TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

PROJECT 1

QUẢN LÝ HỆ THỐNG TẬP TIN TRÊN WINDOWS

Lớp: 19_1
Môn: Hệ điều hành
GVHD: ThS. Lê Viết Long

Niên khóa: 2021-2022

Mục lục

I. Thông tin nhóm, phân công công việc và mức độ hoàn thành

1. Thông tin nhóm:

MSSV	Họ và tên	Email
19120189	Lê Tiến Đạt	19120189@student.hcmus.edu.vn
19120190	Nguyễn Văn Đạt	19120190@student.hcmus.edu.vn
19120399	Nguyễn Tiến Toàn	19120399@student.hcmus.edu.vn
19120492	Đỗ Thái Duy	19120492@student.hcmus.edu.vn
19120603	Nguyễn Bá Ngọc	19120603@student.hcmus.edu.vn

2. Bảng phân công công việc:

MSSV	Học và tên	Nhiệm vụ	Tiến độ
19120189	Lê Tiến Đạt	+ Viết code: Đọc phân tích bảng FAT của FAT32, format code. + Viết báo cáo: Mô tả bước thực hiện phần đọc thông tin cây thư mục.	100%
19120190	Nguyễn Văn Đạt	+ Viết code: tìm hiểu và code phần thông tin của Boot Sector (FAT32) như viết struct FAT32 và đọc thông tin của Boot Sector. + Viết báo cáo: mô tả các bước cần thực hiện của việc đọc thông tin Boot Sector trong FAT32.	100%

19120399	Nguyễn Tiến Toàn	+ Viết code: Đọc và phân tích bảng RDET, và đọc cây thư mục. + Viết báo cáo: Mô tả kỹ các bước đọc cây thư mục FAT32	100%
19120492	Đỗ Thái Duy	+ Viết code: viết các hàm chính như: chuyển cơ số 16 sang cơ số 10, đọc nội dung trong file TXT, hỗ trợ các phần viết code ở tất cả các phần khác, format hàm main. + Viết báo cáo: soạn lại bố cục, hiệu chỉnh báo cáo và các phần mô tả còn thiếu, làm video Demo.	100%
19120603	Nguyễn Bá Ngọc	+ Viết code: tìm hiểu và code phần thông tin của NTFS như: struct NTFS, đọc thông tin của Partition Boot Sector của NTFS. + Viết báo cáo: Mô tả các bước thực hiện của việc đọc thông tin Partition Boot Sector của NTFS.	100%

3. Đánh giá mức độ hoàn thành:

STT	Tên công việc	Mức độ hoàn thành
1	Đọc các thông tin chi tiết của phân vùng FAT32	100%
2	Đọc các thông tin chi tiết của phân vùng NTFS	100%
3	Hiển thị cây thư mục phân vùng FAT32	100%
4	Hiển thị cây thư mục phân vùng NTFS	0%
	Toàn bộ project	75%

II. Tổng quan đồ án nhóm

Ngôn ngữ lập trình: C++.

Phần mềm thực hiện: Visual Studio, Visual Studio Code.

Phần mềm hỗ trợ: Disk Editor.

1. Đọc thông tin chi tiết vùng Boot Sector của phân vùng FAT32

a. Mô tả Boot Sector của phân vùng FAT32

Offset	Số byte	Nội dung
0	3	Jump_Code: lệnh nhảy qua vùng thông số (như FAT)
3	8	OEM_ID: nơi sản xuất – version, thường là "MSWIN4.1"
B	2	Số byte trên Sector, thường là 512 (như FAT)
D	**1**	**S_C: số sector trên cluster (như FAT)**
E	**2**	**S_B: số sector thuộc vùng Bootsector (như FAT)**
10	**1**	**N_F: số bảng FAT, thường là 2 (như FAT)**
11	2	Không dùng, thường là 0 (số entry của RDET – với FAT)
13	2	Không dùng, thường là 0 (số sector của vol – với FAT)
15	1	Loại thiết bị (F8h nếu là đĩa cứng - như FAT)
16	2	Không dùng, thường là 0 (số sector của bảng FAT – với FAT)
18	2	Số sector của track (như FAT)
1A	2	Số lượng đầu đọc (như FAT)
1C	4	Khoảng cách từ nơi mô tả vol đến đầu vol (như FAT)
20	**4**	**S_V: Kích thước volume** (như FAT)
24	**4**	**S_F: Kích thước mỗi bảng FAT**
28	2	bit 8 bật: chỉ ghi vào bảng FAT active (có chỉ số là 4 bit đầu)
2A	2	Version của FAT32 trên vol này
2C	**4**	**Cluster bắt đầu của RDET**
30	**2**	**Sector chứa thông tin phụ (về cluster trống), thường là 1**
32	**2**	**Sector chứa bản lưu của Boot Sector**
34	C	Dành riêng (cho các phiên bản sau)
40	1	Kí hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
41	1	Dành riêng
42	1	Kí hiệu nhận diện HĐH
43	4	SerialNumber của Volume
47	B	Volume Label
52	**8**	**Loại FAT, là chuỗi "FAT32"**
5A	1A4	Đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy
1FE	2	Dấu hiệu kết thúc BootSector /Master Boot (luôn là AA55h)

(Hình 1.a: Mô tả Boot Sector FAT32)

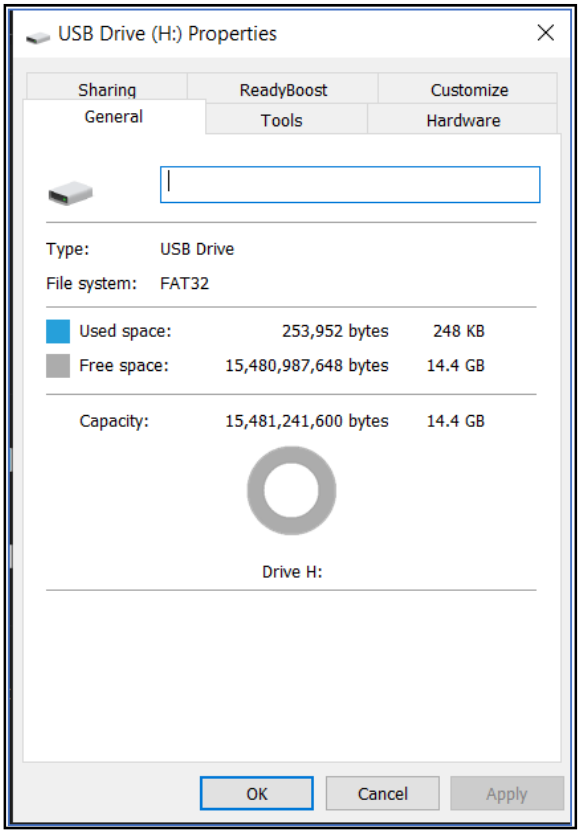
Boot Sector của FAT32 thường chứa nhiều trường dữ liệu. Tuy nhiên, các thông số quan trọng sẽ được sử dụng lại nhiều trong đồ án này mà ta cần lưu ý

là:

- Số byte trên Sector
- Số sector trên cluster (**SC**)
- Số sector thuộc vùng Boot Sector (**SB**)
- Số bảng FAT (**NF**)
- Số entry của RDET
- Kích thước volume (**SV**)
- Kích thước mỗi bảng FAT (**SF**)
- Cluster bắt đầu của RDET

b. Môi trường thử nghiệm

Đối với FAT32, chúng em sử dụng USB dung lượng 16GB để chạy mã nguồn và đánh giá kết quả. Ổ đĩa tương ứng trên Windows là ổ đĩa H. Phân vùng được format thành định dạng FAT32. Thông tin cơ bản về phân vùng này như sau:



(Hình 1.b: Thông tin cơ bản phân vùng FAT32)

c. Quy trình thực hiện

Trước tiên, để đọc thông tin Boot Sector thuận tiện, chúng em khai báo một struct FAT32 có các trường dữ liệu là các thông tin quan trọng của phân vùng FAT32 đã được nhắc đến ở trên.

```
struct FAT32 {
    int bytePerSector;          //  Số byte của một Sector      (B - 2)
    int sectorPerCluster;       //  Số Sector của một Cluster   (D - 1) SC
    int reservedSector;         //  Số Sector của BootSector    (E - 2) SB
    int numFat;                 //  Số bảng FAT                 (10 - 1) NF
    int rdetEntry;              //  Số Entry của RDET           (11 - 2) SRDET
    int totalSector;            //  Kích thước Volume           (32 - 4) Sv
    int sectorPerFAT;           //  Số Sector của một bảng FAT  (24 - 4) Sf
    int rootCluster;            //  Cluster bắt đầu của RDET    (2C - 4)
};
```

(Hình 1.c.1: Khai báo struct FAT32)

Tiếp theo, chúng em định nghĩa hàm ***hexToDecimal( )*** để đọc trên một ***sector*** từ vị trí ***offset*** một số byte là ***count***. Vì đối với FAT32, dữ liệu được lưu trữ theo kiểu Little Endian, có nghĩa là các dữ liệu số phải được đọc ngược từ byte cuối, nên hàm này được cài đặt để đảo thứ tự các byte, đảm bảo đọc đúng thông tin.

```
int hexToDecimal(BYTE* sector, int offset, int count)
{
    int power, sum;
    power = sum = 0;
    BYTE X;
    for (int i = 0; i < count; i++)
    {
        X = sector[i + offset];
        for (int j = 0; j < 8; j++)
        {
            if (X & 1) sum = sum + (int)pow(2, power);
            X = X >> 1; power++;
        }
    }
    return sum;
}
```

