

Báo Cáo Đồ Án Hệ Điều Hành

Project 01 – Quản lý hệ thống tập trên Windows

19127102 – Võ Hoàng Gia Bảo

19127406 – Ngô Huy Hoàng

19127457 – Nguyễn Tuấn Kiệt



Bộ môn Cơ sở trí tuệ nhân tạo

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP. HCM

I.Mục lục:

1. Bảng phân công công việc
2. Đánh giá mức độ hoàn thành trên từng yêu cầu và toàn bộ project
3. Mô tả các bước thực hiện
 - a. FAT32
 - b. NTFS
4. Hình ảnh (chụp màn hình) demo chương trình ứng với các trường yêu cầu
 - a. USB FAT32
 - b, USB NTFS
5. Nguồn tham khảo
 - a. Vùng FAT32
 - b. Vùng NTFS

II. Mô tả mức độ hoàn thành và quy trình thực hiện:

1. Bảng phân công công việc:

MSSV	Họ tên	Chức vụ nhóm	Công việc phân công
19127102	Võ Hoàng Gia Bảo	Trưởng nhóm	FAT32 boot sector, RDET (100%)
19127406	Ngô Huy Hoàng	Thành viên	FAT32 SDET, in cây thư mục (100%)
19127457	Nguyễn Tuấn Kiệt	Thành viên	FAT32 đọc nội dung file text, NTFS boot sector (100%)

Cả 3: + NTFS đọc RDET, SDET: 100%

+ NTFS hiển thị cây thư mục: 50%

+ NTFS đọc nội dung file text: 0%

2. Đánh giá mức độ hoàn thành trên từng yêu cầu và toàn bộ project:

		Mức độ hoàn thành
1. Đọc các thông tin chi tiết của một phân vùng	Với phân vùng FAT32	100%
	Với phân vùng NTFS	100%
2. Hiển thị thông tin cây Thư mục của phân vùng	Với phân vùng FAT32	100%
	Với phân vùng NTFS	60%

Chức năng còn thiếu NTFS:

- + Chưa sắp xếp cây thư mục theo trật tự
- + Chưa đọc được nội dung file text

3. Mô tả các bước thực hiện:

a. FAT32

Tạo các struct để dễ dàng đọc và lưu dữ liệu

```
struct BOOTSECTORFAT32
{
    BYTE JUMP[3];
    BYTE OEM[8];
    WORD BytesPerSector;
    BYTE SectorPerCluster;
    WORD ReservedSector;
    BYTE FatNum;
    WORD EntryRDET;
    WORD LowNumberSectors;
    BYTE DeviceType;
    WORD SectorPerFat16;
    WORD SectorPerTrack;
    WORD HeadPerDisk;

    DWORD NumberHiddenSectors;
    DWORD HighNumberSectors;
    DWORD SectorPerFat32;
    WORD Bit8Flag;
    WORD FAT32Ver;
    DWORD FirstRDETCluster;
    WORD AddInfoSector;
    WORD BackupSector;
    BYTE LaterVerReserved[12];

    BYTE PhysicDisk;
    BYTE Reserved;
    BYTE Signature;
    DWORD VolumeSerial;
    BYTE VolumeLabel[11];
    BYTE FATID[8];
    BYTE BootProgram[420];
    WORD EndSignature;
};

struct RDET FAT32 {
    // Main entry
    BYTE FileName[11];
    BYTE FileAttributes;
    BYTE Reserved;
    BYTE CreatedHour[3];
    WORD CreatedDate;
    WORD LastAccessedDate;
    WORD FirstClusterHigh;
    WORD LastModifiedHour;
    WORD LastModifiedDate;
    WORD FirstClusterLow;
    DWORD FileSize;
};

struct LongFileDir {
    BYTE Flag;
    BYTE Name1[10];
    BYTE Attribute; // always 0F
    BYTE Reserved;
    BYTE Checksum;
    BYTE Name2[12];
    BYTE RelativeCluster[2];
    BYTE Name3[4];
};
```

Sau khi nhập thông tin của ổ đĩa cần đọc, chương trình sẽ đọc thông tin của boot sector của ổ đĩa FAT32

```
BOOTSECTORFAT32 bs32;

int ReadBootSectorFAT32(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());
        return 1;
    }
    else
    {
        // Success
    }
}
```

