武汉大学国家网络安全学院

课程作业报告

题 目： 软件安全实验

专业(班)： 信安5班

学 号： 20222302181161

姓 名： 王亚鹏

课程名称： 软件安全实验

任课教师： 赵磊

2024年 10 月 29日

**目录**

[**实验内容 3**](#_Toc182770108)

[**实验原理 3**](#_Toc182770109)

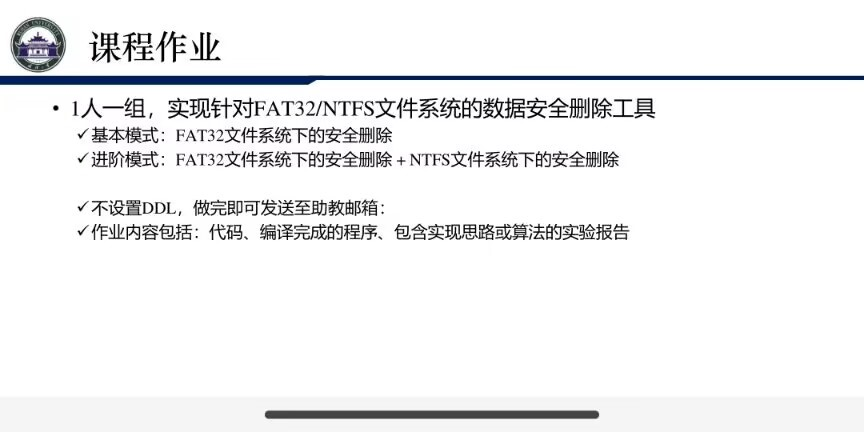
[**实验步骤 4**](#_Toc182770110)

[**代码讲解 6**](#_Toc182770111)

[**实验心得 10**](#_Toc182770112)

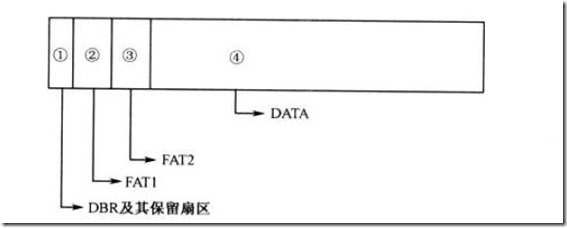
[**附录 11**](#_Toc182770113)

## 实验内容



## 实验原理

FAT32文件系统结构：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：DOS引导记录，为操作系统进入文件系统以后可以访问的第一个扇区（512byte），通常用来解释文件系统，DBR是由硬盘的MBR装载的程序段。DBR装入内存后，即开始执行该引导程序段，其主要功能是完成操作系统的自举并将控制权交给操作系统。每个分区都有引导扇区，但只有被设为活动分区才会被MBR装的DBR入内存运行

