编号：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总评 | 教师签名 |
| 成绩 |  |  |  |  |  |  |  |  |  |  |

武汉大学国家网络安全学院

课程实验(设计)报告

题 目： FAT32文件系统数据安全删除

专业(班)： 信息安全

学 号： 2021302181059

姓 名： 黄静

课程名称： 软件安全

任课教师： 赵磊

2023年 12 月

一、实验名称：FAT32文件系统数据安全删除

二、实验内容：

FAT32文件系统数据安全删除

要求：

1）作业的重点是分析FAT32文件系统的数据结构；

2）对于给定的文件，要求能够通过分析FAT表识别出来文件所占的簇链；

3）对簇的内容重写，以达到安全删除的目的。

三、实验环境：

* 操作系统：Windows10
* 实验软件：Visual Studio Code
* 磁盘设备：FAT32格式的U盘

四、实验原理：

**FAT32文件系统**

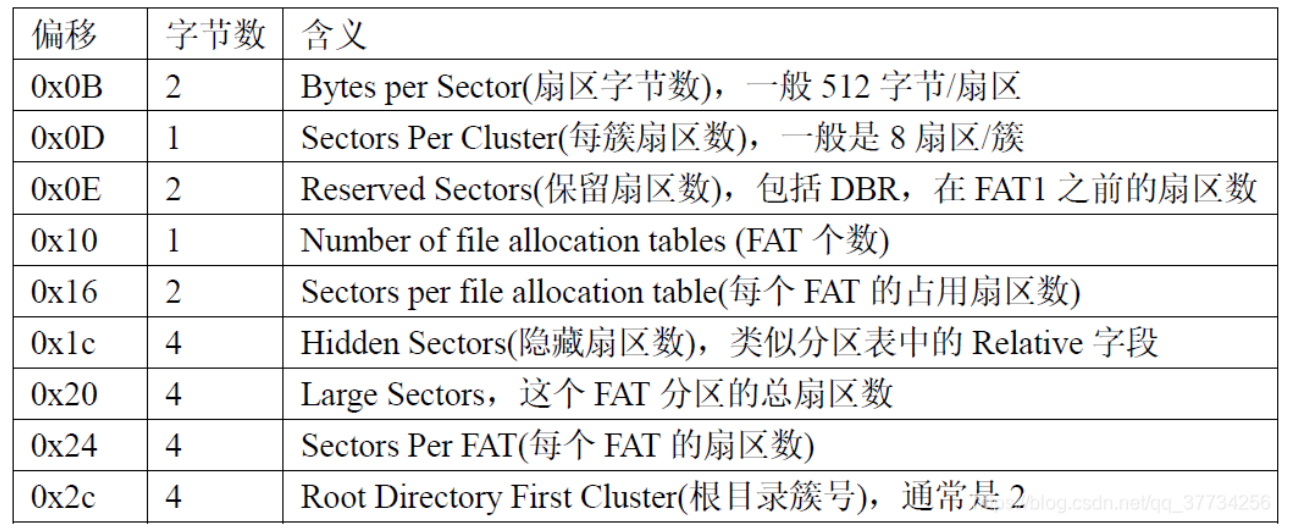
FAT32文件系统由4部分构成：DRB和保留扇区，FAT1，FAT2和数据区。其中数据区中有根目录、子目录和数据三部分。



**DBR的结构：**



开始的3字节是一个跳转指令，指出引导代码的开始位置，EB 58就是跳转到0x58+2(相对于当前位置，当前位置是2)。之后的BPB结构存储了和该分区有关的重要信息。BPB中重要的字段如下：



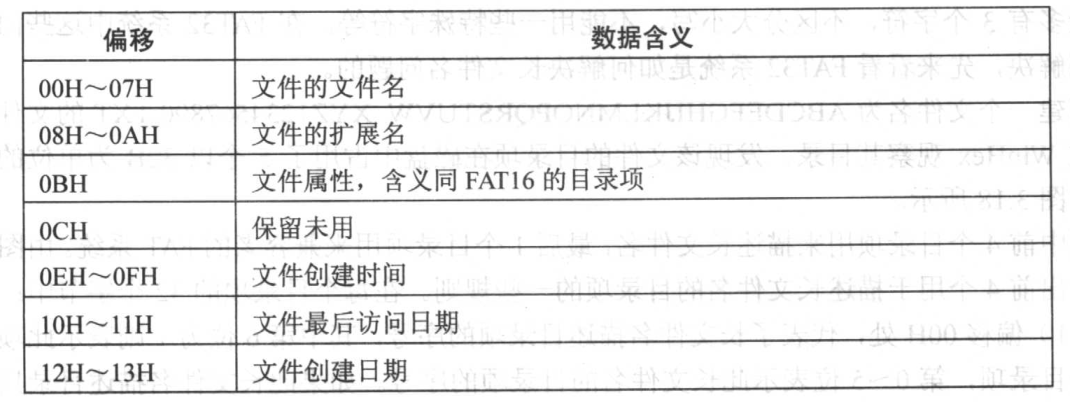
打开磁盘首先进入的是DBR，此时FILE\_BEGIN指针为零，是指对DBR的开始位置而言偏移0，保留扇区数的作用是可以通过它，获得FAT 1相对于FILE\_BEGIN的偏移。FAT的个数一般是2个，在知道每个FAT占用的扇区数之后，可以通过该字段，获得根目录相对FILE\_BEGIN的偏移。

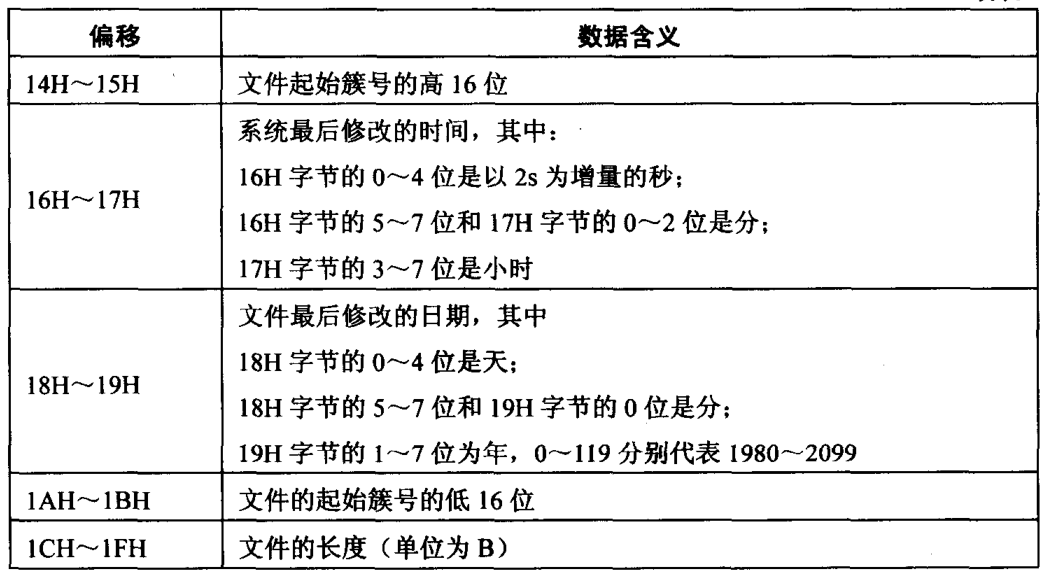
**根目录的相对开始扇区号=保留扇区数+FAT个数\*每个FAT的扇区数。**

**目录项的结构：**

获得根目录相对扇区号之后，打开到根目录，根目录里面存放着根目录下文件和子目录的目录项，每个项占32字节。

目录项数据结构：





**FAT表项：**

FAT像一个巨大的链表，每个表项4字节。在目录中找到一个文件的开始簇号，来查FAT表，找到这个簇号对应的FAT表项。如果该表项是0x0FFFFFFF，表示是该文件的最后一簇，否则该表项指向的是该文件占用的下一个簇号。

