ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ TRI THỨC

BÁO CÁO ĐỒ ÁN

Quản lý hệ thống tập tin trên Windows

Môn học: Hệ Điều Hành

Nhóm: 11Cent
22127213 - Võ Minh Khôi
22127158 - Nhâm Đức Huy
22127129 - Nguyễn Tấn Hoàng
22127042 - Lê Nguyễn Minh Châu
22127283 - Phan Hải Minh

Giáo viên hướng dẫn:
Ths. Lê Viết Long

TP. Hồ Chí Minh, ngày 13 tháng 3 năm 2024

Mục lục

1 Thông tin nhóm 2
2 Bảng phân công công việc 2
3 Đánh giá mức độ hoàn thành 3
4 Mô tả các bước thực hiện 4
1. FAT32 4
a. Tổng quan cấu trúc sourcecode 4
b. Đọc thông tin chi tiết của một phân vùng 7
c. Hiển thị thông tin cây thư mục của một phân vùng 8
d. Đọc nội dung tập tin có phần mở rộng ".txt" và hiển thị thông báo đối với các tập tin có định dạng khác 12
2. NTFS 13
a. Tổng quan cấu trúc sourcecode 13
b. Đọc thông tin chi tiết của một phân vùng 15
c. Hiển thị thông tin cây thư mục của một phân vùng 15
d. Đọc nội dung tập tin có đuôi "txt" và hiển thị thông báo đối với các tập tin có định dạng khác 18
3. Menu 19
a. Tổng quan cấu trúc sourcecode 19
5 Hình ảnh demo chương trình 20
6 Nguồn tham khảo 24

1 Thông tin nhóm

Nhóm: 11Cent

STT	Họ và Tên	MSSV	Email	Mức độ đóng góp
1	Võ Minh Khôi	22127213	Vmkhoi22@clc.fitus.edu.vn	20%
2	Nhâm Đức Huy	22127158	Ndhuy22@clc.fitus.edu.vn	25%
3	Phan Hải Minh	22127283	Phminh22@clc.fitus.edu.vn	5%
4	Nguyễn Tấn Hoàng	22127129	Nthoang22@clc.fitus.edu.vn	25%
5	Lê Nguyễn Minh Châu	22127042	Lnmchau22@clc.fitus.edu.vn	25%

2 Bảng phân công công việc

STT	Công việc	Mô tả	Người thực hiện
1	Viết báo cáo	Thiết kế bố cục, tổng hợp nội dung và chỉnh sửa, đảm bảo chất lượng báo cáo	Võ Minh Khôi
2	Tổng hợp và kiểm tra	Tổng hợp và kiểm tra tính chính xác của nội dung, chỉnh sửa bố cục báo cáo	Nhâm Đức Huy
3	Thiết kế giao diện	Thiết kế và cài đặt giao diện console của ứng dụng	Phan Hải Minh
4	FAT32 – Đọc thông tin phân vùng	Cài đặt chức năng đọc thông tin phân vùng FAT32	Nguyễn Tấn Hoàng
5	FAT32 – Hiển thị thông tin cây thư mục của phân vùng	Cài đặt chức năng hiển thị thông tin cây thư mục của phân vùng FAT32	Nguyễn Tấn Hoàng
6	FAT32 – Hiển thị nội dung tập tin .txt và thông báo dùng phần mềm tương thích	Cài đặt chức năng hiển thị nội dung tập tin có phần mở rộng .txt và thông báo dùng phần mềm tương thích để đọc nội dung	Nguyễn Tấn Hoàng
7	NTFS – Đọc thông tin phân vùng	Cài đặt chức năng đọc thông tin phân vùng NTFS	Lê Nguyễn Minh Châu

8	NTFS – Hiển thị thông tin cây thư mục của phân vùng	Cài đặt chức năng hiển thị thông tin cây thư mục của phân vùng NTFS	Lê Nguyễn Minh Châu
9	NTFS – Hiển thị nội dung tập tin .txt và thông báo dùng phần mềm tương thích	Cài đặt chức năng hiển thị nội dung tập tin có phần mở rộng .txt và thông báo dùng phần mềm tương thích để đọc nội dung	Lê Nguyễn Minh Châu

3 Đánh giá mức độ hoàn thành

STT	Yêu cầu		Mức độ hoàn thành
1	Đọc thông tin chi tiết của một phân vùng	FAT32	100%
		NTFS	100%
2	Chương trình hiển thị cây thư mục gốc gồm tên tập tin/thư mục, trạng thái, kích thước (nếu có), chỉ số sector lưu trữ trên đĩa cứng	FAT32	100%
		NTFS	100%
3	Hiển thị nội dung tập tin đối với tập tin có phần mở rộng là .txt	FAT32	100%
		NTFS	100%
4	Hiển thị thông báo dùng phần mềm tương thích để đọc nội dung đối với tập tin có phần mở rộng khác .txt	FAT32	100%
		NTFS	100%
5	Hiển thị đối tượng là cây thư mục con với trường hợp đối tượng là thư mục	FAT32	100%
		NTFS	100%
	Mức độ hoàn thành Project		**100%**

4 Mô tả các bước thực hiện

1. FAT32

a. Tổng quan cấu trúc sourcecode

- Tên file: *FAT32.py*
- Các class chính:
  - ***Attribute**:* Chứa các thuộc tính của tập tin hoặc thư mục trong hệ thống tập tin FAT32. Định nghĩa một số flags như READ_ONLY *(chỉ đọc)*, HIDDEN *(ẩn)*, SYSTEM *(hệ thống)*, VOLUME_ID *(phân vùng)*, DIRECTORY *(thư mục),* ARCHIVE *(nén)* và LONG_NAME *(tên dài)*

```
class Attribute(Flag):
    READ_ONLY = 0x01
    HIDDEN = 0x02
    SYSTEM = 0x04
    VOLUME_ID = 0x08
    DIRECTORY = 0x10
    ARCHIVE = 0x20
    LONG_NAME = 0x0F
```

