# 大程：磁盘操作

**应承峻 3170103456**

## 1 实验描述

编写程序模拟磁盘工作。以大文件模拟磁盘，建立`FAT`、文件目录等数据结构，进行列目录、文件增加删除等操作，实现文件操作命令：`dir`、`copy`、`del`。

可考虑子目录操作：`md`(建子目录)、`cd`(转子目录)

读写`VHD`虚拟磁盘只能整块(簇)读写，每次能且只能读写一个块(簇)。所以基础的`VHD`读写可编写以下函数：

```
int clusterRead(byte&, int); //int为簇号
int clusterWrite(byte*, int);
```

重点一：熟悉对于二进制文件的随机读写。

重点二：熟悉掌握`FAT`文件系统的基本结构。

## 2 算法设计

### 2.1 二进制文件随机读写

簇读写主要执行的是文件操作，通过传入的缓冲区`buffer`和簇编号`cid`，从磁盘中读入相应的字节到缓冲区中或是从缓冲区写入到磁盘中。在`FAT32`文件系统中，一个簇由32个扇区组成，而一个扇区则由512个字节组成，因此一个簇相当于$512 \times 32 = 16384$个字节。因此根据簇编号在文件中的定位则可通过$16384 \times cid$得到。

```
bool Dir::clusterRead(char* buf , int cid) {
    fstream file;
    file.open(filename , ios::binary | ios::in);
    if (!file) return false;
    file.seekg(cid * sectorSize * clusterSize); //offset
    file.read(buf , sectorSize * clusterSize);  //1 cluster with 512*32 bytes
    file.close();
    return true;
}

bool Dir::clusterWrite(char* buf , int cid) {
    fstream file;
    file.open(filename , ios::binary | ios::out | ios::in);
    if (!file) return false;
    file.seekp(cid * sectorSize * clusterSize); //offset
    file.write(buf , sectorSize * clusterSize); //1 cluster with 512*32 bytes
    file.close();
    isfree.at(cid) = false;  //set unabled
    return true;
```

```
20 }
```

## 2.2 文件系统结构的建立

为了方便测试，整个文件系统总共分为256个簇，每个簇16$KB$，因此总容量为4$MB$，由于第1簇用于配置，且此后的每一簇中都有4个字节用于存放指向下一簇的数据。故实际可存储：$4096KB-16KB-1KB=4079KB$的数据。

在第一簇中，簇中的32个扇区被用于如下安排：

> Sector 0~1 BOOT [0,1)：第1字节用于存放文件数量，后面每4字节存放一个簇的占用情况 Sector 2~10 FAT1 [1,10)：保留 Sector 11~19 FAT2 [11,19)：保留 Sector 20~31 DIR [20,31)：存放文件目录信息

每当系统启动时，需要从磁盘中加载数据。具体的实现方法是先将第一簇从磁盘读入到缓冲区中，然后从缓冲区中取出第一字节用于计算虚拟磁盘中文件的数量。然后根据簇的数量，依次按照每一簇读入4个字节的整数，如果读入的是0则簇没有被占用，读入1则簇被占用，实现代码如下：

```
void Dir::initconfig(char* buffer) {
    int num = readInt(buffer , 0);
    this->dirNum = num >= 0 && num <= 255 ? num : 0;
    for (int i = 1; i < clusterNum; i++) {
        if (readInt(buffer , 4 * i) == 0) isfree.push_back(true);
        else isfree.push_back(false);
    }
    cout << "direction number : " << dirNum << endl;
    for (int i = 0; i < dirNum; i++) {
        int ofs = 20 * 512 + i * 32;
        string str = "";
        for (int j = 0; j < 32; j++) str += buffer[ofs + j];
        dat.push_back(str);
    }
}
```

同理，当系统关闭时，需要将数据写回磁盘。写回磁盘的方法类似于前者，只需要将目录数量以及每一簇的占用情况写回磁盘即可。实现代码如下：

```
void Dir::storeconfig(char* buffer) {
    writeInt(buffer , 0 , this->dirNum);
    for (int i = 1; i < clusterNum; i++) {
        if (isfree.at(i) == true) writeInt(buffer , 4 * i , 0);
        else writeInt(buffer , 4 * i , 1);
    }
    for (int i = 0; i < dat.size(); i++) {
        int ofs = 20 * 512 + i * 32;
        writeData(buffer , ofs , dat.at(i));
    }
}
```