**数据区：**

数据区存储着文件的具体内容。

数据区偏移 = (保留扇区数 + FAT表扇区数 \* FAT表个数(通常为2) + (起始簇号-2) \* 每簇扇区数) \* 每扇区字节数

五、实现思路及步骤

**实现思路：**

首先找到文件的目录项，找到起始簇号，然后找到FAT链，得到文件的簇，对簇的内容重写。

**实验步骤：**

**1、从输入中得到文件名**

将输入的路径保存在filepath字符串中，调用getpath函数解析文件路径，在该函数中再调用getfilename得到文件名，将文件名的前缀和后缀分别保存在filename结构体的prename和rearname中。对于不足8字节的前缀和不足3字节的后缀，用数据0x20填充。

        t = **getfilename**(filepath, nextname, filenameindex); *// 得到文件名*

        if (t == 0) *// t==0，是文件名*

        {

**printf**("file is %s  length is %d\n", nextname, **strlen**(nextname));

            p = (struct **filename** \*)**malloc**(sizeof(struct **filename**));

**memset**(p->prename, 0, **strlen**(p->prename));

**memset**(p->rearname, 0, **strlen**(p->rearname));

**dividefilename**(nextname, prename, rearname); *// 分成前缀和后缀*

**strncpy**(p->prename, prename, **strlen**(prename));

            paddingnum = 8 - **strlen**(p->prename); *// 0x20填充*

            if (paddingnum > 0)

**memset**(&p->prename[**strlen**(p->prename)], 0x20, paddingnum);

**strncpy**(p->rearname, rearname, **strlen**(rearname));

            paddingnum = 3 - **strlen**(p->rearname);

            if (paddingnum > 0)

**memset**(&p->rearname[**strlen**(p->rearname)], 0x20, paddingnum);

            FILENAME = p;

        }

**2、寻找文件的目录项FDT**

使用findFDT函数寻找文件的目录项。

首先通过CreateFile函数打开对应的磁盘，然后读取DBR

**DISK\_GEOMETRY** \*pdg; *// 保存磁盘参数的结构体*

**HANDLE** hDevice; *// 设备句柄*

**DWORD** junk; *// discard resultscc*

hDevice = **CreateFile**(**TEXT**(partition), *// 磁盘*

**GENERIC\_READ** | **GENERIC\_WRITE**, *// 读或写*

**FILE\_SHARE\_READ** | **FILE\_SHARE\_WRITE**, *// share mode*

**NULL**, *// default security attributes*

**OPEN\_EXISTING**, *// disposition*

                         0, *// file attributes*

**NULL**);

    if (hDevice == **INVALID\_HANDLE\_VALUE**) *// cannot open the drive*

    {

**printf**("May be no permission!Or no such partition!\n");

        return;

    } else

    {

**printf**("Successfully open the drive\n");

    }

**LARGE\_INTEGER** offset; *// 读取位置*

    offset.QuadPart = (**ULONGLONG**)0; *// 0*

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

**DWORD** dwCB;

    struct **DBR** the\_DBR;

*// 从这个位置开始读DBR，一开始的512字节有些信息有用*

**BOOL** bRet = **ReadFile**(hDevice, &the\_DBR, 512, &dwCB, **NULL**);

从the\_DBR中获取磁盘的FAT1表的相对偏移地址FAT1\_reladdr：

**ULONGLONG** FAT1\_reladdr = (**ULONGLONG**)**uint8to16**(the\_DBR.reserve\_sectors) \*

                             (**ULONGLONG**)512; *// 得到FAT1的具体地址*

从the\_DBR中获取磁盘的根目录相对偏移地址root\_reladdr：

**ULONGLONG** root\_reladdr = FAT1\_reladdr + (**ULONGLONG**)(the\_DBR.FATnum) \*

     (**ULONGLONG**)**uint8to32**(the\_DBR.sectors\_per\_FAT) \* (**ULONGLONG**)512; *// 根目录的起始相对位置*

将文件指针定位到根目录开始处，读取根目录的第一个扇区到lpBuffer中。

定义the\_short\_FDT用于保存找到的文件的目录项。

    offset.QuadPart = root\_reladdr; *// 根目录区*

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

**memset**(lpBuffer, 0, sizeof(lpBuffer));

    bRet = **ReadFile**(hDevice, lpBuffer, 512, &dwCB, **NULL**); *//读一个扇区到lpBuffer*

*// show\_onesector(lpBuffer);   //correct*

    char prename[101] = {0};

    char rearname[10] = {0};

int needreadmore = 0; *// 需要继续读扇区，还没有找到文件*

    struct **FDT** tempFDT[16] = {0};

    struct **shortFDT** the\_short\_FDT = {0};

    int index = 0;

    int findflag = 0;

    p = FILENAME;

int i = 0;

**memset**(tempFDT, 0, sizeof(tempFDT));

**memcpy**(tempFDT, lpBuffer, 512); *// 根目录前16个目录项*

遍历根目录的扇区，在每一个扇区中，遍历16个目录项，将目录项中的前缀和后缀名和指定文件的前缀名和后缀名比对，如果相同，说明找到了指定文件的目录项。

如果在一个扇区遍历完了16个目录项还没有找到，就读取下一个扇区，继续寻找。

    findflag = 0;

    index = 0;

    while (findflag == 0)

    {

*// printf("tempFDT[%d]:%s\n",index,tempFDT[index].content);*

        if (tempFDT[index].content[0] == 0x00)

        {

**printf**("\npath error!\n");

            return;

        }

        if (tempFDT[index].content[11] != 0x0F)

        {

**memset**(prename, 0, sizeof(prename));

**memset**(rearname, 0, sizeof(rearname));

**strncpy**(prename, &tempFDT[index].content[0], 8);

**strncpy**(rearname, &tempFDT[index].content[8], 3);

*// 比对文件前缀和后缀名*

            if (!**strncmp**(prename, p->prename, 8) && !**strncmp**(rearname, p->rearname, 3))

            {

                findflag = 1;