  - ***Status**:* Các trạng thái của tập tin hoặc thư mục trong hệ thống file tập tin FAT32. Định nghĩa các flags như DELETED *(đã xóa),* EMPTY *(trống),* NORMAL *(bình thường)*

```
class Status(Flag):
    DELETED = 0xE5
    EMPTY = 0x00
    NORMAL = 0xFF
```

  - ***BootSector**:* Phần boot sector của hệ thống tập tin FAT32. Phân tích và chứa các thông tin quan trọng chứa trong boot sector và cung cấp các hàm bổ sung như tính toán vị trí offset bắt đầu của bảng RDET và offset từ một cluster index cho trước

```
class BootSector:
    def __init__(self, data, name):
        self.oem_name
        self.bytes_per_sector
        self.sectors_per_cluster
        self.reserved_sectors
        self.fat_count
        self.total_sectors
        self.fat_size
        self.root_cluster
        self.volume_label
        self.fat_type
        self.boot_code
        self.boot_signature
        self.RDET_start

    def offset_from_cluster(self, cluster_idx)

    def __str__(self)
```

- ***FAT:*** Đại diện cho Bảng Cấp phát Tệp *(File Allocation Table – FAT)* của hệ thống tệp FAT32. Phân tích và lưu trữ dữ liệu FAT, được sử dụng để theo dõi việc cấp phát và liên kết các cluster. Đồng thời cung cấp các phương thức để lấy chuỗi cluster *(cluster chain)* từ một cluster index cho trước

```
class FAT:
    def __init__(self, data):
        self.data
        self.size
        self.FAT

    def get_cluster_chain(self, start)
```

- ***Entry:*** Đại diện cho mục tệp hoặc thư mục trong hệ thống tệp FAT32. Phân tích và lưu trữ các thuộc tính của một entry

```
class Entry:
    def __init__(self, data):
        self.name
        self.longFileName
        self.attr
        self.status
        self.reserved
        self.create_time
        self.create_date
        self.last_access_date
        self.last_write_time
        self.last_write_date
        self.starting_cluster
        self.file_size
        self.extension

    def __str__(self)
```

- ***RDET:*** Đại diện cho Bảng Mục Thư mục Gốc *(Root Directory Entry Table – RDET)* của hệ thống tệp FAT32. Đọc và lưu trữ các entry trong RDET, chứa thông tin về các tệp và thư mục trong thư mục gốc. Cung cấp các hàm để tìm một mục cụ thể theo tên và lấy tổng số mục

```
class RDET:
    def __init__(self, pointer, offset, bytes_per_sector):
        self.entries
        self.size
        self.sector
        self.read_entries(pointer, bytes_per_sector)

    def read_entries(self, pointer, bytes_per_sector=512)
    def find_entry(self, name)
    def __str__(self)
    def read_chain(pointer, starting_cluster, sectors_per_cluster, bytes_per_sector, fat,
RDET_start)
```

- ***SDET**: Đại diện cho Bảng Mục Thư mục Cụm (Sub-Directory Entry Table – SDET)* của hệ thống tập tin FAT32. Đọc và lưu trữ các entry trong một CDET, chứa thông tin về các tập tin và thư mục trong một cluster cụ thể. Cung cấp các phương thức để tìm một entry cụ thể

```
class SDET:
    def __init__(self, data):
        self.entries
        self.read_entries(data)

    def read_entries(self, data)
    def find_entry(self, name):
    def __str__(self)
```

- ***Node**: Đại diện cho một node trong cấu trúc cây thư mục của hệ thống tệp FAT32. Mỗi node tương ứng một tệp hoặc thư mục và chứa thông tin như node cha, node con, thông tin entry và đường dẫn thư mục. Cung cấp phương thức để đặt tên cho node*

```
class Node:
    def __init__(self, dir = None, entry = None, isRoot = False):
        self.isRoot
        self.parent
        self.children
        self.info = entry
        self.dir = dir
    def setName(self, name)
```

- ***FAT32**: Đại diện cho toàn bộ hệ thống tệp FAT32. Khởi tạo các thành phần cần thiết như boot sector, RDET và node gốc. Cung cấp các phương thức như duyệt cây thư mục, đọc nội dung của tệp .txt và hiển thị cấu trúc cây thư mục*

```
class FAT32:
    def __init__(self, name):
        self.name
        self.ptr
        self.RDET
        self.root

    def offset_from_cluster(self, cluster_index)
    def vis(self, start_cluster, dir, parRoot = None)
    def get_dir_tree(self, start_cluster, curRoot)
    def read_txt_file(self, txtNode)
    def draw_dir_tree(self, curNode, depth = 0)
```