*(Hình 1.c.2: Hàm **hexToDecimal( )**)*

Để đọc một sector trên phân vùng, chúng em sử dụng hàm ***readSector( )*** được giảng viên Lê Viết Long cung cấp từ trước. Hàm này đọc trên một phân vùng ***drive*** một sector ***sector*** (512 byte) từ vị trí ***readPoint***.

```
int readSector(LPCWSTR  drive, int readPoint, BYTE sector[512])
{
    int retCode = 0;
    DWORD bytesRead;
    HANDLE device = NULL;
    device = CreateFile(drive,  // Drive to open
        GENERIC_READ,           // Access mode
        FILE_SHARE_READ | FILE_SHARE_WRITE,        // Share Mode
        NULL,                   // Security Descriptor
        OPEN_EXISTING,          // How to create
        0,                      // File attributes
        NULL);                  // Handle to template

    if (device == INVALID_HANDLE_VALUE) // Open Error
    {
        printf("CreateFile: %u\n", GetLastError());
        return 1;
    }
    SetFilePointer(device, readPoint, NULL, FILE_BEGIN);    //Set a Point to Read

    if (!ReadFile(device, sector, 512, &bytesRead, NULL)) printf("ReadFile: %u\n", GetLastError());
    else printf("Success!\n");
}
```

*(Hình 1.c.3: Hàm **readSector( )**)*

Đã có được các hàm cần thiết cho việc đọc thông tin Boot Sector, dựa vào bảng mô tả Boot Sector (Hình 1.a) để có được thông tin về offset và số lượng byte cần đọc của các trường dữ liệu, ta định nghĩa hàm ***initBootSector( )*** để đọc thông tin Boot Sector vào một biến có kiểu là ***FAT32*** (Hình 1.c.1).

```
void initBootSector(BYTE* sector, FAT32& fat32) {

    fat32.bytePerSector = hexToDecimal(sector, 11, 2);
    fat32.sectorPerCluster = hexToDecimal(sector, 13, 1);
    fat32.reservedSector = hexToDecimal(sector, 14, 2);
    fat32.numFat = hexToDecimal(sector, 16, 1);
    fat32.rdetEntry = hexToDecimal(sector, 17, 2);
    fat32.totalSector = hexToDecimal(sector, 32, 4);
    fat32.sectorPerFAT = hexToDecimal(sector, 36, 4);
    fat32.rootCluster = hexToDecimal(sector, 44, 4);
}
```

*(Hình 1.c.4: Hàm **initBootSector( )**)*

Hàm cuối mà ta định nghĩa trong phần này là hàm ***readInformationBootSector( )***. Hàm này nhận vào một biến ***fat32*** có kiểu là ***FAT32*** đã đọc được thông tin

Boot Sector thông qua hàm ***initBootSector( )*** (Hình 1.c.4), thực hiện một số tính toán và in ra thông tin của Boot Sector.

```
void readInformationBootSector(FAT32 fat32)
{
    cout << "FAT: FAT32" << endl;
    cout << "So Byte cho 1 sector: " << fat32.bytePerSector << endl;
    cout << "So Sector cho 1 cluster: " << fat32.sectorPerCluster << endl;
    cout << "So Sector cua BootSector (Sb): " << fat32.reservedSector << endl;
    cout << "So bang FAT (NF): " << fat32.numFat << endl;
    cout << "Tong so sector, kich thuoc Volume (Sv): " << fat32.totalSector << endl;
    cout << "Sector per FAT: " << fat32.sectorPerFAT << endl;
    cout << "So Sector cho bang RDET: " << fat32.rdetEntry << endl;
    cout << "Cluster bat dau cua RDET: " << fat32.rootCluster << endl;
    cout << "Sector dau tien FAT1: " << fat32.reservedSector << endl;

    int firstSectorDATA = fat32.reservedSector + fat32.numFat * fat32.sectorPerFAT;
    cout << "Sector dau tien cua vung RDET: " << firstSectorDATA << endl;
    cout << "Sector dau tien cua vung DATA: " << firstSectorDATA << endl;
}
```

*(Hình 1.c.5: Hàm **readInformationBoootSector( )**)*

Ở đây có một lưu ý là biến ***firstSectorDATA***. Đây là biến lưu thông tin về sector đầu tiên của vùng DATA, và đối với FAT32, sector đầu tiên của vùng DATA cũng là sector đầu tiên của RDET. Vì trong phân vùng FAT32, trước vùng DATA có Boot Sector và các bảng FAT (thường là 2) nên ta có công thức:

<Sector dau tien vung DATA> = SB+NF*SF

Với **SB** là số sector của Boot Sector, **NF** là số bảng FAT và **SF** là số sector mỗi bảng FAT.

Ngoài ta, ta cũng lưu ý rằng sector đầu tiên của bảng FAT1 cũng chính là số sector của Boot Sector.

Cuối cùng, để thực hiện đọc và hiển thị thông tin Boot Sector của FAT32, trong hàm main, ta lần lượt thực hiện như sau:

```
BYTE sector[512];
string a = "\\\\.\\" + nameDisk + ":";
CString str(a.c_str());
CStringW strw(str);
LPCWSTR drive1 = strw;


readSector(drive1, 0, sector);
initBootSector(sector, fat32);

//Đọc các thông tin được mô tả trong Boot Sector
cout << "++++++++++++++++++    THONG TIN BOOT SECTOR     ++++++++++++++++++\n" << endl;
readInformationBootSector(fat32);
```

(Hình 1.c.6: Đọc và hiển thị thông tin phân vùng FAT32)

d. Kiểm tra kết quả

Kết quả thu được sau khi lần lượt thực hiện các bước trong phần 3 như sau:

```
Success!
++++++++++++++++++    THONG TIN BOOT SECTOR     ++++++++++++++++++

FAT: FAT32
So Byte cho 1 sector: 512
So Sector cho 1 cluster: 16
So Sector cua BootSector (Sb): 3238
So bang FAT (NF): 2
Tong so sector, kich thuoc Volume (Sv): 30269568
Sector per FAT: 14765
So Sector cho bang RDET: 0
Cluster bat dau cua RDET: 2
Sector dau tien FAT1: 3238
Sector dau tien cua vung RDET: 32768
Sector dau tien cua vung DATA: 32768
```

(Hình 1.d.1: Kết quả thu được)

Dữ liệu phần đầu Boot Sector của phân vùng được hiển thị trong Disk Editor như sau:

Offset	00 01 02 03 04 05 06 07	08 09 10 11 12 13 14 15	ASCII
00000000000	EB 58 90	4D 53 44 4F 53 35 2E 30 00 02 10 A6 0C	ëX.MSDOS5.0...¦.
00000000016	02 00 00 00 00 F8 00 00	3F 00 FF 00 80 1F 00 00	.....ø..?.ÿ.....
00000000032	80 E0 CD 01 AD 39 00 00	00 00 00 00 02 00 00 00	.àÍ.9..........
00000000048	01 00 06 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00000000064	80 00 29 0C 64 51 F0 4E	4F 20 4E 41 4D 45 20 20	..).dQðNO NAME
00000000080	20 20 46 41 54 33 32 20	20 20 33 C9 8E D1 BC F4	FAT32 3É.Ñ¼ô

(Hình 1.d.2: Phần đầu Boot Sector)

Đối chiếu với thông tin phân vùng của Disk Editor, ta khẳng định tính chính xác của chương trình:

BIOS Parameter Block	011		
Bytes per sector	011	512	512
Sectors per cluster	013	16	16
Reserved sectors	014	3,238	3,238
Number of FATs	016	2	2
(unused)	017	00 00	00 00
(unused)	019	00 00	00 00
Media descriptor	021	0xF8	0xF8
(unused)	022	00 00	00 00
Sectors per track	024	63	63
Number of heads	026	255	255
Hidden sectors	028	8,064	8,064
Total sectors	032	30,269,568	30,269,568
Sectors per FAT	036	14,765	14,765
Extended flags	040	0	0
Version	042	0	0
Root cluster	044	2	2
System Information...	048	1	1
Backup Boot sector	050	6	6
(reserved)	052	00 00 00 ...	00 00 00 00 00 00 00 00 00 00 00 00

(Hình 1.d.3: Thông tin phân vùng của Disk Editor)

2. Hiển thị cây thư mục phân vùng FAT32

a. Mô tả các bước thực hiện

Ta tạo một struct tên là DIRECTORY để lưu trữ các thông tin của thư mục, tập tin như tên, trạng thái, cluster bắt đầu, kích thước, 1 con trỏ *next để liên kết đến địa chỉ của DIRECTORY tiếp theo, 1 con trỏ *dir để liên kết đến địa chỉ của DIRECTORY con (nếu có).

```
struct DIRECTORY {
    char Name[256];             // Tên thư mục/ tập tin
    int Attr;                   // Thuộc tính (thường là thư mục/ tập tin)
    int StartCluster;           // Cluster bắt đầu
    int FileSize;               // Kích cỡ (tính theo byte)
    DIRECTORY* next;            // Trỏ đến thư mục/ tập tin tiếp theo
    DIRECTORY* dir;             // Trỏ đến thư mục con
};
```

Đầu tiên ta sẽ phân tích bảng FAT bằng hàm ***initFAT( )*** bằng cách tìm và đọc từ ổ đĩa nhờ 2 hàm ***seekg( )*** và ***read( )*** bắt đầu bằng chỉ số *số Sector của BootSector * Kích thước của một sector,* (bởi vì sector đầu tiên của bảng FAT là SB – số Sector của BootSector) và truyền vào mảng động ***a*** được khai báo với số lượng phần tử là *kích cỡ theo byte của số sector của bảng FAT*, sau đó gán lại cho tham số đầu vào ***FAT.***

```
// Đọc bảng FAT của FAT32
void initFAT(int*& FAT, FAT32 fat32, LPCWSTR drive1)
{
    //There are 2 FAT tables
    //FAT32 -- 32 bits (4 Bytes) but only 28 bits are used and high 4 bits are reserved
    int fatTableSize = fat32.sectorPerFAT * fat32.bytePerSector;
    BYTE* readBytes = new BYTE[fatTableSize];
    fstream diskStream(drive1, std::ios::in);
    char* a = new char[fatTableSize];
    diskStream.seekg(fat32.reservedSector * fat32.bytePerSector, 0);
    diskStream.read(a, fatTableSize);

    readBytes = (BYTE*)a;
    int numberOfFATEntries = fatTableSize / 4; //FAT_RECORD_SIZE;
    int count = 0;
    //8946
    for (int i = 0; i < numberOfFATEntries; i++)
    {
        if (hexToDecimal(readBytes, 4 * i, 4) == 0)
        {
            count += 1;
            if (count >= 200)
                break;
        }
        FAT[i] = hexToDecimal(readBytes, 4 * i, 4);
    }

}
```

Sau đó, ta cần biết được sector đầu của RDET để làm. Ta có thể kiếm được sector đầu tiên của RDET bằng các thông tin kiếm được ở phần 1 (Đọc thông tin chi tiết phân vùng FAT32).

