fff

操作系统FAT12实现

1.FAT12简介

FAT12 是最早的文件分配表（File Allocation Table, FAT）格式之一，主要用于软盘和小容量的存储设备。

1.1 FAT12组成

- 引导扇区 1扇区
- FAT1 9扇区
- FAT2 9扇区
- 根目录区 14扇区
- 数据区 2847扇区

> 1. 引导扇区：主要存放一些磁盘的重要参数，如 每扇区字节数，共有几个FAT表，每个FAT表占用的扇区数，磁头的数量等等。
> 2. FAT1： FAT1的作用主要是与簇进行关联，我们可以通过FAT1表来把文件的簇连接起来，像是链表一样形成一个链条，从而完整的读取文件内容。如果指向了0xff8 - 0xfff，表示文件结束了。
> 3. FAT2： FAT1的备份，冗余备份作用，FAT1出错直接读取FAT2和恢复。
> 4. 根目录区：存放文件，文件夹的基本信息。根目录区一般数量有限，一般是224个或者更少，不能存放太多的文件。

5. 数据区：存放实际内容的区域，每个簇的固定大小在MBR中均有叙述，由于有FAT1我们可以把分散的扇区内的数据连接起来。同时也可以把写入数据分散存储。

1.2 簇的概念

簇是文件系统中的一个概念，指的是一组扇区的集合。文件在磁盘上是以簇为单位进行存储的。由于单个扇区的大小通常较小（例如512字节），直接以扇区为单位管理文件可能会导致大量的文件碎片和管理开销，因此文件系统将多个扇区组合成一个簇。在FAT12中 基本上是一个扇区为一个簇。

1.3 软盘的大小

标准的3.5英寸软盘可以存储1.44MB数据。它是由2880个512字节的扇区组成的。

计算如下：

2880扇区 * 512B = 1,474,560 字节

1,474,560B / 1MB = 1,440MB

1.4 根目录的格式

序号	字节编号	偏移量	偏移量	字节	含义
1	1~11	0	00h	11	文件名8字节，扩展名3字节
2	12	11	0bh	1	文件属性
3	13~22	12	0ch	10	保留位
4	23, 24	22	16h	2	最后一次写入时间
5	25, 26	24	18h	2	最后一次写入日期
6	27, 28	26	1ah	2	此条目对应的开始簇号
7	29~32	28	1ch	4	文件大小

1.5 时间提取

最后一次写入时间和最后一次写入日期各占用2个字节.我们使用位运算将其转换为我们熟悉的标准格式。

格式如下:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
时间	时0-23					分0-59						秒（*2）0-29				
日期	年（-1980）0-119							月1-12				日1-31				

如何提取:

例如: F1 AB 56 59

我们可以获取其最后一次写入时间是 F1 AB。小端存储，我们进行分析

```
    A       B          F          1

   1010    1011      1111       0001
```

由于我们需要获取最后一次写入时间。

```
lastWriteTime = (0xAB) << 8 | 0xF1

    1010 1011 0000 0000

|              1111 0001
```

```
    1010 1011  1111 0001
```

组成16位

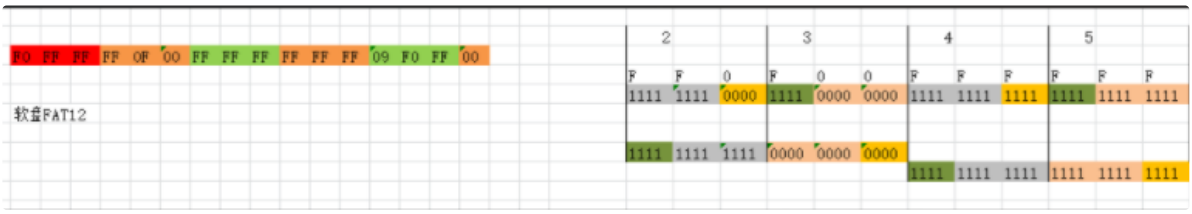
我们根据图示，将其右移11位可获得其最低地址的五位，将其与0x1F按位与即可获得小时。