b. Đọc thông tin chi tiết của một phân vùng

- Khởi tạo: Khởi tạo đối tượng *BootSector* với các tham số *data* là dữ liệu 512 bytes đầu tiên và *name* là tên của phân vùng (*C:, D:,…*) được chọn, constructor của class là *__init__* sẽ được thực thi để phân tích 512 bytes *data* chứa thông tin của phân vùng *name:*

```
class BootSector:
    def __init__(self, data, name):
        self.oem_name = data[3:11]
        self.bytes_per_sector = int.from_bytes(data[11:13], byteorder='little')
        self.sectors_per_cluster = int.from_bytes(data[13:14], byteorder='little')
        self.reserved_sectors = int.from_bytes(data[14:16], byteorder='little')
        self.fat_count = int.from_bytes(data[16:17], byteorder='little')
        self.total_sectors = int.from_bytes(data[32:36], byteorder='little')
        self.fat_size = int.from_bytes(data[36:40], byteorder='little')
        self.root_cluster = int.from_bytes(data[44:48], byteorder='little')
        self.volume_label = name
        self.fat_type = data[54:62]
        self.boot_code = data[62:510]
        self.boot_signature = data[510:512]
        self.RDET_start = self.reserved_sectors + self.fat_count * self.fat_size

    def offset_from_cluster(self, cluster_idx):
        offset = self.reserved_sectors + (self.fat_size * self.fat_count) + ((cluster_idx - 2) *
self.sectors_per_cluster)
        return offset

    def __str__(self):
        return f'{self.oem_name.decode("utf-8").strip()}'
```

- Các giá trị, thông tin của phân vùng được đọc từ boot sector gồm có:
  - **self.oem_name:** Tên OEM *(Original Equipment Manufacturer)*
  - **self.bytes_per_sector:** Số byte trên mỗi sector
  - **self.sectors_per_cluster:** Số sector trên mỗi cluster
  - **self.reserved_sectors:** Số sector dành riêng
  - **self.fat_count:** Số bảng FAT
  - **self.total_sectors:** Tổng số sector trên phân vùng
  - **self.fat_size:** Kích cỡ bảng FAT
  - **self.volume_label:** Tên phân vùng
  - **self.fat_type:** Loại bảng FAT
  - **self.boot_code:** Boot code
  - **self.boot_signature:** Boot signature
  - **self.RDET_start:** Địa chỉ bắt đầu của RDET
  - **offset_from_cluster():** Tính offset dữ liệu từ cluster index cho trước
- Bảng FAT được sử dụng để theo dõi vị trí và tổ chức của các tệp trên thiết bị lưu trữ. Việc đọc và lưu bảng FAT vào bộ nhớ đệm được thực hiện trong hàm **__init__** của class **FAT**:
  - Hàm **__init__** nhận dữ liệu của bảng FAT thông qua tham số *data*
  - Kích thước của bảng FAT được lưu trong biến **self.size** và được tính bằng cách lấy độ dài của *data* chia 4 (mỗi entry trong bảng FAT chiếm 4 bytes)
  - Một danh sách rỗng **self.FAT** được tạo để lưu trữ các entry của bảng FAT
  - Sử dụng vòng lặp for để duyệt qua dữ liệu data với bước nhảy là 4:

- Trong mỗi lần lặp, đoạn dữ liệu từ vị trí *i* đến *i+4* được trích xuất bằng cách sử dụng cú pháp cắt *data[i:i+4]*
- Đoạn dữ liệu này được chuyển đổi thành một số nguyên sử dụng phương thức *int.from_bytes()* với đối số *byteorder='little'* để chỉ định thứ tự byte là little-endian
- Số nguyên này đại diện cho giá trị của một mục trong bảng FAT và được thêm vào danh sách **self.FAT** bằng phương thức *append()*

- Sau khi vòng lặp kết thúc, danh sách **self.FAT** sẽ chứa tất cả các entry trong bảng FAT, mỗi entry là một số nguyên 4 byte.
- Khi cần tìm cluster chain của một tệp tin, phương thức *get_cluster_chain()* của lớp FAT được sử dụng. Phương thức này sử dụng danh sách **self.FAT** đã được lưu trong bộ nhớ để truy cập các entry liên tiếp trong bảng FAT và xây dựng cluster chain tương ứng.

```
class FAT:
    def __init__(self, data):
        self.data = data
        self.size = len(data) / 4
        self.FAT = []
        for i in range(0, len(data), 4):
            self.FAT.append(int.from_bytes(data[i:i+4], byteorder='little'))

    def get_cluster_chain(self, start):
        chain = []
        while True:
            chain.append(start)
            if start >= 0x0FFFFFF8:
                break
            start = self.FAT[start]
        return chain
```

c. Hiển thị thông tin cây thư mục của một phân vùng

- Thông tin về cây thư mục của phân vùng được lưu trong class **FAT32** bằng thuộc tính **self.RDET.** Thuộc tính này chứa các entry đại diện cho các tập tin và thư mục nằm trực tiếp trong thư mục gốc của phân vùng.
- Đọc thư mục gốc (RDET):
  - RDET bắt đầu ngay sau các sector dành riêng (reserved sectors) và các bảng sao của bảng FAT. Vị trí bắt đầu của RDET được tính trong constructor **__init__** của class **BootSector:**

```
self.RDET_start = self.reserved_sectors + self.fat_count * self.fat_size
```

- Đọc các entry trong RDET:

```
def read_entries(self, pointer, bytes_per_sector=512):
        nameBuffer = ''
        self.sector = 0
        while True:
            data = pointer.read(bytes_per_sector)
            self.sector += 1
            for i in range(0, len(data), 32):
                entry = Entry(data[i:i+32])
                if entry.status == Status.EMPTY:
                    self.size = len(self.entries)
                    return
                if entry.status == Status.DELETED:
                    continue
                if entry.attr == Attribute.LONG_NAME:
                    string = data[i+1:i+11] + data[i+14:i+26] + data[i+28:i+32]
                    for i in range(0, len(string), 2):
                        if string[i:i+2] == b'\xff\xff':
                            string = string[:i]
                    nameBuffer = string.decode('utf-16-le') + nameBuffer
                    continue
                if nameBuffer:
                    entry.longFileName = nameBuffer
                    nameBuffer = ''
                self.entries.append(entry)
```

