fit@hcmus

Lớp: 22CLC06

Nhóm: PTHT

BÁO CÁO ĐỒ ÁN

HỆ ĐIỀU HÀNH

Project: Quản lý hệ thông tập tin

Mục lục

Giới thiệu đồ án

Thành viên nhóm

MSSV	Họ và tên	Tỉ lệ đóng góp
22127121	Đào Việt Hoàng	25%
22127320	Bùi Tá Phát	25%
22127432	Đặng Thanh Tú	25%
22127486	Đào Ngọc Thiện	25%

Lời mở đầu

- Hệ thống tập tin là phương thức mà hệ điều hành sử dụng để quản lý và lưu trữ tập tin. Bằng cách tách dữ liệu thành từng mảnh và đặt tên cho chúng, các dữ liệu có thể dễ dành được tách riêng ra rồi truy cập. Một nhóm dữ liệu như vậy được gọi là một tập tin, và hệ thống, phương thức dùng để quản lý và truy cập những tập tin này được gọi là hệ thống tập tin.
- Hai hệ thống tập tin phổ biến nhất trong các loại máy tính sử dụng hệ điều hành Windows hiện nay là FAT32 (32-bit File Allocation Table) và NTFS (New Technology File System).

Giới thiệu sơ lược về các hệ thống tập tin

FAT32: 32-bit File Allocation Table

- FAT32 là hệ thống tập tin được phát triển bởi Microsoft, được sử dụng trên nhiều máy tính chạy hệ điều hành Windows.
- Tận dụng tối đa một bảng 32 bit để quản lý dữ liệu.
- Hỗ trợ nhiều loại máy tính và hệ điều hành, tuy nhiên bị giới hạn bởi kích cỡ tập tin và kích cỡ ổ đĩa.

NTFS: New Technology File System

- NTFS là hệ thống tập tin tân tiến hơn được phát triển bởi Microsoft để loại bỏ khuyết điểm của FAT32.
- NTFS hỗ trợ nhiều hơn về bảo mật, độ tin cậy và hiệu suất cao hơn khả năng của FAT32.
- NTFS phù hợp với các tập tin có kích cỡ lớn, tối ưu hóa khả năng nén, mã hóa thông tin.

Về đồ án

- Bài báo cáo này dựa trên quá trình mà nhóm đã trải qua khi viết và chạy chương trình đọc và phân tích hai hệ thống tập tin đó bằng cách áp dụng các kiến thức đã học trên lớp, cũng như tìm hiểu kỹ lưỡng các tài liệu trên Internet, và sử dụng các công cụ hỗ trợ viết chương trình Visual Studio Code, GitHub Desktop,…
- Chương trình này được viết hoàn toàn bằng ngôn ngữ C++ và được kiểm thử, sửa lỗi và cải tiến trên máy tính cá nhân, USB gồm hai ổ đĩa được lần lượt định dạng theo hệ thống tập tin FAT32 và NTFS, môi trường phát triển tích hợp Visual Studio Code chạy với quyền hạn admin.

Mô tả đồ án

Nguyên lý hoạt động

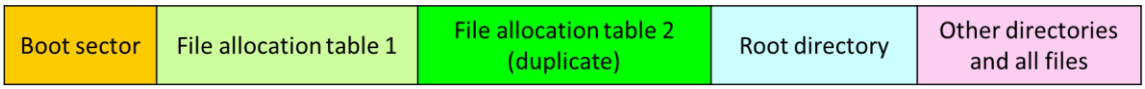
Little Endian

- Khi đọc các sector, ta sẽ có một bảng lưu các dữ liệu của sector theo offset.
- Các hệ thống tập tin của Windows lưu dữ liệu dưới dạng Little Endian, nghĩa là khi đọc các byte dữ liệu đó, ta phải đọc ngược các byte.
- Ví dụ: 4 bytes sau được lưu trữ theo kiểu little endian, do đó dữ liệu được đọc từ 4 bytes này là 0x00000BC4 = 3012.

```
C4 0B 00 00
```

FAT32

- Dữ liệu trong hệ thống tập tin FAT32 được tổ chức thành 2 vùng:
  - Vùng hệ thống gồm: Vùng BootSector, bảng FAT, bảng thư mục gốc (có thể nằm trên vùng dữ liệu).
  - Vùng dữ liệu:



- Bằng cách sử dụng hàm printSectorNum, in ra bảng sector (gồm 3 cột – cột offset, cột nội dung BYTE trong hệ thống lưu dưới dạng hex, cột các chữ có thể đọc được dưới dạng ASCII, hoặc '.' nếu không phải ASCII).

```
  Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000   46 49 4C 45 30 00 03 00 50 6F 4B 9A 13 00 00 00   FILE0...PoK.....
00000010   0A 01 02 00 38 00 01 00 D8 01 00 00 00 04 00 00   ....8...........
00000020   00 00 00 00 00 00 00 00 05 00 00 00 C4 0B 00 00   ................
00000030   45 00 00 00 00 00 00 00 10 00 00 00 60 00 00 00   E...........`...
00000040   00 00 00 00 00 00 00 00 48 00 00 00 18 00 00 00   ........H.......
00000050   C0 B2 BA 82 9B C8 D9 01 32 5C C1 62 AA CC D9 01   ........2\.b....
00000060   32 5C C1 62 AA CC D9 01 14 23 75 DE BC 17 DA 01   2\.b.....#u.....
00000070   20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
00000080   00 00 00 00 57 1F 00 00 00 00 00 00 00 00 00 00   ....W...........
00000090   C8 9D 4B FA 02 00 00 00 30 00 00 00 70 00 00 00   ..K.....0...p...
000000A0   00 00 00 00 00 00 03 00 52 00 00 00 18 00 01 00   ........R.......
```

Ví dụ về bảng dữ liệu

- Từ đó, kết hợp với bảng BootSector, ta có thể biết được các thông tin cần thiết của ổ đĩa.

Name	Offset Hex	Size (bytes)	Description	Ký hiệu
BS_jmpBoot	0	3	Lệnh nhảy đến đoạn boot code.	
BS_OEMName	3	8	Version/tên HĐH	
BPB_BytsPerSec	**B**	**2**	**Số byte/sector**	
BPB_SecPerClus	**D**	**1**	**Số sector/cluster**	S_C
BPB_RsvdSecCnt	**E**	**2**	**Số sector để dành (khác 0) (Số sector trước bảng FAT)**	S_B
BPB_NumFATs	**10**	**1**	**Số bảng FAT**	N_F
BPB_RootEntCnt	**11**	**2**	**FAT32: có giá trị là 0**	N_{RDET}
BPB_TotSec16	**13**	**2**	**FAT32: có giá trị là 0**	S_V
BPB_Media	15	1	Loại Voulme	
BPB_FATSz16	**16**	**2**	**FAT32: có giá trị là 0 (BPB_FATSz32)**	S_F
BPB_SecPerTrk	18	2	Số sector/track	
BPB_NumHeads	1A	2	Số heads	
BPB_HiddSec	1C	4	Số sector ẩn trước Volume	
BPB_TotSec32	**20**	**4**	**Số sector trong Volume.**	N_V

Bảng BootSector của FAT32 (36 byte đầu)

Name	Offset hexa	Size (bytes)	Description
BPB_FATSz32	**24**	**4**	**số sector trong 1 bảng FAT** **BPB_FATSz16 must be 0.**
BPB_ExtFlags	28	2	0-3: chỉ số bảng FAT active Bits 4-6: dành riêng 7: 0 – cập nhật lên tất cả các bảng FAT 1 – chỉ cập nhật lên bảng FAT active 8-15: dành riêng
BPB_FSVer	2A	2	Version FAT32 (byte thấp mirror)
BPB_RootClus	**2C**	**4**	**Chỉ số cluster đầu tiên của RDET (thông thường: 2)**
BPB_FSInfo	30	2	Chỉ số sector chứa FSINFO – thông tin sector trống. (thông thường: 1)
BPB_BkBootSec	32	2	Chỉ số sector chứa bản sao của bootsector (thông thường: 6)
BPB_Reserved	34	12	Dành riêng
BS_DrvNum	**40**	**1**	**Ký hiệu vật lý đĩa (0x00: floppy disks, 0x80: hard disks).**
BS_Reserved1	41	1	Dành riêng
BS_BootSig	42	1	Ký hiệu nhận diện HĐH (0x29).
BS_VolID	43	4	Volume serial number..
BS_VolLab	47	11	Volume label.
BS_FilSysType	52	8	Chuỗi nhận diện loại FAT: ”**FAT32** ”.
	5A	420	Boot code
	1FE	2	Dấu hiệu kết thúc bootsector (0x55AA)

Bảng BootSector của FAT32 (476 byte còn lại)