同理，根据图示，将其右移 5位可获得其分钟位，将其与0x3F按位与即可获得分钟。

对于年份以及月份和日期我们采用同样的方法，唯一注意的是获得的年份是距离1980年的差值。需要加上1980才可以获取目前的时间。

1.6 FAT1表的理解

FAT1表的结构：



值得注意的是FAT1的前两个FAT表项留作特殊用途，所以我们一般从FAT的第三个表项开始计算。

由于FAT12每个表项占据12位，计算机为对齐（n*8字节），我们考虑将一个FAT表项囊括两个簇号，以便节省空间同时加快计算机读取速度。

```
     FAT[0]        FAT[1]       FAT[2]      FAT[3]      FAT[4]

    F0 FF FF     03 40 00     05 F0 FF    FF FF FF  FF FF FF

     FF0   FFF     003    004     005   FFF    FFF FFF FFF FFF

     0簇   1簇       2簇     3簇       4簇   5簇     6簇   7簇   8簇   9簇
```

对于FAT1[1] 作为起始，我们有其低字节与中字节部分，高字节部分。

所以我们的首个可用是由第二个字节的高地址部分 拼接 第一个字节的8位地址部分组成簇号。

第二个簇是由 第三个字节的全部部分 拼接 第二个字节的低位地址部分形成簇号。

在我们根据簇号寻找FAT表项时候只需要分奇偶项即可，若是偶数则直接寻找其低地址部分的二字节。

如果是奇数则直接寻找其高地址部分的二字节。

1.7 MBR结构

序号	字节编号	偏移量	偏移量	字节	含义
1	1,2,3	0	00h	3	一条短跳转指令
2	4~11	3	03h	8	厂商名
3	12,13	11	0bh	2	每扇区字节数
4	14	13	0dh	1	每簇扇区数
5	15,16	14	0eh	2	保留扇区数（boot记录）
6	17	16	10h	1	FAT表的数目
7	18,19	17	11h	2	根目录登记项数
8	20,21	19	13h	2	总扇区数
9	22	21	15h	1	磁盘介质描述
10	23,24	22	16h	2	每个FAT的扇区数
11	25,26	24	18h	2	每个磁道的扇区数
12	27,28	26	1ah	2	磁头数
13	29~32	28	1ch	4	隐藏扇区数
14	33~36	32	20h	4	逻辑驱动器总扇区数
15	37	36	24h	1	中断13的驱动器号
16	38	37	25h	1	保留
17	39	38	26h	1	扩展引导标志
18	40~43	39	27h	4	卷序列号
19	44~54	43	2bh	11	卷标
20	55~62	54	36h	8	文件系统类型
21	63~510	62	3eh	448	引导代码，数据，填充字符
22	511,512	510	1feh	2	055h, 0AAh

3.实现FAT12文件系统

3.1 简介介绍

为进一步了解文件系统的构造，了解FAT12文件系统的思想。我实现了一下功能：

- `dir` 命令，展示当前目录下的文件以及文件夹。
- `cat` 命令，使用cat 文件名 的命令，可以展示其文件的内容。
- `cd` 命令， 使用cd命令可以进入当前目录子文件夹以及上一层文件夹或回到根目录。
- `pwd` 命令，通过pwd命令可以获取当前用户所在路径。