Khi đó:

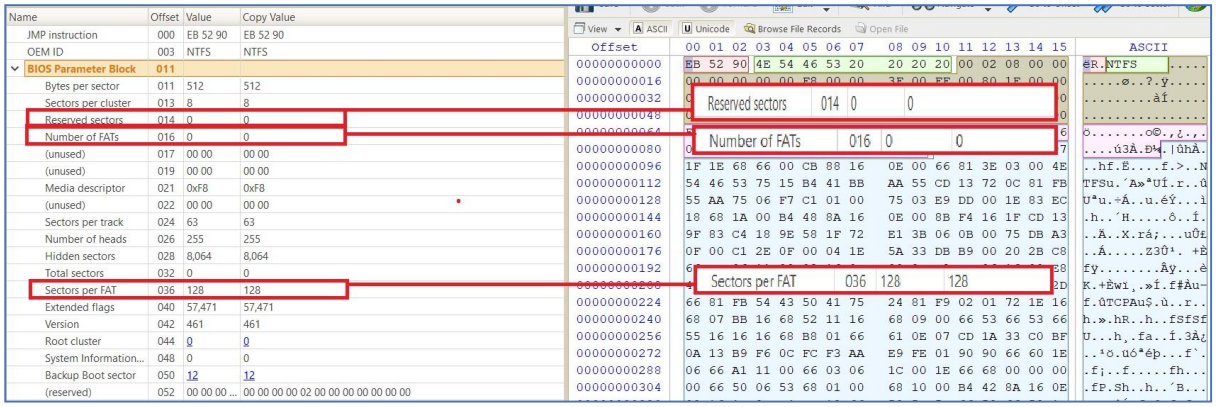
• Reserved sector = SB = 3238

• Number of FATS = NF = 2

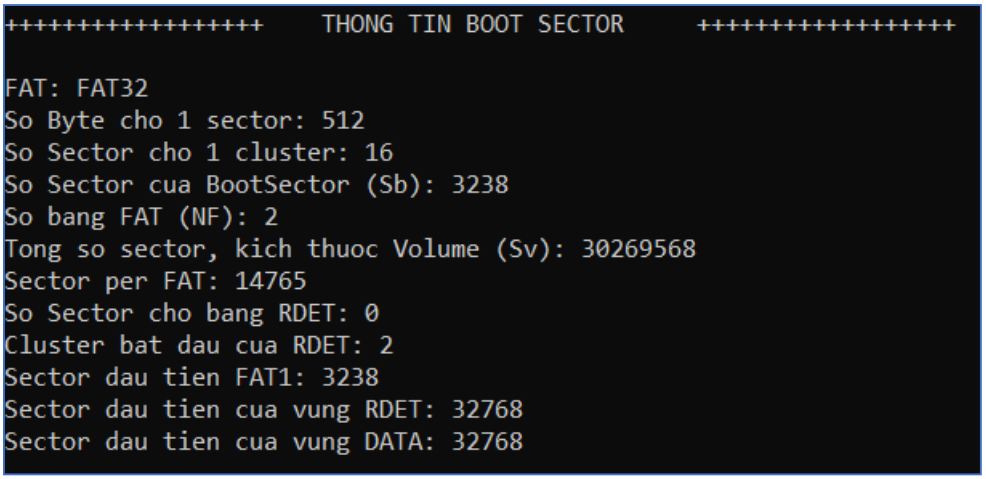
• Sectors per FAT = SF = 14765

=>Sector đầu tiên của RDET = SB + NF*SF = 32768

Có sector này giờ ta đơn giản là tìm cách chạy đến vị trí sector này để đọc thông tin của RDET.



(Hình 2.a.1: Những thông tin lấy được từ phần mềm)



(Hình 2.a.2: Thông tin lấy được từ code)

Lúc này, ta được sector đầu tiên của RDET = 32768

Bây giờ ta cần chương trình có thể đọc được thông tin ở vị trí sector 32768

Khi đã biết đọc được thông tin của bảng FAT, ta bắt đầu thực hiện đọc cây thư mục qua hàm ***readDirectory( ).***

```
DIRECTORY* readDirectory( int firstEntryIndex, int clusIndex, int* FAT, FAT32 fat32, LPCWSTR drive1, string space)
{
    int clusSize = fat32.sectorPerCluster * fat32.bytePerSector;
    //Root directory entries có kích thước 32 Bytes
    //và số lượng các entries bị giới hạn bởi kích thước của cluster
    //và kích thước dựa trên FAT entries

    int totalEntries = (fat32.sectorPerCluster * fat32.bytePerSector) / 32;
    char* a = new char[clusSize];
    int firstSector = firstSectorIndexOfCluster(clusIndex, fat32);
    fstream diskStream(drive1, std::ios::in);

    diskStream.seekg(firstSector * fat32.bytePerSector, SEEK_SET);
    diskStream.read(a, clusSize);
    BYTE* readBytes = new BYTE[clusSize];
    readBytes = (BYTE*)a;

    //đọc từng entry
    DIRECTORY* dirHeadNode, * dirTempNode, * dirTailNode;
    dirHeadNode = dirTempNode = dirTailNode = NULL;

    //holder to store longer named
    int byteHolder[13];
    int name[256];
    int holderIndex = -1;
    int nameIndex = -1;
```

(Hình 2.a.3: phần đầu hàm readDirectory)

Đầu tiên ta khai báo các biến cần thiết như số entry, kích thước của cluster (tính theo byte) với kích thước của cluster là kích thước của mảng các phần tử (khai báo ở mảng động ***a*** và ***readBytes***) cần được đọc vào.

Sau đó ta sẽ tìm kiếm từ sector đầu tiên của tham số ***clusIndex*** truyền vào của hàm trong ổ đĩa bằng hàm ***seekg( )*** và đọc thông tin này vào mảng ***a*** với số lượng bằng ***clusSize*** nhờ hàm ***read( )*** để có được một mảng các chỉ số hex của thư mục. Ép kiểu mảng ***a*** thành mảng động ***readBytes*** có kiểu dữ liệu ***BYTE*** (mục đích để đọc các phần tử unsigned char).

```
for (int i = firstEntryIndex; i < totalEntries; i++)
{
    //cuối bảng ghi
    if (readBytes[32 * i] == 0x0) break;

    //tập tin không được sử dụng do bị xóa
    if (readBytes[32 * i] == 0xE5) continue;

    int status = hexToDecimal(32 * i + readBytes , 11, 1);

    //LONG FILE NAME ENTRY
    if (status == 0xF)
    {
        //Code để lấy được Long file name, đọc lần lượt 2 Bytes để lấy cái kí tự
        //1 -- 10
        for (int j = 1; j < 10; j += 2)
        {
            if (hexToDecimal(32 * i + readBytes, j, 2) == 0xffff) break; //kết thúc file name
            byteHolder[++holderIndex] = hexToDecimal(32 * i + readBytes, j, 2);
        }
        //14 -- 25
        for (int j = 14; j < 25; j += 2)
        {
            if (hexToDecimal(32 * i + readBytes, j, 2) == 0xffff) break; //kết thúc file name
            byteHolder[++holderIndex] = hexToDecimal(32 * i + readBytes, j, 2);
        }
        //28 -- 31
        for (int j = 28; j < 31; j += 2)
        {
            if (hexToDecimal(32 * i + readBytes, j, 2) == 0xffff) break; //kết thúc file name
            byteHolder[++holderIndex] = hexToDecimal(32 * i + readBytes, j, 2);
        }
        //POP VÀ PUSH VÀO LONG NAME STACK
        while (holderIndex != -1)
            name[++nameIndex] = byteHolder[holderIndex--];
        continue; //Bởi vì Long file name entry được xác định nên bỏ qua
    }
    //Chỉ có các tập tin và thư mục được xét
    //và các tập tin liên quan đến hệ thống được bỏ qua dựa vào Attr
```

Tiếp đến là ta sẽ duyệt bằng vòng lặp được trỏ đến từng byte để xét các thông tin của thư mục.