每次创建一个文件时，文件系统会为文件分配一个簇的存储空间，簇的前4个字节用于指向该文件的下一个簇，如果该文件的大小在一个簇内则该4字节整数存储0，否则存储对应的簇编号。分配簇空间的算法如下：

```
int Dir::getfreecluster() {
    for (int i = 1; i < clusterNum; i++) {
        if (isfree.at(i)) return i;
    }
    return 0;  //not exist a free cluster
}
```

每个文件在目录项里以32个字节存放：

> 0~7 filename 8 8~10 extended 3 11 attribute 1 12~21 reserved 10 22~23 time 2 [99]-[12]-[31] 7bits-4bits-5bits 24~25 date 2 [23]-[59] 1byte-1byte 26~27 first cluster 2 28~31 size 4

## 2.3 文件系统命令

本文件系统主要支持`md`（新建文件）、`copy`（从外部磁盘拷如文件）、`del`（删除文件）、`dir`（列目录）以及`cat`（查看文件信息）这几种命令。

### 2.3.1 md

通过命令传入文件名，首簇编号以及文件大小，我们可以建立一个新的文件。首先需要分割文件名以得到其前缀和后缀（扩展名）。然后将空余的字节用特殊符号补上。再通过自定义函数获取时间、首簇编号以及大小的编码结果，将其组合成32字节字符串存入即可，整体算法如下：

```
bool Dir::makedir(string dirname, int firstcluseter = 1, int size = 0) {
    int len = dirname.length();
    string dirstr;
    string filename , extended , reversed;
    for (int i = 0; i < len && i < 32; i++) {
        if (dirname[i] == '.') {
            for (int j = i + 1; j < len && j < 32; j++)
                extended = extended + dirname[j];
            break;
        } else {
            filename = filename + dirname[i];
        }
    }
    filename = filename.substr(0 , 8);
    extended = extended.substr(0 , 3);
    while (filename.length() < 8) filename += "&";
    while (extended.length() < 3) extended += "&";
    for (int i = 11; i <= 21; i++) reversed += "&";
    dirstr = filename + extended + reversed + get_byte_time() +
int_to_2byte_str(firstcluseter) + int_to_4byte_str(size);
    dat.push_back(dirstr);
    char filebuffer[sectorSize * clusterSize] = { 0 }; //copy file
    writeInt(filebuffer , 0 , 0);  //next pointer
    //write cluster back to disk and also make clusterid locked
    clusterWrite(filebuffer , firstcluseter);
    dirNum++;
    isfree.at(firstcluseter) = false;
    return true;
}
```

获取当前时间并进行编码算法如下：日期和时间各占用两个字节，其中日期的最大值为$[99]-[12]-[31]$，分别占据$[7bits]-[4bits]-[5bits]$，时间（精确到分）的最大值为$[23]-[59]$，各用一个字节表示。因此可以通过如下位运算来得到相应的编码字符：

```
string Dir::get_byte_time() {
    char ret[5];
    time_t times;
    struct tm *pt;
    time(&times);
    pt = localtime(&times);
    int year = pt->tm_year - 100;
    int month = pt->tm_mon + 1;
    int day = pt->tm_mday;
    int hour = pt->tm_hour;
    int min = pt->tm_min;
    unsigned char x = 0 , y = 0 , u = 0 , v = 0;
    x = (unsigned char)year << 1;
    x = x | (unsigned char)month & 8; //month & 1000
    y = ((unsigned char)month & 7) << 5; //month & 111
    y = y | ((char)day & 31);
    u = (unsigned char)hour;
    v = (unsigned char)min;
    sprintf(ret , "%c%c%c%c" , x , y , u , v);
    string s = ret;
    return s;
}
```

将对应的编码转换回字符串时间的算法如下：

```
string Dir::get_format_time(string s) {
    char ret[24];
    int year , month , day , hour , min;
    unsigned char x = s.at(0) , y = s.at(1) , u = s.at(2) , v = s.at(3);
    year = (x & 254) >> 1; //x & 11111110
    month = ((x & 1) << 3) | ((y & 224) >> 5);  //y & 11100000
    day = y & 31;
    hour = (int)u;
    min = (int)v;
    sprintf(ret , "%02d-%02d-%02d %02d:%02d" , year , month , day , hour , min);
    return ret;
}
```