Sau đó tiến hành copy số bytes từ sector vào các nội dung của boot sector

```
else
{
    memset(&bs32, 0, 512);

    memcpy(&bs32.JUMP, sector, sizeof(bs32.JUMP));
    memcpy(&bs32.OEM, sector + 3, sizeof(bs32.OEM));
    memcpy(&bs32.BytePerSector, sector + 11, sizeof(bs32.BytePerSector));
    memcpy(&bs32.SectorPerCluster, sector + 13, sizeof(bs32.SectorPerCluster));
    memcpy(&bs32.ReservedSector, sector + 14, sizeof(bs32.ReservedSector));
    memcpy(&bs32.FatNum, sector + 16, sizeof(bs32.FatNum));
    memcpy(&bs32.EntryRDET, sector + 17, sizeof(bs32.EntryRDET));
    memcpy(&bs32.LowNumberSectors, sector + 19, sizeof(bs32.LowNumberSectors));
    memcpy(&bs32.DeviceType, sector + 21, sizeof(bs32.DeviceType));
    memcpy(&bs32.SectorPerFat16, sector + 22, sizeof(bs32.SectorPerFat16));
    memcpy(&bs32.SectorPerTrack, sector + 24, sizeof(bs32.SectorPerTrack));
    memcpy(&bs32.HeadPerDisk, sector + 26, sizeof(bs32.HeadPerDisk));

    memcpy(&bs32.NumberHiddenSectors, sector + 28, sizeof(bs32.NumberHiddenSectors));
    memcpy(&bs32.HighNumberSectors, sector + 32, sizeof(bs32.HighNumberSectors));
    memcpy(&bs32.SectorPerFat32, sector + 36, sizeof(bs32.SectorPerFat32));
    memcpy(&bs32.Bit8Flag, sector + 40, sizeof(bs32.Bit8Flag));
    memcpy(&bs32.FAT32Ver, sector + 42, sizeof(bs32.FAT32Ver));
    memcpy(&bs32.FirstRDETCluster, sector + 44, sizeof(bs32.FirstRDETCluster));
    memcpy(&bs32.AddiInfoSector, sector + 48, sizeof(bs32.AddiInfoSector));
    memcpy(&bs32.BackupSector, sector + 50, sizeof(bs32.BackupSector));
    memcpy(&bs32.LaterVerReserved, sector + 52, sizeof(bs32.LaterVerReserved));

    memcpy(&bs32.PhysicDisk, sector + 64, sizeof(bs32.PhysicDisk));

    memcpy(&bs32.PhysicDisk, sector + 64, sizeof(bs32.PhysicDisk));
    memcpy(&bs32.Reserved, sector + 65, sizeof(bs32.Reserved));
    memcpy(&bs32.Signature, sector + 66, sizeof(bs32.Signature));
    memcpy(&bs32.VolumeSerial, sector + 67, sizeof(bs32.VolumeSerial));
    memcpy(&bs32.VolumeLabel, sector + 71, sizeof(bs32.VolumeLabel));
    memcpy(&bs32.FATID, sector + 82, sizeof(bs32.FATID));
    memcpy(&bs32.BootProgram, sector + 90, sizeof(bs32.BootProgram));
    memcpy(&bs32.EndSignature, sector + 510, sizeof(bs32.EndSignature));
}

CloseHandle(device);
return 0;
```

Tiến hành đọc cây thư mục gốc và bảng fat

```
int retCode = 0;
DWORD bytesRead;
HANDLE device = NULL;
LongFileDir LFD;
std::string SubName = "";

device = CreateFile(drive,      // Drive to open
    GENERIC_READ,              // Access mode
    FILE_SHARE_READ,           // 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;
}

// Read FAT Table
int FATSize = reversedDWORD(bs32.SectorPerFat32) * reversedWORD(bs32.BytePerSector);
BYTE* FATTable = new BYTE[FATSize];
SetFilePointer(device, reversedWORD(bs32.ReservedSector) * reversedWORD(bs32.BytePerSector), NULL, FILE_BEGIN);
if (!ReadFile(device, FATTable, FATSize, &bytesRead, 0))
{
    printf("ReadFile: %u\n", GetLastError());
    return 1;
}
```

```
//// SB + SF * NF
ULONG distance = reversedWORD(bs32.ReservedSector) + bs32.FatNum * reversedDWORD(bs32.SectorPerFat32);
distance *= bs32.BytePerSector; // convert to bytes

Start:
SetFilePointer(device, distance, NULL, FILE_BEGIN);

int clusterSize = bs32.BytePerSector * bs32.SectorPerCluster; // cluster size
int NumberOfEntries = clusterSize / sizeof(RDETFAT32); // number of record inside cluster
RDETFAT32* root = new RDETFAT32[NumberOfEntries]; // describe the partition

if (!ReadFile(device, (BYTE*)root, clusterSize, &bytesRead, 0))
{
    printf("ReadFile: %u\n", GetLastError());
    return 1;
}
else {
    bool breakPoint = false;
    int ind = 0;
    for (int i = 0; i < NumberOfEntries; i++)
    {
        // Xet entry chinh
        if (root[i].FileName[0] == 0x00) {
            breakPoint = true;
            break;
        }
        if (root[i].FileName[0] == 0xE5)
            continue;

        // Xet entry phu
    }
}
```

Nối tên từ các entry phụ của entry chính

```
// Xet entry phu
if (root[i].FileAttributes == 0x0F) {
    memset(&LFD, 0, sizeof(LFD));
    memcpy(&LFD, &root[i], sizeof(LFD));

    std::string temp = "";
    for (int j = 0; j < 3; j++) {
        if (j == 0) {
            for (int k = 0; k < sizeof(LFD.Name1); k++)
                if (32 <= (int)LFD.Name1[k] && (int)LFD.Name1[k] <= 126)
                    temp += LFD.Name1[k];
        }
        else if (j == 1) {
            for (int k = 0; k < sizeof(LFD.Name2); k++)
                if (32 <= (int)LFD.Name2[k] && (int)LFD.Name2[k] <= 126)
                    temp += LFD.Name2[k];
        }
        else {
            for (int k = 0; k < sizeof(LFD.Name3); k++)
                if (32 <= (int)LFD.Name3[k] && (int)LFD.Name3[k] <= 126)
                    temp += LFD.Name3[k];
        }
    }

    SubName.insert(0, temp);
    continue;
}

if (SubName != "") {
    std::cout << SubName;
}
```

```
if (SubName != "") {
    std::cout << SubName;
}

else {
    std::string tmp = "";
    for (int j = 0; j < 8; j++) {
        std::cout << root[i].FileName[j];
        tmp += root[i].FileName[j];
    }
    if ((root[i].FileAttributes & 0x10) != 0x10) {
        std::cout << ".";
        tmp += '.';
        for (int j = 8; j < 11; j++) {
            std::cout << root[i].FileName[j];
            tmp += root[i].FileName[j];
        }
    }
    SubName = tmp;
}
```