Khi đọc từng entry, ta cần kiểm tra xem nó là entry chính, entry phụ, đã bị xóa hay không có

```
if (status == 0xF)
{
```

=> Entry phụ

```
//tập tin không được sử dụng do bị xóa
if (readBytes[32 * i] == 0xE5) continue;
```

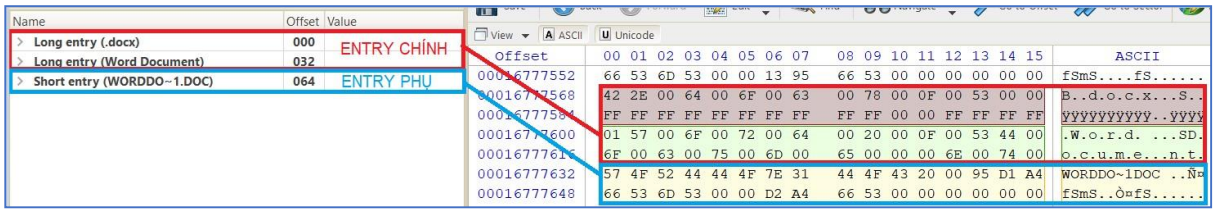
=> Entry bị xóa

```
//cuối bảng ghi
if (readBytes[32 * i] == 0x0) break;
```

=> Không có entry nào hết

Còn lại là entry chính

Mà trong RDET có 2 loại sector: entry chính và entry phụ.



Cứ có n entry phụ thì sẽ có 1 entry chính của các entry đó.

Nhiệm vụ của entry chính là lưu giữ các thông tin tên, thời gian, size, tình trạng của file, … Còn nhiệm vụ của entry phụ là để lưu giữ tên của file nếu như tên file dài quá mức giữ được của entry chính.

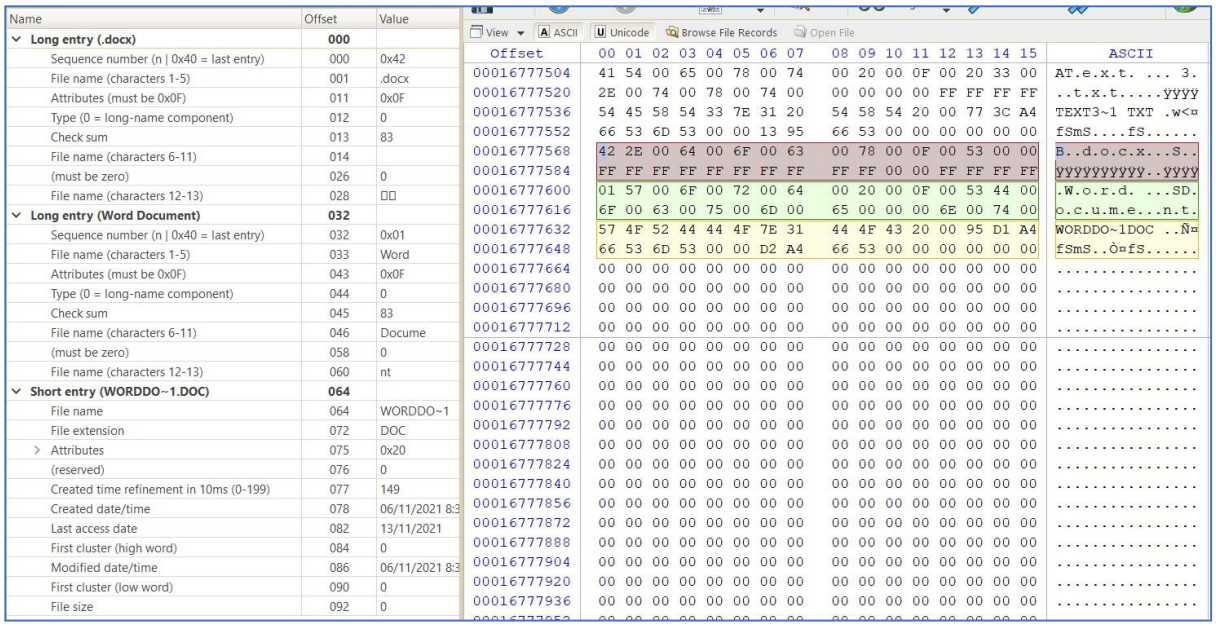
Ta có tính chất của 1 file sẽ gồm 1 entry chính và 0 hoặc nhiều entry phụ Mà nó đọc entry phụ trước rồi mới đọc tới entry chính.

Mỗi entry chiếm 32 byte trong sector với cấu trúc khác nhau tùy vào loại chính hay phụ, ta có thể tìm hiểu trong FAT.

```
for (int i = firstEntryIndex; i < totalEntries; i++)
{
    //cuối bảng ghi
    if (readBytes[32 * i] == 0x0) break;

    //tập tin không được sử dụng do bị xóa
    if (readBytes[32 * i] == 0xE5) continue;

    int status = hexToDecimal(32 * i + readBytes , 11, 1);

    //LONG FILE NAME ENTRY
    if (status == 0xF)
    {
        //Code để lấy được Long file name, đọc lần lượt 2 Bytes để lấy cái kí tự
        //1 -- 10
        for (int j = 1; j < 10; j += 2)
        {
            if (hexToDecimal(32 * i + readBytes, j, 2) == 0xffff) break; //kết thúc file name
            byteHolder[++holderIndex] = hexToDecimal(32 * i + readBytes, j, 2);
        }
        //14 -- 25
        for (int j = 14; j < 25; j += 2)
        {
            if (hexToDecimal(32 * i + readBytes, j, 2) == 0xffff) break; //kết thúc file name
            byteHolder[++holderIndex] = hexToDecimal(32 * i + readBytes, j, 2);
        }
        //28 -- 31
        for (int j = 28; j < 31; j += 2)
        {
            if (hexToDecimal(32 * i + readBytes, j, 2) == 0xffff) break; //kết thúc file name
            byteHolder[++holderIndex] = hexToDecimal(32 * i + readBytes, j, 2);
        }
        //POP VÀ PUSH VÀO LONG NAME STACK
        while (holderIndex != -1)
            name[++nameIndex] = byteHolder[holderIndex--];
        continue; //Bởi vì Long file name entry được xác định nên bỏ qua
    }
    //Chỉ có các tập tin và thư mục được xét
    //và các tập tin liên quan đến hệ thống được bỏ qua dựa vào Attr
```

Bằng cách đó em xây dựng các vòng lặp for để đọc từng kí tự của ***Long file name*** và Push vào Stack như đoạn code bên trên mô tả.

Đọc thông tin của 1 tập tin có tên dài sử dụng entry chính và entry phụ



Name	Offset	Value
Long entry (.docx)	**000**	
Sequence number (n | 0x40 = last entry)	000	0x42
File name (characters 1-5)	001	.docx
Attributes (must be 0x0F)	011	0x0F
Type (0 = long-name component)	012	0
Check sum	013	83
File name (characters 6-11)	014	
(must be zero)	026	0
File name (characters 12-13)	028	□□
Long entry (Word Document)	**032**	
Sequence number (n | 0x40 = last entry)	032	0x01
File name (characters 1-5)	033	Word
Attributes (must be 0x0F)	043	0x0F
Type (0 = long-name component)	044	0
Check sum	045	83
File name (characters 6-11)	046	Docume
(must be zero)	058	0
File name (characters 12-13)	060	nt
Short entry (WORDDO~1.DOC)	**064**	
File name	064	WORDDO~1
File extension	072	DOC
Attributes	075	0x20
(reserved)	076	0
Created time refinement in 10ms (0-199)	077	149
Created date/time	078	06/11/2021 8:3
Last access date	082	13/11/2021
First cluster (high word)	084	0
Modified date/time	086	06/11/2021 8:3
First cluster (low word)	090	0
File size	092	0

View ▾ | ASCII | Unicode | Browse File Records | Open File

Offset	00 01 02 03 04 05 06 07	08 09 10 11 12 13 14 15	ASCII
00016777504	41 54 00 65 00 78 00 74	00 20 00 0F 00 20 33 00	AT.e.x.t. ... 3.
00016777520	2E 00 74 00 78 00 74 00	00 00 00 00 FF FF FF FF	..t.x.t.....ÿÿÿÿ
00016777536	54 45 58 54 33 7E 31 20	54 58 54 20 00 77 3C A4	TEXT3~1 TXT .w<¤
00016777552	66 53 6D 53 00 00 13 95	66 53 00 00 00 00 00 00	fSmS....fS.....
00016777568	42 2E 00 64 00 6F 00 63	00 78 00 0F 00 53 00 00	B..d.o.c.x...S..
00016777584	FF FF FF FF FF FF FF FF	FF FF 00 00 FF FF FF FF	ÿÿÿÿÿÿÿÿÿÿ..ÿÿÿÿ
00016777600	01 57 00 6F 00 72 00 64	00 20 00 0F 00 53 44 00	.W.o.r.d. ...SD.
00016777616	6F 00 63 00 75 00 6D 00	65 00 00 00 6E 00 74 00	o.c.u.m.e...n.t.
00016777632	57 4F 52 44 44 4F 7E 31	44 4F 43 20 00 95 D1 A4	WORDDO~1DOC ..Ñ¤
00016777648	66 53 6D 53 00 00 D2 A4	66 53 00 00 00 00 00 00	fSmS..Ò¤fS.....
00016777664	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777680	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777696	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777712	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777728	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777744	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777760	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777776	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777792	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777808	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777824	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777840	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777856	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777872	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777888	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777904	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777920	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
00016777936	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....

Kết quả:

```
FileName: Word Document.docx
FileSize: 0
StartCluster: 0
FileAttr: 32(0x20,FILE)
Dung phan mem tuong thich de doc noi dung!
```

Tiếp đến ta xét trạng thái (1 byte tại 0B) của 1 thư mục/ tập tin và thực hiện hiển thị thông tin của nó lên màn hình console.