                Findindex = index; *// 记录索引号，用于后续将FDT首字节置E5*

**memcpy**(&the\_short\_FDT, &tempFDT[index], 32); *// 记录文件目录项到the\_short\_FDT中*

                FDToffset.QuadPart = offset.QuadPart; *// 记录扇区偏移，用于后续将FDT首字节置E5*

**findFDT**(the\_short\_FDT, FAT1\_reladdr, hDevice);

            }

        }

        if (index < 15 && findflag == 0) *// 没找到*

            index++;

        else if (index == 15 && findflag == 0) *// 读下一个扇区*

        {

            index = 0;

**memset**(tempFDT, 0, sizeof(tempFDT));

**memset**(lpBuffer, 0, sizeof(lpBuffer));

            offset.QuadPart = offset.QuadPart + (**ULONGLONG**)512;

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

            bRet = **ReadFile**(hDevice, lpBuffer, 512, &dwCB, **NULL**);

**memcpy**(tempFDT, lpBuffer, 512);

        }

    }

如果寻找到了文件目录项，用the\_short\_FDT保存目录项，然后调用findFAT寻找文件的簇链。

**3、寻找文件的簇链**

使用findFAT函数寻找文件的簇链

在findFAT函数中，根据找到的文件目录项，可以得到文件的起始簇号。

**uint8\_t** cluster[4] = {the\_short\_FDT.low\_cluster[0],the\_short\_FDT.low\_cluster[1],

 the\_short\_FDT.high\_cluster[0],the\_short\_FDT.high\_cluster[1]}; *//记录起始簇号*

**uint32\_t** firstcluster=**uint8to32**(cluster); *// 起始簇号*

AllClustor[count++] = firstcluster; *// 记录簇链，用于后面重写簇的内容*

**printf**("The starting cluster number of the file is %08X\n", firstcluster);

将起始簇号除以128可以得到扇区号，然后计算出起始簇号在扇区中的偏移

    int next\_sector = firstcluster / 128; *// 在第几个扇区*

    int next\_byte = (firstcluster % 128) \* 4; *//偏移*

定义FATentry用于保存簇号，Buffer用于保存读出的扇区。

将文件指针定位到起始簇号所在的扇区，读取扇区到Buffer中。

**uint32\_t** FATentry = 0; *//保存簇号*

**uint8\_t** Buffer[512] = {0}; *//保存扇区*

    int needreadmore = 1; *//需要读下一个扇区*

**BOOL** bRet;

**DWORD** dwCB;

**LARGE\_INTEGER** offset;

offset.QuadPart = FAT1\_reladdr + (**ULONGLONG**)(512 \* next\_sector); *// 定位到扇区*

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

    bRet = **ReadFile**(hDevice, Buffer, 512, &dwCB, **NULL**); *// 读出扇区到Buffer*

然后根据起始簇号在扇区中的偏移找到FAT项，得到FAT项中的下一个簇号。

        FATentry = \*(**uint32\_t** \*)&Buffer[next\_byte]; *// 下一个簇号*

        AllClustor[count++] = FATentry; *// 记录簇链，用于后面重写簇的内容*

**printf**("%08X\n", FATentry);

        Buffer[next\_byte + 0] = 0x00;

        Buffer[next\_byte + 1] = 0x00;

        Buffer[next\_byte + 2] = 0x00;

        Buffer[next\_byte + 3] = 0x00;

        offset.QuadPart = FAT1\_reladdr + (**ULONGLONG**)(512 \* next\_sector);

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

        bRet = **WriteFile**(hDevice, Buffer, 512, &dwCB, **NULL**); *// 读出扇区到Buffer*

        if (bRet)

**printf**("delete FAT %d successfully!\n", count - 1);

如果FAT项的内容不为0x0FFFFFFF，说明簇链还没有结束，需要继续找下一个簇号，如果下一个簇号位于下一个扇区，需要读取下一个扇区。

        if (FATentry != 0x0FFFFFFF) *// 尚未结束*

        {

            if (next\_sector != FATentry / 128)

            {

                next\_sector = FATentry / 128;

                offset.QuadPart = FAT1\_reladdr + (**ULONGLONG**)(512 \* next\_sector);

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

**memset**(Buffer, 0, 512);

                bRet = **ReadFile**(hDevice, Buffer, 512, &dwCB, **NULL**);

            }

            next\_byte = (FATentry % 128) \* 4;

        }

        else

            needreadmore = 0;

如果读到内容为0x0FFFFFFF的FAT项，说明簇链结束了，已经得到了所有簇号。

**4、将文件目录项首字节置为“E5”**

首先根据之前保存的文件目录项的扇区偏移，读出目录项所在的扇区，

然后根据目录项在扇区中的索引Findindex，将目录项首字节置为0xE5，表示该文件被删除。然后再将扇区内容写入磁盘。

*// 删除FDT->E5=======================================================*

offset.QuadPart = FDToffset.QuadPart;

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

**uint8\_t** tmpSector[512] = {0};

**memset**(tmpSector, 0, sizeof(tmpSector));

    bRet = **ReadFile**(hDevice, tmpSector, 512, &dwCB, **NULL**);

*// show\_onesector(tmpSector);*

    if (bRet)

**printf**("read FDT successfully!\n");

*// printf("Findindex:%d\n", Findindex);*

tmpSector[Findindex \* 32] = 0xE5;  *// 首字节置E5*

    offset.QuadPart = FDToffset.QuadPart;

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

    bRet = **WriteFile**(hDevice, tmpSector, 512, &dwCB, **NULL**);

*// show\_onesector(tmpSector);*

    if (bRet)

**printf**("FDT set E5 successfully!\n");

**5、用0重写文件数据区簇的内容**

根据之前保存的簇链的数组AllClustor[ ]得到文件的所有簇号，然后定位到每个簇所在的起始位置，将该簇的128个扇区全部重写为0，达到安全删除的目的。

**printf**("Clustor count:%d\n", count - 1);

    for (int i = 0; i < count - 1; i++)

    {

**uint8\_t** buffer[512 \* 128] = {0};

**memset**(buffer, 0, sizeof(buffer)); *//簇的内容置0*

        offset.QuadPart = root\_reladdr + (**ULONGLONG**)(AllClustor[i] - 2) \* 512 \* 128; *// 定位到簇内容*

**SetFilePointer**(hDevice, offset.LowPart, &offset.HighPart, **FILE\_BEGIN**);

        bRet = **WriteFile**(hDevice, buffer, 512 \* 128, &dwCB, **NULL**); *// 重写簇的128个扇区*

        if (!bRet)

        {

**DWORD** dwErr = **GetLastError**();

**printf**("Errorcode:%d\n", dwErr);

**printf**("Write failed!\n");

**CloseHandle**(hDevice);

            return 0;