Tiến hành in các thông tin của tập tin / thư mục (mặc định thư mục có size = 0)

```
if (root[i].FileAttributes == 0x01)
    printf("    <Read Only>\n");
if (root[i].FileAttributes == 0x02)
    printf("    <Hidden>\n");
if (root[i].FileAttributes == 0x04)
    printf("    <System>\n");
if (root[i].FileAttributes == 0x08)
    printf("    <Volume Label>\n");
if (root[i].FileAttributes == 0x10)
    printf("    <Directory>\n");
if (root[i].FileAttributes == 0x20)
    printf("    <Archive>\n");

WORD nYear = (root[i].CreatedDate >> 9);
WORD nMonth = (root[i].CreatedDate << 7);
nMonth = nMonth >> 12;
WORD nDay = (root[i].CreatedDate << 11);
nDay = nDay >> 11;
printf("Create Date: %d/%d/%d\n", nDay, nMonth, (nYear + 1980));

nYear = (root[i].LastModifiedDate >> 9);
nMonth = (root[i].LastModifiedDate << 7);
nMonth = nMonth >> 12;
nDay = (root[i].LastModifiedDate << 11);
nDay = nDay >> 11;
printf("Modification Date: %d/%d/%d\n", nDay, nMonth, (nYear + 1980));

WORD nHour = (root[i].LastModifiedHour >> 11);
WORD nMin = (root[i].LastModifiedHour << 5);
nMin = nMin >> 10;
WORD nSec = (root[i].LastModifiedHour << 11);
nSec = nSec >> 11;
printf("Modification Hours: %02d:%02d:%02d\n", nHour, nMin, nSec/2);

nYear = (root[i].LastAccessedDate >> 9);
nMonth = (root[i].LastAccessedDate << 7);
nMonth = nMonth >> 12;
nDay = (root[i].LastAccessedDate << 11);
nDay = nDay >> 11;
printf("Accessed Date: %d/%d/%d\n", nDay, nMonth, (nYear + 1980));

DWORD clusterNumber = root[i].FirstClusterHigh << 16;
clusterNumber |= root[i].FirstClusterLow;
std::cout << "Size: " << root[i].FileSize << " bytes" << "    " << clusterNumber << " cluster\n" << std::endl;
```

Sau khi đọc thông tin của tập tin / thư mục, nếu là thư mục thì sẽ đọc tiếp cây thư mục con, và nếu là tập tin file text thì sẽ in ra nội dung file text đó

```
if (root[i].FileAttributes == 0x10) {
    ReadSRDETFAT32(drive, device, clusterNumber, 0);
    std::cout << std::endl;
}

if (SubName != "") {
    if (SubName.length() - 4 >= 0 &&
        (SubName.substr(SubName.length() - 3) == "txt" || SubName.substr(SubName.length() - 3) == "TXT"))
        ReadTextFile(drive, device, clusterNumber);
}

SubName = "";
}
```

Hàm ReadSRDETFAT32 tương tự hàm ReadRDETFAT32 đọc cây thư mục gốc, lấy số cluster của thư mục và tra bảng fat để ra được cây thư mục con

b. NTFS

Sử dụng thư viện Sean Barrett's data structures để mọi thứ đơn giản

```
#include "NTFS.h"
#include "GUI.h"

#define STB_DS_IMPLEMENTATION
#include "../Sublib/stb_ds.h"
```

Hàm đọc từ 1 vị trí, giá trị truyền vào tên ổ đĩa, buffer lưu các bytes đọc, vị trí, số bytes cần đọc

```
int Read(LPCWSTR driveLabel, void* buffer, uint64_t from, uint64_t count)
{
    int retCode = 0;
    DWORD bytesRead;
    HANDLE device = NULL;

    device = CreateFile(driveLabel,    // 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;
    }

    LONG high = from >> 32;
    SetFilePointer(device, from & 0xFFFFFFFF, &high, FILE_BEGIN);
    ReadFile(device, buffer, count, &bytesRead, NULL);
    assert(bytesRead == count);

    CloseHandle(device);
    return 0;
}
```

Sử dụng assert trong suốt quá trình thay vì kiểm tra lỗi, ta có thể muốn thay thế những thứ này sau, nhưng việc đọc trực tiếp từ MFT không phải là việc ta phải làm. Trình điều khiển hệ thống tệp có thể

đang cập nhật MFT khi ta đọc nó, vì vậy có thể có conflicts trong dữ liệu thu được.

Tạo struct như bên FAT32 để lưu trữ boot sector

```
struct BOOTSECTORNTFS {
    uint8_t    JUMP[3];
    char       OEM[8];

    // BPB
    uint16_t   BytePerSector;
    uint8_t    SectorPerCluster;
    uint16_t   ReservedSector;
    uint8_t    Unknown1[3];
    uint16_t   Unknown2;
    uint8_t    MediaDescriptor;
    uint16_t   Unknown3;
    uint16_t   SectorPerTrack;
    uint16_t   HeadPerDisk;
    uint32_t   NumberHiddenSectors;
    uint32_t   Unknown4;

    // Extended BPB
    uint32_t   Unknown5;
    uint64_t   TotalSectors;
    uint64_t   $MFTCluster;
    uint64_t   $MFTMirrCluster;
    uint8_t    BytesPerFileRecordSegment;    // > 0 -> cluster || < 0 : Ex: -10 -> 2^10 bytes = 1024 bytes
    uint8_t    Unknown6[3];
    uint8_t    BytesPerIndexBuffer;          // Like above
    uint8_t    Unknown7[3];
    uint64_t   VolumeSerial;
    uint32_t   Checksum;

    uint8_t    Bootstrap[426];
    uint16_t   EndOfSectorMarker;
};
```