**常规删除：**

在 FAT32 文件系统中，常规删除操作并不会真正清除文件数据，而是进行以下处理：

1. **标记目录项为删除状态**：

将目录项的第一个字节改为 0xE5，表示文件已删除。

1. **释放簇链**：

在 FAT 表中将该文件的簇链标记为未使用。

1. **数据未被覆盖**：

文件实际数据仍保留在磁盘上，直到被新数据覆盖。

这意味着普通删除操作后的**数据是可以通过恢复工具找回的**

**安全删除：**

为了确保文件不可恢复，安全删除需要以下步骤：

**1.清除文件的簇链数据**

**覆盖文件数据**：

根据文件起始簇号和簇链获取所有簇。

在每个簇中多次写入伪随机数据、全零或特定模式的数据。

目的：使原始数据无法通过磁盘扫描工具恢复。

**2.清除文件的目录项**

**覆盖目录项**：

定位到文件的目录项位置，将其第一个字节改为 0xE5，标记为删除。

同时清空其他元数据字段（如文件名、起始簇号、文件大小等）。

目的：防止通过目录项元数据快速找到文件位置。

**3.重复覆盖**

**多次覆盖文件数据和目录项**：

安全删除的强度与覆盖次数相关。实验表明，覆盖多次能有效防止数据残留

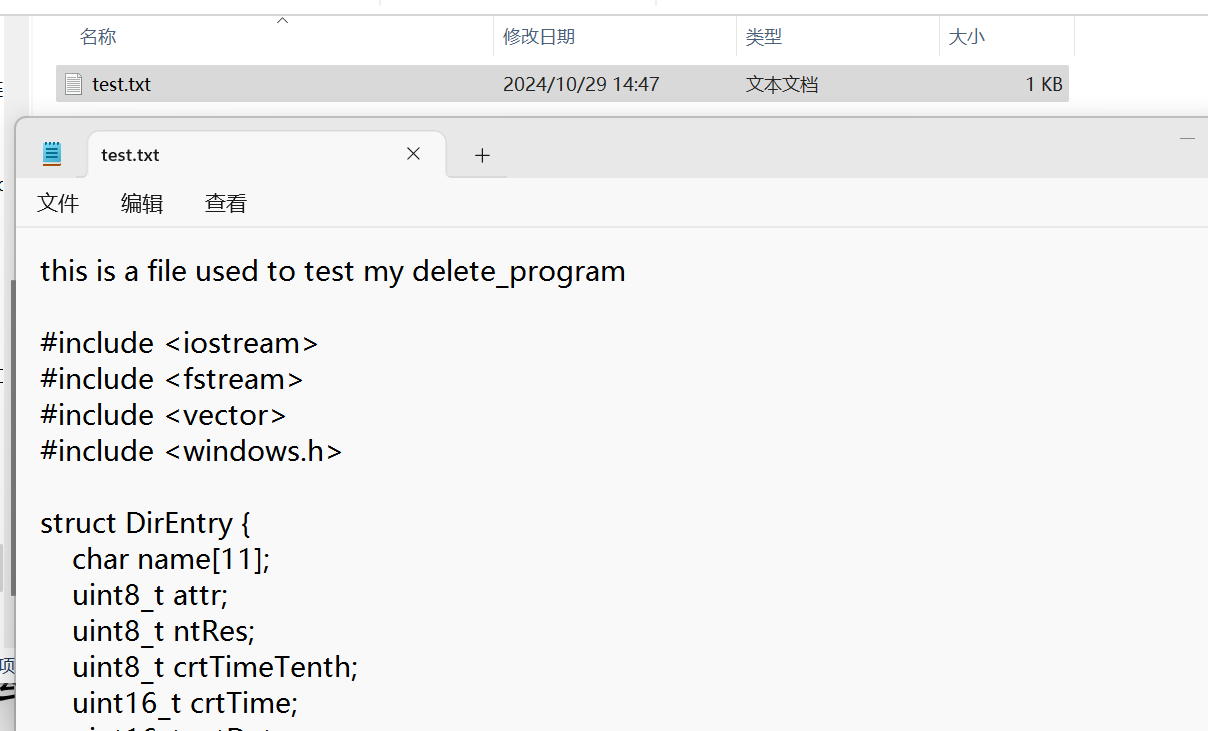
## 实验步骤

**（1）确定方法：**

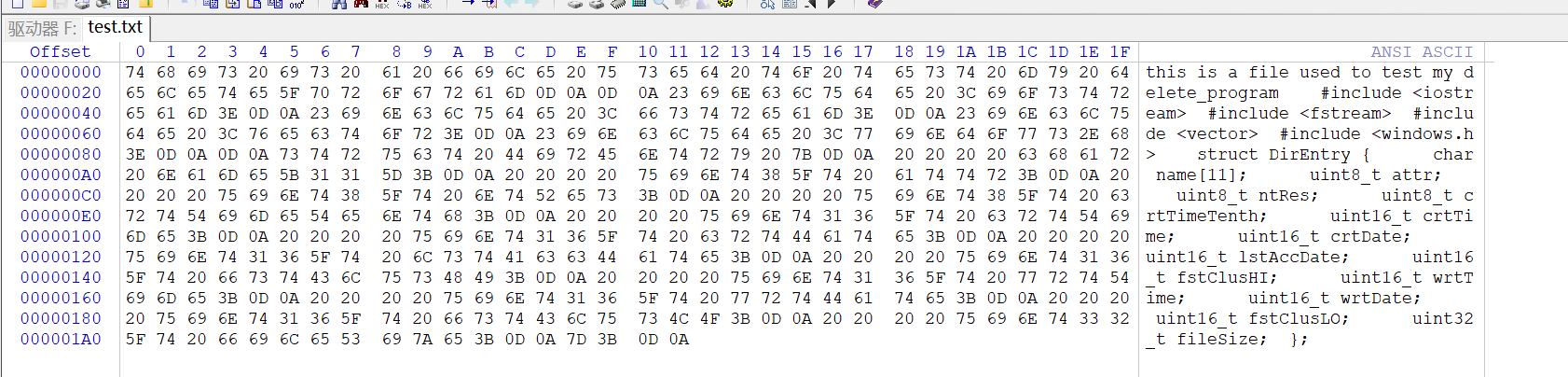
首先找到文件的目录项，找到起始簇号，然后找到FAT链，得到文件的簇，对簇的内容多次填充0的覆盖重写