        }

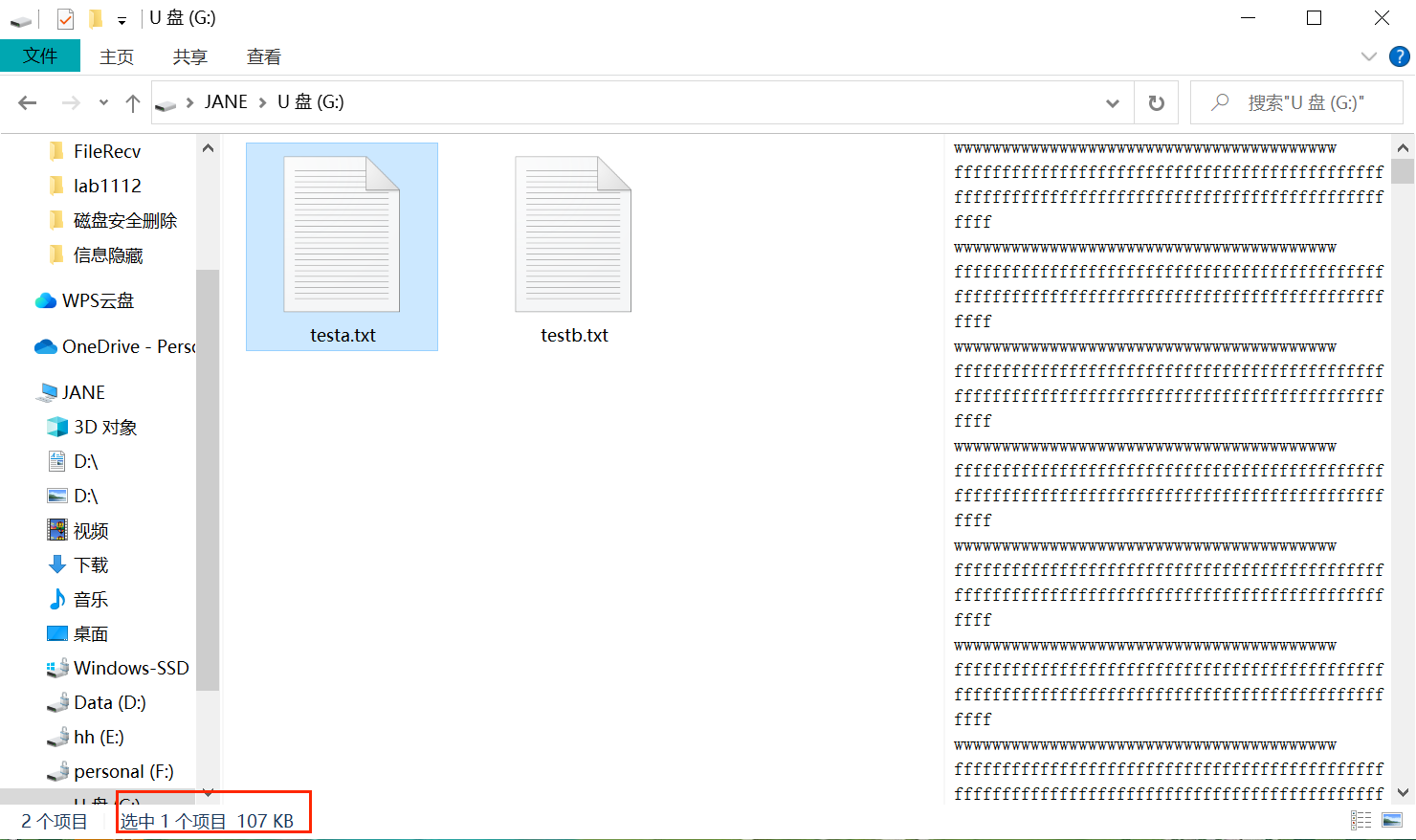
    }

**printf**("Safe delete data successfully\n");

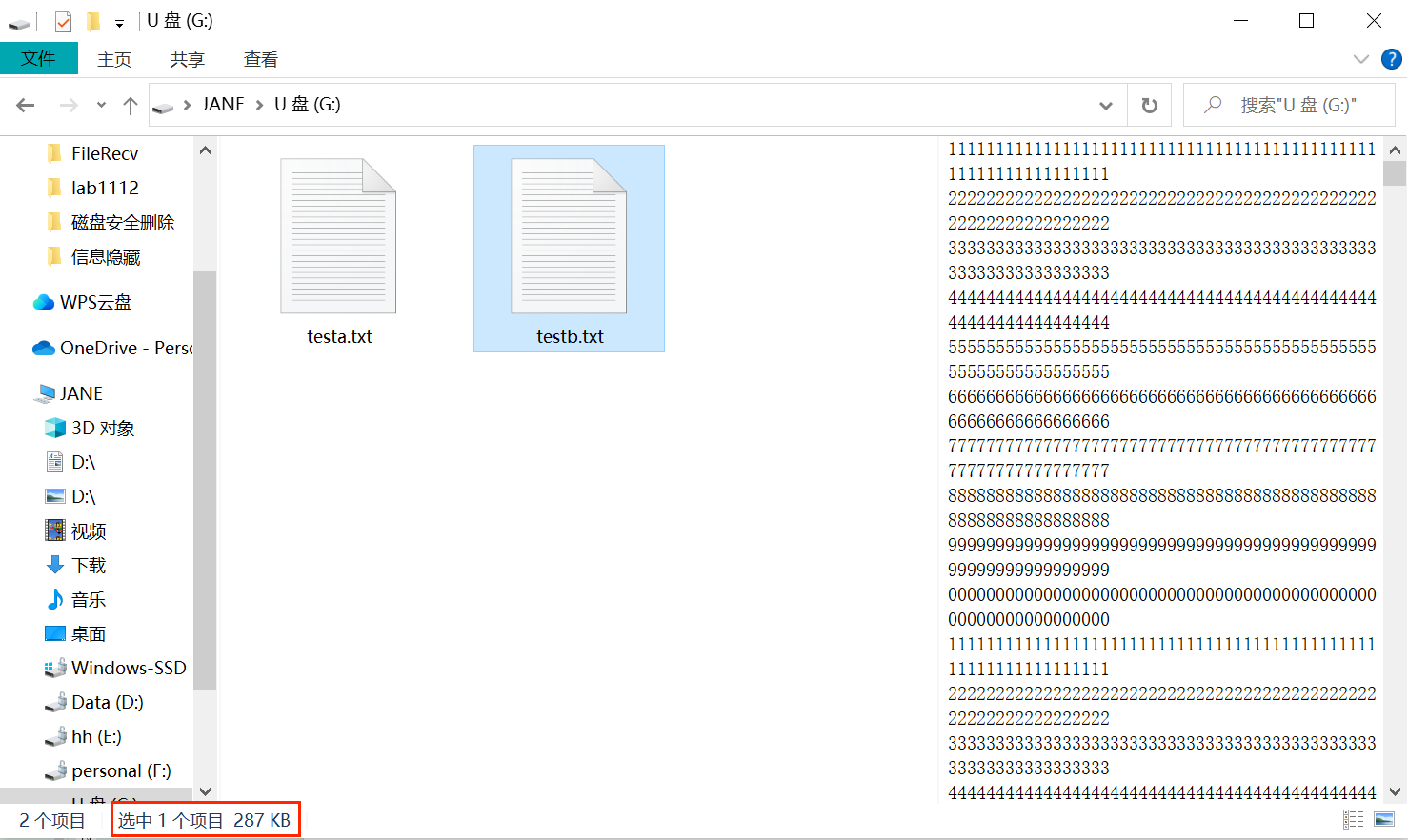
六、实验结果与分析

将U盘格式化，方便观察，新建两个文本文件testa.txt和testb.txt。