Các trường cụ thể của header

```

struct FileRecordHeader {
    uint32_t    signature;
    uint16_t    updateSequenceOffset;
    uint16_t    updateSequenceSize;
    uint64_t    logSequence;
    uint16_t    sequenceNumber;
    uint16_t    hardLinkCount;
    uint16_t    firstAttributeOffset;
    uint16_t    inUse : 1;
    uint16_t    isDirectory : 1;
    uint32_t    usedSize;
    uint32_t    allocatedSize;
    uint64_t    fileReference;
    uint16_t    nextAttributeID;
    uint16_t    unused;
    uint32_t    recordNumber;
};

struct AttributeHeader {
    uint32_t    attributeType;
    uint32_t    length;
    uint8_t     nonResident;
    uint8_t     nameLength;
    uint16_t    nameOffset;
    uint16_t    flags;
    uint16_t    attributeID;
};

struct ResidentAttributeHeader : AttributeHeader {
    uint32_t    attributeLength;
    uint16_t    attributeOffset;
    uint8_t     indexed;
    uint8_t     unused;
};

struct FileNameAttributeHeader : ResidentAttributeHeader {
    uint64_t    parentRecordNumber : 48;
    uint64_t    sequenceNumber : 16;
    uint64_t    creationTime;
    uint64_t    modificationTime;
    uint64_t    metadataModificationTime;
    uint64_t    readTime;
    uint64_t    allocatedSize;
    uint64_t    realSize;
    uint32_t    flags;
    uint32_t    repase;
    uint8_t     fileNameLength;
    uint8_t     namespaceType;
    wchar_t     fileName[1];
};

struct NonResidentAttributeHeader : AttributeHeader {
    uint64_t    firstCluster;
    uint64_t    lastCluster;
    uint16_t    dataRunsOffset;
    uint16_t    compressionUnit;
    uint32_t    unused;
    uint64_t    attributeAllocated;
    uint64_t    attributeSize;
    uint64_t    streamDataSize;
};

struct RunHeader {
    uint8_t     lengthFieldBytes : 4;
    uint8_t     offsetFieldBytes : 4;
};

```

Có một số attributes bổ sung khác nhau cho các thuộc tính Non-Resident và Resident.

Bọc các cấu trúc này trong pragma để trình biên dịch không thêm phần padding.

MFT được chia nhỏ thành các phần bằng nhau gọi là MFT entry. Kích thước của một MFT entry được quy định trong BPB, thường là 1024 byte.

```
#define MFT_FILE_SIZE (1024)
extern uint8_t mftFile[MFT_FILE_SIZE];

#define MFT_FILES_PER_BUFFER (65536)
extern uint8_t mftBuffer[MFT_FILES_PER_BUFFER * MFT_FILE_SIZE];
```

Tiến hành đọc boot sector

```
Read(DriveLabel, &bootSector, 0, 512);
```

Đọc 1KB đầu tiên của MFT (con trỏ bit đến các khối khác trong MFT)

```
uint64_t bytesPerCluster = bootSector.BytePerSector * bootSector.SectorPerCluster;
Read(DriveLabel, &mftFile, bootSector.$MFTCluster * bytesPerCluster, MFT_FILE_SIZE);
```

Chúng tôi sẽ tìm kiếm thuộc tính \$DATA trong FileRecord. Thuộc tính \$DATA được sử dụng để lưu trữ nội dung của một tệp, vì vậy trong trường hợp của MFT, nó lưu trữ MFT. Vì toàn bộ MFT không thể nằm gọn trong một bản ghi tệp MFT, nên thuộc tính \$DATA sẽ là non-resident. Điều đó có nghĩa là thay vào đó, thuộc tính sẽ liệt kê các khối liên kế có chứa MFT.

```

FileRecordHeader* fileRecord = (FileRecordHeader*)mftFile;
AttributeHeader* attribute = (AttributeHeader*)(mftFile + fileRecord->firstAttributeOffset);
NonResidentAttributeHeader* dataAttribute = nullptr;
uint64_t approximateRecordCount = 0;
assert(fileRecord->magic == 0x454C4946);

while (true) {
    if (attribute->attributeType == 0x80) {
        dataAttribute = (NonResidentAttributeHeader*)attribute;
    }
    else if (attribute->attributeType == 0xB0) {
        approximateRecordCount = ((NonResidentAttributeHeader*)attribute)->attributeSize * 8;
    }
    else if (attribute->attributeType == 0xFFFFFFFF) {
        break;
    }

    attribute = (AttributeHeader*)((uint8_t*)attribute + attribute->length);
}

assert(dataAttribute);

```

Tiến hành parse các dữ liệu non-resident

Như đã mô tả ở trên, thuộc tính không cư trú liệt kê các khối liên kế tạo nên nội dung của thuộc tính. Do đó, với thuộc tính \$DATA của MFT, chúng ta có danh sách các khối chứa MFT.

Mỗi khối được gọi là một lần chạy dữ liệu. Việc chạy dữ liệu phải bắt đầu và kết thúc ở ranh giới cụm (thường là mỗi 4KB - kích thước một cluster là: $8 \times 512 = 4096 \text{ B} = 4 \text{ KB}$)

Loop từng lần chạy dữ liệu trong thuộc tính \$DATA

```

RunHeader* dataRun = (RunHeader*)((uint8_t*)dataAttribute + dataAttribute->dataRunsOffset);
uint64_t clusterNumber = 0, recordsProcessed = 0;

while (((uint8_t*)dataRun - (uint8_t*)dataAttribute) < dataAttribute->length && dataRun->lengthFieldBytes) {
    uint64_t length = 0, offset = 0;

    for (int i = 0; i < dataRun->lengthFieldBytes; i++) {
        length |= (uint64_t)((uint8_t*)dataRun)[1 + i] << (i * 8);
    }

    for (int i = 0; i < dataRun->offsetFieldBytes; i++) {
        offset |= (uint64_t)((uint8_t*)dataRun)[1 + dataRun->lengthFieldBytes + i] << (i * 8);
    }

    if (offset & ((uint64_t)1 << (dataRun->offsetFieldBytes * 8 - 1))) {
        for (int i = dataRun->offsetFieldBytes; i < 8; i++) {
            offset |= (uint64_t)0xFF << (i * 8);
        }
    }

    clusterNumber += offset;
    dataRun = (RunHeader*)((uint8_t*)dataRun + 1 + dataRun->lengthFieldBytes + dataRun->offsetFieldBytes);
}

```

Đối với mỗi lần chạy dữ liệu, ta cần tính toán độ dài và độ lệch tuyệt đối của nó (tức là từ đầu phân vùng). Sau đó, ta tính được số cluster và độ dài.