  - Mỗi entry trong RDET có kích thước cố định là 32 byte
  - Đọc từng entry cho đến khi gặp entry rỗng hoặc đã đọc hết entry trong RDET
- Xử lý các loại entry:
  - Có 3 loại entry chính trong RDET:
    - Entry thư mục (directory entry): Chứa thông tin về thư mục con
    - Entry tập tin (file entry): Chứa thông tin về tập tin
    - Entry tên dài (long name entry): Chứa phần tên dài của tệp tin hoặc thư mục
      - Cần xử lý từng loại entry tương ứng dựa trên giá trị cỉa trường *attr* (thuộc tính) trong entry
      - Nếu entry là tên dài, cần ghép các phần tên lại để tạo thành tên dài hoàn chỉnh.
      - Việc xử lý các loại entry được thực hiện trong phương thức *read_entries:*

```
if entry.status == Status.EMPTY:
    self.size = len(self.entries)
    return
if entry.status == Status.DELETED:
    continue
if entry.attr == Attribute.LONG_NAME:
    string = data[i+1:i+11] + data[i+14:i+26] + data[i+28:i+32]
    for i in range(0, len(string), 2):
        if string[i:i+2] == b'\xff\xff':
            string = string[:i]
    nameBuffer = string.decode('utf-16-le') + nameBuffer
    continue
```

- Lưu thông tin về các entry:
  - Sau khi đọc và xử lý, lưu trữ thông tin về các entry trong danh sách **self.entries** để sử dụng cho các thao tác tiếp theo
  - Mỗi phần tử là một đối tượng của class **Entry**

- Việc xây dựng và hiển thị cây thư mục được thực hiện thông qua class **FAT32:**
  - Khởi tạo đối tượng **FAT32:**

```
class FAT32:
    def __init__(self, name):
        self.name = name
        with open(f'\\\\.\\{self.name}:', 'rb') as f:
            self.data = f.read(512)
            self.boot_sector = BootSector(data, self.name + ':')
        self.ptr = open(f'\\\\.\\{self.name}:', 'rb')
        self.RDET = RDET(self.ptr, self.boot_sector.RDET_start *
self.boot_sector.bytes_per_sector, self.boot_sector.bytes_per_sector)
        self.root = Node(entry = None, isRoot = True)
        self.curNode = self.root
```

    - Khởi tạo bằng constructor **__init__** với tham số *name* là tên phân vùng, *name* được sử dụng để đọc boot sector của ổ đĩa và lưu thông tin vào thuộc tính **self.boot_sector.**
    - Khởi tạo đối tượng **RDET** bằng cách truyền con trỏ tập tin, vị trí bắt đầu của RDET và kích thước của một sector.
    - Khởi tạo một node gốc (root node) cho cây thư mục.
  - Xây dựng cây thư mục:
    - Sử dụng phương thức *get_dir_tree:*

```
    def get_dir_tree(self):
        self.root = Node(entry = None, isRoot = True)
        self.vis(0, f'\\\\.\\{self.name}:')
        self.curNode = self.root
```

- Trong phương thức này, gọi đệ quy hàm *vis* để duyệt qua các thư mục và tập tin trong hệ thống FAT32:

```
    def vis(self, start_cluster, dir, parRoot = None):
        if (parRoot == None):
            parRoot = self.root
        curEntry = []
        if (parRoot == self.root):
            curEntry = self.RDET.entries
        else:
            tmp = read_chain(self.ptr, start_cluster, boot_sector.sectors_per_cluster,
boot_sector.bytes_per_sector, fat, boot_sector.RDET_start)
            curEntry = SDET(tmp).entries
        for i in curEntry:
            if (i.starting_cluster == start_cluster):
                continue
            if (i.name.strip() == b'.' or i.name.strip() == b'..'):
                continue
            if ((not (i.attr & Attribute.DIRECTORY)) and (not (i.attr &
Attribute.ARCHIVE))):
                continue
            if ((i.attr & Attribute.HIDDEN)):
                continue
            if ((i.status == Status.DELETED) or (i.status == Status.EMPTY)):
                continue
            curNode = Node(entry = i)
            curNode.parent = parRoot
            parRoot.children.append(curNode)
            tmpDir = dir
            if (i.longFileName != ''):
                extension = curNode.info.extension.decode().strip()
                if (extension != ''):
                    extension = '.' + extension
                curNode.setName(i.longFileName + extension)
                tmpDir = dir + '\\' + i.longFileName + extension
            else:
                extension = curNode.info.extension.decode().strip()
                curName = i.name.decode().strip(extension).strip()
                if (extension != ''):
                    extension = '.' + extension
                curNode.setName(curName + extension)
                tmpDir = dir + '\\' + curName + extension
            print('file name: ', curNode.name, curNode.info.attr)
            if (i.attr & Attribute.DIRECTORY):
                self.vis(i.starting_cluster, tmpDir, curNode)
```

- Lấy danh sách các entry trong thư mục hiện tại (RDET hoặc SDET) dựa trên cluster bắt đầu. Sau đó, với mỗi entry trong danh sách:
  - Nếu entry là thư mục hoặc tập tin hợp lệ (không phải các entry đặc biệt như "." hoặc ".."), tạo một node mới.
  - Nếu entry có tên dài (longFileName), sử dụng tên dài làm tên node. Ngược lại, sử dụng tên ngắn (name) và phần mở rộng (extension) để tạo tên node.
  - Nếu entry là một thư mục, gọi đệ quy hàm *vis* với cluster bắt đầu của thư mục con và node hiện tại làm node cha.
- Lặp lại đến khi duyệt hết các thư mục và tập tin trong hệ thống tập tin