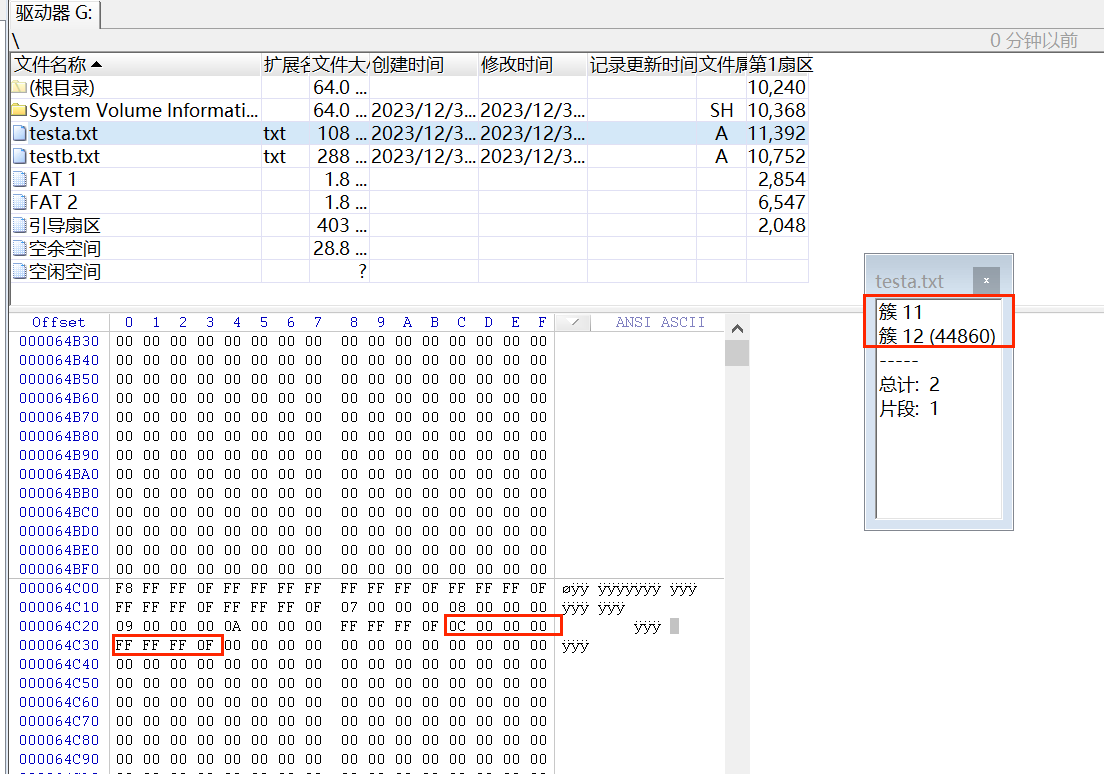
其中，testa.txt中内容为字母，大小为107KB，应该占用2个簇（64KB\*2）



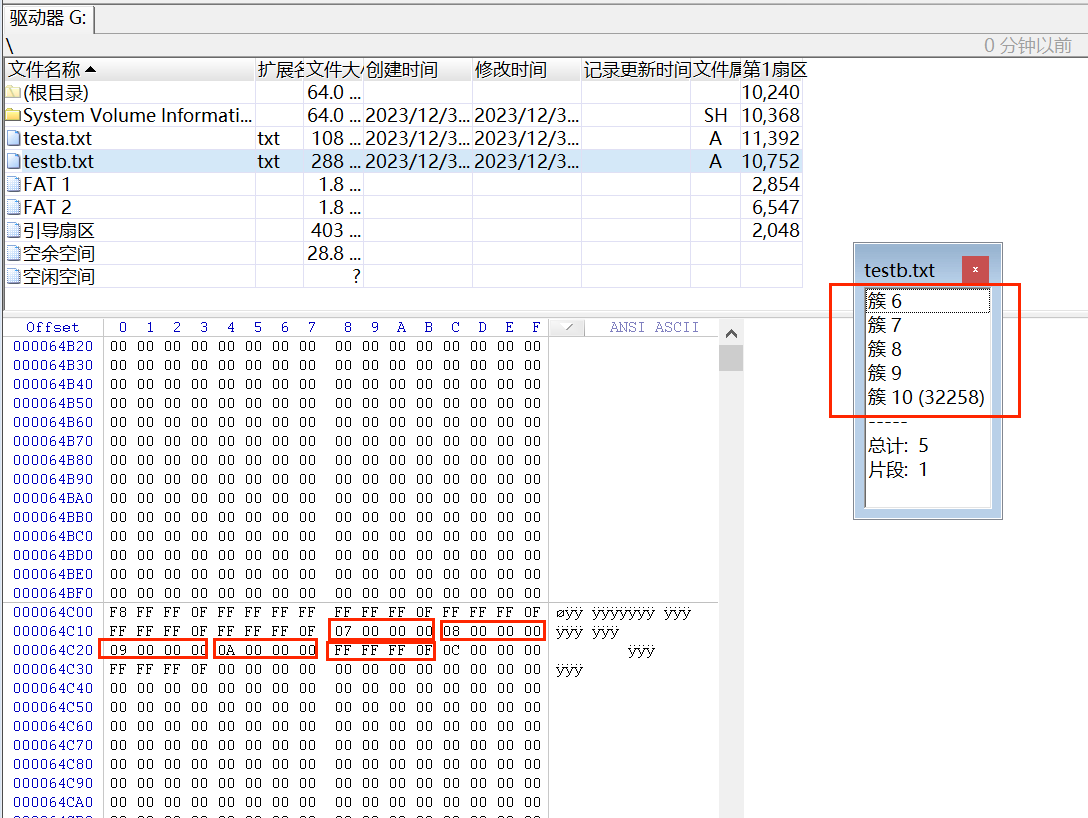
testb.txt中内容为数字，大小为287KB，应该占用5个簇（64KB\*5）