Tiến hành liệt kê các tệp

Đã đến lúc tải từng block MFT và quét file records. Bởi vì một khối MFT có thể khá lớn (hàng trăm MB), ta sẽ xử lý nó trong các khối 64MB

```
uint64_t filesRemaining = length * bytesPerCluster / MFT_FILE_SIZE;
uint64_t positionInBlock = 0;

while (filesRemaining) {
    //fprintf(stderr, "%d%% ", (int)(recordsProcessed * 100 / approximateRecordCount));

    uint64_t filesToLoad = MFT_FILES_PER_BUFFER;
    if (filesRemaining < MFT_FILES_PER_BUFFER) filesToLoad = filesRemaining;
    Read(DriveLabel, &mftBuffer, clusterNumber * bytesPerCluster + positionInBlock, filesToLoad * MFT_FILE_SIZE);
    positionInBlock += filesToLoad * MFT_FILE_SIZE;
    filesRemaining -= filesToLoad;

    for (int i = 0; i < filesToLoad; i++) {
        // Even on an SSD, processing the file records takes only a fraction of the time to read the data,
        // so there's not much point in multithreading this.
    }
}
```

Bỏ qua các file records đang không được sử dụng

```
FileRecordHeader* fileRecord = (FileRecordHeader*)(mftBuffer + MFT_FILE_SIZE * i);
recordsProcessed++;

if (!fileRecord->inUse) continue;
```

Bây giờ ta cần tìm thuộc tính \$FILENAME. (Thực tế có thể có nhiều thuộc tính \$FILENAME - đó là cách các liên kết cứng hoạt động trong NTFS.) Code parse attribute giống như trước đó.

```

AttributeHeader* attribute = (AttributeHeader*)((uint8_t*)fileRecord + fileRecord->firstAttributeOffset);
assert(fileRecord->magic == 0x454C4946);

while ((uint8_t*)attribute - (uint8_t*)fileRecord < MFT_FILE_SIZE) {
    if (attribute->attributeType == 0x30) {
        FileNameAttributeHeader* fileNameAttribute = (FileNameAttributeHeader*)attribute;

        if (fileNameAttribute->namespaceType != 2 && !fileNameAttribute->nonResident) {
            File file = {};
            file.parent = fileNameAttribute->parentRecordNumber;
            file.name = DuplicateName(fileNameAttribute->fileName, fileNameAttribute->fileNameLength);

            if (file.name[0] != '$' && file.name[0] != '.') {
                std::cout << file.name << std::endl;

                SYSTEMTIME stSystemTime, stLocalTime;
                FILETIME fileTime;
                TIME_ZONE_INFORMATION tZone;
                GetTimeZoneInformation(&tZone);

                fileTime.dwHighDateTime = fileNameAttribute->creationTime >> 32;
                fileTime.dwLowDateTime = fileNameAttribute->creationTime & 0xFFFFFFFF;
                if (FileTimeToSystemTime(&fileTime, &stSystemTime) && SystemTimeToTzSpecificLocalTime(&tZone, &

```

```

                fileTime.dwHighDateTime = fileNameAttribute->creationTime >> 32;
                fileTime.dwLowDateTime = fileNameAttribute->creationTime & 0xFFFFFFFF;
                if (FileTimeToSystemTime(&fileTime, &stSystemTime) && SystemTimeToTzSpecificLocalTime(&tZone, &
                {
                    std::cout << "Creation Time: " << dayOfWeek(stSystemTime.wDayOfWeek);
                    printf(", %d/%d/%d, %02d:%02d:%d\n", stLocalTime.wDay, stLocalTime.wMonth, stLocalTime.wYear,
                }
                fileTime.dwHighDateTime = fileNameAttribute->modificationTime >> 32;
                fileTime.dwLowDateTime = fileNameAttribute->modificationTime & 0xFFFFFFFF;
                if (FileTimeToSystemTime(&fileTime, &stSystemTime) && SystemTimeToTzSpecificLocalTime(&tZone, &
                {
                    std::cout << "Modification Time: " << dayOfWeek(stSystemTime.wDayOfWeek);
                    printf(", %d/%d/%d, %02d:%02d:%d\n", stLocalTime.wDay, stLocalTime.wMonth, stLocalTime.wYear,
                }

                std::cout << "Size on disk: " << fileNameAttribute->allocatedSize << " bytes\n";
                std::cout << std::endl;
            }

            uint64_t oldLength = arrlenu(files);

            if (fileRecord->recordNumber >= oldLength) {

```

```

                uint64_t oldLength = arrlenu(files);

                if (fileRecord->recordNumber >= oldLength) {
                    arrsetlen(files, fileRecord->recordNumber + 1);
                    memset(files + oldLength, 0, sizeof(File) * (fileRecord->recordNumber - oldLength));
                }

                files[fileRecord->recordNumber] = file;
            }
        }
    }
    else if (attribute->attributeType == 0xFFFFFFFF) {
        break;
    }

    attribute = (AttributeHeader*)((uint8_t*)attribute + attribute->length);
}

```

Thuộc tính \$FILE_NAME chứa tên file trong nội dung của nó. Tên file có thể không quá dài, vì vậy thuộc tính này sẽ là resident.