**（2）实验步骤：**

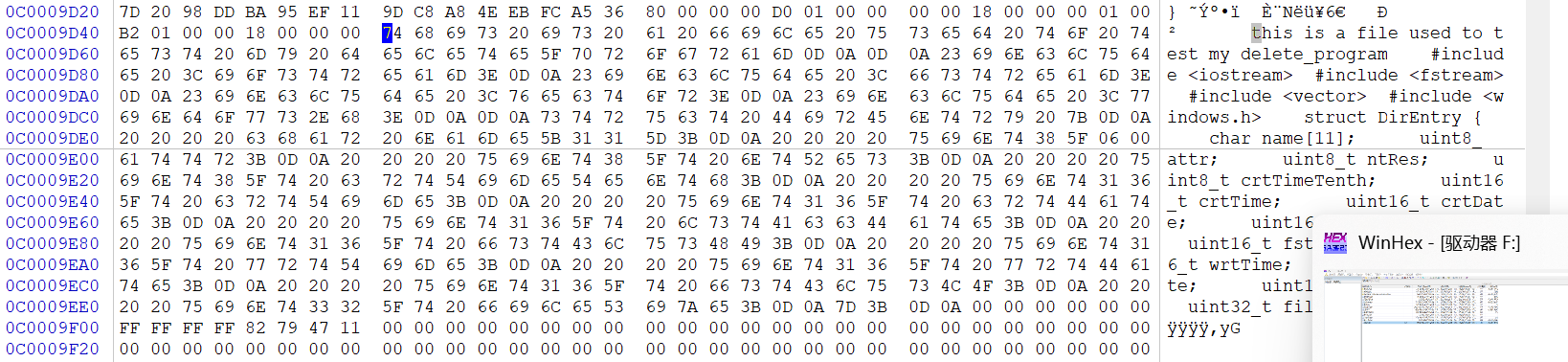
**准备采用FAT32文件系统的U盘和待删除的文件test.txt**

****

**使用winhex查看test.txt的基本信息**

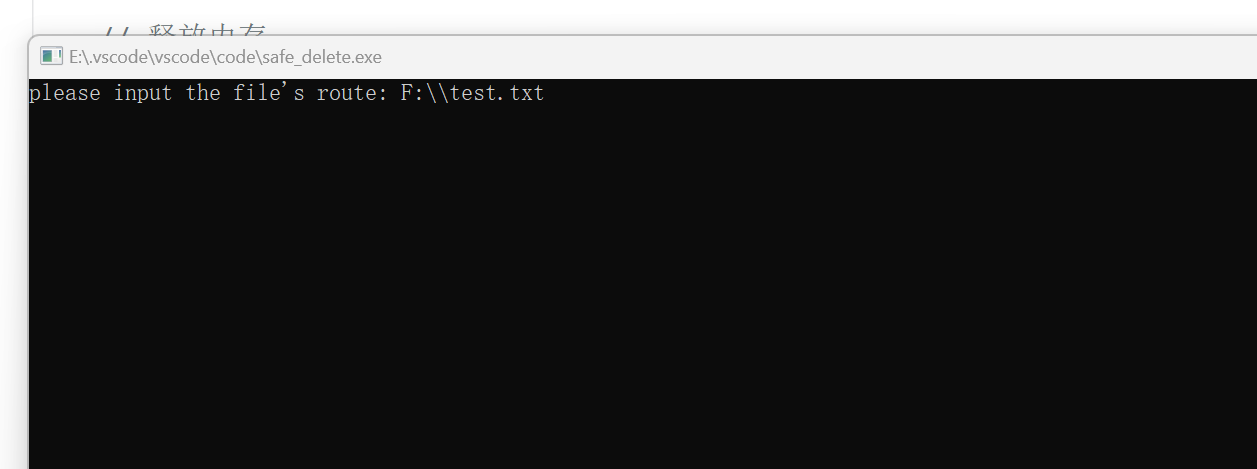