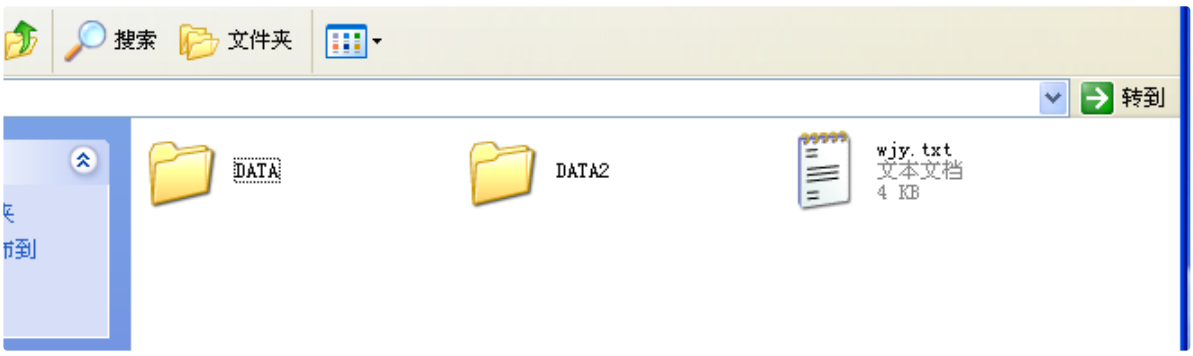
- `touch` 命令，通过touch命令用户可以创建需要的文件并且填入相应的内容保存到软盘。
- `rm` 命令，通过rm 文件名 命令可以移除根目录下的某个文件名。具体的实现过程在下方介绍。
- `mkdir` 命令，使用mkdir命令可以建立一个文件夹。
- `format` 命令，通过format命令可以格式化磁盘至初始化状态。

3.2 实验环境配置

本地使用GCC 11.2.0 进行开发，使用Windows11操作系统，使用JetBrains旗下的Clion 集成工具进行编码，同时使用winImage软件进行软盘的制造，以及使用了windows 7（虚拟机）挂载网盘查看具体细节。

3.2.1 软盘的制作

使用winImage软件创建vfd即可，添加一些文件以及文件夹，子文件夹。



软盘的制作也可以采用本地配置InitMBR和InitFat表进行。本实验未采用代码模拟软盘（配有部分代码模拟软盘）。

```
122  void InitMBR(Disk *disk) {
123      memset( Dst: disk->MBR.BS_jmpBOOT,  Val: 3,  Size: 0);
124      memcpy( Dst: disk->MBR.BS_OEMName,  Src: "LNNU WJY",  Size: 8);
125      disk->MBR.BPB_BytesPerSec = 512;
126      disk->MBR.BPB_SecPerClus = 1;
127      disk->MBR.BPB_ResvdSecCnt = 1;
128      disk->MBR.BPB_NumFATs = 2;
129      disk->MBR.BPB_RootEntCnt = ROOT_DICT_NUM;
130      disk->MBR.BPB_TotSec16 = 2880;
131      disk->MBR.BPB_Media = 0xf0;
132      disk->MBR.BPB_FATSz16 = FAT_SECTOR_NUM;
133      disk->MBR.BPB_SecPerTrk = 0x12;
134      disk->MBR.BPB_NumHeads = 0x2;
135      disk->MBR.BPB_HiddSec = 0;
136      disk->MBR.BPB_TotSec32 = 0;
137      disk->MBR.BS_DrvNum = 0;
138      disk->MBR.BS_Reserved1 = 0;
139      disk->MBR.BS_BootSig = 0x29;
140      disk->MBR.BS_VollD = 0;
141      memcpy( Dst: disk->MBR.BS_VolLab,  Src: "C",  Size: 1);
142      memcpy( Dst: disk->MBR.BS_FileSysType,  Src: "FAT12",  Size: 5);
143      memset( Dst: disk->MBR.code,  Val: 0,  Size: 448);
144      disk->MBR.end_point[0] = 0x55;
145      disk->MBR.end_point[1] = 0xAA;
146  }
147
148
149  //初始化FAT表
150  void InitFAT() {
151      memset( Dst: disk.FAT1,  Val: 0x00,  Size: sizeof(disk.FAT1));
152      memset( Dst: disk.FAT2,  Val: 0x00,  Size: sizeof(disk.FAT2));
```

f InitFAT

3.3 FAT12介绍

在第一章节已详细介绍。

特别地：

笔者使用的软盘，为1簇1扇区，更方便我们定位位置。

3.4 初始化代码介绍

```
#pragma pack(1)
--1. fat表项的创建
typedef struct FATEntry {
    uint8_t data[3]; // 3字节 24位，存储两个簇号
} FATEntry;
```