Trước tiên, ta khai báo con trỏ dirTempNode – đây được xem là con trỏ tượng trưng cho cây thư mục với kiểu dữ liệu là DIRECTORY. Thực hiện khởi tạo các giá trị của thuộc tính của thư mục/ tập tin này nhờ mảng ***readBytes.***

```
if (status == 0x10 || status == 0x20)
{
    dirTempNode = new DIRECTORY;
    dirTempNode->next = dirTempNode->dir = NULL;

    if (dirHeadNode == NULL)
        dirHeadNode = dirTempNode;
    else
        dirTailNode->next = dirTempNode;

    dirTailNode = dirTempNode;

    int count = 0; //sử dụng để đếm các ký tự
    for (int j = 0; j < 11; j++)
        dirTempNode->Name[count++] = readBytes[32 * i + j];

    if (nameIndex != -1)
    {
        count = 0;
        for (int j = nameIndex; j > -1; j--)
            dirTempNode->Name[count++] = (char)name[j];

        holderIndex = nameIndex = -1;
    }
    dirTempNode->Name[count] = '\0';
    dirTempNode->Attr = status;
    dirTempNode->FileSize = hexToDecimal(32 * i + readBytes, 28, 4);
    dirTempNode->StartCluster = (hexToDecimal(32 * i + readBytes, 20, 2)
        * pow(2, 16)) + hexToDecimal(32 * i + readBytes, 26, 2);


    //to read directories in directories

    cout << space << "FileName: " << dirTempNode->Name << endl;
    cout << space << "FileSize: " << dirTempNode->FileSize << endl;
    cout << space << "StartCluster: " << dirTempNode->StartCluster << endl;
    cout << space << "FileAttr: " << dirTempNode->Attr << (dirTempNode->Attr == 0x10 ? "(0x10,DIR)" : "(0x20,FILE)") << endl;
```

Sau khi biết được các thông tin cần thiết của một thư mục/ tập tin. Ta thực hiện in cây thư mục như đoạn code bên dưới.

```
        if (status == 0x10)
        {
            space = space + "     ";
            cout << endl;
            dirTempNode->dir = readDirectory( 2, dirTempNode->StartCluster, FAT, fat32, drive1,space); //2 to skip . and .. dir entries
            space = space.substr(0, space.length() - 5);
        }
        else if (status == 0x20)
        {
            int len = strlen(dirTempNode->Name) - 3;
            char string[4];
            strcpy(string, dirTempNode->Name + len);

            if (strcmp(string, "txt") == 0 || strcmp(string, "TXT") == 0)
            {
                cout << space << "Noi dung file TXT: " << endl;
                readContentOfFile(fat32,  dirTempNode->StartCluster, drive1, space);
                cout << endl;
            }
            else
                cout << space << "Dung phan mem tuong thich de doc noi dung!" << endl << endl;
        }

        if (FAT[clusIndex] != 0x0fffffff)
            dirTailNode->next = readDirectory( firstEntryIndex, FAT[clusIndex], FAT, fat32, drive1, space);
    }
}

return dirHeadNode;
}
```

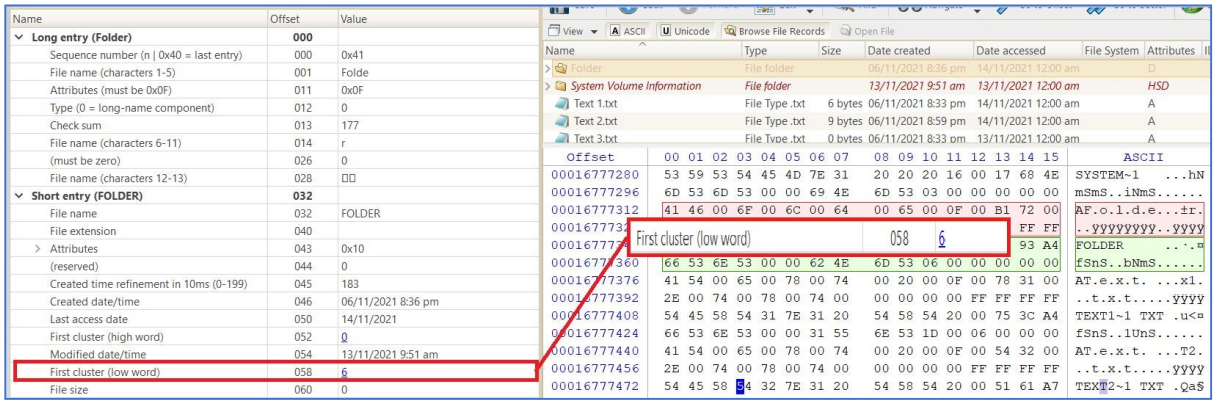
Nếu trạng thái hiện tại đọc được tại 0B là *0x10* thì đây là thư mục và cần phải hiển thị cây thư mục con bên trong nên ta sẽ gọi hàm readDirectory( ) với tham số clusterIndex là cluster bắt đầu của thư mục hiện tại.

Còn ngược lại, trạng thái đọc được tại 0B là *0x20* thì đây là tập tin và ta thực hiện xem xét phần mở rộng của tập tin này có phải là "txt" (hoặc "TXT") hay không? Nếu không phải đuôi "txt" thì ta phải dùng phần mềm tương thích để đọc nội dung. Ngược lại ta sẽ hiển thị nội dung của tập tin "txt" qua hàm ***readContentOfFile( ).***

```
void readContentOfFile(FAT32 fat32, int clusIndex, LPCWSTR drive1, string space) {
    int content_sector = firstSectorIndexOfCluster(clusIndex, fat32);
    int clusterSizeInBytes = fat32.sectorPerCluster * fat32.bytePerSector;

    char* a = new char[clusterSizeInBytes];
    fstream G(drive1, std::ios::in);
    G.seekg(content_sector * fat32.bytePerSector, 0);
    G.read(a, clusterSizeInBytes);

    char* content = new char[clusterSizeInBytes];
    int i = 0;
    while (a[i] != '\x00')
    {
        content[i] = a[i];
        i++;
        if (i == 4095) break;
    }
    content[i] = '\0';

    cout << space << content << endl;
}
```

Dùng phần mềm Disk Editor để ta có thể thấy trực quan về các thông số.

Đọc thông tin tập tin hoặc các thư mục con của thư mục gốc ở đây ta có cluster dẫn tới vị trí đọc file của thư mục FOLDER là 6.



Biết 1 cluster có 16 sector:

Name	Offset	Value	Copy Value
JMP instruction	000	EB 58 90	45 56 49
OEM ID	003	MSDOS5.0	L JPG
BIOS Parameter Block	**011**		
Bytes per sector	011	512	6,176
Sectors per cluster	013	16	132
Reserved sectors	014	3,238	19,910
Number of FATs	016	2	109

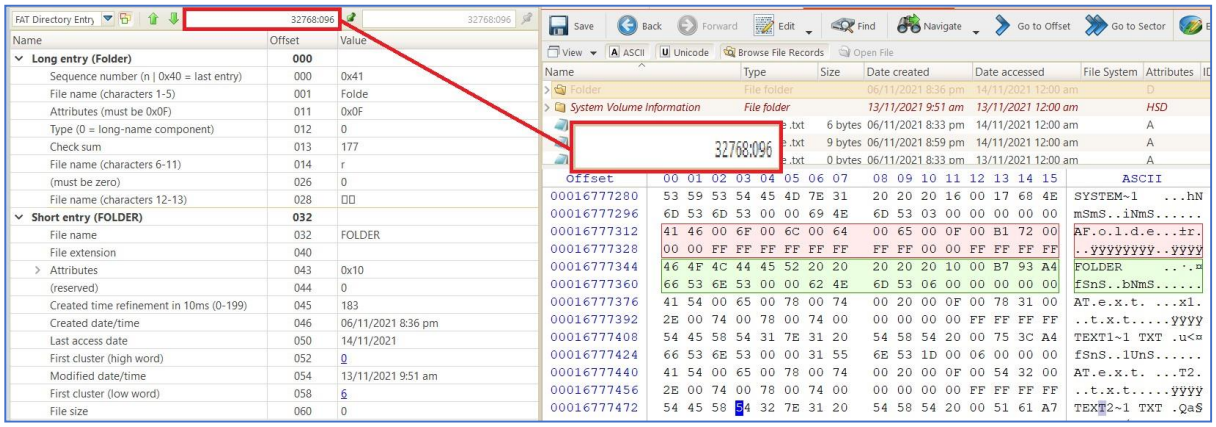
Ta có công thức:

Tính sector bắt đầu khi biết giá trị cluster

Nếu biết giá trị cluster (kí hiệu: N) bất kì, có thể tính được sector bắt đầu (sector tương đối) của cluster đó bằng công thức sau:

FirstSectorOfCluster = ((N – 2) * SecPerClus) + FirstDataSector

- FirstSectorOfCluster: sector bắt đầu của cluster
- SecPerClus: số sector/cluster
- FirstDataSector: sector đầu tiên của cluster số 2 trong vùng Data



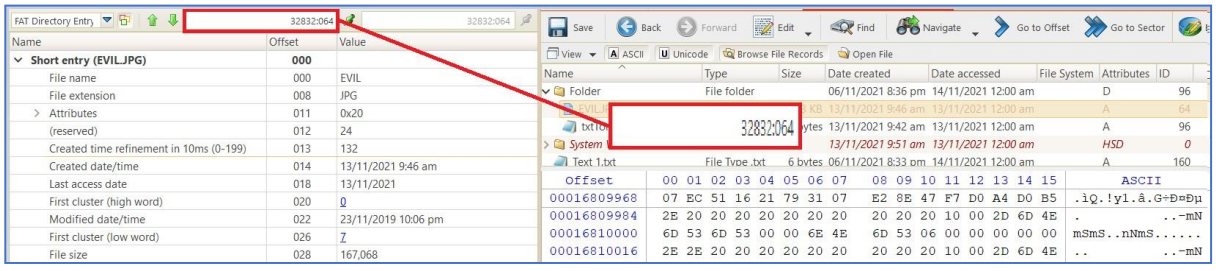
Ta có:

First data sector =

32768 N = 6

SectorPerCluster = 16

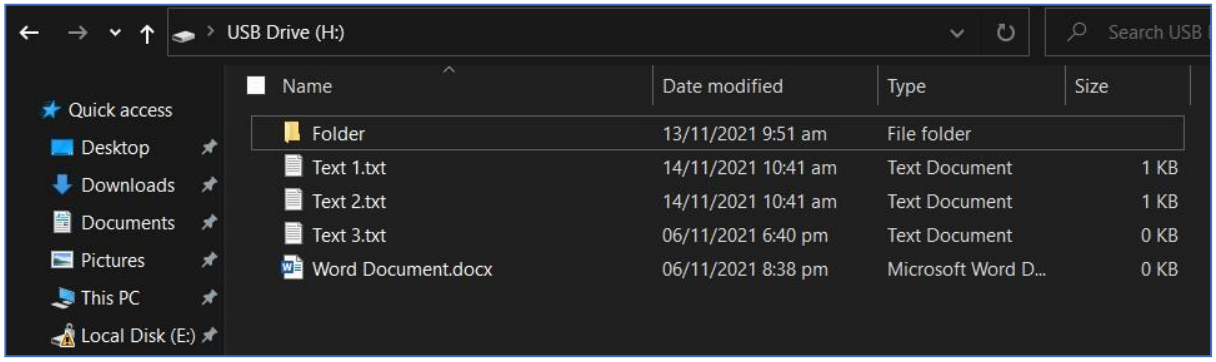
=> 16 * (6 - 2) + 32768 = 32832 là sector đầu tiên chứa thông tin của file/folder đó



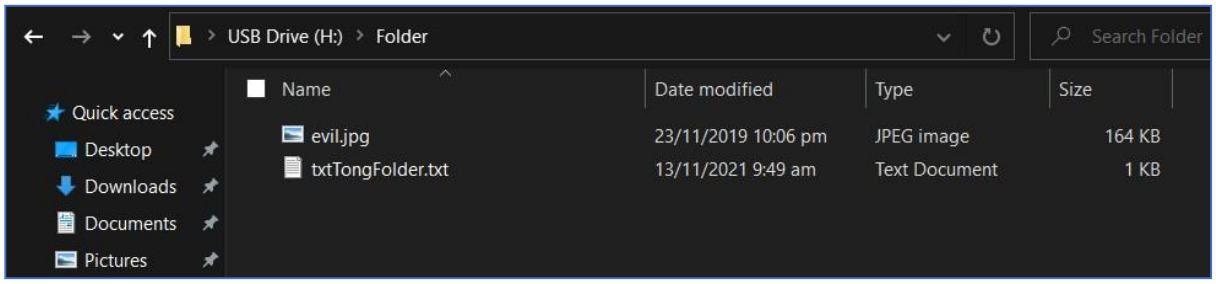
Ta có vị trí sector đầu của file/folder và có 2 cách tìm thông tin. Vì nếu là file thì nó sẽ chạy tới nơi có thông tin. Còn folder thì sẽ chạy ra thông tin các file/ folder con khác nằm trong nó và tương tự như trên.

b. Demo chương trình với các trường yêu cầu:

Cây thư mục gốc:



Bên trong thư mục Folder:



Kết quả chương trình:

```
==========  CAY THU MUC  ===========

FileName: Folder
FileSize: 0
StartCluster: 6
FileAttr: 16(0x10,DIR)

     FileName: EVIL     JPG
     FileSize: 167068
     StartCluster: 7
     FileAttr: 32(0x20,FILE)
     Dung phan mem tuong thich de doc noi dung!

     FileName: txtTongFolder.txt
     FileSize: 28
     StartCluster: 28
     FileAttr: 32(0x20,FILE)
     Noi dung file TXT:
     Day la file txt trong Folder

FileName: Text 1.txt
FileSize: 6
StartCluster: 29
FileAttr: 32(0x20,FILE)
Noi dung file TXT:
abcdef
```

```
FileName: Text 2.txt
FileSize: 9
StartCluster: 30
FileAttr: 32(0x20,FILE)
Noi dung file TXT:
123456789

FileName: Text 3.txt
FileSize: 0
StartCluster: 0
FileAttr: 32(0x20,FILE)
Noi dung file TXT:


FileName: Word Document.docx
FileSize: 0
StartCluster: 0
FileAttr: 32(0x20,FILE)
Dung phan mem tuong thich de doc noi dung!
```

Đọc thông tin và nội dung các file có phần mở rộng là “txt”:

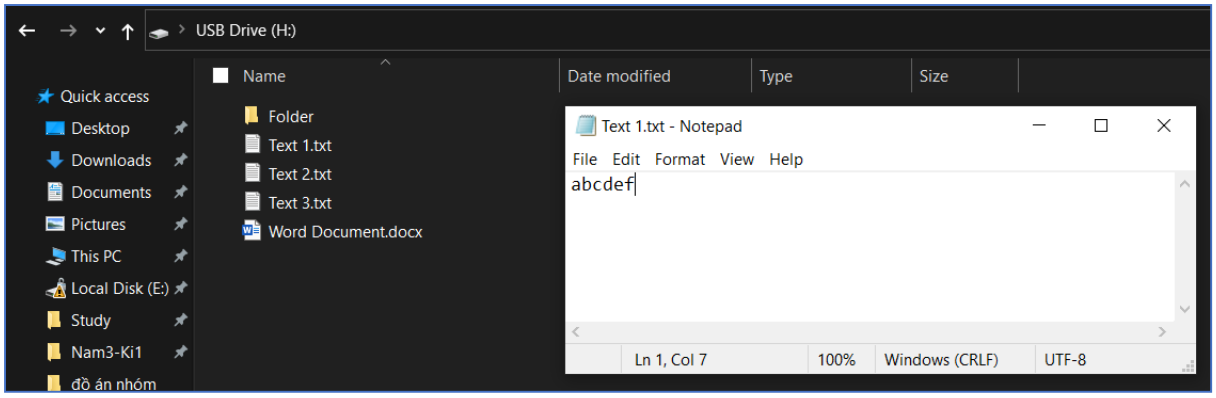
```
FileName: Text 2.txt
FileSize: 9
StartCluster: 30
FileAttr: 32(0x20,FILE)
Noi dung file TXT:
123456789
```

```
FileName: Text 1.txt
FileSize: 6
StartCluster: 29
FileAttr: 32(0x20,FILE)
Noi dung file TXT:
abcdef
```

```
    FileName: txtTongFolder.txt
    FileSize: 28
    StartCluster: 28
    FileAttr: 32(0x20,FILE)
    Noi dung file TXT:
    Day la file txt trong Folder
```

Minh chứng cho 1 file txt đã đọc:

```
FileName: Text 1.txt
FileSize: 6
StartCluster: 29
FileAttr: 32(0x20,FILE)
Noi dung file TXT:
abcdef
```



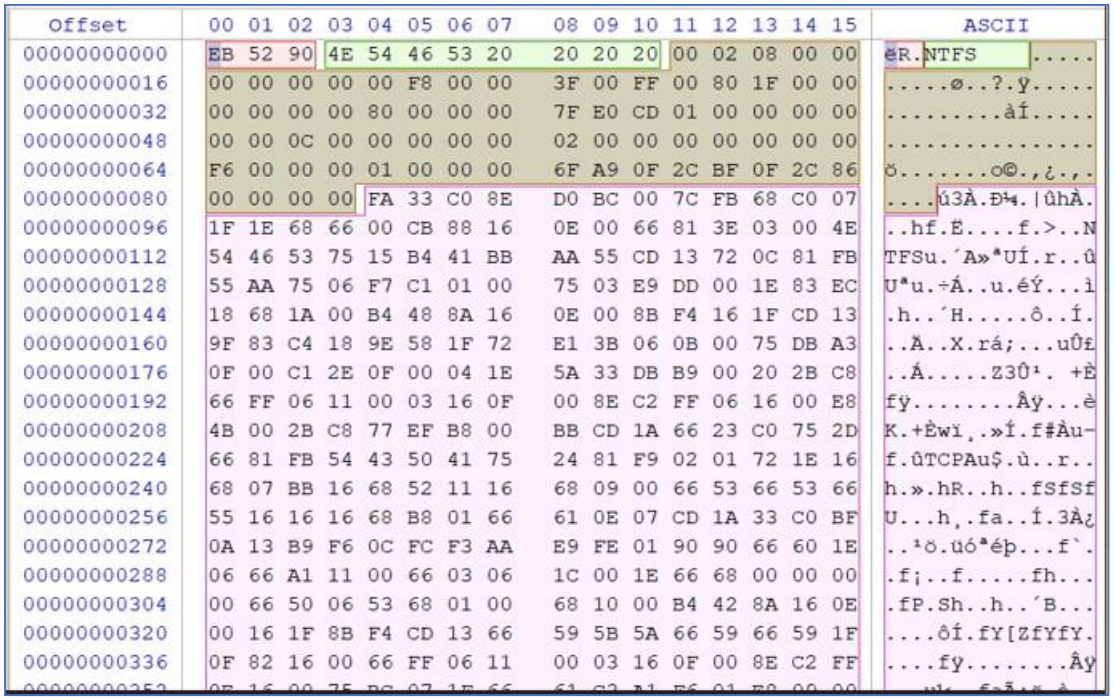
3. Đọc thông tin chi tiết phân vùng NTFS:

a. Mô tả BPB của NTFS:

Địa chỉ (offset)	Kích thước (byte)	Mô tả
0Bh	2	Kích thước một sector. Đơn vị tính là byte.
0Dh	1	Số sector trong một cluster.
0Eh	2	Chưa sử dụng.
10h	1	Với hệ thống NTFS luôn mang giá trị 0.
11h	2	Với hệ thống NTFS luôn mang giá trị 0.
13h	2	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
15h	1	Mã xác định loại đĩa.
16h	2	Với hệ thống NTFS luôn mang giá trị 0.
18h	2	Số sector/track.
1Ah	2	Số mặt đĩa (head hay side).
1Ch	4	Sector bắt đầu của ổ đĩa logic.
20h	4	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
24h	4	Hệ thống NTFS luôn thiết lập giá trị này là "80008000".
28h	8	Số sector của ổ đĩa logic.
30h	8	Cluster bắt đầu của MFT.
38h	8	Cluster bắt đầu của MFT dự phòng (MFTMirror).
40h	1	Kích thước của một bản ghi trong MFT (MFT entry), đơn vị tính là byte.
41h	3	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
44h	1	Số cluster của Index Buffer.
45h	3	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
48h	8	Số seri của ổ đĩa (volume serial number).
50h	4	Không được sử dụng bởi NTFS.