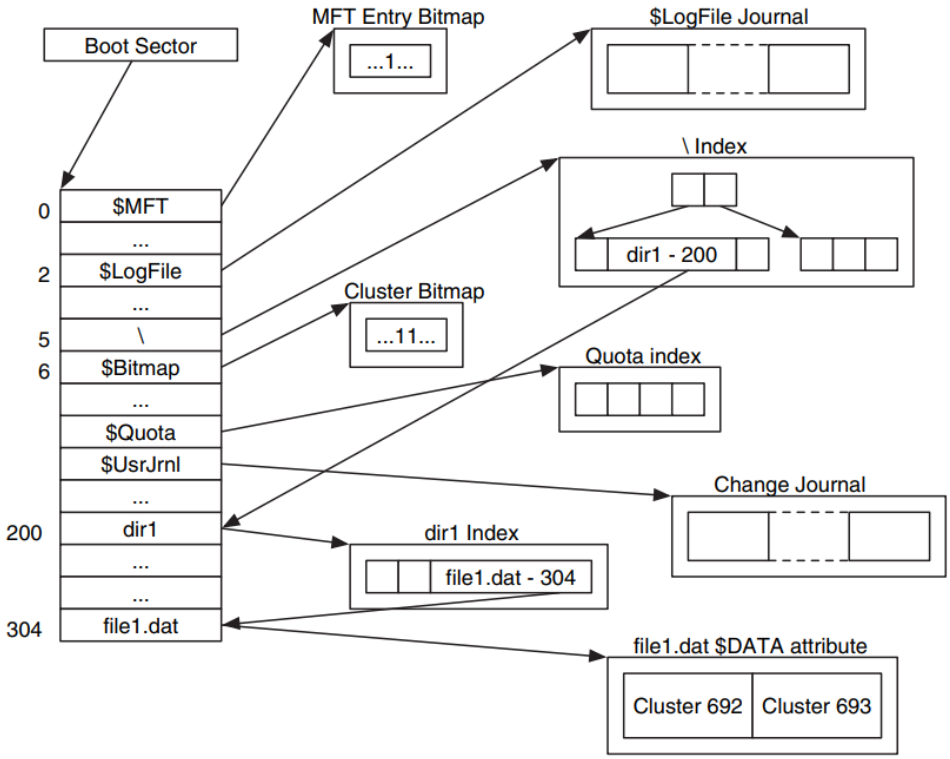
- Dựa vào bảng mô tả này, ta có thể nhận biết và đọc được các thông tin cần thiết như:
  - Số sector trên mỗi cluster
  - Số sector thuộc vùng Boot Sector
  - Số bảng FAT
  - Kích thước ổ đĩa
  - Kích thước mỗi bảng FAT
  - Cluster bắt đầu của RDET
  - Sector chứa thông tin phụ
  - Sector chứa bản lưu của Boot Sector loại FAT (chuỗi "FAT32")

- Ví dụ bảng offset của một BootSector của FAT32:

```
---------------------------------------
-- Enter volume's name (enter 0 to exit): E
  Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000   EB 58 90 4D 53 44 4F 53 35 2E 30 00 02 20 74 0D   .X.MSDOS5.0.. t.
00000010   02 00 00 00 00 F8 00 00 3F 00 FF 00 00 08 00 00   ........?.......
00000020   00 D8 94 03 46 39 00 00 00 00 00 00 02 00 00 00   ....F9..........
00000030   01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000040   80 00 29 2A 72 FA 92 4E 4F 20 4E 41 4D 45 20 20   ..)*r..NO NAME
00000050   20 20 46 41 54 33 32 20 20 20 33 C9 8E D1 BC F4     FAT32   3.....
00000060   7B 8E C1 8E D9 BD 00 7C 88 56 40 88 4E 02 8A 56   {......|.V@.N..V
00000070   40 B4 41 BB AA 55 CD 13 72 10 81 FB 55 AA 75 0A   @.A..U..r...U.u.
00000080   F6 C1 01 74 05 FE 46 02 EB 2D 8A 56 40 B4 08 CD   ...t..F..-.V@...
00000090   13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 0F B6   .s......f...@f..
000000A0   D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F B7 C9   ...?.......Af...
000000B0   66 F7 E1 66 89 46 F8 83 7E 16 00 75 39 83 7E 2A   f..f.F..~..u9.~*
000000C0   00 77 33 66 8B 46 1C 66 83 C0 0C BB 00 80 B9 01   .w3f.F.f........
000000D0   00 E8 2C 00 E9 A8 03 A1 F8 7D 80 C4 7C 8B F0 AC   ..,......}..|...
000000E0   84 C0 74 17 3C FF 74 09 B4 0E BB 07 00 CD 10 EB   ..t.<.t.........
000000F0   EE A1 FA 7D EB E4 A1 7D 80 EB DF 98 CD 16 CD 19   ...}...}........
00000100   66 60 80 7E 02 00 0F 84 20 00 66 6A 00 66 50 06   f`.~.... .fj.fP.
00000110   53 66 68 10 00 01 00 B4 42 8A 56 40 8B F4 CD 13   Sfh.....B.V@....
00000120   66 58 66 58 66 58 66 58 EB 33 66 3B 46 F8 72 03   fXfXfXfX.3f;F.r.
00000130   F9 EB 2A 66 33 D2 66 0F B7 4E 18 66 F7 F1 FE C2   ..*f3.f..N.f....
00000140   8A CA 66 8B D0 66 C1 EA 10 F7 76 1A 86 D6 8A 56   ..f..f....v....V
00000150   40 8A E8 C0 E4 06 0A CC B8 01 02 CD 13 66 61 0F   @............fa.
00000160   82 74 FF 81 C3 00 02 66 40 49 75 94 C3 42 4F 4F   .t.....f@Iu..BOO
00000170   54 4D 47 52 20 20 20 20 00 00 00 00 00 00 00 00   TMGR    ........
00000180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

NTFS

- Trong hệ thống tập tin NTFS, các tập tin và thư mục được lưu theo một cấu trúc khá độc đáo. MFT entry của thư mục cha lưu thông tin về địa chỉ MFT entry của các tập tin và thư mục con. Và ngược lại, MFT entry của tập tin và thư mục con lại lưu địa chỉ MFT entry của thư mục cha. Nên ta có 2 cách đọc cây thư mục như sau:
  - Đối với cách đầu tiên:
    - Trong MFT entry của thư mục cha có 2 attribute khá đặc biệt là $INDEX_ROOT và $INDEX_ALLOCATION.
    - Hai attribute này lưu thông tin về địa chỉ MFT entry của các tập tin và thư mục con có trong thư mục cha.
    - Và thư mục gốc (Root directory) được lưu ở MFT entry thứ 5, entry này cho biết các tập tin và thư mục con có trong thư mục gốc.
    - Do đó từ MFT entry thứ 5, ta có thể xác định toàn bộ cây thư mục bằng việc sử dụng địa chỉ MFT entry của thư mục cha để đi tim địa chỉ MFT entry của thư mục con.
  - Đối với cách thứ hai:
    - Trong attribute $FILE_NAME của mỗi MFT entry đều chứa thông tin về địa chỉ MFT entry của thư mục cha của nó (đối với thư mục gốc - Root directory - thì MFT entry của thư mục này chứa chính địa chỉ của nó).
    - Nghĩa là ta có thể tìm ra toàn bộ cây thư mục dựa vào việc đọc địa chỉ cha từ các MFT entry con.
- Nhưng nếu ta chỉ muốn đọc một vài thư mục mà không phải tất cả các thư mục trong ổ đĩa thì cách thứ hai sẽ không thuận tiện hơn so với cách thứ nhất. Do đó trong đồ án này, chương trình sẽ đọc theo cách đầu tiên – Từ thư mục cha tìm tới tập tin và thư mục con.



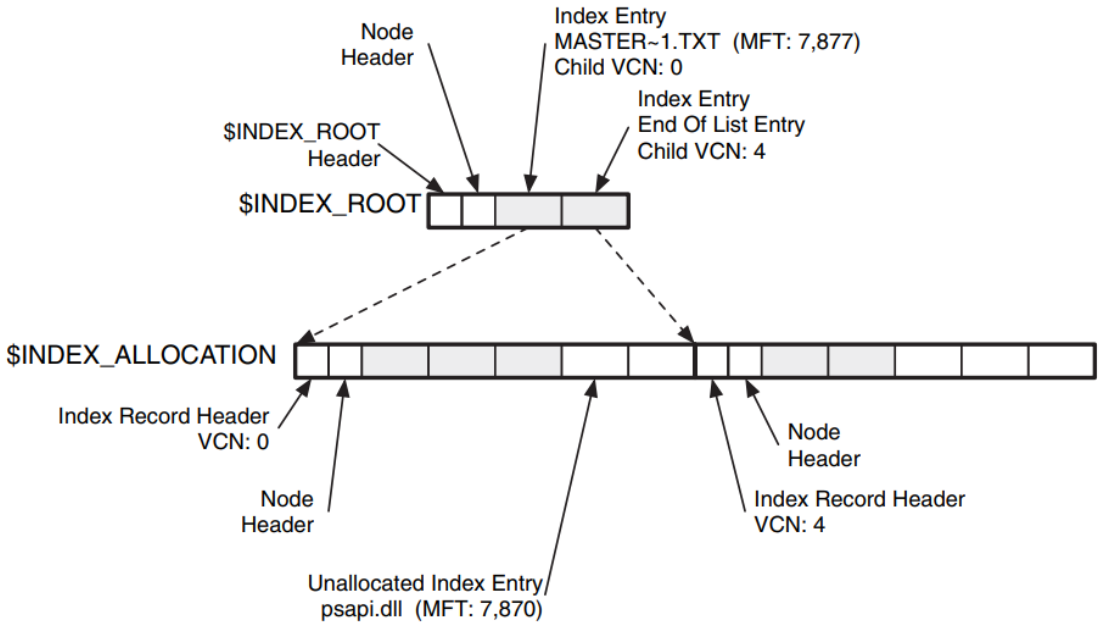
(Cách tổ chức của các MFT entry trong NTFS)

- Đối với cách lưu địa chỉ MFT entry của tập tin và thư mục con ở trong $INDEX_ROOT và $INDEX_ALLOCATION, mỗi địa chỉ này được "gói" trong một cấu trúc gọi là index entry.

  - 6 bytes đầu trong từng cấu trúc này sẽ lưu địa chỉ MFT entry của tập tin hoặc thư mục mà nó trỏ tới.

  - Attribute $INDEX_ROOT luôn là một attribute resident, còn $INDEX_ALLOCATION là non-resident, nghĩa là nếu số lượng tập tin và thư mục ít thì sẽ được lưu hết trong $INDEX_ROOT và sẽ không tồn tại attribute $INDEX_ALLOCATION trong trường hợp này, còn khi số lượng tập tin và thư mục đủ lớn thì $INDEX_ALLOCATION sẽ xuất hiện để hỗ trợ $INDEX_ROOT.

- Do là một attribute non-resident nên $INDEX_ALLOCATION lưu dữ liệu theo các datarun, mỗi datarun là một dãy các cluster liên tiếp nhau.
    - Mỗi cluster được định danh bằng một con số riêng được gọi là VCN – Virtual Cluster Number.
    - Attribute $BITMAP sẽ phụ trách việc lưu thông tin về các VCN để biết cluster nào đang được sử dụng.
- NTFS lưu danh sách các địa chỉ MFT entry của tập tin và thư mục con theo một dạng cấu trúc gọi là cây B (B Tree), mỗi node của cây B chính là một cluster (node gốc của cây là $INDEX_ROOT, các node khác của cây là các cluster trong $INDEX_ALLOCATION được định danh bởi VCN.
    - Mỗi cấu trúc index entry có thông tin về việc liệu index entry này có trỏ đến một cluster nào khác hay không, và nếu có thì cho biết thông tin về VCN của cluster đó.
    - Ta cần kiểm tra định danh VCN này với thông tin trong $BITMAP, giá trị của bit tại vị trí của VCN đang xét được bật lên 1 thì VCN này hợp lệ.



(Cách các thông tin về địa chỉ MFT entry của các tập tin và thư mục con được lưu trong NTFS)

Mô tả chi tiết

Quá trình đọc dữ liệu

- Sau khi nhập volume của drive cần đọc, chương trình sẽ đọc 512 byte đầu bằng hàm readSector (đọc được nhiều sector).
- Sau đó, kiểm tra 5 byte từ offset 82 có phải là dãy "FAT32" không, nếu không thì kiểm tra 4 byte từ offset 3 có phải là dãy "NTFS" không sau đó tạo con trỏ tới class file system tương ứng hoặc yêu cầu người dùng nhập lại tên volume nếu không phải là FAT32 hoặc NTFS.

