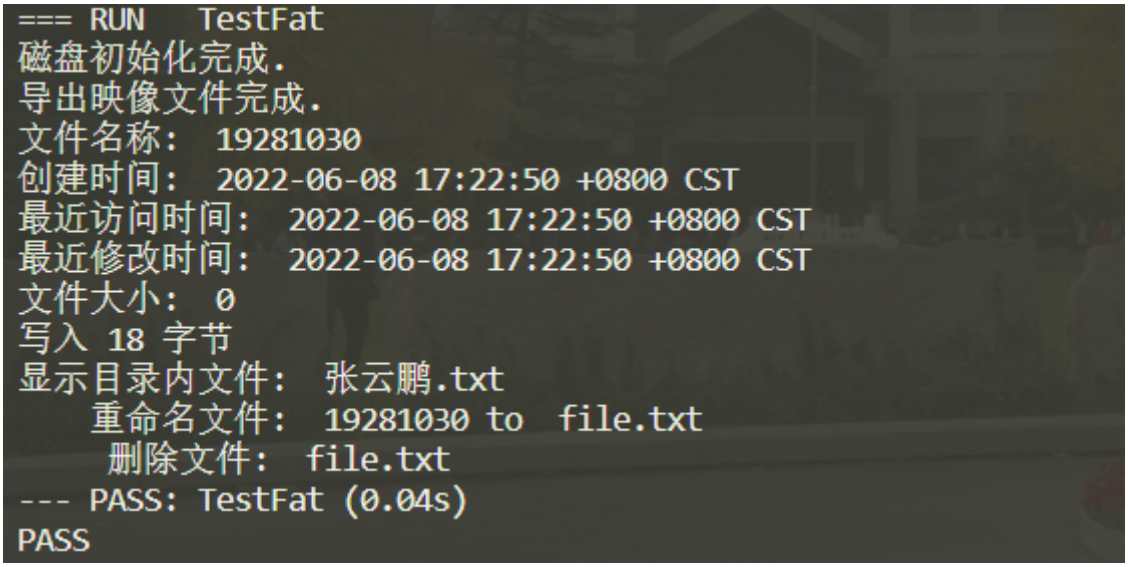
FAT 文件系统模拟设计与实现

19281030-张云鹏

实验截图



实验环境

- Ubuntu20.04 LTS

数据结构设计

磁盘

```
type Disk struct {
    DiskData []byte //磁盘所有的字节
    MBR      []byte
    FAT1     []byte
    FAT2     []byte
    RootDir  []byte //根目录
    FileData []byte //数据区
    CurDir   string //当前目录
}
```

INode目录/文件

```
type INoder interface {
    create()               //创建目录/文件
    rename(newName string) //重命名目录/文件
    cd(path string)        //进入目录
    show()                 //现实当前目录
```

```
    delete(address int)    //删除地址
}

type INode struct {
    name       string //目录/文件名
    createTime int64  //创建时间
    accessTime int64  //最近访问时间
    modifyTime int64  //最近修改时间
    size       int64  //文件大小
    fatHead    int    //fat首地址
    fatTail    int    //fat尾地址
    fileBytes  []byte //文件内容
}
```

磁盘常量

```
const (
    DISK_SIZE   = 1440 * 1024 //硬盘大小1.44MB
    SECTOR_SIZE = 512         //扇区大小
)
```

磁盘格式化和映像文件生成

磁盘格式化

- 磁盘划分为MBR,FAT1,FAT2,根目录,数据区,共5个区域
- 保存每个区域的首位地址.

```
func (disk *Disk) Init() {
    disk.DiskData = make([]byte, DISK_SIZE, DISK_SIZE)              //磁盘大小
1474560字节
    disk.MBR = disk.DiskData[0:SECTOR_SIZE]                         //引导扇区
    disk.FAT1 = disk.DiskData[SECTOR_SIZE : 10*SECTOR_SIZE]         //FAT1
    disk.FAT2 = disk.DiskData[10*SECTOR_SIZE : 19*SECTOR_SIZE]      //FAT2
    disk.RootDir = disk.DiskData[20*SECTOR_SIZE : 33*SECTOR_SIZE]   //根目录
    disk.FileData = disk.DiskData[33*SECTOR_SIZE : 2879*SECTOR_SIZE] //数据区
    fmt.Println("磁盘初始化完成.")
}
```

磁盘映像文件导出

原封不动将字节流写入到文件

```
func (disk *Disk) Export() {
    imgFileName := "img"
    os.Create(imgFileName)
```

```
    file, err := os.OpenFile(imgFileName, os.O_RDWR, os.ModePerm) //读写模式打开文
件
    if err != nil {
        fmt.Println("open file error")
    }
    file.Write(disk.DiskData) //写入字节流到文件，导出镜像文件
    fmt.Println("导出映像文件完成.")
}
```