Dùng struct File để lưu tên file và thư mục gốc của nó

```
struct File {  
    uint64_t parent;  
    char* name;  
};  
  
extern File* files;
```

Đối với mỗi thuộc tính \$FILE_NAME, ta sẽ thêm 1 entry vào array.

```
uint64_t oldLength = arrlenu(files);  
  
if (fileRecord->recordNumber >= oldLength) {  
    arrsetlen(files, fileRecord->recordNumber + 1);  
    memset(files + oldLength, 0, sizeof(File) * (fileRecord->recordNumber - oldLength));  
}  
  
files[fileRecord->recordNumber] = file;
```

Ta cần lưu trữ tên file ở nơi khác, vì MFT buffer sẽ được sử dụng lại. Hàm sau phân bổ khoảng trống cho tên tệp và chuyển đổi nó thành UTF-8. Ta nhóm các phân bổ lại với nhau thành các phần 16MB.

```
char* DuplicateName(wchar_t* name, size_t nameLength) {  
    static char* allocationBlock = nullptr;  
    static size_t bytesRemaining = 0;  
  
    size_t bytesNeeded = WideCharToMultiByte(CP_UTF8, 0, name, nameLength, NULL, 0, NULL, NULL) + 1;  
  
    if (bytesRemaining < bytesNeeded) {  
        allocationBlock = (char*)malloc((bytesRemaining = 16 * 1024 * 1024));  
    }  
  
    char* buffer = allocationBlock;  
    buffer[bytesNeeded - 1] = 0;  
    WideCharToMultiByte(CP_UTF8, 0, name, nameLength, allocationBlock, bytesNeeded, NULL, NULL);  
  
    bytesRemaining -= bytesNeeded;  
    allocationBlock += bytesNeeded;  
  
    return buffer;  
}
```


Đối phần thông tin của các file, phần này sẽ in ra các thông tin của file

```
if (file.name[0] != '$' && file.name[0] != '.') {
    std::cout << file.name << std::endl;

    SYSTEMTIME stSystemTime, stLocalTime;
    FILETIME fileTime;
    TIME_ZONE_INFORMATION tZone;
    GetTimeZoneInformation(&tZone);

    fileTime.dwHighDateTime = fileNameAttribute->creationTime >> 32;
    fileTime.dwLowDateTime = fileNameAttribute->creationTime & 0xFFFFFFFF;
    if (FileTimeToSystemTime(&fileTime, &stSystemTime) && SystemTimeToTzSpecificLocalTime(&tZone, &stSystemTime, &stLocalTime))
    {
        std::cout << "Creation Time: " << dayOfWeek(stSystemTime.wDayOfWeek);
        printf(", %d/%d/%d, %02d:%02d:%d\n", stLocalTime.wDay, stLocalTime.wMonth, stLocalTime.wYear, stLocalTime.wHour, stLocalTime.wMinute, stLocalTime.wMilliseconds);
    }
    fileTime.dwHighDateTime = fileNameAttribute->modificationTime >> 32;
    fileTime.dwLowDateTime = fileNameAttribute->modificationTime & 0xFFFFFFFF;
    if (FileTimeToSystemTime(&fileTime, &stSystemTime) && SystemTimeToTzSpecificLocalTime(&tZone, &stSystemTime, &stLocalTime))
    {
        std::cout << "Modification Time: " << dayOfWeek(stSystemTime.wDayOfWeek);
        printf(", %d/%d/%d, %02d:%02d:%d\n", stLocalTime.wDay, stLocalTime.wMonth, stLocalTime.wYear, stLocalTime.wHour, stLocalTime.wMinute, stLocalTime.wMilliseconds);
    }

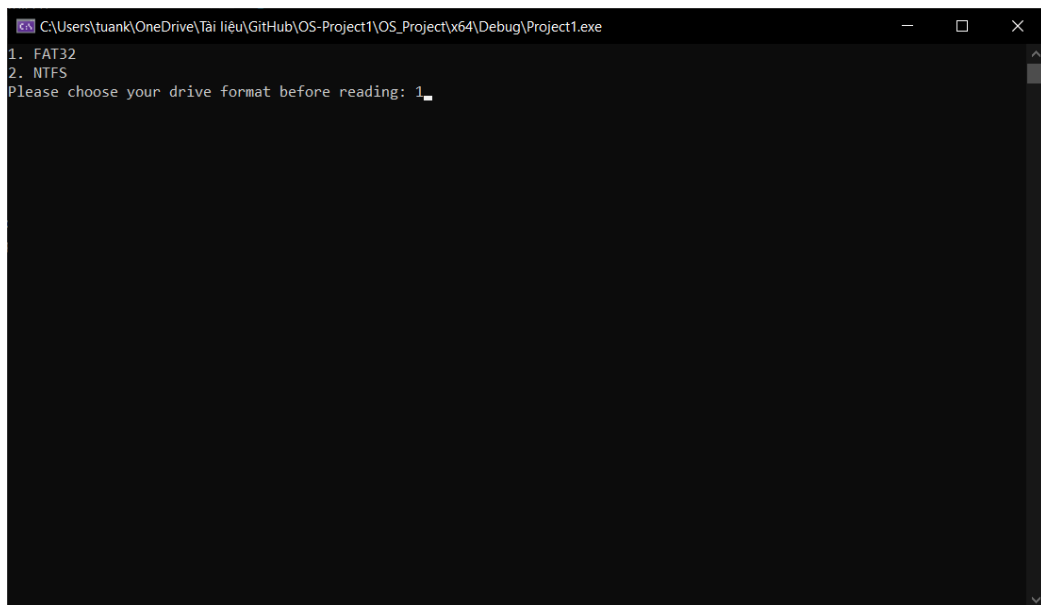
    std::cout << "Size on disk: " << fileNameAttribute->allocatedSize << " bytes\n";
    std::cout << std::endl;
}
```

4. Hình ảnh (chụp màn hình) demo chương trình ứng với các trường yêu cầu:

Đối với các ổ đĩa chính để không lỗi thì cần phải chạy visual studio dưới quyền admin, còn nếu đọc ổ đĩa rời như USB thì có thể chạy như bình thường. Ở đây sẽ demo đọc USB