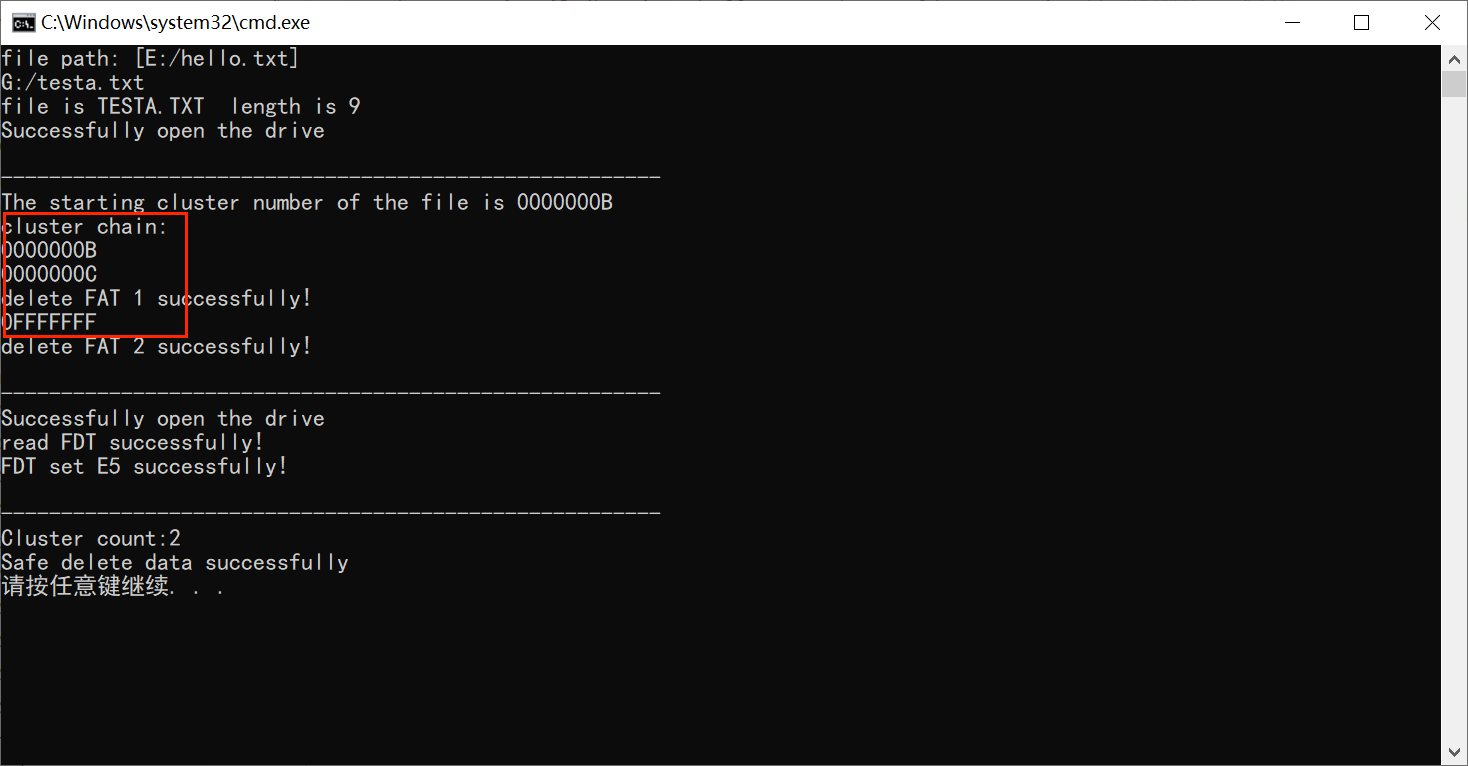
在winhex中列出testa的簇为：11、12



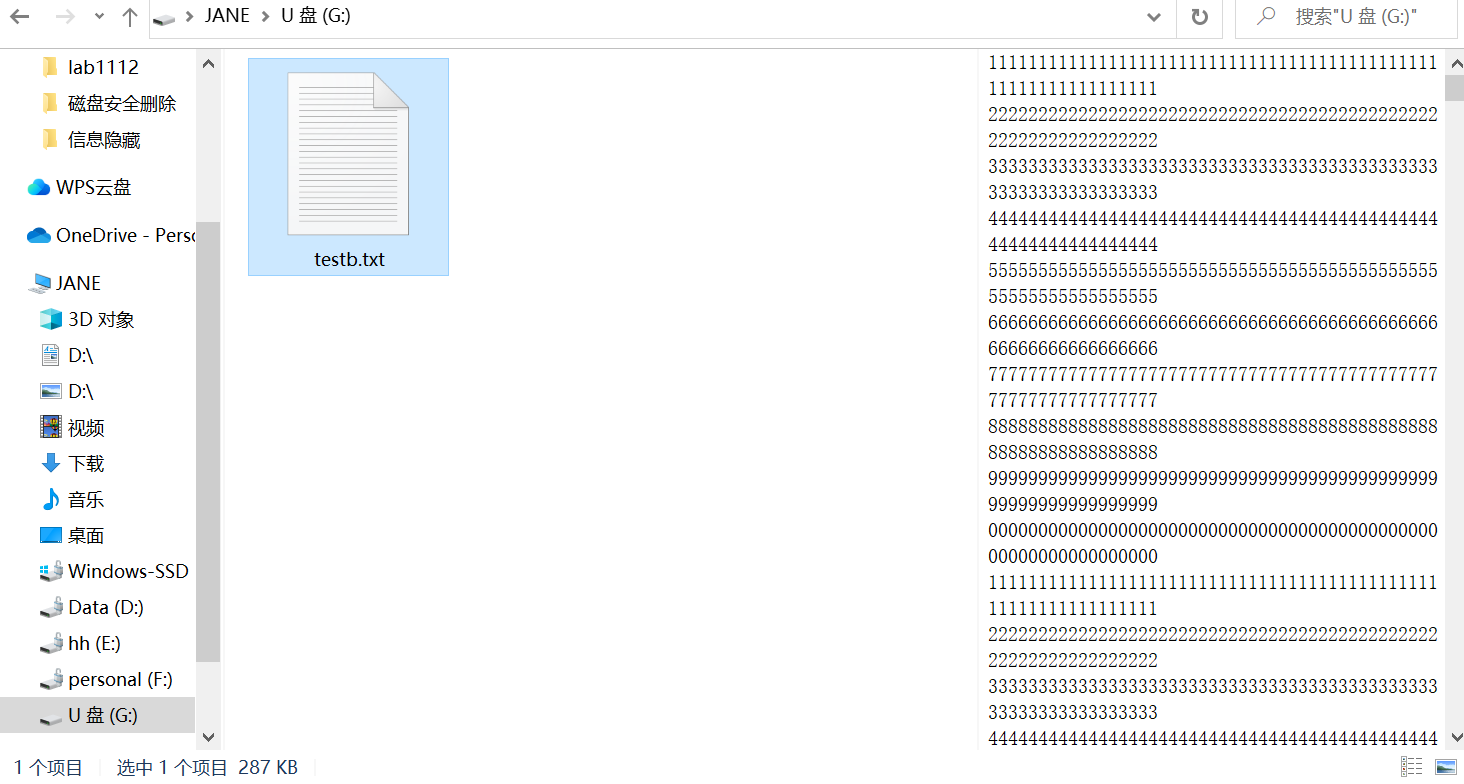