- Hiển thị cây thư mục:
  - Sử dụng phương thức *draw_dir_tree* để vẽ cây thư mục:

```
    def draw_dir_tree(self, curNode, depth = 0):
        if (depth == 0):
            print(self.name + ':')
        for child in curNode.children:
            for i in range(depth + 1):
                print('--', end = '')
            print(child.name)
            if (child.info.attr & Attribute.DIRECTORY):
                self.draw_dir_tree(curNode = child, depth = depth + 1)
```

  - Phương thức này nhận vào node hiện tại và độ sâu (depth) của node trong cây thư mục.
  - Với mỗi node con của node hiện tại:
    - Hiển thị các dấu "--" tương ứng với độ sâu để thể hiện cấu trúc cây thư mục.
    - Nếu node con là một thư mục, gọi đệ quy phương thức *draw_dir_tree()* với node con và độ sâu tăng lên 1.
    - Quá trình này được lặp lại cho đến khi duyệt hết các node trong cây thư mục.

d. Đọc nội dung tập tin có phần mở rộng ".txt" và hiển thị thông báo đối với các tập tin có định dạng khác

- Việc đọc nội dung tập tin có phần mở rộng ".txt" và hiển thị thông báo đối với các tập tin có định dạng khác được thực hiện trong phương thức *printFile* của lớp **FAT32**:

```
def printFile(self, txtNode):
        rawData = read_chain(self.ptr, txtNode.info.starting_cluster,
self.boot_sector.sectors_per_cluster, self.boot_sector.bytes_per_sector, fat,
self.boot_sector.RDET_start)
        textSize = txtNode.info.file_size
        fileContent = rawData[:textSize]
        fileName = txtNode.name
        if (fileName.lower().endswith('.txt')):
            print(fileContent.decode('utf-8', errors = 'replace'))
        elif (fileName.lower().endswith('.docx')):
            print('Please use MS Word to open this file!')
        elif (fileName.lower().endswith('.pdf')):
            print('Please use Adobe Acrobat Reader to open this file!')
        elif (fileName.lower().endswith('.png')):
            print('Please use an image viewer to open this file!')
        elif (fileName.lower().endswith('.jpg')):
            print('Please use an image viewer to open this file!')
        elif (fileName.lower().endswith('.jpeg')):
            print('Please use an image viewer to open this file!')
        elif (fileName.lower().endswith('.gif')):
            print('Please use an image viewer to open this file!')
        elif (fileName.lower().endswith('.mp4')):
            print('Please use a video player to open this file!')
        elif (fileName.lower().endswith('.mp3')):
            print('Please use a music player to open this file!')
        elif (fileName.lower().endswith('.cpp')):
            print('Please use a code editor to open this file!')
        elif (fileName.lower().endswith('.c')):
            print('Please use a code editor to open this file!')
        elif (fileName.lower().endswith('.java')):
            print('Please use a code editor to open this file!')
        else:
            print('Please use an appropriate program to open this file!')
```

- Phương thức *printFile* nhận vào một tham số txtNode, đại diện cho một node (tập tin) trong cây thư mục.
- Lấy tên của tập tin bằng cách sử dụng thuộc tính *name* của đối tượng *txtNode*
- Kiểm tra phần mở rộng của tập tin bằng phương thức *endswith* của chuỗi tên tập tin. Nếu phần mở rộng là ".txt", in nội dung của tập tin bằng cách giải mã dữ liệu thô thành chuỗi Unicode và in ra.
- Đối với các tập tin có định dạng khác, hiển thị thông báo tương ứng cho từng loại tập tin.

2. NTFS

a. Tổng quan cấu trúc sourcecode

- Tên file: NTFS.py
- Các class chính:
  - **BPB**: Đọc các thông tin chi tiết của một phân vùng chứa trong BPB. Đối tượng BPB được khởi tạo với đối số là con trỏ file và đường dẫn đến ổ đĩa mà người dùng nhập vào.

```
class BPB:
    def __init__(self, ptr, name):
        self.name
        self.ptr
        self.data
        self.byte_per_sector
        self.sector_per_cluster
        self.sector_per_track
        self.number_of_sector
        self.MFT_start_sector
        self.MFT_reserve_start_sector
```

  - **Entry**: Đối tượng khởi tạo để lưu trữ thông tin của các MFT Entry. Đối tượng này lưu trữ các thông tin cơ bản như**:**

```
class Entry:
    def __init__(self, parDirectory, name, timeCreated, timeAccessed, timeModified,
fileContent):
        self.isFolder
        self.name
        self.timeCreated
        self.timeAccessed
        self.timeModified
        self.parDir
        self.fileContent
```

    - **self.isFolder**: Nhận dạng xem MFT Entry hiện tại có phải là một thư mục hay không.
    - **self.name**: tên của tập tin/thư mục
    - **self.timeCreated**: thời gian khởi tạo
    - **self.timeAccessed**: thời gian truy cập gần đây nhất
    - **self.timeModified**: thời gian chỉnh sửa gần đây nhất
    - **self.parDir**: thư mục cha (nếu có)
    - **self.fileContent**: nội dung của tập tin (nếu có)
  - **Node**: Đối tượng khởi tạo để lưu trữ các nút của cây thư mục. Đối tượng này lưu trữ các thông tin cơ bản như:

```
class Node:
    def __init__(self, entry, parent, address):
        self.entry
        self.parent
        self.children
        self.address

    def __str__(self)
```