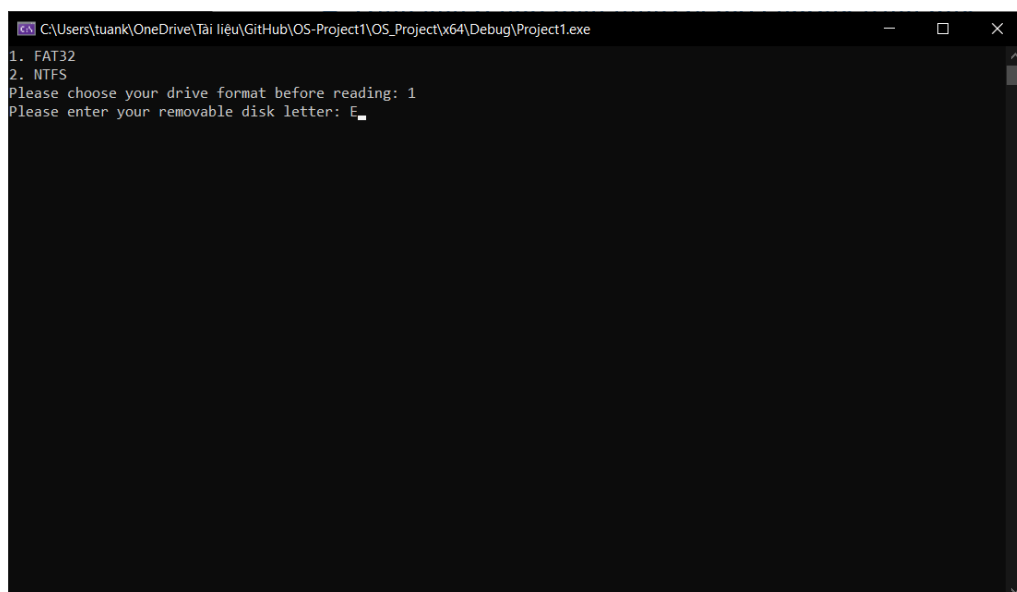
a. USB FAT32

Đối với USB ở format FAT32 vui lòng chọn option 1



```
C:\Users\tuank\OneDrive\Tài liệu\GitHub\OS-Project1\OS_Project\x64\Debug\Project1.exe
1. FAT32
2. NTFS
Please choose your drive format before reading: 1_
```

Hãy chọn drive letter của ổ đĩa USB. Ở đây USB là ổ E



```
C:\Users\tuank\OneDrive\Tài liệu\GitHub\OS-Project1\OS_Project\x64\Debug\Project1.exe
1. FAT32
2. NTFS
Please choose your drive format before reading: 1
Please enter your removable disk letter: E_
```

Chương trình sẽ hiển thị thông tin cây thư mục gốc và cây thư mục con cho người dùng

```
Microsoft Visual Studio Debug Console

1. FAT32
2. NTFS
Please choose your drive format before reading: 1
Please enter your removable disk letter: E

YORHA . <Volume Label>
Create Date: 0/0/1980
Modification Date: 7/7/2021
Modification Hours: 00:19:10
Accessed Date: 0/0/1980
Size: 0 bytes 0 cluster

System Volume Information
Create Date: 7/7/2021
Modification Date: 7/7/2021
Modification Hours: 00:19:10
Accessed Date: 7/7/2021
Size: 0 bytes 3 cluster

New folder <Directory>
Create Date: 4/7/2021
Modification Date: 7/7/2021
Modification Hours: 00:17:05
Accessed Date: 7/7/2021
Size: 0 bytes 5 cluster

|___ 19127102.PDF <Archive>
Create Date: 4/7/2021
Modification Date: 28/6/2021
Modification Hours: 20:06:02
```

```
Microsoft Visual Studio Debug Console

New folder <Directory>
Create Date: 4/7/2021
Modification Date: 7/7/2021
Modification Hours: 00:17:05
Accessed Date: 7/7/2021
Size: 0 bytes 5 cluster

|___ 19127102.PDF <Archive>
Create Date: 4/7/2021
Modification Date: 28/6/2021
Modification Hours: 20:06:02
Accessed Date: 7/7/2021
Size: 791255 bytes 6 cluster

|___ Test <Directory>
Create Date: 7/7/2021
Modification Date: 7/7/2021
Modification Hours: 00:17:05
Accessed Date: 7/7/2021
Size: 0 bytes 56 cluster

|___ BEAUZ - Outerspace (feat. Dallas).mp3 <Archive>
Create Date: 7/7/2021
Modification Date: 20/3/2021
Modification Hours: 01:03:01
Accessed Date: 7/7/2021
Size: 2787569 bytes 57 cluster

|___ Test.txt <Archive>
```

Nếu tập tin có phần mở rộng là txt thì dòng content sẽ hiển thị nội dung của file text

```
Microsoft Visual Studio Debug Console

Create Date: 5/7/2021
Modification Date: 7/7/2021
Modification Hours: 00:20:12
Accessed Date: 7/7/2021
Size: 36 bytes 55 cluster

Content:
Future YoRHa. Jinrui ni eikou are!

New folder (2) <Directory>
Create Date: 4/7/2021
Modification Date: 7/7/2021
Modification Hours: 00:17:05
Accessed Date: 7/7/2021
Size: 0 bytes 228 cluster

|___ BTCN04_19127102_TKM.png <Archive>
Create Date: 4/7/2021
Modification Date: 29/6/2021
Modification Hours: 00:37:05
Accessed Date: 7/7/2021
Size: 145193 bytes 229 cluster

New folder (3) <Directory>
Create Date: 4/7/2021
Modification Date: 4/7/2021
Modification Hours: 23:16:13
Accessed Date: 7/7/2021
Size: 0 bytes 238 cluster
```

```
Microsoft Visual Studio Debug Console

|___ BTCN04_19127102_TKM.png <Archive>
Create Date: 4/7/2021
Modification Date: 29/6/2021
Modification Hours: 00:37:05
Accessed Date: 7/7/2021
Size: 145193 bytes 229 cluster

New folder (3) <Directory>
Create Date: 4/7/2021
Modification Date: 4/7/2021
Modification Hours: 23:16:13
Accessed Date: 7/7/2021
Size: 0 bytes 238 cluster

19127102_19127406_19127457.docx <Archive>
Create Date: 4/7/2021
Modification Date: 4/7/2021
Modification Hours: 19:06:01
Accessed Date: 7/7/2021
Size: 4265795 bytes 239 cluster

C:\Users\snowl\OneDrive - VNU-HCMUS\Documents\GitHub\OS-Project1\OS_Project\x64\Debug\Project1.exe (process 14368) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

b. USB NTFS

Đối với ổ đĩa định dạng NTFS, hãy chọn 2, sau đó chọn drive letter của ổ đĩa USB. Ở đây USB là ổ E