```
readSector(device, 0, sector.data(), 512);
```

- Sau đó, chương trình sẽ đưa ra 5 lựa chọn cho người dùng: xem thông tin volume, xem cây thư mục hiện tại, mở folder hoặc đọc nội dung file text, quay lại thư mục trước đó, hoặc quay lại thư mục gốc.

Đọc hệ thống tập tin FAT32

- Class Fat32 sẽ có các thông tin cơ bản của 1 volume thuộc Fat32 và 1 vector 2 chiều thuộc kiểu dữ liệu Entry (1 Entry nghĩa là 1 thư mục hoặc file).
- Phần tử đầu mảng sẽ là mảng Entry của Root Directory, cuối mảng sẽ là mảng Entry của thư mục hiện tại.

```
private:
    // Fat32 specific
    uint64_t SectorsPerBootSector;
    uint64_t NumOfFAT;
    uint64_t SizeOfVolume;
    uint64_t SectorsPerFAT;
    uint64_t StartOfRDET;

    // List of entries
    // Last vector is the entries of current directory
    std::vector<std::vector<Entry>> Entries;
```

- Khi được khởi tạo sẽ đọc các thông tin của volume từ BootSector.
- Sau khi có được thông tin thì sẽ đọc bảng rdet để lưu thông tin thư mục và tập tin trong thư mục gốc vào mảng Entries.

```
Fat32::Fat32(std::vector<BYTE>& bootSector, HANDLE volumeHandle) : Volume(volumeHandle){
    this->ReadBootSector(bootSector);
    this->Entries.push_back(readRDETSDET(this->VolumeHandle, this->StartOfRDET, true));
    if (this->Entries.back().size() == 0) {
        std::wcout << "No entries found!" << std::endl;
    }
}
```

```
void Fat32::ReadBootSector(std::vector<BYTE>& bootSector) {
    this->BytesPerSector = nBytesToNum(bootSector.data(), 0x0B, 2);
    this->SectorsPerCluster = bootSector[0x0D];
    this->SectorsPerTrack = nBytesToNum(bootSector.data(), 0x18, 2);
    this->NumOfHeads = nBytesToNum(bootSector.data(), 0x1A, 2);

    this->SectorsPerBootSector = nBytesToNum(bootSector.data(), 0x0E, 2);
    this->NumOfFAT = bootSector[0x10];
    this->SizeOfVolume = nBytesToNum(bootSector.data(), 0x20, 4);
    this->SectorsPerFAT = nBytesToNum(bootSector.data(), 0x24, 4);

    this->StartOfRDET = rdetStartPoint(bootSector.data()) * this->BytesPerSector;
}       Dao Viet Hoang, 4 weeks ago • Add the method to read the Boot Sector
```

- Ví dụ BootSector:
  - Tại 0x0B, 2 bytes có giá trị 0x0200 = 512 là số bytes mỗi sector.
  - Tại 0x0D, 1 byte có giá trị 0x20 = 32 là số sector mỗi cluster.
  - Tại 0x18, 2 bytes có giá trị 0x003F = 63 là số sector trên mỗi track.
  - Tại 0x1A, 2 bytes có giá trị 0x00FF = 255 là số đầu đọc mà ổ đĩa có.
  - Tại 0x0E, 2 bytes có giá trị 0x0D74 = 3444 là số sectors thuộc vùng bootsector (cũng như vị trí bắt đầu của bảng FAT) .
  - Tại 0x10, 1 byte có giá trị 0x02 = 2 là số bảng FAT của ổ đĩa.
  - Tại 0x20, 4 bytes có giá trị 0x0394D800 = 60086272 là kích thước volume.
  - Tại 0x24, 4 bytes có giá trị 0x00003946 = 14662 là số sectors dùng để lưu 1 bảng fat.
  - Tại 0x2C, 4 bytes có giá trị 0x00000002 = 2, nghĩa là bảng rdet nằm ở cluster thứ (2-2) = 0 trên vùng data (bắt đầu vùng data), ta áp dụng công thức tương ứng tìm được vị trí là tại byte 16777216.

