SECTOR\_IN\_ROOT\_DIR: 44. add word [wSectorNo], 1 45. jmp LABEL\_SEARCH\_IN\_ROOT\_DIR\_BEGIN 46. LABEL\_NO\_LOADERBIN: 47. mov dh, 2   ; "No LOADER." 48. call DispStr   ; 显示字符串 49. %ifdef \_BOOT\_DEBUG\_ 50. mov ax, 4c00h  ; `. 51. int 21h   ; /  没有找到 LOADER.BIN, 回到 DOS 52. %else 53. jmp $   ; 没有找到 LOADER.BIN, 死循环在这里 54. %endif 55. LABEL\_FILENAME\_FOUND:   ; 找到 LOADER.BIN 后便来到这里继续 56. 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  m  **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   20231107095814  **图13 调试1**  o  **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  ; / 14. pop ax 15. mov byte [bOdd], 0 16. mov bx, 3 17. mul bx   ; dx:ax = ax \* 3 18. mov bx, 2 19. div bx   ; dx:ax / 2  ==>  ax <- 商, dx <- 余数 20. cmp dx, 0 21. jz LABEL\_EVEN 22. mov byte [bOdd], 1 23. LABEL\_EVEN:;偶数 24. ; 现在 ax 中是 FATEntry 在 FAT 中的偏移量,下面来 25. ; 计算 FATEntry 在哪个扇区中(FAT占用不止一个扇区) 26. xor dx, dx 27. mov bx, [BPB\_BytsPerSec] 28. div bx ; dx:ax / BPB\_BytsPerSec 29. ;  ax <- 商 (FATEntry 所在的扇区相对于 FAT 的扇区号) 30. ;  dx <- 余数 (FATEntry 在扇区内的偏移) 31. push dx 32. mov bx, 0 ; bx <- 0 于是, es:bx = (BaseOfLoader - 100):00 33. add ax, SectorNoOfFAT1 ; 此句之后的 ax 就是 FATEntry 所在的扇区号 34. mov cl, 2 35. call ReadSector ; 读取 FATEntry 所在的扇区, 一次读两个, 避免在边界 36. ; 发生错误, 因为一个 FATEntry 可能跨越两个扇区 37. pop dx 38. add bx, dx 39. mov ax, [es:bx] 40. cmp byte [bOdd], 1 41. jnz LABEL\_EVEN\_2 42. shr ax, 4 43. LABEL\_EVEN\_2: 44. and ax, 0FFFh 45. LABEL\_GET\_FAT\_ENRY\_OK: 46. pop bx 47. pop es 48. 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：   1. LABEL\_FILENAME\_FOUND:   ; 找到 LOADER.BIN 后便来到这里继续 2. mov ax, RootDirSectors 3. and di, 0FFE0h  ; di -> 当前条目的开始 4. add di, 01Ah  ; di -> 首 Sector 5. mov cx, word [es:di] 6. push cx   ; 保存此 Sector 在 FAT 中的序号 7. add cx, ax 8. add cx, DeltaSectorNo ; cl <- LOADER.BIN的起始扇区号(0-based) 9. mov ax, BaseOfLoader 10. mov es, ax   ; es <- BaseOfLoader 11. mov bx, OffsetOfLoader ; bx <- OffsetOfLoader 12. mov ax, cx   ; ax <- Sector 号 13. LABEL\_GOON\_LOADING\_FILE: 14. push ax   ; `. 15. push bx   ;  | 16. mov ah, 0Eh   ;  | 每读一个扇区就在 "Booting  " 后面 17. mov al, '.'   ;  | 打一个点, 形成这样的效果: 18. mov bl, 0Fh   ;  | Booting ...... 19. int 10h   ;  | 20. pop bx   ;  | 21. pop ax   ; / 22. mov cl, 1 23. call ReadSector 24. pop ax   ; 取出此 Sector 在 FAT 中的序号 25. call GetFATEntry 26. cmp ax, 0FFFh 27. jz LABEL\_FILE\_LOADED 28. push ax   ; 保存 Sector 在 FAT 中的序号 29. mov dx, RootDirSectors 30. add ax, dx 31. add ax, DeltaSectorNo 32. add bx, [BPB\_BytsPerSec] 33. jmp LABEL\_GOON\_LOADING\_FILE 34. LABEL\_FILE\_LOADED: 35. mov dh, 1   ; "Ready." 36. 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：   1. jmp BaseOfLoader:OffsetOfLoader ; 这一句正式跳转到已加载到内 2. ; 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 5. 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。   * + 1. **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    **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。    **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    拷贝成功 | | | | |
| 1. 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） | | | | |
| * 1. **思考题**      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”   * + 1. **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。    * 1. **如何在软盘中找到指定的文件？** 5. 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。 6. 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。 7. 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。 8. 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。 9. 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。    * 1. **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。   * + 1. **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。   * + 1. **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。   * + 1. **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。    1. **对实验的改进意见**       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。   * + 1. **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。   * + 1. **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   * + 1. **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。 | | | | |
| 1. 各人实验贡献与体会（每人各自撰写） | | | | |
| * 1. **王卓**      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。   * + 1. **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！   * 1. **程子洋**      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。   * + 1. **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。   * 1. **聂森**      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。   * + 1. **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。   * 1. **刘虓**      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。   * + 1. **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。 | | | | |
| 1. 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 | | | | |