目录/文件存取操作

创建

1. 获取空闲磁盘地址
2. 创建iNode项
3. 通过size判断是文件或是目录
4. 如果是文件则讲fileBytes分成多个块写入磁盘
5. 更新fat

```
func (iNode *INode) create(newNode *INode) {
    availableBlockAddress := getAvailableFatAddress()
    iNode.fatTail = availableBlockAddress
    fmt.Println("创建文件/目录，iNode保存fat地址: ", iNode.fatTail)
    fmt.Println("创建文件/目录，更新内容: ", iNode.fatTail)
    copy(disk.DiskData[availableBlockAddress:availableBlockAddress+64],
newNode.toBytes())
    if iNode.size != 0 {
        writeBytesToDisk(iNode.fileBytes)
    }
}
```

```
//获取空闲的磁盘块地址
func getAvailableFatAddress() int {
    for item := range disk.FAT1 {
        if item == 0 {
            return item
        }
    }
    panic("No Available Address")
    return -1
}
```

```
//将字节流复制到硬盘
func writeBytesToDisk(sourceBytes []byte) int {
    // var n = 0
    blockNum := len(sourceBytes)/64 + 1
```

```
    if len(sourceBytes)%64 == 0 {
        blockNum--
    }
    for i := 0; i < blockNum; i++ {
        availableFatAddress := getAvailableFatAddress()
        diskBlock := disk.DiskData[availableFatAddress : availableFatAddress+64]
        diskBlockBuffer := bytes.NewBuffer(diskBlock)
        limit := Min(len(sourceBytes), (i+1)*64)
        n, err := diskBlockBuffer.Write(sourceBytes[i*64 : limit])
        if err != nil {
            fmt.Println("写入硬盘失败")
        }
        fmt.Println("写入", n, "字节")
    }

    return 0
}
```

进入

1. 全局变量保存当前的目录
2. 需要显示,文件时以此为工作目录

```
func (iNode *INode) cd(dir INode) {
    disk.CurDir = dir.name
    fmt.Println("进入目录: ", dir.name)
}
```

重命名

1. 解析字节流
2. 修改字节块的前16个字节

```
func (iNode *INode) rename(newName string) {
    bytes.NewBuffer(iNode.fileBytes[:16]).Write([]byte(newName))
    fmt.Println("    重命名文件: ", iNode.name, "to ", newName)
}
```

显示

- 输出iNode的基本属性
- 当iNode为文件输出内容, 大文件同样适用
- 当iNode为目录时,输出目录内的文件信息

```
func (iNode *INode) toBytes() []byte {
    iNodeBytes := make([]byte, 0, 64)
```

```
    nameBytes := []byte(iNode.name)
    nameBytes = nameBytes[:16] //名称保留前16字节
    iNodeBytes = append(iNodeBytes, nameBytes...)
    fmt.Println("文件名称: ", string(nameBytes))

    t := new(bytes.Buffer)
    binary.Write(t, binary.LittleEndian, iNode.createTime) //创建时间8字节，小端序
    iNodeBytes = append(iNodeBytes, t.Bytes()...)
    fmt.Println("创建时间: ", time.Unix(iNode.createTime, 0))

    binary.Write(t, binary.LittleEndian, iNode.accessTime) //访问时间8字节，小端序
    iNodeBytes = append(iNodeBytes, t.Bytes()...)
    fmt.Println("最近访问时间: ", time.Unix(iNode.accessTime, 0))

    binary.Write(t, binary.LittleEndian, iNode.modifyTime) //修改时间8字节，小端序
    iNodeBytes = append(iNodeBytes, t.Bytes()...)
    fmt.Println("最近修改时间: ", time.Unix(iNode.modifyTime, 0))

    binary.Write(t, binary.LittleEndian, iNode.size) //文件大小8字节，小端序，定位Byte
    iNodeBytes = append(iNodeBytes, t.Bytes()...)
    fmt.Println("文件大小: ", iNode.size)
    // 如果不是目录则打印文件内容
    if iNode.size !=0{
        fmt.Println(string(iNode.fileBytes))
    }
    return iNodeBytes
}
```

删除

```
func (iNode *INode) delete(address int) {
    disk.FAT1[address] = 0 //fat置0，当再次申请时覆盖即视为删除
    fmt.Println("     删除文件: ", string(iNode.name))
}
```

重定向

- 文件特有, 大文件同样适用
- 重新设定输出字节数组
- 当标志为0时输出到终端

```
func Redirect(sourceBytes []byte, fileName string) {
    file, err := os.OpenFile(fileName, os.O_RDWR, os.ModePerm)
    if err != nil {
        fmt.Println("file open err!")
    }
    file.Write(sourceBytes)
}
```

虚拟机互通测试

本实验运行在真实主机, 互操作通过