```
--------------------------------------
-- Enter volume's name (enter 0 to exit): E
  Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000   EB 58 90 4D 53 44 4F 53 35 2E 30 00 02 20 74 0D   .X.MSDOS5.0.. t.
00000010   02 00 00 00 00 F8 00 00 3F 00 FF 00 08 00 00      ........?.......
00000020   00 D8 94 03 46 39 00 00 00 00 00 00 02 00 00 00   ....F9..........
00000030   01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000040   80 00 29 2A 72 FA 92 4E 4F 20 4E 41 4D 45 20 20   ..)*r..NO NAME
00000050   20 20 46 41 54 33 32 20 20 20 33 C9 8E D1 BC F4     FAT32   3.....
00000060   7B 8E C1 8E D9 BD 00 7C 88 56 40 88 4E 02 8A 56   {......|.V@.N..V
00000070   40 B4 41 BB AA 55 CD 13 72 10 81 FB 55 AA 75 0A   @.A..U..r...U.u.
00000080   F6 C1 01 74 05 FE 46 02 EB 2D 8A 56 40 B4 08 CD   ...t..F..-.V@...
00000090   13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 0F B6   .s......f...@f..
000000A0   D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F B7 C9   ...?.......Af...
000000B0   66 F7 E1 66 89 46 F8 83 7E 16 00 75 39 83 7E 2A   f..f.F..~..u9.~*
000000C0   00 77 33 66 8B 46 1C 66 83 C0 0C BB 00 80 B9 01   .w3f.F.f........
000000D0   00 E8 2C 00 E9 A8 03 A1 F8 7D 80 C4 7C 8B F0 AC   ..,......}..|...
000000E0   84 C0 74 17 3C FF 74 09 B4 0E BB 07 00 CD 10 EB   ..t.<.t.........
000000F0   EE A1 FA 7D EB E4 A1 7D 80 EB DF 98 CD 16 CD 19   ...}...}........
00000100   66 60 80 7E 02 00 0F 84 20 00 66 6A 00 66 50 06   f`.~.... .fj.fP.
00000110   53 66 68 10 00 01 00 B4 42 8A 56 40 8B F4 CD 13   Sfh.....B.V@....
00000120   66 58 66 58 66 58 66 58 EB 33 66 3B 46 F8 72 03   fXfXfXfX.3f;F.r.
00000130   F9 EB 2A 66 33 D2 66 0F B7 4E 18 66 F7 F1 FE C2   ..*f3.f..N.f....
00000140   8A CA 66 8B D0 66 C1 EA 10 F7 76 1A 86 D6 8A 56   ..f..f....v....V
00000150   40 8A E8 C0 E4 06 0A CC B8 01 02 CD 13 66 61 0F   @............fa.
00000160   82 74 FF 81 C3 00 02 66 40 49 75 94 C3 42 4F 4F   .t.....f@Iu..BOO
00000170   54 4D 47 52 20 20 20 20 00 00 00 00 00 00 00 00   TMGR    ........
00000180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Bảng offset Boot Sector

- Vì bảng RDET và SDET có cấu trúc gần giống nhau ngoài trừ việc 64 bytes đầu của SDET là thông tin của Entry cha, do đó hàm đọc RDET SDET là hàm chung.

```
uint64_t rdetStartPoint(BYTE bootSector[]) {      You, last week •
    uint64_t sb = bootSector[15] * (1 << 7) * 2 + bootSector[14];
    uint64_t nf = bootSector[16];
    uint64_t sf = bootSector[37] * (1 << 7) * 2 + bootSector[36];
    uint64_t sc = bootSector[13];
    uint64_t k = bootSector[45] * (1 << 7) * 2 + bootSector[44];
    return sb + sf * nf + (k - 2) * sc;
}
```

- Kiểm tra RDET hoặc SDET để dịch offset khi bắt đầu đọc entry đầu tiên.

```
int start = isRDET ? 0 : 64; // True: RDET, False: SDET
```

```
        if (start != 0) start = 0;
    }
```

- Khi đọc sẽ kiểm tra nếu entry đã bị xóa hoặc đã kết thúc bảng rdet, sdet (i là byte đầu entry).

- Khi gặp entry phụ sẽ đọc tên và chạy liên tục đến khi hết entry phụ.

```
for (int i = start; i < 512; i += 32) {
    if (sector[i] == 0xE5) continue; // Deleted entry
    if (sector[i] == 0x00) { // End of RDET table
        isStopped = true;
        break;
    }

    if (sector[i + 11] == 0x0F) { // Sub entry
        std::wstring tempName = L"";
        for (int j = 1; j < 11; j += 2) if (sector[i + j] != 0x00 && sector[i + j] != 0xFF) {
            tempName += byteToWString(std::vector<BYTE>(sector + i + j, sector + i + j + 1), 1);
        }
        for (int j = 14; j < 26; j += 2) if (sector[i + j] != 0x00 && sector[i + j] != 0xFF) {
            tempName += byteToWString(std::vector<BYTE>(sector + i + j, sector + i + j + 1), 1);
        }
        for (int j = 28; j < 32; j += 2) if (sector[i + j] != 0x00 && sector[i + j] != 0xFF) {
            tempName += byteToWString(std::vector<BYTE>(sector + i + j, sector + i + j + 1), 1);
        }
        name = tempName + name;
        hasSubEntry = true;
    } else { // Main entry ···
```

- Sau khi gặp Entry chính thì sẽ đọc từ 32 byte các thuộc tính như extension (byte 8 -> 10) (TXT, DOC, e.t.c), attribute (byte 11) (Archive, System, Subdirectory, e.t.c), time (byte 22-23), date (byte 24-25), cluster thứ k chứa dữ liệu (byte high 20-21 và low 26-27) (SDET đối với Subdirectory và Data đối với Archive) và kích thước entry (byte 28-31).

```
// Set extension
std::wstring ext = L"";
for (int j = 0; j < 3; j++) ext += (sector[i + 8 + j]);
if (sector[i + 11] == 0x10) cur.setExt(L""); // Folder
else cur.setExt(ext); // Others

// Attribute
std::wstring attr = L"";
for (int j = 0; j < 8; j++) {
    if (sector[i + 11] & (1 << j)) {
        attr += flag32ToMeaningMapping[1 << j] + L" | ";
    }
}
attr.erase(attr.length() - 3, 3);
cur.setAttr(attr);
```

- Các lựa chọn của người dùng:
  - Đối với việc xem thông tin volume, chương trình sẽ in ra các thuộc tính của class Fat32.

```
You, yesterday | 2 authors (You and others)
class Fat32 : public Volume {
private:
    // Fat32 specific
    uint64_t SectorsPerBootSector;
    uint64_t NumOfFAT;
    uint64_t SizeOfVolume;
    uint64_t SectorsPerFAT;
    uint64_t StartOfRDET;
protected:
    // Volume specific
    HANDLE VolumeHandle;
    int BytesPerSector;
    int SectorsPerCluster;
    int SectorsPerTrack;
    int NumOfHeads;
};
```

  - Đối với việc xem cây thư mục hiện tại, chương trình sẽ in ra thông tin các entry trong mảng entry cuối (chỉ in các entry thuộc loại tập tin hoặc thư mục.

```
void printFileAndFolder(std::vector<Entry> vect) {
    bool isPrinted = false;
    int size = static_cast<int>(vect.size());
    for (int i = 0; i < size; i++) {
        if (vect[i].getAttr() == L"Archive" || vect[i].getAttr() == L"Subdirectory") {
            std::wcout << "Position: " << std::dec << i << std::endl;
            std::wcout << vect[i] << std::endl;
            isPrinted = true;
        }
    }
    if (!isPrinted) std::cout << "No file or folder here!" << std::endl;
}       You, last week • Reformat project
```

  - Đối với việc mở folder hoặc đọc nội dung file, người dùng cần nhập vị trí (position) được in ra khi chọn lựa chọn thứ 2, chương trình sẽ đọc file nếu nó là file txt hoặc di chuyển vào thư mục nếu nó là một thư mục (thêm mảng các entry của thư mục từ việc đọc bảng sdet vào cuối mảng Entries).
  - Đối với việc đọc file txt, hàm sẽ nhận thông tin về thứ tự cluster bắt đầu ở vùng data (bảng fat) và độ lớn của file từ Entry.
  - Từ cluster bắt đầu, dùng hàm để nhận cluster tiếp theo và vị trí cluster trên ổ đĩa.

- Từ cluster trên ổ đĩa ta đọc data sau đó lặp lại bước trên cho đến khi cluster tiếp theo nhận được từ bảng fat không thuộc đoạn từ 2 đến 0x0FFFFFEF.

```
// Vector to store the data of a cluster
std::vector<BYTE> buffer;
buffer.resize(BytesPerSector * SectorsPerCluster);
```

```
while (remainingBytes > 0 && currentCluster <= 0x0FFFFFEF) {
    // Read the data of the current cluster
    this->ReadDataCluster(this->GetDataCluster(currentCluster), buffer);

    // Number of bytes to read in the current cluster
    uint64_t bytesReaded = remainingBytes < (BytesPerSector * SectorsPerCluster) ? remainingBytes : (BytesPerSector * SectorsPerCluster);

    // Print the data
    for (uint64_t i = 0; i < bytesReaded; i++) {
        std::wcout << wchar_t(buffer[i]);
    }

    // Update the remaining bytes and the current cluster
    remainingBytes -= bytesReaded;
    currentCluster = this->GetNextFATCluster(currentCluster);
}
std::wcout << std::endl;
```

```
void Fat32::ReadDataCluster(uint64_t cluster, std::vector<BYTE>& buffer) {
    readSector(this->VolumeHandle, cluster * this->BytesPerSector, buffer.data(), this->BytesPerSector * this->SectorsPerCluster);
}
```