FATEntry表示FAT表的一个条目，每个条目 使用了3个字节来存储12位的簇号。

```
--2. mbrHeader的创建
typedef struct MBRHeader {
    char BS_jmpBOOT[3];
    char BS_OEMName[8];
    // 每个扇区字节数 512
    uint16_t BPB_BytesPerSec;
    // 每簇扇区数 1
    uint8_t BPB_SecPerClus;
    // boot引导占扇区数 1

    省略...

} MBRHeader;

--3. 根目录的创建
typedef struct RootEntry {
    uint8_t DIR_Name[11];//文件名与扩展名
    uint8_t DIR_Attr;//文件属性
    uint8_t DIR_reserve[10];//保留位
    uint8_t DIR_WrtTime[2];//最后一次写入时间
    uint8_t DIR_WrtDate[2];//最后一次写入日期
    uint16_t DIR_FstClus;//文件开始的簇号
    uint32_t DIR_FileSize;//文件大小
} RootEntry;
```

```
--4. 定义磁盘属性
struct Disk {
    MBRHeader MBR;  //1个扇区
    FATEntry FAT1[1536]// 9个扇区 全部表示出来的 所以
9 * 512 = 4608B , 每个占3字节, 1608 / 3 = 1536
    FATEntry FAT2[1536]; // copy fat1
    RootEntry rootDirectory[ROOT_DICT_NUM];
};
#pragma pack()
//注意一定要取消字节对齐   1表示不对齐
```

3.4.1 获取软盘的MBR代码（删减了一部分）

```
void read_mbr_from_vfd(char *vfd_file) {
    //一个MBR也就是512字节 申请512字节空间
    MBRHeader *mbr = (MBRHeader *)
malloc(sizeof(MBRHeader));
    fseek(diskFile, 0, SEEK_SET);
    uint32_t readed_size = fread(mbr, 1,
SECTOR_SIZE, diskFile);
    disk.MBR = *mbr;
    uint32_t rootDirStartSec = 1 +
(disk.MBR.BPB_NumFATs * disk.MBR.BPB_FATSz16); //
1 MBR, 2 是 FAT 表的扇区数
    fseek(diskFile, rootDirStartSec *
SECTOR_SIZE, SEEK_SET); //移动到根目录区了
    uint16_t read_size =
fread(disk.rootDirectory, sizeof(RootEntry),
disk.MBR.BPB_RootEntCnt, diskFile);
    free(mbr);
}
```

3.4.2获取软盘的FAT表

```
void read_fat_from_vfd(char *vfd_file) {
    fseek(diskFile, 512, SEEK_SET);
    fread(&disk.FAT1, sizeof(FATEntry), 1536,
diskFile);
}
```

3.5 获取FAT表

由于一个FAT表项对应两个簇，我们获取簇号时，只需要根据规则：

偶数：

第二个字节的高地址部分（4位） 拼接 第一个字节的8位地址部分组成簇号。

奇数：

第三个字节的全部部分 拼接 第二个字节的低位地址部分形成簇号。

```
uint8_t  getClus(uint8_t *buf) {
    uint16_t res = 0;
    if (flag % 2 == 0) {  //偶数
        res += buf[0];
        res += (buf[1] << 8) & 0xFFF;
    } else { //奇数
        res += buf[1] << 4;
        res += buf[0] >> 4;
    }
    return ans;
}
```