列出testb的簇为：6、7、8、9、10



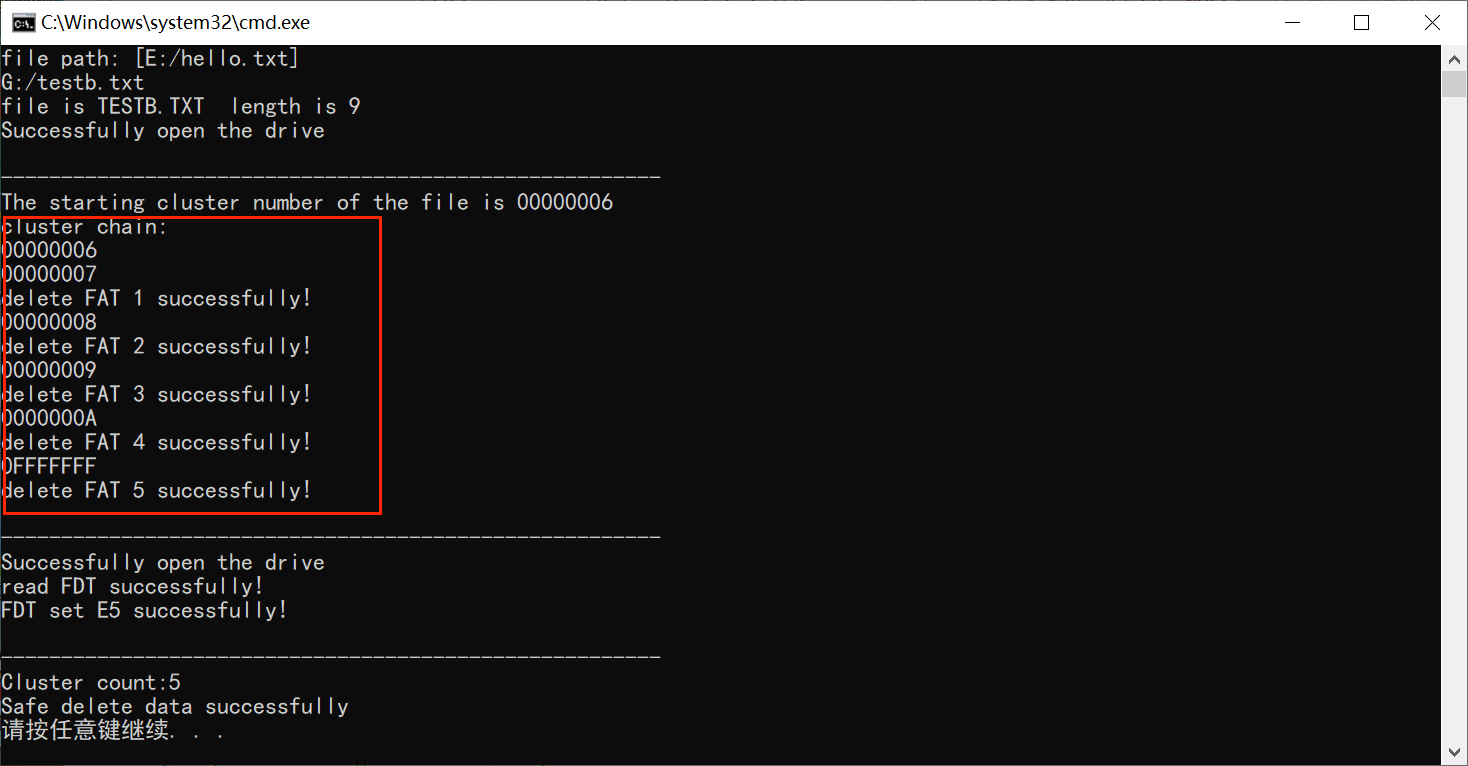
首先删除testa，可以看到程序找出的簇链为0000000B、0000000C，和winhex分析的簇链相同。