- Đối với hàm lấy thứ tự cluster tiếp theo, ta dựa vào việc đọc bảng fat, bởi vì mỗi phần tử của bảng fat có độ dài là 4 bytes, do đó khi vị trí cluster hiện tại vượt qua một mức nào đó (thường là bội 128) thì ta sẽ dịch vị trí sector của bảng fat cần đọc và trừ thứ tự cluster hiện tại đi để được vị trí phần tử đúng.

```
uint64_t Fat32::GetNextFATCluster(uint64_t currentCluster) {
    // Calculate the begin of the FAT
    uint64_t BeginOfFat = this->SectorsPerBootSector;

    // If the current cluster is not in the first sector of the FAT
    while (currentCluster > this->BytesPerSector / 4) {
        currentCluster -= this->BytesPerSector / 4;
        BeginOfFat += this->BytesPerSector;
    }

    // Read the current FAT sector
    BYTE FAT[512];
    readSector(this->VolumeHandle, BeginOfFat * this->BytesPerSector, FAT, BytesPerSector);

    // Get the next cluster
    uint64_t nextCluster = nBytesToNum(FAT, currentCluster * 4, 4);
    return nextCluster;
}
```

- Khi đến được đúng sector, ta sẽ đọc 4 byte của phần tử hiện tại để biết phần tử tiếp theo (thứ tự của cluster tiếp theo).

- Đối với hàm lấy vị trí trên ổ đĩa, ta áp dụng công thức Sb + Nf * Sf + (k – 2) * Sc (do RDET nằm trên vùng Data).

```
uint64_t Fat32::GetDataCluster(uint64_t cluster) {
    return SectorsPerBootSector + (NumOfFAT * SectorsPerFAT) + (cluster - 2) * SectorsPerCluster;
}
```

Ví dụ với file Dunno.txt (đã giảm bớt còn 742 bytes để tiện minh họa):

- Tại phần tử thứ 102 của bảng fat (tức offset 0x184 = 102 * 4 bytes) ta được 0x0FFFFFFF, tức là file này chỉ có data tại duy nhất cluster thứ 100 của vùng data
- Tại cluster thứ 100 của vùng data, toàn bộ 32 * 512 bytes đều là dữ liệu của file

```
00000170   5D 00 00 00 FF FF FF 0F FF FF FF 0F FF FF FF 0F   ]...............
00000180   FF FF FF 0F FF FF FF 0F 63 00 00 00 FF FF FF 0F   ........c.......
```

```
00000000   57 68 61 74 20 69 73 20 74 68 69 73 3F 20 42 72   What is this? Br
00000010   75 68 61 73 64 6C 6A 61 73 64 61 6A 73 6B 61 73   uhasdljasdajskas
00000020   68 64 73 61 6B 6A 64 68 61 6B 6A 73 64 68 61 73   hdsakjdhakjsdhas
00000030   6A 6B 64 68 61 73 64 68 61 73 6A 64 68 61 73 6A   jkdhasdhasjdhasj
00000040   64 61 6A 73 64 61 73 64 61 73 64 6A 68 61 6A 73   dajsdasdasdjhajs
00000050   64 68 61 6A 73 64 68 61 73 6A 64 68 61 6A 73 64   dhajsdhasjdhajsd
00000060   68 61 6A 73 68 64 61 6A 73 64 68 61 73 6A 68 61   hajshdajsdhasjha
00000070   73 6A 64 20 48 61 68 61 68 61 0D 0A 57 68 61 74   sjd Hahaha..What
00000080   20 69 73 20 74 68 69 73 3F 20 42 72 75 68 61 73    is this? Bruhas
00000090   64 6C 6A 61 73 64 61 6A 73 6B 61 73 68 64 73 61   dljasdajskashdsa
000000A0   6B 6A 64 68 61 6B 6A 73 64 68 61 73 6A 6B 64 68   kjdhakjsdhasjkdh
000000B0   61 73 64 68 61 73 6A 64 68 61 73 6A 64 61 6A 73   asdhasjdhasjdajs
000000C0   64 61 73 64 61 73 64 6A 68 61 6A 73 64 68 61 6A   dasdasdjhajsdhaj
000000D0   73 64 68 61 73 6A 64 68 61 6A 73 64 68 61 6A 73   sdhasjdhajsdhajs
000000E0   68 64 61 6A 73 64 68 61 73 6A 68 61 73 6A 64 20   hdajsdhasjhasjd
000000F0   48 61 68 61 68 61 0D 0A 57 68 61 74 20 69 73 20   Hahaha..What is
00000100   74 68 69 73 3F 20 42 72 75 68 61 73 64 6C 6A 61   this? Bruhasdlja
00000110   73 64 61 6A 73 6B 61 73 68 64 73 61 6B 6A 64 68   sdajskashdsakjdh
00000120   61 6B 6A 73 64 68 61 73 6A 6B 64 68 61 73 64 68   akjsdhasjkdhasdh
00000130   61 73 6A 64 68 61 73 6A 64 61 6A 73 64 61 73 64   asjdhasjdajsdasd
00000140   61 73 64 6A 68 61 6A 73 64 68 61 6A 73 64 68 61   asdjhajsdhajsdha
00000150   73 6A 64 68 61 6A 73 64 68 61 6A 73 68 64 61 6A   sjdhajsdhajshdaj
00000160   73 64 68 61 73 6A 68 61 73 6A 64 20 48 61 68 61   sdhasjhasjd Haha
00000170   68 61 0D 0A 57 68 61 74 20 69 73 20 74 68 69 73   ha..What is this
00000180   3F 20 42 72 75 68 61 73 64 6C 6A 61 73 64 61 6A   ? Bruhasdljasdaj
00000190   73 6B 61 73 68 64 73 61 6B 6A 64 68 61 6B 6A 73   skashdsakjdhakjs
000001A0   64 68 61 73 6A 6B 64 68 61 73 64 68 61 73 6A 64   dhasjkdhasdhasjd
000001B0   68 61 73 6A 64 61 6A 73 64 61 73 64 61 73 64 6A   hasjdajsdasdasdj
000001C0   68 61 6A 73 64 68 61 6A 73 64 68 61 73 6A 64 68   hajsdhajsdhasjdh
000001D0   61 6A 73 64 68 61 6A 73 68 64 61 6A 73 64 68 61   ajsdhajshdajsdha
000001E0   73 6A 68 61 73 6A 64 20 48 61 68 61 68 61 0D 0A   sjhasjd Hahaha..
```

```
000001F0   57 68 61 74 20 69 73 20 74 68 69 73 3F 20 42 72   What is this? Br
00000200   75 68 61 73 64 6C 6A 61 73 64 61 6A 73 6B 61 73   uhasdljasdajskas
00000210   68 64 73 61 6B 6A 64 68 61 6B 6A 73 64 68 61 73   hdsakjdhakjsdhas
00000220   6A 6B 64 68 61 73 64 68 61 73 6A 64 68 61 73 6A   jkdhasdhasjdhasj
00000230   64 61 6A 73 64 61 73 64 61 73 64 6A 68 61 6A 73   dajsdasdasdjhajs
00000240   64 68 61 6A 73 64 68 61 73 6A 64 68 61 6A 73 64   dhajsdhasjdhajsd
00000250   68 61 6A 73 68 64 61 6A 73 64 68 61 73 6A 68 61   hajshdajsdhasjha
00000260   73 6A 64 20 48 61 68 61 68 61 0D 0A 57 68 61 74   sjd Hahaha..What
00000270   20 69 73 20 74 68 69 73 3F 20 42 72 75 68 61 73    is this? Bruhas
00000280   64 6C 6A 61 73 64 61 6A 73 6B 61 73 68 64 73 61   dljasdajskashdsa
00000290   6B 6A 64 68 61 6B 6A 73 64 68 61 73 6A 6B 64 68   kjdhakjsdhasjkdh
000002A0   61 73 64 68 61 73 6A 64 68 61 73 6A 64 61 6A 73   asdhasjdhasjdajs
000002B0   64 61 73 64 61 73 64 6A 68 61 6A 73 64 68 61 6A   dasdasdjhajsdhaj
000002C0   73 64 68 61 73 6A 64 68 61 6A 73 64 68 61 6A 73   sdhasjdhajsdhajs
000002D0   68 64 61 6A 73 64 68 61 73 6A 68 61 73 6A 64 20   hdajsdhasjhasjd
000002E0   48 61 68 61 68 61 00 00 00 00 00 00 00 00 00 00   Hahaha..........
000002F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000300   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000310   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000320   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

- Đối với lựa chọn quay về thư mục gốc, ta sẽ xóa cho đến khi còn đúng 1 mảng (1 chiều) Entries ở đầu.

- Đối với lựa chọn quay về thư mục cha, ta sẽ xóa 1 Entry cuối nếu Entry hiện tại không phải là thư mục gốc.

Đọc hệ thống tập tin NTFS

- Class NTFS cũng sẽ có các thông tin cơ bản của volume thuộc NTFS và 1 mảng 1 chiều chứa MFTEntries chứa các thông tin của thư mục hiện tại.