可以看到填充的内容



可以看到相对于这个磁盘，test.txt的起始地址在图中标注为蓝色的地方

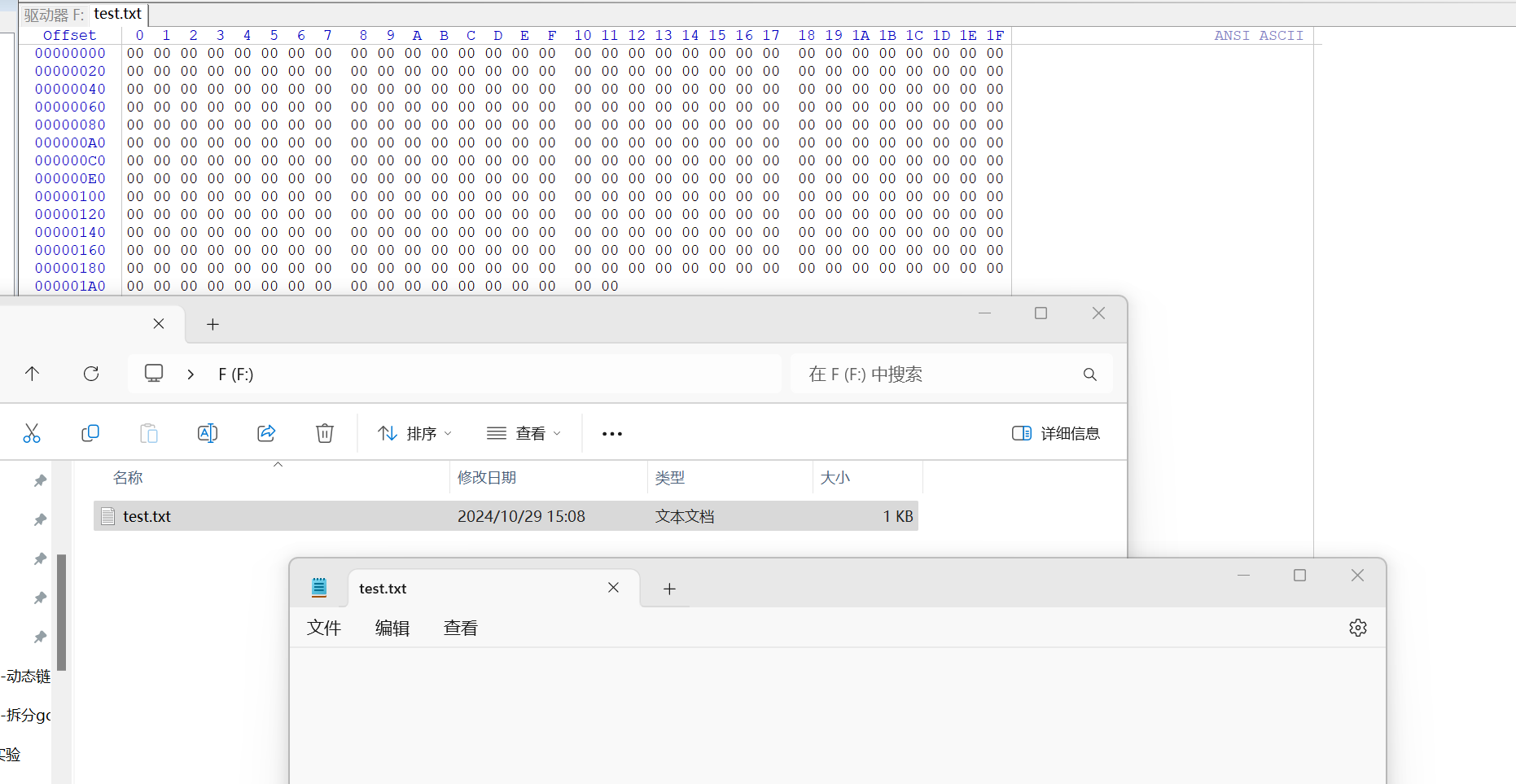
接下来我们进行文件的删除工作，运行安全删除代码safe\_delete.cpp（附在压缩包，稍后有代码讲解）