### 2.3.2 copy

在拷贝前，需要先检测外部文件的大小，如果文件超过一个簇（这里指$32\times 512-4=16380Bytes$）的大小，则需要分簇存放，分簇存放主要通过迭代法完成，每一次迭代从源文件中读入一个簇到缓冲区中，然后将其转储到文件系统中对应的簇，同时生成下一个簇的簇号，迭代簇号直至不存在空闲簇或文件读完为止。迭代结束后，将剩余内容读进簇即可。

```
bool Dir::copy(string dest , string src) {
    int clusterid = getfreecluster();
```

```
    if (!clusterid) {
        throw new exception("can not find free cluster!");
        return false;
    }
    ifstream file(src , ios::in | ios::binary);  //open source file
    file.seekg(0 , ios::end);
    streampos pos = file.tellg();  //calculate file size
    file.clear(); //move file pointer to the front
    file.seekg(0 , ios::beg);

    //create file directory and also make clusterid locked
    makedir(dest.c_str() , clusterid , (int)pos);

    char filebuffer[sectorSize * clusterSize] = { 0 }; //copy file

    //16384bytes for one cluster, except for a integer pointer, remains 16380 bytes
    while ((int)pos > sectorSize * clusterSize - 4) {
        int nextclusterid = getfreecluster();  //get next free id
        if (!nextclusterid) {
            throw new exception("can not find free cluster!");
            return false;
        }
        writeInt(filebuffer , 0 , nextclusterid);  //next pointer
        //read nearly 1 cluster from file
        file.read(filebuffer + 4 , sectorSize * clusterSize - 4);
        //write cluster back to disk and also make clusterid locked
        clusterWrite(filebuffer , clusterid);

        pos = (int)pos - sectorSize * clusterSize + 4;
        clusterid = nextclusterid;
    }

    //next pointer point to zero cluster mean end of file
    writeInt(filebuffer , 0 , 0);
    file.read(filebuffer + 4 , pos);  //read all remain files
    //write cluster back to disk and also make clusterid locked
    clusterWrite(filebuffer , clusterid);

    return true;
}
```

#### 2.3.3 del

删除时，通过获取的文件名，在文件列表中使用迭代器依次检索。当检索到相应的条目时，根据其簇号依次进行迭代，将其内容清空并写回虚拟磁盘，然后释放在该簇上的锁。最后将文件数量减一即可。

```
bool Dir::rmdir(string dirname) {
    char filebuffer[sectorSize * clusterSize];
    vector<string>::iterator it;
    for (it = dat.begin(); it != dat.end();) {
        string filename;
        string extended = replace((*it).substr(8 , 3));
        if (extended == "") filename = replace((*it).substr(0 , 8));
```

```
        else filename = replace((*it).substr(0 , 8)) + "." + extended;
        if (filename == dirname) {
            int clusterid = getclusternum((*it));
            while (clusterid) {
                clusterRead(filebuffer , clusterid);
                int nextcluster = readInt(filebuffer , 0);
                memset(filebuffer , 0 , sizeof(filebuffer));
                clusterWrite(filebuffer , clusterid);
                isfree.at(clusterid) = true;
                clusterid = nextcluster;
            }
            it = dat.erase(it);
            dirNum--;
        } else it++;
    }
    return true;
}
```

#### 2.3.4 dir

列目录主要是通过遍历`vector`中的每一条记录，并将32字节字符串解析成信息即可。

```
bool Dir::showdir() {
    try {
        char output[30];
        string s , filename , extended , datetime , size;
        printf("+%-10s-%-14s+%-13s+\n" , "----------" , "--------------", "-------------");
        printf("|%-10s|%-14s|%-13s|\n" , "filename" , "created time", "file size (b)");
        for (int i = 0; i < dirNum; i++) {
            s = dat.at(i);
            int size = getsize(s);
            extended = replace(s.substr(8 , 3));
            if (extended == "") filename = replace(s.substr(0 , 8));
            else filename = replace(s.substr(0 , 8)) + "." + extended;
            datetime = get_format_time(s.substr(22 , 4));
            printf("+%-10s-%-14s+%-13s+\n" , "----------" , "--------------" , "-------------");
            printf("|%-10s|%s|%-13d|\n" , filename.c_str() , datetime.c_str(), size );
        }
        printf("+%-10s-%-14s+%-13s+\n" , "----------" , "--------------" , "-------------");
        return true;
    } catch (exception e) {
        cout << e.what() << endl;
        return false;
    }
}
```