```
private:
    uint64_t ReservedSectors;
    uint64_t HiddenSectors;
    uint64_t TotalSectors;
    uint64_t StartOfMFT;
    uint64_t StartOfMFTMirr;

    std::vector<MFTEntry> MFTEntries;
```

- Khi được khởi tạo, chương trình sẽ gọi constructor của một đối tượng NTFS

```
NTFS::NTFS(std::vector<BYTE>& bootSector, HANDLE volumeHandle) : Volume(volumeHandle) {
    ReadBootSector(bootSector);

    // Read root directory
    this->curEntry = uint64_t(5);
    uint64_t start = this->curEntry * 1024 + this->StartOfMFT * this->BytesPerSector;
    std::vector<uint64_t> listEntries = readFolder(this->VolumeHandle, start);
    formatListEntries(listEntries);
    this->MFTEntries = readNTFSTree(this->VolumeHandle,
        this->StartOfMFT * this->BytesPerSector,
        listEntries
    );
}
```

- Trong constructor này, trước tiên chương trình sẽ đọc bootsector, cụ thể như sau:

    - Đọc 2 bytes tại offset 0B để lấy kích thước mỗi sector (số byte trên mỗi sector).

- Đọc 1 byte tại offset 0D để lấy kích thước mỗi cluster (số sector trên mỗi cluster).
- Đọc 2 bytes tại offset 18 để lấy kích thước mỗi track (số sector trên mỗi track).
- Đọc 2 bytes tại offset 1A để lấy số head.
- Đọc 2 bytes tại offset 0E để lấy số sector dự trữ (chưa sử dụng).
- Đọc 4 bytes tại offset 1C để lấy số sector ẩn.
- Đọc 8 bytes tại offset 28 để lấy tổng số sector của ổ đĩa.
- Đọc 8 bytes tại offset 30 để lấy vị trí bắt đầu (đơn vị là cluster) của MFT (Master File Table).

```
void NTFS::ReadBootSector(std::vector<BYTE>& bootSector) {
    if (bootSector.size() < 512) {
        return;
    }

    this->BytesPerSector = nBytesToNum(bootSector.data(), 0x0B, 2);
    this->SectorsPerCluster = bootSector[0x0D];
    this->SectorsPerTrack = nBytesToNum(bootSector.data(), 0x18, 2);
    this->NumOfHeads = nBytesToNum(bootSector.data(), 0x1A, 2);

    this->ReservedSectors = nBytesToNum(bootSector.data(), 0x0E, 2);
    this->HiddenSectors = nBytesToNum(bootSector.data(), 0x1C, 4);
    this->TotalSectors = nBytesToNum(bootSector.data(), 0x28, 8);
    this->StartOfMFT = MFTStartPoint(bootSector.data());
    this->StartOfMFTMirr = nBytesToNum(bootSector.data(), 0x38, 8);
}
```

- Hàm ReadBootSector ở trên gọi tới hàm MFTStartPoint khi đọc vị trí bắt đầu của bảng MFT.

- Hàm MFTStartPoint trả về giá trị của vị trí bắt đầu của MFT nhân với giá trị của số sector trên mỗi cluster, nghĩa là hàm này trả về vị trí bắt đầu của MFT dưới dạng vị trí sector.

- Kế đến, chương trình sẽ gọi hàm readFolder để đọc nội dung của thư mục gốc (Root directory), bao gồm vị trí các MFT entry của các tập tin và thư mục con có trong thư mục gốc, hàm readFolder sẽ trả về một vector bao gồm các địa chỉ MFT entry của các tập tin và thư mục con.

- Hàm readFolder hoạt động như sau:
  - Đọc 1024 bytes từ vị trí đang xét, đây chính là dữ liệu MFT entry của thư mục gốc.
    - Ví dụ nội dung đọc ra 4 bytes ở offset 2C có giá trị là 0x00000005 = 5, nghĩa là đây là MFT entry thứ 5 – là MFT entry của thư mục gốc.

```
std::vector<uint64_t> readFolder(HANDLE device, uint64_t readPoint) {
    // This funtion returns list of entries of all files and folders in a directory

    DWORD bytesRead;
    BYTE sector[1024];

    std::vector<uint64_t> result;
    result.clear();

    LONG high = readPoint >> 32;
    LONG low = readPoint;
    SetFilePointer(device, low, &high, FILE_BEGIN); // Start reading from readPoint
    ReadFile(device, sector, 1024, &bytesRead, NULL);
```

  - Kế đến hàm sẽ đi tìm vị trí của các attribute $INDEX_ROOT có định danh là 144, $INDEX_ALLOCATION có định danh 160 và $BITMAP có định danh 176.
    - Để tìm các vị trí này thì ta đọc 4 bytes đầu tiên của attribute, rồi so sánh với định danh của từng attribute, nếu không giống nhau thì đọc 4 bytes kế của attribute để lấy kích cỡ attribute hiện tại và dùng kích cỡ đó để nhảy qua attribute tiếp theo.
  - Sau đó hàm sẽ đọc nội dung của $BITMAP, attribute này lưu tình trạng được phân bố của các VCN (Virtual Cluster Number – Định danh cluster ảo).
    - Hàm sẽ đọc ra các địa chỉ VCN để biết các cluster nào trong datarun của $INDEX_ALLOCATION đang được sử dụng, rồi lưu kết quả đọc được dưới dạng vector.
  - Tiếp theo, hàm đọc nội dung của $INDEX_ROOT, lấy ra các giá trị về vị trí MFT entry của các tập tin và thư mục con, đồng thời cũng lấy ra giá trị về các VCN của các cluster trong $INDEX_ALLOCATION rồi đưa vào vector listVCN.
  - Cuối cùng, hàm sẽ lấy các giá trị VCN trong vector listVCN và đọc các cluster tương ứng trong các datarun của $INDEX_ALLOCATION.
    - Ở mỗi cluster, cũng tương tự như đọc nội dung $INDEX_ROOT, hàm sẽ lấy ra các giá trị về vị trí MFT entry của các tập tin và thư mục con, và đồng thời lấy ra giá trị của các VCN.
    - Việc đọc này sẽ tiếp tục cho tới khi trong vector listVCN không còn bất kì phần tử nào.

- Hàm formatListEntries được dùng để chắc chắn rằng mỗi entry trong vector đọc ra là duy nhất (không bị trùng với các entry khác có trong vector), vì thư mục cha có thể chứa nhiều thể hiện của các thư mục con.

- Cuối cùng, constructor của NTFS sẽ gọi hàm readNTFSTree. Hàm này sẽ đọc nội dung của cây thư mục nhờ vào vector MFTEntry được đọc ra từ hàm readFolder. Nghĩa là với mỗi entry trong vector, hàm sẽ di chuyển vị trí đọc tới entry đó trong ổ đĩa. Ở đây, hàm sẽ làm 4 công việc chính, lần lượt là:

    - Đọc cờ hiệu của entry từ 2 bytes ở offset 0x16 và 0x17.
        - Cờ hiệu này cho ta biết liệu entry này là của tập tin hay của thư mục, hay không phải tập tin và thư mục
    - Đọc attribute $STANDARD_INFORMATION, attribute này chứa 5 thông tin quan trọng: thời gian tạo tập tin, thời gian cập nhật tập tin, thời gian cập nhật hai attribute $DATA và $INDEX, thời gian truy cập tập tin mới nhất, và cờ báo của tập tin (hầu như các thư mục đều không có cờ báo ở trường dữ liệu này).
        - Nhưng để chương trình thuận lợi cho người dùng (có những thông tin mà người dùng có thể không cần biết) thì hàm này chỉ lấy ra 2 thông tin là cờ báo và thời gian cập nhật tập tin, trong đó thời gian cập nhật sẽ được tách ra thành ngày và giờ
    - Đọc attribute $FILE_NAME để lấy tên tập tin
    - Đọc attribute $DATA để lấy kích thước tập tin

- Như vậy constructor của đối tượng NTFS sẽ lấy ra nội dung về các tập tin và thư mục con của thư mục gốc (Root directory)
- Trong chương trình có 6 chức năng chính:

    - Chức năng đầu tiên là xem thông tin volume, chương trình sẽ in ra các thuộc tính của class NTFS.

```
JackArrowdeh, 23 hours ago | 2 authors (You
class NTFS : public Volume {
private:
    // NTFS specific
    uint64_t ReservedSectors;
    uint64_t HiddenSectors;
    uint64_t TotalSectors;
    uint64_t StartOfMFT;
    uint64_t StartOfMFTMirr;
    uint64_t curEntry;

protected:
    // Volume specific
    HANDLE VolumeHandle;
    int BytesPerSector;
    int SectorsPerCluster;
    int SectorsPerTrack;
    int NumOfHeads;
};
```

- Chức năng thứ hai là xem cây thư mục hiện tại, chương trình sẽ in ra thông tin về các tập tin và thư mục con trong thư mục đang xét.