- **self.entry**: Tất cả các thông tin của entry mà tập tin/thư mục này thuộc về
- **self.parent**: nút cha của nút hiện tại
- **self.children**: mảng chứa các nút con của nút hiện tại
- **self.address**: Offset của MFT Entry mà nút này thuộc về trong MBR (Master Boot Record)

- **NTFS**: Đây là đối tượng chính của chương trình quản lí hệ thống tập tin định dạng NTFS. Đối tượng này lưu trữ các thông tin cơ bản như:

```
class NTFS:
    def __init__(self, name):
        self.name
        self.root
        self.curNode
        self.ptr

    def get_info(self)
    def clusterToSectorList(self)
    def readEntry(self)
    def getDirTree(self, curNode, depth)
    def getDir(self)
    def listDir(self)
    def moveIntoDir(self)
    def printFile(self, txtNode)
    def readFile(self)
```

- **self.name**: tên ổ đĩa mà người dùng muốn truy vấn
- **self.root**: gốc của cây thư mục
- **self.curNode**: nút hiện tại
- **self.ptr**: con trỏ đọc thông tin phân vùng

b. Đọc thông tin chi tiết của một phân vùng

Sử dụng đối tượng NTFS và BPB

- Khởi tạo đối tượng **BPB**:
  - Một đối tượng BPB được khởi tạo với đối số **self.ptr** và **self.name** là con trỏ để đọc thông tin phân vùng và tên ổ đĩa được chọn để trích xuất thông tin.

```
class NTFS:
    def __init__(self):
        self.name = name
        self.root = None
        self.curNode = None
        self.map = {}
        self.ptr = open(f'\\\\.\\{self.name}:', 'rb')
        with open(f'\\\\.\\{self.name}:','rb') as f:
              self. BPB = BPB(self.ptr, self.name + ':')
        self.readEntry()
```

- Đọc các thông số từ Boot Sector:
  - Dãy byte được trích xuất từ phân vùng **VBR** và lưu vào mảng **self.data**. Sau khi đọc thông tin từ các offset, dãy byte được đảo ngược (được chuyển thành little endian) bằng đối số **byteorder = 'little'.** Điều này đảm bảo rằng byte thấp sẽ đứng trước.
  - Dãy byte little endian sau khi đảo ngược được chuyển thành một số nguyên bằng cách ép kiểu **int.**
  - Các giá trị thông tin của phần vùng được đọc từ BPB gồm có:
    - **self.byte_per_sector**: Số byte của mỗi sector
    - **self.sector_per_cluster**: Số sector của mỗi cluster
    - **self.sector_per_track**: Số sector mỗi track
    - **self.number_of_sector**: Số lượng phân vùng của sector
  - Riêng:
    - **self.MFT_start_sector**: Sector bắt đầu của MFT (Master File Table)
    - **self.MFT_reserve_start_sector**: Sector bắt đầu của MFT dự phòng
  - Được tính trực tiếp bằng cách lấy **self.sector_per_cluster** nhân với cluster bắt đầu của MFT.

```
class NTFS:
    def get_info(self):
        print('sector_size: ', self.BPB.byte_per_sector)
        print('sector_per_cluster: ', self.BPB.sector_per_cluster)
        print('sector per track: ', self.BPB.sector_per_track)
        print('number_of_sector: ', self.BPB.number_of_sector)
```

c. Hiển thị thông tin cây thư mục của một phân vùng

Sử dụng đối tượng **NTFS** và các hàm:

- **readEntry()**: Hàm dùng để đọc thông tin từng entry theo các bước:
  - Đọc lần lượt các MFT Entry trong NTFS Volume, với mỗi entry có độ lớn 1024 bytes, tương ứng 2 sectors, chương trình sẽ đọc mỗi 2 sector cho đến khi số sector vượt quá số sector của Volume.

- Đọc chữ ký của MFT Entry để nhận diện nếu MFT Entry này có bắt đầu bằng một chữ kí (signature) "FILE" hay không, nếu chữ kí này được tìm thấy, mã sẽ thực hiện đọc thông tin về thư mục hoặc tệp tương ứng, nếu không sẽ bỏ qua và đọc 2 sector tiếp theo.
- Đọc thông tin về cờ của Entry hiện tại từ offset 0x16 –> 0x18:

Flag	Description
0x01	Record is in use
0x02	Record is a directory (FileName index present)
0x04	Record is an exension (Set for records in the $Extend directory)
0x08	Special index present (Set for non-directory records containing an index: $Secure, $ObjID, $Quota, $Reparse)

Trường hợp cờ báo đang không trong tình trạng sử dụng và đồng thời không phải là thư mục, mã sẽ bỏ qua và thực hiện đọc Entry tiếp theo.

- Sau khi có thông tin offset của Attribute đầu tiên, mã sẽ tiến hành đọc thông tin các Attribute của Entry hiện tại:
  Quy trình đọc thông tin các Attribute:
  - Trích xuất các thông tin cơ bản từ header của Attribute.

```
#get attribute type
attrType = int.from_bytes(data[attrOffset:attrOffset + 4], byteorder = 'little')
 #break if end of attribute
if (attrType == 0xFFFFFFFF or attrType == 0x0):
        break

#get attribute length
attrLength = int.from_bytes(data[attrOffset + 4:attrOffset + 8], byteorder='little')

#get attribute type (resident/non-resident)
attrResident = int.from_bytes(data[attrOffset + 8:attrOffset + 9], byteorder='little')
isResident = True
if(attrResident == 1):  #non-resident
        isResident = False
if(attrResident > 1):
        break

#get content's size of the attribute
attrContentSize = int.from_bytes(data[attrOffset + 16:attrOffset + 20], byteorder='little')

#get attribute's content's offset
attrContentOffset = int.from_bytes(data[attrOffset + 20:attrOffset + 21],
byteorder='little')
```

  - Trích xuất các thông tin từ nội dung của Attribute.
- Sau khi đã xác định được loại Attribute nào, mã sẽ tiến hàng đọc dựa trên các trường hợp.