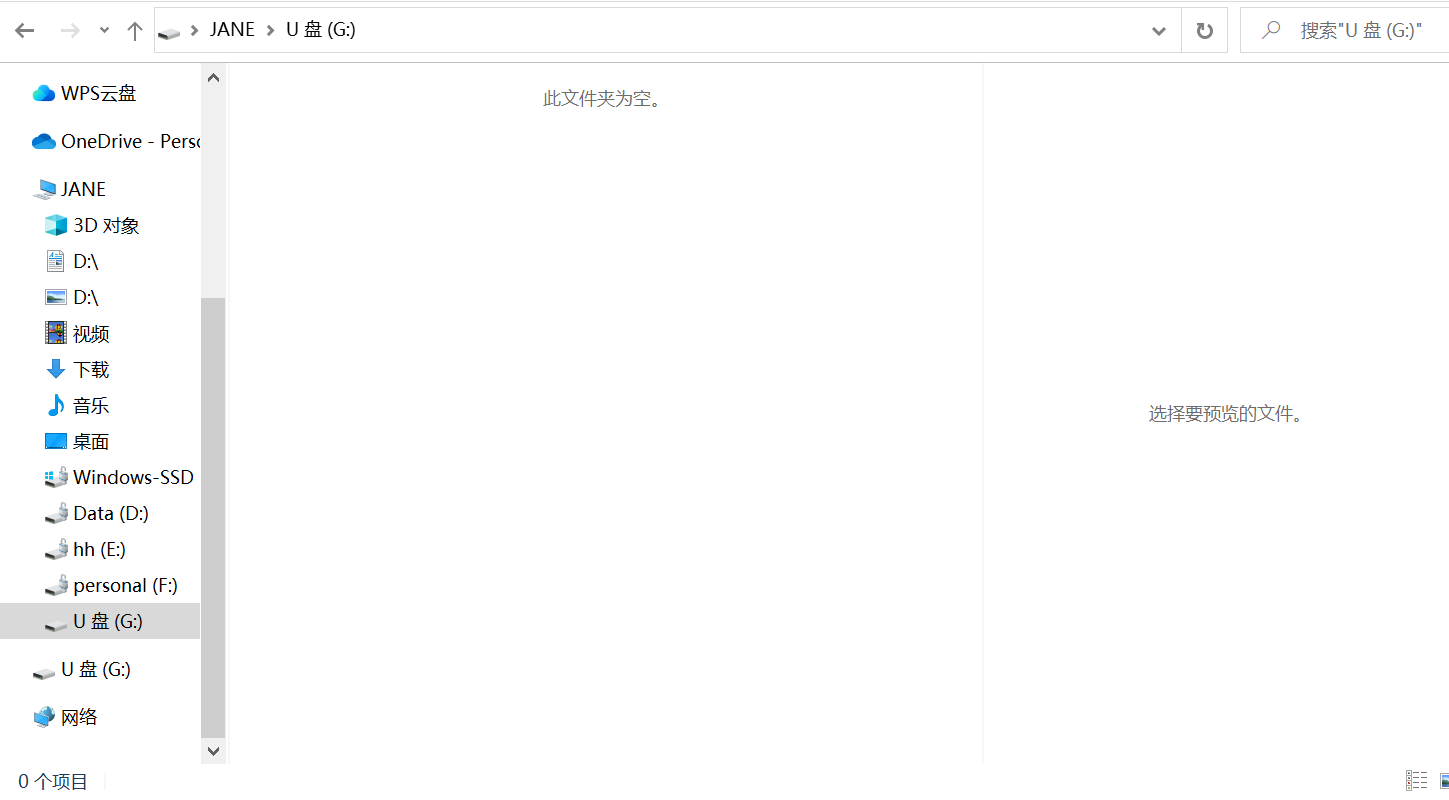
删除后U盘中只剩下了testb。



接着安全删除testb，可以看到程序找到的簇链为6、7、8、9、A，和用winhex分析结果一致。

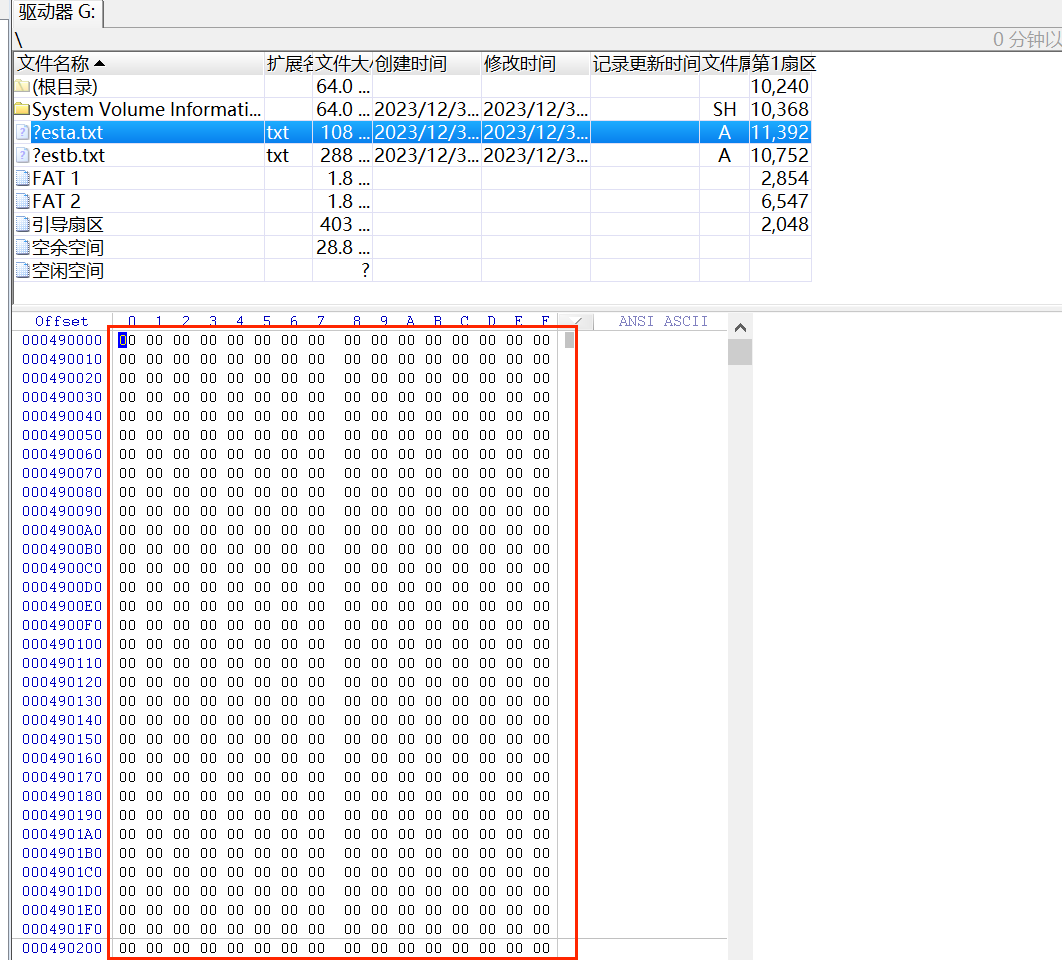


此时U盘中testa和testb都被删除了。

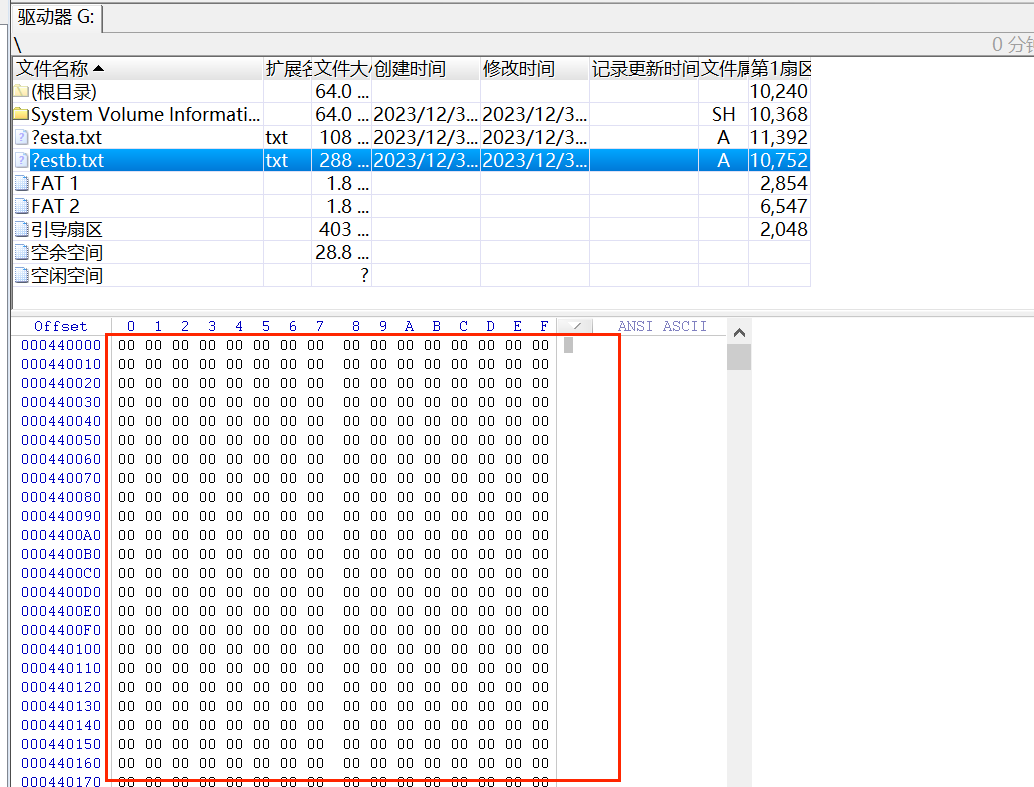


再次使用winhex打开U盘，观察磁盘中的信息。

可以看到testa的内容已被清零：



Testb的内容已被清零：



可以看到，程序成功达到预期，将FDT首字节置为E5，清空了FAT链中的信息，并用0重写了簇的内容。

七、总结及心得体会