```
No. 1
Name: 1stYear
Type: Folder
Last modified time: 13/3/2024, 8:18:14
MFT entry index: 200
Size: 325

No. 2
Name: Cam Nang 2022 (1)
Extension: pdf
Type: File | Archive
Last modified time: 13/3/2024, 8:18:12
MFT entry index: 192
Size: 325

No. 3
Name: CLC-Danh sách SV khoá 2022
Extension: pdf
Type: File | Archive
Last modified time: 13/3/2024, 8:18:13
MFT entry index: 193
Size: 325
```

Ví dụ về in thông tin về tập tin và thư mục con trong một thư mục đang xét

- Chức năng thứ ba là đọc tập tin hoặc thư mục tại vị trí do người dùng nhập:
  - Người dùng cần nhập vị trí (position) được in ra khi chọn lựa chọn thứ 2, chương trình sẽ kiểm tra entry mà người dùng chọn là entry của tập tin hay không phải tập tin
  - Nếu là entry của tập tin thì sẽ đọc nội dung tập tin đối với tập tin có phần mở rộng là .txt. hoặc di chuyển vào thư mục nếu nó là một

thư mục (thay thế mảng MFTEntries hiện tại bằng việc đọc nội dung MFTEntry của thư mục).

- Trong class NTFS có một thuộc tính là curEntry, thuộc tính này lưu địa chỉ MFT entry của thư mục đang xét tại thời điểm chạy.
- Khi người dùng mở thư mục con thì curEntry sẽ được cập nhật thành địa chỉ MFT entry của thư mục con.

```
void NTFS::ReadAtPosition(uint64_t position) {
    if (position >= this->MFTEntries.size()) {
        std::wcout << "Invalid position!" << std::endl;
        return;
    }

    if (this->MFTEntries[position].getType().find(L"File") != std::wstring::npos) {
        if (this->MFTEntries[position].getExt() == L"txt") {
            std::wcout << "File content: " << std::endl;
            this->ReadAndDisplayFileData(this->MFTEntries[position].getEntry());
        }
        else {
            std::wcout << "File type not supported!" << std::endl;
        }
    }
    else {
        uint64_t start = this->MFTEntries[position].getEntry() * 1024 + this->StartOfMFT * this->BytesPerSector;
        this->curEntry = this->MFTEntries[position].getEntry();
        std::vector<uint64_t> listEntries = readFolder(this->VolumeHandle, start);
        formatListEntries(listEntries);
        this->MFTEntries = readNTFSTree(this->VolumeHandle,
            this->StartOfMFT * this->BytesPerSector,
            listEntries
        );
    }
}
```

- Chức năng thứ tư là quay về thư mục cha: Khi người dùng chọn chức năng này, chương trình sẽ đọc thông tin về địa chỉ MFT entry của thư mục cha trong attribute $FILE_NAME của thư mục hiện tại. Sau đó sẽ gán giá trị địa chỉ vừa đọc vào biến curEntry. Sau đó tiến hành đọc cây thư mục sử dụng địa chỉ được lưu trong curEntry.

```
void NTFS::ReturnToParent() {
    if (this->curEntry == uint64_t(5)) return; // Exception/Quick exit

    // Prepare for reading data
    DWORD bytesRead;
    uint64_t readPoint = this->StartOfMFT * this->BytesPerSector + this->curEntry * 1024;
    LONG high = readPoint >> 32;
    LONG low = readPoint;
    SetFilePointer(this->VolumeHandle, low, &high, FILE_BEGIN);
    BYTE sector[1024];
    ReadFile(this->VolumeHandle, sector, 1024, &bytesRead, NULL);

    // Find $FILE_NAME position
    uint64_t startAttr = nBytesToNum(sector, 0x14, 2);
    while (startAttr < 1024 && nBytesToNum(sector, startAttr, 4) != uint64_t(48)) {
        uint64_t size = nBytesToNum(sector, startAttr + uint64_t(4), 4);
        if (size == 0) return;
        startAttr += size;
    }
    if (startAttr >= 1024) return;
    uint64_t offFileName = startAttr;

    // Seek to parent position
    uint64_t startContent = offFileName + nBytesToNum(sector, offFileName + 0x14, 2); // Starting position of content section
    uint64_t parentEntry = nBytesToNum(sector, startContent, 6); // Six bytes not eight bytes
    this->curEntry = parentEntry;

    // Read parent directory
    uint64_t start = this->curEntry * 1024 + this->StartOfMFT * this->BytesPerSector;
    std::vector<uint64_t> listEntries = readFolder(this->VolumeHandle, start);
    formatListEntries(listEntries);
    this->MFTEntries = readNTFSTree(this->VolumeHandle,
        this->StartOfMFT * this->BytesPerSector,
        listEntries
    );
}
```

- Chức năng thứ năm là quay về thư mục gốc (Root directory): Do thư mục gốc luôn nằm ở entry thứ năm của MFT nên chỉ cần gán giá trị của biến curEntry cho 5 và tiến hành đọc nội dung tại entry này.

```
void NTFS::ReturnToRoot() {
    // Reset current entry
    this->curEntry = uint64_t(5);
    // Read root directory
    uint64_t start = this->curEntry * 1024 + this->StartOfMFT * this->BytesPerSector;
    std::vector<uint64_t> listEntries = readFolder(this->VolumeHandle, start);
    formatListEntries(listEntries);
    this->MFTEntries = readNTFSTree(this->VolumeHandle,
        this->StartOfMFT * this->BytesPerSector,
        listEntries
    );
}
```

- Đối với việc đọc file txt, hàm sẽ nhận vào số thứ tự của mft entry sau đó đọc mft entry của file và tìm kiếm vị trí của attribute $DATA.

```
readSector(this->VolumeHandle, this->StartOfMFT * this->BytesPerSector + mftEntry * 1024, buffer.data(), 1024); // Read MFT entry
```

- Hàm sẽ dùng 2 biến để chứa thông tin mã code của attribute và kích thước của attribute để tìm đến attribute $DATA.

```
// Current offset in the buffer
uint64_t attributeOffset = 0;
```

```
// Attribute information
uint64_t attributeCode = 0;
uint64_t attributeSize = 0;
```

Type Identifier	Name	Description
16	$STANDARD_ INFORMATION	General information, such as flags; the last accessed, written, and created times; and the owner and security ID.
32	$ATTRIBUTE_LIST	List where other attributes for file can be found.
48	$FILE_NAME	File name, in Unicode, and the last accessed, written, and created times.
64	$VOLUME_VERSION	Volume information. Exists only in version 1.2 (Windows NT).
64	$OBJECT_ID	A 16-byte unique identifier for the file or directory. Exists only in versions 3.0+ and after (Windows 2000+).
80	$SECURITY_ DESCRIPTOR	The access control and security properties of the file.
96	$VOLUME_NAME	Volume name.
112	$VOLUME_ INFORMATION	File system version and other flags.
128	$DATA	File contents.
144	$INDEX_ROOT	Root node of an index tree.
160	$INDEX_ ALLOCATION	Nodes of an index tree rooted in $INDEX_ROOT attribute.
176	$BITMAP	A bitmap for the $MFT file and for indexes.
192	$SYMBOLIC_LINK	Soft link information. Exists only in version 1.2 (Windows NT).
192	$REPARSE_POINT	Contains data about a reparse point, which is used as a soft link in version 3.0+ (Windows 2000+).
208	$EA_INFORMATION	Used for backward compatibility with OS/2 applications (HPFS).
224	$EA	Used for backward compatibility with OS/2 applications (HPFS).
256	$LOGGED_UTILITY_ STREAM	Contains keys and information about encrypted attributes in version 3.0+ (Windows 2000+).

- Hàm sẽ dùng biến offset để chứa vị trí đọc hiện tại.

- Hàm sẽ có kiểm tra attribute $DATA có resident hay không để điều chỉnh offset byte cần đọc.

```
// Flag to check if the attribute is resident or non-resident
bool isResident = true;
```

Table 13.2 Data structure for the first 16 bytes of an attribute.

Byte Range	Description	Essential
0–3	Attribute type identifier	Yes
4–7	Length of attribute	Yes
8–8	Non-resident flag	Yes
9–9	Length of name	Yes
10–11	Offset to name	Yes
12–13	Flags	Yes
14–15	Attribute identifier	Yes

<Header of $DATA>

- Hàm sẽ có biến nameLength để kiểm tra vùng data này có phải là vùng data của mft entry đang đọc hay không.