- Loại **$STANDARD_INFOMATION**
  - Trích xuất cờ từ offset 0x20 → 0x24 để kiểm tra xem tập tin/thư mục hiện tại thuộc loại nào.

Flag	Description
0x0001	Read-Only
0x0002	Hidden
0x0004	System
0x0020	Archive

- Loại **$FILE_NAME**
  - Trích xuất các thông tin như tên của tệp/thư mục, thư mục cha (nếu có), thời gian tạo, thời gian chỉnh sửa gần đây, thời gian truy cập gần đây.

```
#get file name
nameLength = int.from_bytes(data[attrOffset + attrContentOffset + 64:attrOffset +
attrContentOffset + 65], byteorder='little')
fileName = data[attrOffset + attrContentOffset + 66:attrOffset + attrContentOffset + 66
+ nameLength * 2]
fileName = fileName.decode('utf-16le')

if(fileName.startswith('$')):
    break
 #get parent directory
parDir = int.from_bytes(data[attrOffset + attrContentOffset + 0:attrOffset +
attrContentOffset + 6], byteorder='little')
parDir = hex(parDir)

parDir2 = int.from_bytes(data[attrOffset + attrContentOffset + 6:attrOffset +
attrContentOffset + 8], byteorder='little')
parDir2 = hex(parDir2)

#get file create time
createTime = int.from_bytes(data[attrOffset + attrContentOffset + 8:attrOffset +
attrContentOffset + 16], byteorder='little')
createTime = convertToTime(createTime)

#get file last modified time
modifiedTime = int.from_bytes(data[attrOffset + attrContentOffset + 16:attrOffset +
attrContentOffset + 24], byteorder='little')
modifiedTime = convertToTime(modifiedTime)

#get file last accessed time
accessedTime = int.from_bytes(data[attrOffset + attrContentOffset + 32:attrOffset +
attrContentOffset + 40], byteorder='little')
accessedTime = convertToTime(accessedTime)
```

- Loại **$DATA**
  - Trích xuất các thông tin về cờ resident: Nếu cờ báo là resident thì đọc tiếp nội dung của file, nếu là non-resident thì đọc clusters bắt đầu nơi chứa nội dung và bắt đầu trích xuất nội dung của file.

```
#in case resident attribute
  if(isResident):
     fileSize = attrContentSize
     fileContent = data[attrOffset + attrContentOffset:attrOffset + attrContentOffset +
fileSize]
     fileContent = fileContent.decode('utf-8', errors = 'replace')
     filePermission = data[attrOffset + attrContentOffset + 0x20:attrOffset +
attrContentOffset + 0x2d]
 #in case non-resident
else:
     dataRunOffset = int.from_bytes(data[attrOffset + 32:attrOffset + 34],
byteorder='little')
     fileSize = int.from_bytes(data[attrOffset + 48:attrOffset + 55],
byteorder='little')
        if fileName.lower().endswith('.txt'):
           numberOfCluster = int.from_bytes(data[attrOffset + dataRunOffset +
1:attrOffset +        dataRunOffset + 2], byteorder='little')
           startClusterIndex = int.from_bytes(data[attrOffset + dataRunOffset +
2:attrOffset + dataRunOffset + 4], byteorder='little')
           clusterList = []
           for cluster in range(startClusterIndex, startClusterIndex + numberOfCluster):
               clusterList.append(cluster)
           sectorList = self.clusterToSectorList(clusterList)
           # read data based on sector list to get file data
           fileContent = self.readSectorChain(sectorList)
```

- Bằng cách ánh xạ Entry với MFT_segment_reference của mỗi file, chương trình lưu lại số thứ tự của mỗi Entry từ đó tính ra xem tệp tin thuộc thư mục nào và xây dựng cây thư mục.

d. Đọc nội dung tập tin có đuôi "txt" và hiển thị thông báo đối với các tập tin có định dạng khác

- Như đã trình bày ở phần trên, ta nhận diện các attribute, nếu như attribute đó là loại **$DATA**, ta đọc nội dung của tập tin đó:
  - Nếu attribute thuộc loại resident, kích thước tập tin chính là kích thước của attribute trong MFT entry. Từ thông tin ở phần header của MFT entry, ta lấy được offset bắt đầu của attribute **$STANDARD_INFORMATION**, sau đó cộng với kích thước của attribute này ta sẽ có được offset bắt đầu của phần nội dung cần đọc.
  - Nếu attribute thuộc loại non-resident, thì nội dung của tập tin/thư mục đó được lưu ở các cluster bên ngoài MFT entry vậy nên từ tên tập tin/thư mục có được ở attribute **$FILE_NAME**, ta kiểm tra nó có phải một tập tin đuôi "txt" hay không, nếu phải ta bắt đầu đọc nội dung: đọc 1 byte tại offset 1 của vùng data lấy số lượng cluster, đọc 2 byte tại offset 2 của vùng data lấy chỉ số cluster đầu của vùng chứa data thực, lập danh sách các sector chứa data thực và đọc nội dung từ các sector đó.
- Với mỗi entry như vậy, ta xuất ra các thông tin cơ bản của mỗi tập tin/thư mục như đã trình bày ở trên, và nếu là một tập tin đuôi "txt", ta in ra thêm nội dung của tập tin đó.
- Sử dụng hàm **printFile**, ta có thể điều chỉnh cách xuất thông tin của tập tin/thư mục: trong trường hợp tập tin có đuôi khác "txt", thay vì in ra nội dung (chưa được decode, vẫn ở dạng

binary), ta in ra thông báo “Sử dụng công cụ tương thích để mở”, nếu tập tin có đuôi "txt", ta decode bằng "utf-8" và xuất ra nội dung tập tin.