再通过之前保存的test.txt在整个磁盘中的相对位移，查看这一部分的内容



可以看到，此时磁盘内的内容都被填充为0

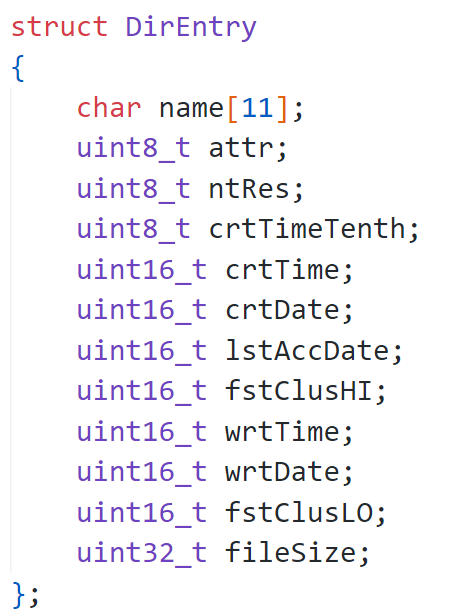


使用实验课上讲过的恢复被删除的文件的方法恢复test.txt，再次右键打开查看内容，可以看到，都被填充为0，使用记事本打开则什么也看不到，实验成功

## 代码讲解

**代码讲解部分：（注意这个代码运行前你需要填充待删除文件的簇号，因此需要winhex的帮助获取相关信息并且记录）**

**（1）结构体DirEntry**



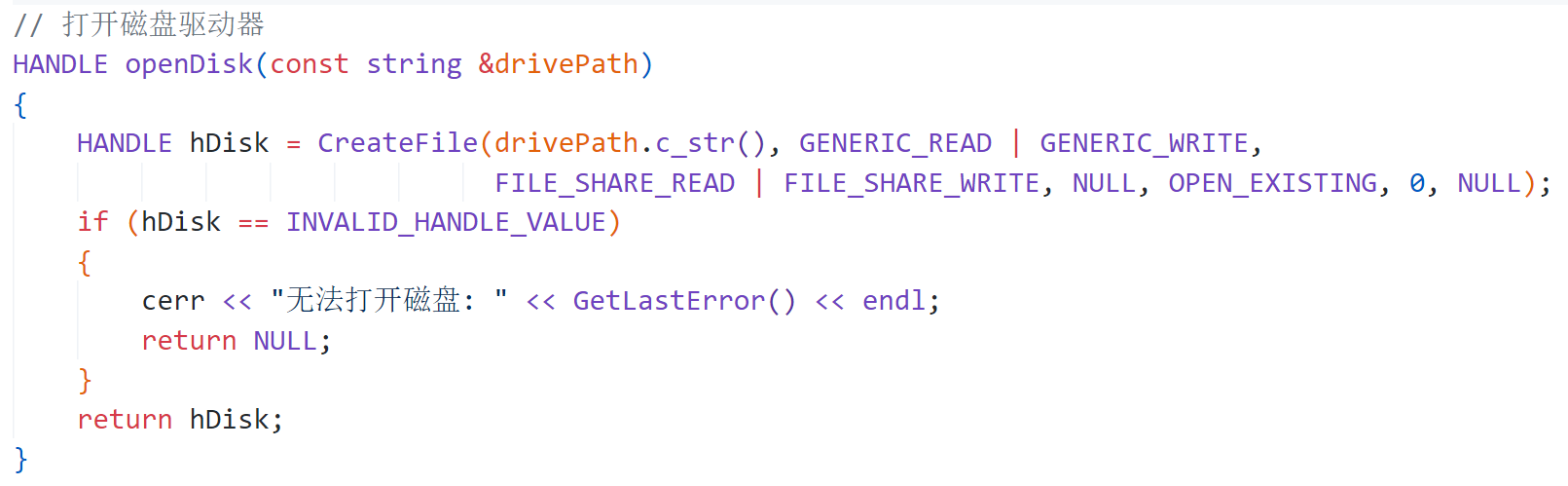
表示 FAT32 文件系统中每个目录项的结构。

**重要字段：**

**fstClusHI 和 fstClusLO：**文件起始簇号（高16位和低16位组合）。

**fileSize：**文件大小

**(2)打开磁盘驱动器**

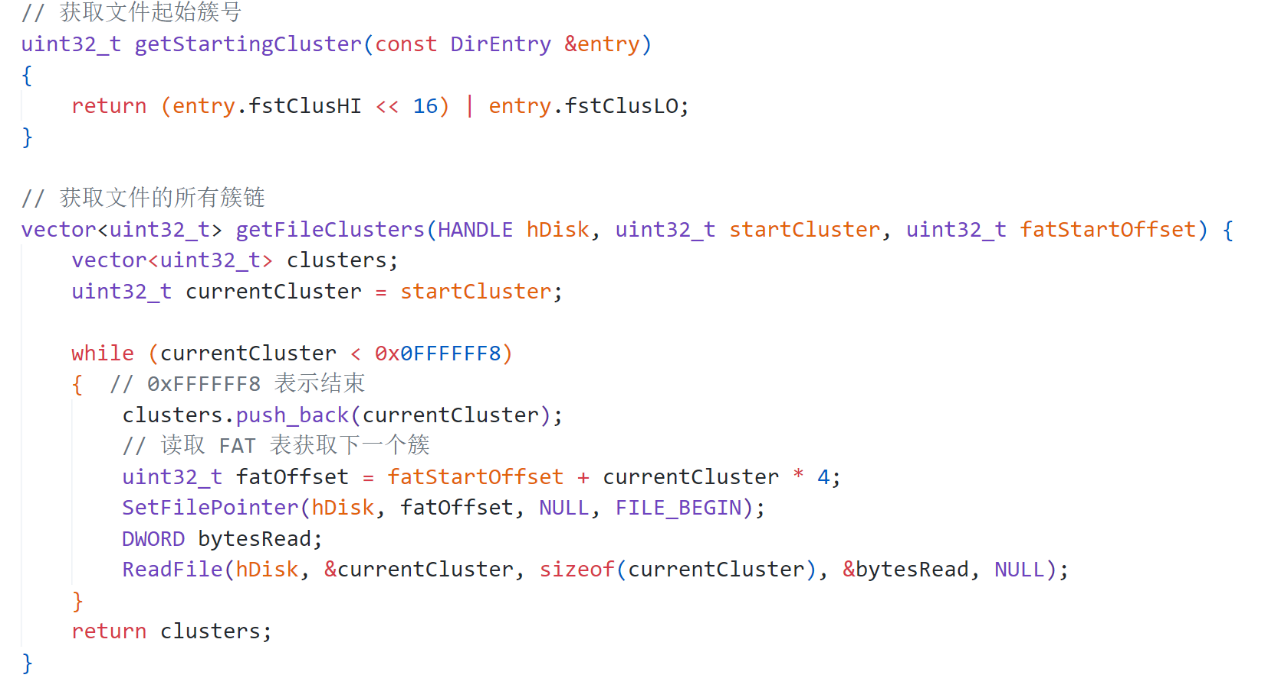


使用 Windows API 函数 CreateFile 打开磁盘。

参数 drivePath 应是类似 \\.\X: 的格式（X为磁盘盘符）。

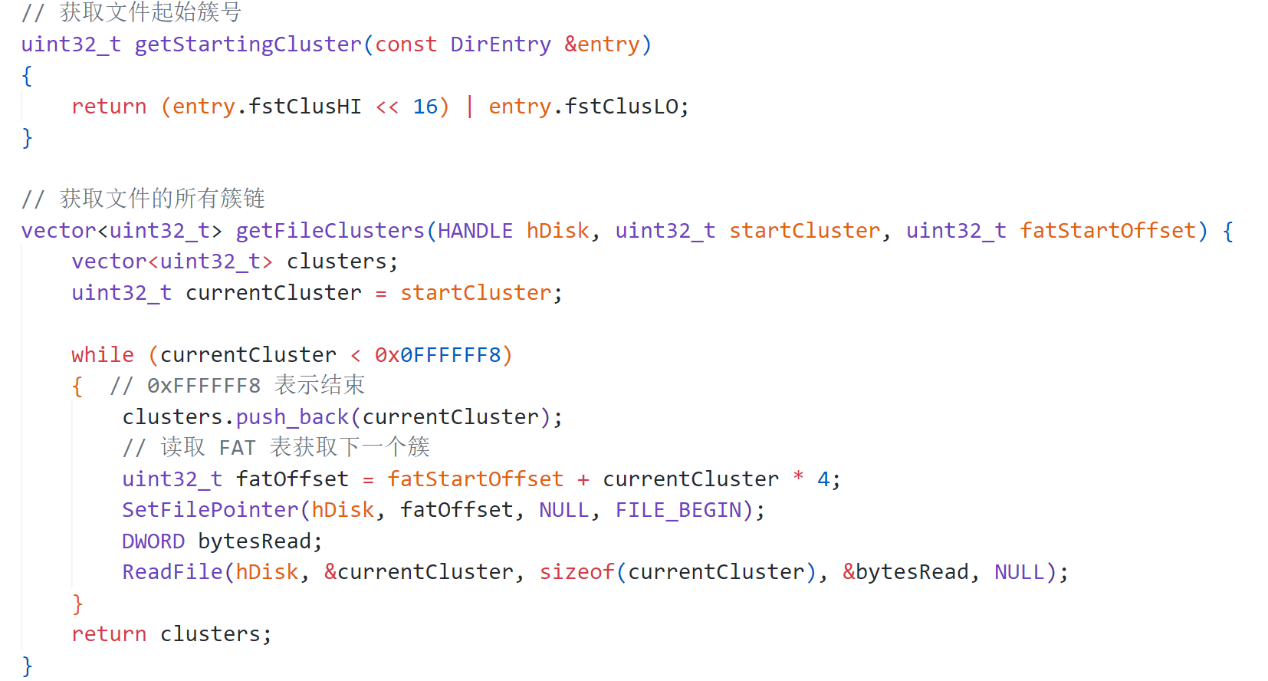
需要管理员权限才能操作

**（3）获取文件起始簇号**



通过组合目录项的 fstClusHI 和 fstClusLO 获取文件的起始簇号