3.6 dir 功能的实现：

实现逻辑： 我们遍历此时的根目录的所有条目（最大224),如果非空获取他的文件名、拓展名、以及文件大小，通过函数来解析 最后一次修改日期 和修后一次修改时间，同时获取他的类型,文件标注 文件夹标注

实现效果：

```
可使用命令:
(1):dir
(2):cat 文件名
(3):cd 文件夹
(4):pwd
(5):touch
(6):rm 文件名
(0):exit
A>:dir
2024/10/29      15:52      <file>  3927 B        WJY.TXT
2024/10/29      15:52      <folder>               DATA2
2024/10/29      15:53      <folder>               DATA
A>:
```

解析时间的部分在1.5部分叙述，在这里粘贴一部分dir代码：

实现代码：

```
void dir() {
    for (int i = 0; i < disk.MBR.BPB_RootEntCnt; i++) {
        if (disk.rootDirectory[i].DIR_Name[0] == 0) {
            continue; // 空条目，跳过
        }

        //文件名处理  略了一些...
        char name[9] = "";
        char ext[4] = "";
```

```
        memcpy(name,
disk.rootDirectory[i].DIR_Name, 8);
        memcpy(ext,
disk.rootDirectory[i].DIR_Name + 8, 3);

        //解析时间操作     略了一些...
        int hours = (lastWriteTime >> 11) &
0x1F; // 获取小时
        int minutes = (lastWriteTime >> 5) &
0x3F; // 获取分钟 略...
            // 解析最后一次写入时间
        int hours = (lastWriteTime >> 11) &
0x1F; // 获取小时
        int minutes = (lastWriteTime >> 5) &
0x3F; // 获取分钟

        //最后解析
        bool isDirectory =
(disk.rootDirectory[i].DIR_Attr &
DIRECTORY_CODE) != 0;
        if (isDirectory) {
            cout << year << "/" << month <<
"/" << day << setw(7) << hours << ":" <<
minutes << setw(13) << "<folder>"
                 << setw(20) << fileName <<
endl;
        }
```

3.7 cat的实现

实现逻辑：

输入cat xx命令之后，我们通过findFile()进行解析(此函数的作用是分离分离它的文件名和拓展名,因为11位有空格部分，所以需要分割,如果找到了返回RootEntry，否则返回null）。

我们获得了他的目录信息则可以通过getClus（）获取下一个簇号（此函数的作用为获取下一个FAT项的簇号）,如果下一个簇号clusterNum > 0x0ff8,说明有问题（停止或者坏簇）。