Các thông số quan trọng của NTFS mà ta cần lưu ý:

- Số byte của một Sector.
- Số Sector của một Cluster:
- Số Sector/track.
- Số mặt đĩa.
- Sector bắt đầu của ổ đĩa logic.
- Số Sector của ổ đĩa logic.
- Cluster bắt đầu của MFT.
- Cluster bắt đầu của MFT dự phòng.
- Kích thước của 1 bản ghi trong MFT Entry.

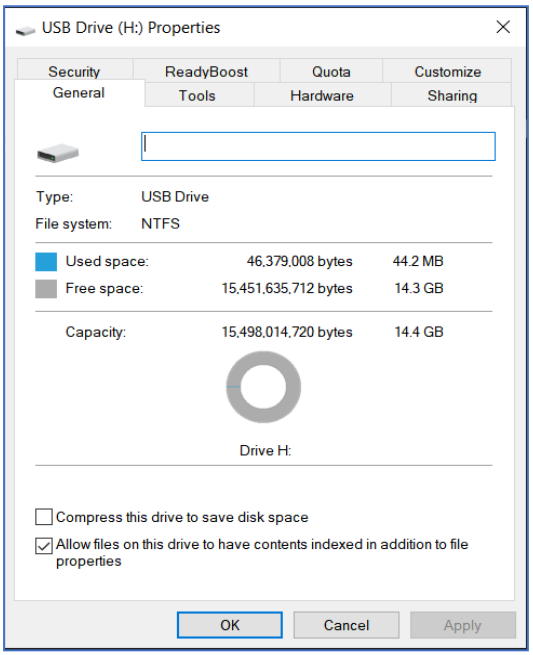


Offset	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	ASCII
00000000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00	ëR.NTFS .....
00000000016	00	00	00	00	00	F8	00	00	3F	00	FF	00	80	1F	00	00	.....ø..?.ÿ.....
00000000032	00	00	00	00	80	00	00	00	7F	E0	CD	01	00	00	00	00	.....àÍ.....
00000000048	00	00	0C	00	00	00	00	00	02	00	00	00	00	00	00	00	.....
00000000064	F6	00	00	00	01	00	00	00	6F	A9	0F	2C	BF	0F	2C	86	ö.....o©.,¿.,.
00000000080	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07	....ú3À.Ð¼.|ûhÀ.
00000000096	1F	1E	68	66	00	CB	88	16	0E	00	66	81	3E	03	00	4E	..hf.Ë....f.>..N
00000000112	54	46	53	75	15	B4	41	BB	AA	55	CD	13	72	0C	81	FB	TFSu.´A»ªUÍ.r..û
00000000128	55	AA	75	06	F7	C1	01	00	75	03	E9	DD	00	1E	83	EC	Uªu.÷Á..u.éÝ...ì
00000000144	18	68	1A	00	B4	48	8A	16	0E	00	8B	F4	16	1F	CD	13	.h..´H.....ô..Í.
00000000160	9F	83	C4	18	9E	58	1F	72	E1	3B	06	0B	00	75	DB	A3	..Ä..X.rá;...uÛ£
00000000176	0F	00	C1	2E	0F	00	04	1E	5A	33	DB	B9	00	20	2B	C8	..Á.....Z3Û¹. +È
00000000192	66	FF	06	11	00	03	16	0F	00	8E	C2	FF	06	16	00	E8	fÿ.........Âÿ...è
00000000208	4B	00	2B	C8	77	EF	B8	00	BB	CD	1A	66	23	C0	75	2D	K.+Èwï¸.»Í.f#Àu-
00000000224	66	81	FB	54	43	50	41	75	24	81	F9	02	01	72	1E	16	f.ûTCPAu$.ù..r..
00000000240	68	07	BB	16	68	52	11	16	68	09	00	66	53	66	53	66	h.».hR..h..fSfSf
00000000256	55	16	16	16	68	B8	01	66	61	0E	07	CD	1A	33	C0	BF	U...h¸.fa..Í.3À¿
00000000272	0A	13	B9	F6	0C	FC	F3	AA	E9	FE	01	90	90	66	60	1E	..¹ö.üó ªéþ...f`.
00000000288	06	66	A1	11	00	66	03	06	1C	00	1E	66	68	00	00	00	.f¡..f.....fh...
00000000304	00	66	50	06	53	68	01	00	68	10	00	B4	42	8A	16	0E	.fP.Sh..h..´B...
00000000320	00	16	1F	8B	F4	CD	13	66	59	5B	5A	66	59	66	59	1F	....ôÍ.fY[ZfYfY.
00000000336	0F	82	16	00	66	FF	06	11	00	03	16	0F	00	8E	C2	FF	....fÿ.........Âÿ

Thông tin để đọc NTFS vẫn nằm ở sector đầu tiên và có dấu hiệu nhận biết ở offset 03 -> 10 với thông tin đọc được là “NTFS” để ta có thể phân biệt nó so với các FAT khác.

b. Môi trường thử nghiệm:

Đối với NTFS, tụi em sử dụng USB tương tự như FAT32 để chạy mã nguồn và đánh giá kết quả. Ổ đĩa tương ứng trên windows là ổ đĩa H. Phân vùng được format thành định dạng NTFS. Thông tin về phân vùng đó như sau:



c. Quy trình thực hiện:

Tương tự như FAT32, tụi em cũng có sử dụng struct ***NTFS*** để lưu các thông tin quan trọng được nhắc đến bên trên của NTFS như sau:

```
struct NTFS {
    int bytesPerSec;           //  Số byte của một Sector          (B - 2)
    int sectorPerClus;         //  Số Sector của một Cluster       (D - 1) SC
    int numSecPerTrack;        //  Số sector/track                 (18 - 2)
    int numSide;               //  Số mặt đĩa (head hay side)      (1A - 2)
    int startSecDisk;          //  Sector bắt đầu của ổ đĩa logic  (1C - 4)
    int totalSector;           //  Số Sector của ổ đĩa logic       (28 - 8)
    int startClusterMFT;       //  Cluster bắt đầu của MFT         (30 - 8)
    int startClusterMFTMirror; //  Cluster bắt đầu của MFT dự phòng(38 - 8)
    int sizeOfMFTEntry;        //  Kích thước của 1 bản ghi trong MFT Entry (40 - 1) đơn vị tính là Byte
};
```

Để đọc NTFS (như FAT32), chúng em sử dụng lại các hàm đã cài đặt phía trên như:

- ***hexToDecimal( )*** để đọc trên một sector từ vị trí offset với một số byte là count.
- Quan trọng nhất vẫn là hàm ***readSector( )*** giúp chúng em đọc một sector (512byte) trên phân vùng từ vị trí readPoint.

Sau đó, dựa vào bảng BPB của NTFS để có được thông tin về offset và kích thước (byte) cần đọc, ta định nghĩa hàm ***initPartitionBootsector()*** để đọc các thông tin vào một biến có kiểu là NTFS.

```
void initPartitionBootsector(BYTE* sector, NTFS& _ntfs) {
    _ntfs.bytesPerSec = hexToDecimal(sector, 11, 2);
    _ntfs.sectorPerClus = hexToDecimal(sector, 13, 1);
    _ntfs.numSecPerTrack = hexToDecimal(sector, 24, 2);
    _ntfs.numSide = hexToDecimal(sector, 26, 2);
    _ntfs.startSecDisk = hexToDecimal(sector, 28, 4);
    _ntfs.totalSector = hexToDecimal(sector, 40, 8);
    _ntfs.startClusterMFT = hexToDecimal(sector, 48, 8);
    _ntfs.startClusterMFTMirror = hexToDecimal(sector, 56, 8);
    _ntfs.sizeOfMFTEntry = sizeOfMFTEntry(sector, 64);
}
```

Hàm cuối cùng mà chúng em cần trong phần đọc thông tin Partition Boot Sector của NTFS là hàm ***readInformationPartitionBootSector( ).*** Hàm này nhận vào 1 biến kiểu NTFS đã đọc được thông tin Partition Boot Sector thông qua hàm ***initPartitionBootsector( )*** để hiển thị thông tin lên màn hình console.

```
void readInformationPartitionBootSector(NTFS _ntfs)
{
    cout << "So byte cua mot Sector: " << _ntfs.bytesPerSec << endl;
    cout << "So Sector cua mot Cluster: " << _ntfs.sectorPerClus << endl;
    cout << "So sector/track: " << _ntfs.numSecPerTrack << endl;
    cout << "So mat dia (head hay side): " << _ntfs.numSide << endl;
    cout << "Sector bat dau cua o dia logic: " << _ntfs.startSecDisk << endl;
    cout << "So Sector cua o dia logic: " << _ntfs.totalSector << endl;
    cout << "Cluster bat dau cua MFT: " << _ntfs.startClusterMFT << endl;
    cout << "Cluster bat dau cua MFT du phong: " << _ntfs.startClusterMFTMirror << endl;
    cout << "Kich thuoc cua 1 ban ghi trong MFT Entry: " << _ntfs.sizeOfMFTEntry << endl;
}
```