**（4）获取文件所有簇链**



遍历 FAT 表，从起始簇号开始读取文件的所有簇链。

使用 SetFilePointer 定位到 FAT 表中对应的簇链记录位置。

每个簇号指向下一个簇号，直到特殊标记 0x0FFFFFF8（簇链结束）

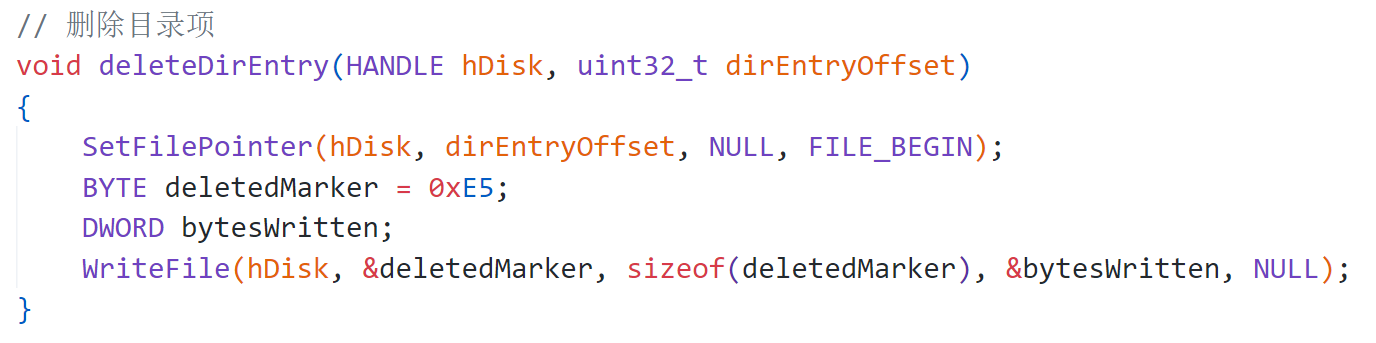
**（5）覆盖每个簇的内容**



将文件所在的簇内容用特定数据（如全零）覆盖多次（默认3次）。

计算每个簇的偏移量，然后使用 WriteFile 写入覆盖数据。

**（6）删除目录项**



定位到文件的目录项位置，将其首字节设置为 0xE5（删除标记）。

使用 WriteFile 写入删除标记。

**（7）主函数**



 **接收输入**：用户提供目标驱动器路径（如 \\.\C:）。

 **打开磁盘**：调用 openDisk 获取磁盘句柄。

 **假定文件信息**：文件起始簇和目录项位置需要提前获取（代码中没有实现获取逻辑）。

 **删除文件**：

* 获取文件的簇链
* 覆盖文件内容
* 清除目录项

总体来说函数的**运行注意事项**如下：

**注意事项**

1. **需要管理员权限**：

使用 CreateFile 打开磁盘和直接写磁盘内容都需要管理员权限

在 Windows 中以管理员身份运行程序

1. **删除逻辑局限性**：