这里粘贴一部分读取文件的 重要 内容的代码：

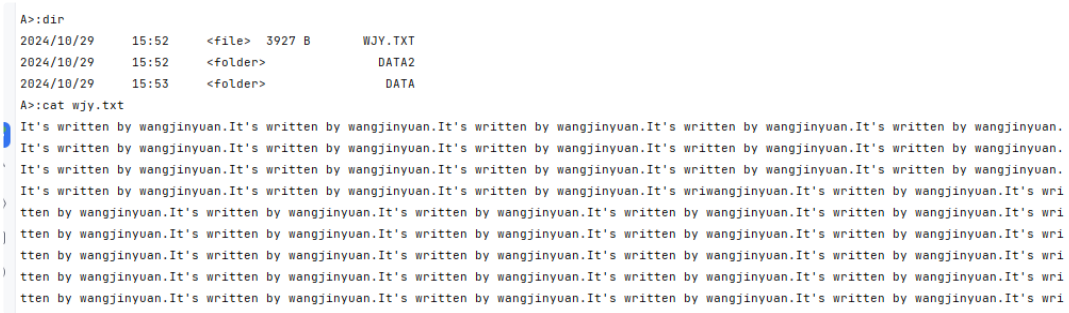
实现代码：

```
 uint16_t currentCluster = firstCluster;
    uint32_t bytesRead = 0;
    while (bytesRead < fileSize) {
        // 计算当前簇对应的起始扇区
        uint32_t clusterSector =
dataStartSector + (currentCluster - 2) *
sectorsPerCluster;
        // 用于存储读取的字节
        uint8_t buffer[SECTOR_SIZE];
        // 逐扇区读取数据
        for (uint8_t sector = 0; sector <
sectorsPerCluster; ++sector) {
            if (bytesRead ≥ fileSize) break;
// 如果已读取完毕，退出循环
            FILE *boot = fopen(PATH, "rb+");
            // 读取当前扇区
            fseek(boot, (clusterSector +
sector) * SECTOR_SIZE, SEEK_SET);
            size_t readSize = fread(buffer, 1,
SECTOR_SIZE, boot);
            fclose(boot);
            // 输出读取的数据
```

```
            for (size_t i = 0; i < readSize &&
bytesRead < fileSize; i++, bytesRead++) {
                putchar(buffer[i]); // 输出字符
到控制台
            }
        }
        unsigned short nextClusNum =
getClus(&disk.FAT1[currentCluster /
2].data[currentCluster % 2],

currentCluster % 2);
        currentCluster = nextClusNum ;//+1
//          cout << "将要读取的簇是" <<
currentCluster <<endl;
        // 如果获取的下一个簇是 ff8 ~ fff，结束读取
        if (currentCluster ≥ 0xFF8) { // 对于
FAT12，0x0FF8 及以上的簇表示 over
            break;
        }
    }
```

效果截图：cat一个 > 一个簇的文件（需要寻簇的情况）



3.8 cd 和 pwd 功能的实现

实现逻辑：

cd命令我使用了栈来维护路径，记录下当前所在的簇号和路径号。

- 如果是 `cd .` 则处于当前目录不做处理
- 如果是 `cd ..`，则表示上一级目录，如果当前处于 `/` 根目录则不做处理，否则会弹出栈到上一级目录的位置，通过更新栈内信息
- 如果是 `cd /` 则直接回到主目录全部出战，只有根目录在栈中。

实现效果：

```
pwd
/
A>:dir
2024/10/29     15:52     <file>  3927 B       WJY.TXT
2024/10/29     15:52     <folder>              DATA2
2024/10/29     15:53     <folder>               DATA
A>:cd data
当前目录：data
A>:pwd
/data
A>:cd ..
已回到根目录
A>:pwd
/
A>:dir
2024/10/29     15:52     <file>  3927 B       WJY.TXT
2024/10/29     15:52     <folder>              DATA2
2024/10/29     15:53     <folder>               DATA
```

实现代码

```
void cd(string &_name) {
    string fileFullName = _name.substr( pos: 3); // 从 "cd " 后开始
    toLowerCase( &: fileFullName);
    if (fileFullName == ".") {...}
    if (fileFullName == "/") {...}
    if (fileFullName == "..") {
        //如果此时栈内元素只有一个元素2代表已经在顶级目录 无法返回
        if (clusterStack.top() == 2 || pathStack.top() == "/") {...}
        //栈空不允许弹出
        if (clusterStack.empty() || pathStack.empty()) {...}
        //栈内有多种个元素的情况：
        /**...*/
        string tempPathName = pathStack.top(); //栈顶路径
        clusterStack.pop();
        pathStack.pop();
        now_clu = clusterStack.top();
        if (now_clu == 2) {...}
        navigator_to_specific_directory_position( cluster_num: now_clu);
        return;
    }
    for (uint16_t i = 0; i < disk.MBR.BPB_RootEntCnt; i++) {
        if (disk.rootDirectory[i].DIR_Name[0] == 0) continue; //跳过空目录项
        //文件名整理以便比较
        char file_name[9];
        //把前8位给file_name
        memcpy( Dst: file_name, Src: disk.rootDirectory[i].DIR_Name, Size: 8);
        //转为字符串方便操作
        string actual_name = file_name;
        actual_name = actual_name.substr( pos: 0, n: actual_name.find_last_not_of( c: ' ') +
```