Xong, ở hàm main để đọc và hiển thị thông tin Partition Boot Sector NTFS ta làm như sau:

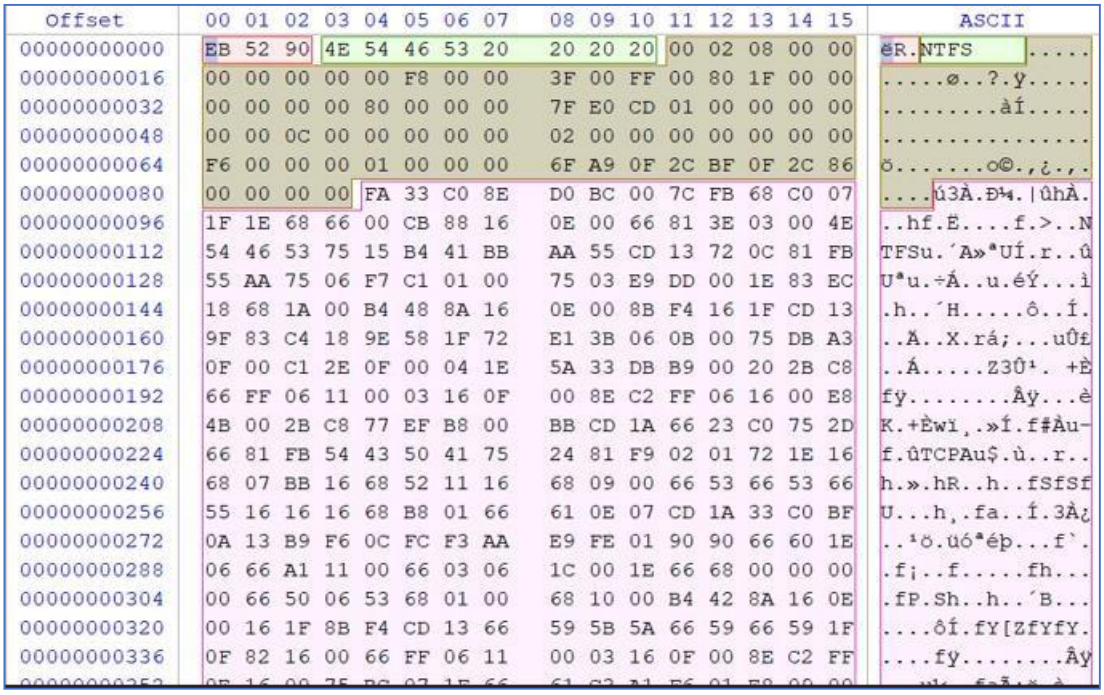
```
    BYTE sector[512];
    string a = "\\\\.\\" + nameDisk + ":";
    CString str(a.c_str());
    CStringW strw(str);
    LPCWSTR drive2 = strw;

    readSector(drive2, 0, sector);
    initPartitionBootsector(sector, _ntfs);

    //Đọc các thông tin được mô tả trong Partition Boot Sector
    cout << "++++++++++++++++++    THONG TIN PARTITION BOOT SECTOR     ++++++++++++++++++\n" << endl;
    readInformationPartitionBootSector(_ntfs);
```

d. Kiểm tra kết quả:

Sau khi chạy chương trình, ta được kết quả như sau:

```
++++++++++++++++++    THONG TIN PARTITION BOOT SECTOR     ++++++++++++++++++

So byte cua mot Sector: 512
So Sector cua mot Cluster: 8
So sector/track: 63
So mat dia (head hay side): 255
Sector bat dau cua o dia logic: 8064
So Sector cua o dia logic: 30269567
Cluster bat dau cua MFT: 786432
Cluster bat dau cua MFT du phong: 2
Kich thuoc cua 1 ban ghi trong MFT Entry: 1024
```

Dữ liệu phần đầu Partiton Bootsector của phân vùng được hiển thị trong Disk Editor như sau:



Offset	00 01 02 03 04 05 06 07	08 09 10 11 12 13 14 15	ASCII
00000000000	EB 52 90 4E 54 46 53 20	20 20 20 00 02 08 00 00	ëR.NTFS .....
00000000016	00 00 00 00 00 F8 00 00	3F 00 FF 00 80 1F 00 00	.....ø..?.ÿ.....
00000000032	00 00 00 00 80 00 00 00	7F E0 CD 01 00 00 00 00	.....àÍ.....
00000000048	00 00 0C 00 00 00 00 00	02 00 00 00 00 00 00 00	.....
00000000064	F6 00 00 00 01 00 00 00	6F A9 0F 2C BF 0F 2C 86	ö.....o©.,¿.,.
00000000080	00 00 00 00 FA 33 C0 8E	D0 BC 00 7C FB 68 C0 07	....ú3À.Ð¼.|ûhÀ.
00000000096	1F 1E 68 66 00 CB 88 16	0E 00 66 81 3E 03 00 4E	..hf.Ë....f.>..N
00000000112	54 46 53 75 15 B4 41 BB	AA 55 CD 13 72 0C 81 FB	TFSu.´A»ªUÍ.r..û
00000000128	55 AA 75 06 F7 C1 01 00	75 03 E9 DD 00 1E 83 EC	Uªu.÷Á..u.éÝ...ì
00000000144	18 68 1A 00 B4 48 8A 16	0E 00 8B F4 16 1F CD 13	.h..´H.....ô..Í.
00000000160	9F 83 C4 18 9E 58 1F 72	E1 3B 06 0B 00 75 DB A3	..Ä..X.rá;...uÛ£
00000000176	0F 00 C1 2E 0F 00 04 1E	5A 33 DB B9 00 20 2B C8	..Á.....Z3Û¹. +È
00000000192	66 FF 06 11 00 03 16 0F	00 8E C2 FF 06 16 00 E8	fÿ.....Âÿ...è
00000000208	4B 00 2B C8 77 EF B8 00	BB CD 1A 66 23 C0 75 2D	K.+Èwï¸.»Í.f#Àu-
00000000224	66 81 FB 54 43 50 41 75	24 81 F9 02 01 72 1E 16	f.ûTCPAu$.ù..r..
00000000240	68 07 BB 16 68 52 11 16	68 09 00 66 53 66 53 66	h.».hR..h..fSfSf
00000000256	55 16 16 16 68 B8 01 66	61 0E 07 CD 1A 33 C0 BF	U...h¸.fa..Í.3À¿
00000000272	0A 13 B9 F6 0C FC F3 AA	E9 FE 01 90 90 66 60 1E	..¹ö.üó ªéþ...f`.
00000000288	06 66 A1 11 00 66 03 06	1C 00 1E 66 68 00 00 00	.f¡..f.....fh...
00000000304	00 66 50 06 53 68 01 00	68 10 00 B4 42 8A 16 0E	.fP.Sh..h..´B...
00000000320	00 16 1F 8B F4 CD 13 66	59 5B 5A 66 59 66 59 1F	....ôÍ.fY[ZfYfY.
00000000336	0F 82 16 00 66 FF 06 11	00 03 16 0F 00 8E C2 FF	....fÿ.....Âÿ

Đối chiếu với thông tin phân vùng Disk Editor, ta khẳng định tính chính xác của chương trình:

Name	Offset	Value	Copy Value
JMP instruction	000	EB 52 90	EB 52 90
OEM ID	003	NTFS	NTFS
BIOS Parameter Block	011		
Bytes per sector	011	512	512
Sectors per cluster	013	8	8
Reserved sectors	014	0	0
(always zero)	016	00 00 00	00 00 00
(unused)	019	00 00	00 00
Media descriptor	021	248	248
(unused)	022	00 00	00 00
Sectors per track	024	63	63
Number of heads	026	255	255
Hidden sectors	028	8,064	8,064
(unused)	032	00 00 00 ...	00 00 00 00
Signature	036	80 00 00 ...	80 00 00 00
Total sectors	040	30,269,567	30,269,567
$MFT cluster number	048	786,432	786,432
$MFTMirr cluster nu...	056	2	2
Clusters per File Re...	064	246	246
Clusters per Index B...	068	1	1
Volume serial num...	072	6F A9 0F ...	6F A9 0F 2C BF 0F 2C 86
Checksum	080	0	0
Bootstrap code	084	FA 33 C0...	FA 33 C0 8E D0 BC 00 7C FB 68 C0 07 1F 1E 68 66 00 CB 88 16 0E 0...
Signature (55 AA)	510	55 AA	55 AA

4. Hiển thị cây thư mục phân vùng NTFS

Phần này tụi em chưa làm được ạ.

III. Video Demo và Tài liệu tham khảo:

1. Video Demo:

https://youtu.be/oHM9osR3yV8

2. Nguồn tham khảo từ tài liệu sách giáo khoa, giáo trình:

[1] Giáo trình Hệ Điều Hành – Khoa Công Nghệ Thông Tin – Trường Đại Học Khoa Học Tự Nhiên, ĐHQG TPHCM (2019).

[2] Tài liệu và video hướng dẫn của Thầy Lê Viết Long.

3. Nguồn tham khảo từ trên Internet:

[3] https://ntcore.com/?tag=fat32

[4]https://codeandlife.com/2012/04/02/simple-fat-and-sd-tutorial-part-1/?fbclid=IwAR0FRXKnDgVmVHbPeZ_IWWDcjzqXJfczOncAhjGDcUABEUQ_cp3LKf2OOB8

[5]https://github.com/shiva-prasad-reddy/Reading-and-writing-raw-data-to-a-FAT32-formatted-disk/blob/master/FAT32.c?fbclid=IwAR3GX7BkXfcNYnH2eQ87PYIs0plcN82F8g9dFGtEvkxO-RvpA5Q39fEX3_k