**文件起始簇和目录项偏移位置需要事先获取**，代码中未提供文件扫描或定位逻辑

需要额外实现功能来识别目标文件。

1. **文件系统限制**：

该代码仅适用于 FAT32 文件系统，其他文件系统（如 NTFS）无法使用相同逻辑

1. **数据不可恢复性**：

覆盖文件多次（如3次）后，数据基本不可恢复

## 实验心得

通过本次实验，我收获颇丰，不仅深入理解了 FAT32 文件系统的结构，还掌握了数据安全删除的基本技术。以下是我的主要收获：

1. **掌握了 FAT32 的目录项和 FAT 表结构**：我理解了如何在目录项中找到文件的起始簇号，并利用 FAT 表获取文件的所有簇链。
2. **实现了安全删除文件的流程**：从提取簇链到多次覆盖簇内容，再到清除目录项，我成功实现了一个完整的文件安全删除工具。
3. **熟悉了 WinHex 的使用**：WinHex 在实验中帮助我验证了文件删除的效果，通过查看十六进制数据，直观地展示了覆盖后的变化。

**改进与反思**

在实验中，我也发现了一些可以改进的地方：

1. **提高覆盖效率**：在多次覆盖过程中，使用不同的填充值（如随机数）可能会进一步提升安全性。
2. **适应更多文件系统**：本实验仅针对 FAT32 进行了实现和验证，在未来的实验中，我希望扩展到 NTFS 或其他文件系统，进一步提高工具的适用性。
3. **自动化检测和覆盖**：目前实验代码需要手动输入簇号和目录项偏移，未来可以考虑将这部分代码自动化，实现批量处理的能力