```
uint64_t nameLength = 0;
```

- Hàm sẽ có 2 biến chứa vị trí bắt đầu của data và kích thước của data.
  - Nếu là resident thì sẽ chứa offset bắt đầu của data tính từ đầu attribute $DATA và kích thước của data, ta dịch offset đến vị trí đó và bắt đầu đọc.

Table 13.3 Data structure for a resident attribute.

Byte Range	Description	Essential
0 - 15	General header (see Table 13.2)	Yes
16–19	Size of content	Yes
20–21	Offset to content	Yes

<Data structure of $DATA if resident>

  - Nếu là non-resident thì sẽ dùng để chứa số byte thuộc vùng run length và vùng run offset, sau đó chứa số cluster liên tục chứa dữ liệu và cluster bắt đầu trên ổ đĩa.

Table 13.4 Data structure for a non-resident attribute.

Byte Range	Description	Essential
0–15	General header (see Table 13.2)	Yes
16–23	Starting Virtual Cluster Number (VCN) of the runlist	Yes
24–31	Ending VCN of the runlist	Yes
32–33	Offset to the runlist	Yes
34–35	Compression unit size	Yes
36–39	Unused	No
40–47	Allocated size of attribute content	No
48–55	Actual size of attribute content	Yes
56–63	Initialized size of attribute content	No

<Data structure of $DATA if non-resident>

```
uint64_t dataSize = 0; //Size of data in bytes if resident and number of clusters if non-resident
uint64_t dataStart = 0; //Offset to the start of the data if resident and start cluster if non-resident
```

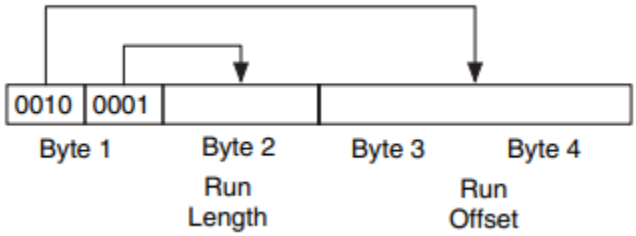
- Hàm có 1 biến offset của data runlist để nhảy tới datarun và để cộng vào offset đọc hiện tại.
- Trong trường hợp non-resident, ta đọc offset bắt đầu của data runlist từ 2 byte 32-33 sau đó dịch vị trí đến để đọc datarun.

```
uint64_t dataRunOffset = 0; //Offset to the start of the data run

attributeOffset += dataRunOffset;
```

```
dataRunOffset = nBytesToNum(buffer.data(), attributeOffset + 32, 2);
```

- Bắt đầu đọc data runlist, ta đọc header của datarun và lưu vào dataRunLength và cộng offset đọc thêm 1 (header có kích thước 1 byte).
- Sau đó ta lưu 4 bit cuối vào dataSize (số byte tiếp theo thuộc về run length), 4 bit đầu vào dataStart (số byte tiếp theo nữa thuộc về run offset).
- Dựa vào số byte cần đọc, ta đọc dữ liệu vào dataSize và dataStart.



- Từ dataStart, ta di chuyển đến vị trí cluster trên ổ đĩa và đọc data vào 1 mảng content, lặp lại cho đến khi số cluster đã đọc bằng giá trị trong data size (in mảng content ra ngay sau khi đọc).
- Tiếp tục đọc data runlist cho đến khi gặp datarun có header có giá trị 0 (kết thúc data runlist).

```
do {
    dataRunLength = nBytesToNum(buffer.data(), attributeOffset, 1);
    attributeOffset++;
    dataSize = dataRunLength & 0x0F;
    dataStart = dataRunLength >> 4;

    dataSize = nBytesToNum(buffer.data(), attributeOffset, dataSize);
    attributeOffset += dataSize;
    dataStart = nBytesToNum(buffer.data(), attributeOffset, dataStart);
    attributeOffset += dataStart;

    for (int i = 0; i < dataSize; i++) {
        readSector(this->VolumeHandle,
            (dataStart + i) * this->SectorsPerCluster * this->BytesPerSector,
            content.data(),
            this->BytesPerSector * this->SectorsPerCluster
        );

        for (int j = 0; j < content.size(); j++) {
            if (content[j] == L'\000') break;
            std::wcout << wchar_t(content[j]);      You, 3 hours ago • Read NT
        }
    }

} while (dataRunLength != 0);
```

Khó khăn gặp phải

- Về vấn đề đi tìm cách đọc cây thư mục dễ hiểu nhất:
    - Mô tả chi tiết:
        - Hầu hết các nguồn trên mạng đều chỉ nhắc tới việc sử dụng thông tin về địa chỉ của MFT entry thư mục cha được lưu ở các bytes đầu tiên trong phần nội dung của attribute $FILE_NAME.
        - Việc tra ngược cây thư mục như vậy rất bất tiện và ta cần phải đọc hết tất cả các MFT entry trong ổ đĩa để xây nên cây thư mục.
        - Ngoài ra, các thông tin trên mạng rất mơ hồ về vị trí kết thúc của việc đọc các MFT entry.
        - Nếu không biết về vị trí kết thúc này thì ta sẽ không thể dừng việc đọc sớm mà phải đọc hết tất cả dữ liệu trong ổ đĩa.

    - Cách khắc phục:
        - Dựa vào hướng dẫn của thầy trong file “Hướng dẫn project 1”, nhóm đã tìm đọc tài liệu “File system forensic analysis” của tác giả Brian Carrier.
        - Ở tài liệu này, tác giả đã nhắc tới việc MFT entry của thư mục cha có các thuộc tính $INDEX_ROOT, $INDEX_ALLOCATION và $BITMAP.
        - Các thuộc tính này được sử dụng để lưu địa chỉ MFT entry của các tập tin và thư mục con của thư mục đang xét.
        - Nghĩa là có một cách khác để đọc cây thư mục của hệ thống tập tin NTFS.
        - Cách này thuận tiện cho việc đọc cây thư mục hơn vì chỉ cần đọc các MFT entry con là có được thông tin bên trong thư mục đang xét thay vì phải đọc hết toàn bộ MFT entry trong ổ đĩa.
        - Nhóm sử dụng thông tin về địa chỉ thư mục cha trong MFT entry thư mục con để trong trường hợp người dùng muốn quay trở lại thư mục cha của thư mục hiện tại.

- Về vấn đề xuất hiện lỗi một số tập tin và thư mục không đọc được nội dung:
    - Vấn đề gặp phải:
        - Trong quá trình đọc nội dung của ổ đĩa C (Ổ đĩa chứa các tập tin quan trọng của hệ thống), một số tập tin và thư mục trả về các MFT

entry mà khi dùng các MFT entry này để đọc nội dung thì chương trình đọc ra các sector chứa 1024 byte đều có giá trị là 0.
    - Và các entry này đều là entry của các tập tin hoặc thư mục hệ thống như Program Files hay ProgramData.

- Nhóm đã tìm hiểu trên các nguồn và đưa ra nhận định rằng có thể các entry đó bị ẩn khỏi hệ thống và phải đọc theo các cách khác mà Microsoft đã không công bố thông tin này trên các nền tảng thông tin.
    - Do đó đây là một vấn đề khá lớn mà nhóm vẫn chưa khắc phục được, rất mong thầy có thể bỏ qua nếu đọc bị lỗi.
    - Ngoài ổ đĩa C ra thì chương trình vẫn hoạt động bình thường trên các ổ đĩa khác, nhóm cũng đã kiểm tra trên USB được định dạng theo NTFS và đọc chương trình thành công.

Lời cuối

Bài học, kinh nghiệm

- Sau khi hoàn thành đồ án, nhóm đã tích lũy thêm được rất nhiều kinh nghiệm trong quá trình xây dựng chương trình quản lý tập tin.
- Cảm ơn thầy đã hỗ trợ, cung cấp cho nhóm kiến thức và tạo điều kiện để nhóm có thể vận dụng những kiến thức đã học, và cả những kiến thức mới lạ có được qua việc tìm hiểu, mày mò nghiên cứu, để có thể hoàn thành đồ án tốt đẹp.

Nguồn, trích dẫn

Nguồn tham khảo

https://z-lib.id/book/file-system-forensic-analysis-1

https://stackoverflow.com/questions/44372286/using-wstring-and-wcout-doesnt-get-the-expected-output?fbclid=IwAR2k2DRRciSkQd9FSF_eqDOdymVf9GKrQ8yU6VKxHVZdhiARLH1A1lznmcU

https://en.wikipedia.org/wiki/File_system

Đường dẫn

GitHub của nhóm:
https://github.com/LancelotMoretti/Reading-FS-Project?fbclid=IwAR2YrvR2xdCZz74iV2qHd2XfgrwPJR7plmcGdNVhl5ow5kkNj2a0mWgzphE

Video Demo trên YouTube:
https://www.youtube.com/watch?v=EyP39YcvNA4