pwd 较为简单，复制一个当前栈内情况，至今拼接路径即可。

3.9 touch命令

实现逻辑

创建一个目录项。首先检查目录下其是否存在，设置它的名称，属性，时间，保留位等。

现在通过`getFreeClus ()`作为簇号，并调用`usedClus(a)`将该簇号标记为已使用。

此时输入写入的内容，是否大于1个扇区，如果大于1个扇区我们需要分配FAT表项。

- 情况1（size ⩽ 512）：

  根据簇号a计算偏移量 直接跳转到对应位置，然后写入软盘，同时标记文件结束了。

- 情况2 (size > 512) :

计算需要的扇区（向上取整），每次写入512B内容，并且重新写入偏移量。如果剩余大小>512字节，分配新的簇，让当前的簇指向下一个簇，并且标记新簇已经使用。如果<512字节，把最后一个簇标记为0XFFF即可。

最后写入文件系统并且刷新即可。

实现效果

```
A>:touch f1.txt
分配的簇号是: 16
请输入内容:it's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written
by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wj
yit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's
 written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's writ
ten by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written b
y wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjy
it's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's
written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's writt
en by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by
 wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyi
t's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's w
ritten by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's writte
n by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by
wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit
's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's wr
itten by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjyit's written
 by wjyit's written by wjyit's written by wjyit's written by wjyit's written by wjy over
```

```
 by wjyit's written by wjyit's written by wjyit's written by wjyit's written b
需要的扇区数为：4
A>:
```

此时dir可以看到

```
dir
2024/10/29     15:52     <file>   3927 B       WJY.TXT
2024/10/29     15:52     <folder>               DATA2
2024/10/29     15:53     <folder>                DATA
2024/11/2     14:10     <file>   1981 B       f1.txt
A>:
```

实现代码（重要部分，其余属性设置在此省略）

```
if (remain_size <= 512) {
        // 直接写入并结束
        uint32_t offset = (31 + a) * 512;
        fseek(diskFile, offset, SEEK_SET);
```

```
        fwrite(content.c_str(), sizeof(char),
remain_size, diskFile);
        usedClus(a);  // 标记最后一个簇为结束
    } else {
        // 计算需要的扇区数
        uint16_t need_sector_num =
(rootEntry.DIR_FileSize + 511) / 512;
        cout << "需要的扇区数为: " <<
(int)need_sector_num << endl;

        //
        for (uint8_t i = 0; i <
need_sector_num; i++) {
            //写入
            uint32_t offset = (31 + a) * 512;
            fseek(diskFile, offset, SEEK_SET);
            uint32_t bytesToWrite =
(remain_size < 512) ? remain_size : 512;
            fwrite(content + written_size,
sizeof(char), bytesToWrite, diskFile);
            written_size += bytesToWrite;
            remain_size -= bytesToWrite;

            if (remain_size > 0) {
                // 获取空闲簇,并且把下个簇标记为已
使用
                uint16_t next_free_clus =
getFreeClusNum();
                usedClus(next_free_clus);
                setFATEntry(a,next_free_clus);
//成功连接到下一个
                a = next_free_clus;
            } else {
                // 最后一个簇直接标记为结束
```

```
                usedClus(a);
            }
        }
    }
```

3.10 rm 命令

实现逻辑：

首先查找文件，服用findFile（）函数（作用是：寻找文件），如果找到了获得它的起始簇号，

然后清理它的数据区（这步可能非必要，因为用FAT判断此空间为00则覆盖，但考虑到新文件写不到512B的情况我选择留下这个函数），以及FAT链表的链式关系.