## 附录

**代码全文如下，同一压缩包下的safe\_delete.cpp文件即是**

#include <iostream>

#include <fstream>

#include <vector>

#include <windows.h>

using namespace std;

struct DirEntry

{

    char name[11];

    uint8\_t attr;

    uint8\_t ntRes;

    uint8\_t crtTimeTenth;

    uint16\_t crtTime;

    uint16\_t crtDate;

    uint16\_t lstAccDate;

    uint16\_t fstClusHI;

    uint16\_t wrtTime;

    uint16\_t wrtDate;

    uint16\_t fstClusLO;

    uint32\_t fileSize;

};

// 打开磁盘驱动器

HANDLE openDisk(const string &drivePath)

{

    HANDLE hDisk = CreateFile(drivePath.c\_str(), GENERIC\_READ | GENERIC\_WRITE,

                              FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, 0, NULL);

    if (hDisk == INVALID\_HANDLE\_VALUE)

    {

        cerr << "无法打开磁盘: " << GetLastError() << endl;

        return NULL;

    }

    return hDisk;

}

// 获取文件起始簇号

uint32\_t getStartingCluster(const DirEntry &entry)

{

    return (entry.fstClusHI << 16) | entry.fstClusLO;

}

// 获取文件的所有簇链

vector<uint32\_t> getFileClusters(HANDLE hDisk, uint32\_t startCluster, uint32\_t fatStartOffset) {

    vector<uint32\_t> clusters;

    uint32\_t currentCluster = startCluster;

    while (currentCluster < 0x0FFFFFF8)

    {  // 0xFFFFFF8 表示结束

        clusters.push\_back(currentCluster);

        // 读取 FAT 表获取下一个簇

        uint32\_t fatOffset = fatStartOffset + currentCluster \* 4;

        SetFilePointer(hDisk, fatOffset, NULL, FILE\_BEGIN);

        DWORD bytesRead;

        ReadFile(hDisk, &currentCluster, sizeof(currentCluster), &bytesRead, NULL);

    }

    return clusters;

}

// 覆盖每个簇的内容

void overwriteCluster(HANDLE hDisk, uint32\_t cluster, uint32\_t clusterSize, int overwriteCount)

{

    vector<char> buffer(clusterSize, 0);  // 填充 0 的缓冲区

    uint32\_t clusterOffset = cluster \* clusterSize;  // 根据簇号计算簇偏移

    for (int i = 0; i < overwriteCount; ++i)

    {

        SetFilePointer(hDisk, clusterOffset, NULL, FILE\_BEGIN);

        DWORD bytesWritten;

        WriteFile(hDisk, buffer.data(), buffer.size(), &bytesWritten, NULL);

    }

}

// 删除目录项

void deleteDirEntry(HANDLE hDisk, uint32\_t dirEntryOffset)

{

    SetFilePointer(hDisk, dirEntryOffset, NULL, FILE\_BEGIN);

    BYTE deletedMarker = 0xE5;

    DWORD bytesWritten;

    WriteFile(hDisk, &deletedMarker, sizeof(deletedMarker), &bytesWritten, NULL);

}

// 主函数：执行文件安全删除

int main()

{

    cout << "please input the file's route:";

    string drivePath;  // 目标驱动器路径（需要管理员权限）

    cin >> drivePath;

    HANDLE hDisk = openDisk(drivePath);

    if (hDisk == NULL) return 1;

    // 假设 FAT 表起始位置和簇大小（需根据分区情况调整）

    uint32\_t fatStartOffset = 0x1000;  // FAT 表的起始偏移，我这里是这个值

    uint32\_t clusterSize = 4096;       // 每个簇的大小，我这里是4kb

    // 假设我们已找到文件的起始簇和目录项位置

    uint32\_t startCluster = ;  // 需替换为实际起始簇号

    uint32\_t dirEntryOffset = ;   // 需替换为实际目录项偏移

    // 获取文件的所有簇

    auto clusters = getFileClusters(hDisk, startCluster, fatStartOffset);

    // 覆盖文件的所有簇内容

    for (auto cluster : clusters)

    {

        overwriteCluster(hDisk, cluster, clusterSize, 3);  // 默认覆盖3次

    }

    // 清除文件的目录项

    deleteDirEntry(hDisk, dirEntryOffset);

    CloseHandle(hDisk);

    cout << "文件已安全删除。" << endl;

    return 0;

}