```
Microsoft Visual Studio Debug Console

1. FAT32
2. NTFS
Please choose your drive format before reading: 2
Please enter your removable disk letter: E

System Volume Information
Creation Time: Wednesday, 7/7/2021, 15:08:408
Modification Time: Wednesday, 7/7/2021, 15:08:408
Size on disk: 0 bytes

WPSettings.dat
Creation Time: Wednesday, 7/7/2021, 15:08:422
Modification Time: Wednesday, 7/7/2021, 15:08:422
Size on disk: 0 bytes

New folder
Creation Time: Wednesday, 7/7/2021, 15:11:55
Modification Time: Wednesday, 7/7/2021, 15:11:55
Size on disk: 0 bytes

19127102.pdf
Creation Time: Wednesday, 7/7/2021, 15:11:73
Modification Time: Wednesday, 7/7/2021, 15:11:73
Size on disk: 794624 bytes

Test.txt
Creation Time: Wednesday, 7/7/2021, 15:11:150
Modification Time: Wednesday, 7/7/2021, 15:11:150
Size on disk: 0 bytes
```

```
Microsoft Visual Studio Debug Console

Test.txt
Creation Time: Wednesday, 7/7/2021, 15:11:150
Modification Time: Wednesday, 7/7/2021, 15:11:150
Size on disk: 0 bytes

Test
Creation Time: Wednesday, 7/7/2021, 15:11:167
Modification Time: Wednesday, 7/7/2021, 15:11:167
Size on disk: 0 bytes

BEAUZ - Outerspace (feat. Dallas).mp3
Creation Time: Wednesday, 7/7/2021, 15:11:180
Modification Time: Wednesday, 7/7/2021, 15:11:180
Size on disk: 2789376 bytes

New folder (2)
Creation Time: Wednesday, 7/7/2021, 15:11:421
Modification Time: Wednesday, 7/7/2021, 15:11:421
Size on disk: 0 bytes

BTCN04_19127102_TKM.png
Creation Time: Wednesday, 7/7/2021, 15:11:431
Modification Time: Wednesday, 7/7/2021, 15:11:431
Size on disk: 147456 bytes

New folder (3)
Creation Time: Wednesday, 7/7/2021, 15:11:469
Modification Time: Wednesday, 7/7/2021, 15:11:469
Size on disk: 0 bytes
```

```
Microsoft Visual Studio Debug Console
Modification Time: Wednesday, 7/7/2021, 15:11:421
Size on disk: 0 bytes

BTCN04_19127102_TKM.png
Creation Time: Wednesday, 7/7/2021, 15:11:431
Modification Time: Wednesday, 7/7/2021, 15:11:431
Size on disk: 147456 bytes

New folder (3)
Creation Time: Wednesday, 7/7/2021, 15:11:469
Modification Time: Wednesday, 7/7/2021, 15:11:469
Size on disk: 0 bytes

19127102_19127406_19127457.docx
Creation Time: Wednesday, 7/7/2021, 15:11:476
Modification Time: Wednesday, 7/7/2021, 15:11:476
Size on disk: 4268032 bytes

New Text Document.txt
Creation Time: Wednesday, 7/7/2021, 20:08:819
Modification Time: Wednesday, 7/7/2021, 20:08:819
Size on disk: 0 bytes

C:\Users\snowl\OneDrive - VNU-HCMUS\Documents\GitHub\OS-Project1\OS_Project\Debug\Project1.exe (process 956) exited
with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console
when debugging stops.
Press any key to close this window . . .
```

5. Nguồn tham khảo:

a. Vùng FAT32:

https://www.codeguru.com/cpp/cpp/cpp_mfc/files/article.php/c13831/FAT-Root-Directory-Structure-on-Floppy-Disk-and-File-Information.htm

https://dev-notes.eu/2019/07/Convert-an-Array-of-Unsigned-Chars-to-an-int32_t-Value/

http://www.cs.fsu.edu/~cop4610t/lectures/project3/Week11/Slides_week11.pdf

https://www.cse.scu.edu/~tschwarz/COEN252_09/Lectures/FAT.html

<http://www.disk-imager.com/data-recovery-blog/?p=1593>

<https://www.programmersought.com/article/8448841987/>

<https://stackoverflow.com/questions/29577114/read-boot-sector-using-c-on-windows8>

<https://lazytrick.wordpress.com/2015/12/27/khai-quạt-ve-fat/>

b. Vùng NTFS:

http://ntfs.com/ntfs_basics.htm

<https://github.com/orkblutt/NTFS-Parser-Lib>

http://inform.pucp.edu.pe/~inf232/Ntfs/ntfs_doc_v0.5/index.html

http://www.dewassoc.com/kbase/windows_nt/ntfs_directories_and_files.htm

<https://www.file-recovery.com/recovery-understanding-file-system-ntfs.htm>

<https://slidetodoc.com/computer-forensics-ntfs-file-system-mbr-and-gpt/>

https://handmade.network/wiki/7002-tutorial_parsing_the_mft