在清理FAT链表的关系中

使用while循环获取下一个簇的簇号，并且把它标记为空闲簇。不要忘记第一个簇还没有标记0X000，所以将最初的firstClus标记为空闲。

实现效果：

未删除：

```
000200 F0 FF FF 03 40 00 05 60 00 07 80 00 09 F0 FF FF
000210 FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00
```

```
2024/10/29      15:52      <file>  3927 B       WJY.TXT
2024/10/29      15:52      <dir>               DATA2
2024/10/29      15:53      <dir>                DATA
A>:rm wjy.txt
待删除的文件信息2
清空了2号簇
清空了3号簇
清空了4号簇
清空了5号簇
清空了6号簇
清空了7号簇
清空了8号簇
清空了9号簇
文件删除成功
A>:dir
2024/10/29      15:52      <dir>               DATA2
2024/10/29      15:53      <dir>                DATA
A>:
```

删除后：

000200	F0	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00	FF
000210	FF	FF	FF	FF	FF	FF	FF	FF	00	00	00	00	00	00	00	00

实现代码：

```
void deleteFile(string filename) {
    // 1. 在根目录中查找文件
    string fileFullName = filename.substr(3);
// 从 "rm " 后开始
    RootEntry *pEntry =
findFile(fileFullName);
    cout << "待删除的文件信息" << pEntry-
>DIR_FstClus<<endl;

    // 2. 获取文件的起始簇号
    uint16_t clus_num = pEntry->DIR_FstClus;
    uint16_t firstClus = clus_num;

    //3.循环清除他的数据区
```

```
    clearDataArea(clus_num);

    // 4. 清除FAT链表中的所有簇
    while (clus_num ≠ 0xFFF) {
        uint16_t next_clus =
getClus(&disk.FAT1[clus_num / 2].data[clus_num
% 2],clus_num %2);
        setFATEntry(clus_num, 0);  // 标记簇为
空闲
        clus_num = next_clus;
    }
    setFATEntry(firstClus, 0);  // 标记簇为空闲


//    setFATEntry(firstClus, 0x000);  // 标记簇
为空闲

    //4.1 找到他的根目录把那个32字节删掉
    uint32_t offset = (firstClus *
sizeof(RootEntry)) + (31 * 512); // 根目录从第
31个扇区开始
    fseek(diskFile, offset, SEEK_SET); // 定位
到目录项位置
    memset(pEntry, 0, sizeof(RootEntry)); //
将目录项清空
```

3.11 mkdir 命令

实现逻辑:

创建一个目录项，设置目录属性、保留字节、时间等等信息之后，分配对应的簇号，同时标记此簇已经被使用过了。此时创建.（s当前目录)和..(上一级目录),并且把新目录项写到根目录项中，最后写回磁盘中。

实现效果:

实现代码:

3.12 Format命令

实现逻辑:

首先MBR区域是固定信息不应该被format。

其次初始化FAT区，把前两个簇（0，1）的值设置为初始值 FF0 FFF。然后将1536大小的数组写入FAT1表的位置，然后将FAT1表的内容赋值到FAT2表，实现冗余备份。

然后初始化根目录区域，根目录区域一共7168字节，相当于14个扇区，将其全部置为0，最后初始化数据区 把数据区的每个簇初始化为0。

实现效果:

1. 其中未改动.
2. format前FAT

forma*后FAT:

3. format前根目录区:

format 后目录区：***

实现代码：

4.实验总结：

通过本次实验，我了解了FAT12的实现结构和实现细节。掌握了使用c和c++进行文件操作的办法，如FSEEK（）以及FWRITE()等等函数的使用。了解了目录项的管理，簇的分配和释放，以及如何维护FAT链表，这些知识都有助于我了解文件创建、删除、管理等操作。

在实验中我也遇到了一些挑战，比如如何正确表示簇的状态，如何管理簇，扇区。在这个过程中我提升了自己的编程思维和逻辑能力。虽然FAT12是较为古老的文件系统，但实现过程中仍然让我对操作系统知识有了重要理解。