#### 2.3.5 cat

显示文件的算法类似于删除算法，通过迭代器找到首簇后，不断迭代将文件打印出来即可。

```
bool Dir::cat(string dirname) {
    char filebuffer[sectorSize * clusterSize] = { 0 };
    vector<string>::iterator it;
    for (it = dat.begin(); it != dat.end();) {
        string filename;
        string extended = replace((*it).substr(8 , 3));
        if (extended == "") filename = replace((*it).substr(0 , 8));
        else filename = replace((*it).substr(0 , 8)) + "." + extended;
        if (filename == dirname) {
            int clusterid = getclusternum(*it);
            while (clusterid) {
                clusterRead(filebuffer , clusterid);
                int nextcluster = readInt(filebuffer , 0);
                puts(filebuffer + 4);
                clusterid = nextcluster;
            }
            break;
        } else it++;
    }
    return true;
}
```

## 3 实验测试

**启动程序可以看到欢迎界面：**

```
direction number : 0
+-------------------------------------------------------+
|           Welcome to simulate disk!                   |
+-------------------------------------------------------+
|[Command] md [filename]: create a new file             |
+-------------------------------------------------------+
|[Command] dir: show all files                          |
+-------------------------------------------------------+
|[Command] copy [src] [dest]: copy a file               |
+-------------------------------------------------------+
|[Command] del [filename]: delete file from disk        |
+-------------------------------------------------------+
|[Command] cat [filename]: show file content            |
+-------------------------------------------------------+
--> 
```

**通过`md`命令分别创建三个不同的文件，再通过列目录命令`dir`查看目录：**

```
--> md file_1
--> md file_2
--> md file_3
--> dir
+------------------------+-------------+
|filename  |created time  |file size (b)|
+------------------------+-------------+
|file_1    |19-06-13 17:02|0            |
+------------------------+-------------+
|file_2    |19-06-13 17:02|0            |
+------------------------+-------------+
|file_3    |19-06-13 17:02|0            |
+------------------------+-------------+
```

**从本地磁盘中拷入`abc.txt`文件，在新磁盘中命名为`hello`，然后再通过列目录命令`dir`查看目录，同时通过`cat`命令查看拷贝的文件内容：**

```
--> copy abc.txt hello
--> dir
+------------------------+-------------+
|filename  |created time  |file size (b)|
+------------------------+-------------+
|file_1    |19-06-13 17:02|0            |
+------------------------+-------------+
|file_2    |19-06-13 17:02|0            |
+------------------------+-------------+
|file_3    |19-06-13 17:02|0            |
+------------------------+-------------+
|hello     |19-06-13 17:03|20           |
+------------------------+-------------+
--> cat hello
hello simulate disk!
```

**再新建一个带后缀的文件`temp.txt`，通过`del`目录删除一部分文件，然后重新列目录查看：**

```
--> md temp.txt
--> del file_1
--> del hellp
--> del hello
--> dir
+------------------------+-------------+
|filename  |created time  |file size (b)|
+------------------------+-------------+
|file_2    |19-06-13 17:02|0            |
+------------------------+-------------+
|file_3    |19-06-13 17:02|0            |
+------------------------+-------------+
|temp.txt  |19-06-13 17:03|0            |
+------------------------+-------------+
```

**删除带后缀的文件`temp.txt`，查看结果：**

```
--> del temp.txt
--> dir
+-------------------------+-------------+
|filename  |created time  |file size (b)|
+-------------------------+-------------+
|file_2    |19-06-13 17:02|0            |
+-------------------------+-------------+
|file_3    |19-06-13 17:02|0            |
+-------------------------+-------------+
-->
```

**通过`exit`或`quit`命令退出，测试结束，能够基本实现文件管理功能。**