3. Menu

a. Tổng quan cấu trúc sourcecode

- Tên file: *main.py*
- Các hàm chính:
  - **check_filesystem_type()** : kiểm tra format ổ đĩa hiện tại.

```
def check_filesystem_type(drive_letter):
     partitions = psutil.disk_partitions(all=True)
     for partition in partitions
         if partition.device.startswith(drive_letter):
             filesystem_type = partition.fstype
                 return filesystem_type
              return None
```

  - **isFAT32(diskTyp)**: kiểm tra xem định dạng ổ đĩa hiện tại có phải FAT32 hay không để chuyển định dạng menu.

```
def isFat32(diskType):
     return diskType.strip() == ‘FAT32’
```

  - **getUserQuery()**: nhận câu lệnh lựa chọn từ người dùng.

```
def getUserQuery():
     print(‘Input command: ‘, end = ‘’)
         query = int(input())
         return query
```

- **helpQuery()**: in ra menu lựa chọn và trung gian chuyển lệnh thực thi các lệnh từ người dùng.

```
def helpQuery(query, disk, diskType):
     isF32 = isFAT32(diskType)
         if (query == 1):
             print('List of all commands:')
             print('1. List all command type')
             print('2. Print file content')
             print('3. Change to directory')
             print('4. Print volume information')
             print('5. List all files and folders in current working directory')
             print('6. Print the tree of working directory')
             print('7. Exit program')
         elif (query == 2):
             print('Input directory: ', end = '')
             dir = input()
             disk.printFileFromDir(dir)
          elif (query == 3):
             print('Input directory: ', end = '')
             dir = input()
         disk.gotoDir(dir)
       elif (query == 4):
         disk.getVolumeInfo()
       elif (query == 5):
         disk.listDir()
       elif (query == 6):
         disk.drawTree()
       elif (query == 7):
             return False
```

5 Hình ảnh demo chương trình

- Giao diện chính của chương trình được cài đặt chung cho cả 2 định dạng FAT32 và NTFS.

```
PS D:\study\os\11cent_explorer\11cent-Cent-Explorer\Source> & C:/Users/HACOM/AppDat
a/Local/Programs/Python/Python311/python.exe d:/study/os/11cent_explorer/11cent-Cen
t-Explorer/Source/main.py
Available volumes:
1/ C:
2/ D:
3/ F:
4/ G:
Choose volume:
```

- Đầu tiên, ta chọn ổ đĩa muốn duyệt, ví dụ:
  - Nhập "3" để chọn ổ đĩa *F:*
- Sau khi chọn ổ đĩa, hệ thống sẽ hiển thị tên ổ đĩa, định dạng cùng với các lệnh chức năng được cài đặt để người dùng thao tác với chương trình:

```
Available volumes:
1/ C:
2/ D:
3/ F:
4/ G:
Choose volume: 3
driveLetter:  F
F uses FAT32 file system.
List of all commands:
1. List all command type
2. Print file content from working directory
3. Print file content from directory
4. Change to directory
5. Print volume information
6. List all files and folders in current working directory
7. Print the tree of working directory
8. Exit program
Type the number that corresponds to the command!
```

- Các lệnh chức năng gồm có:
  - 1 – Hiển thị danh sách các lệnh
  - 2 – In nội dung file ở thư mục hiện tại
  - 3 – In nội dung file ở thư mục khác
  - 4 – Thay đổi thư mục
  - 5 – In thông tin ổ đĩa
  - 6 – Hiển thị danh sách các thư mục, tập tin ở thư mục hiện tại
  - 7 – In cây thư mục
  - 8 – Thoát chương trình

- In thông tin ổ đĩa bằng lệnh số *5:*

```
Input command: 5
Volume name:  FF
OEM_Name:  MSDOS5.0
Bytes per sector:  512
Sectors per cluster:  2
Reserved sectors:  6654
Number of sectors in volume:  204800
```

- Chọn một file ở thư mục hiện tại để in nội dung với lệnh số *2:*
  - Người dùng nhập đường dẫn của của file muốn đọc, nội dung của file được in phía dưới

```
Input command: 2
Input directory: F:\a\test\aaaaa.txt
asfd
```

- Lệnh số *3* in ra danh sách các file ở thư mục hiện tại, cho phép người dùng chọn file để in:
  - Danh sách in ra được đánh số thứ tự, người dùng chọn số thứ tự tương ứng để chọn file

```
Input command: 3
Files in  :
1:      HCMUS.TXT
2:      WPSettings.dat
3:      IndexerVolumeGuid
Select file to print: 1
HCMUS
```

- Thay đổi thư mục làm việc bằng lệnh số *4*:
  - Người dùng nhập đường dẫn của thư mục muốn đi đến

```
Input command: 4
Input directory: F:\a\test
Current working directory:  F:\A\TEST
```

- In ra danh sách các thư mục và tập tin của thư mục hiện tại bằng lệnh số *6*:

```
Input command: 6
Index    | Type          | Date        | Time        | Size(B)   | Name
-------------------------------------------------------------------------------------------------------
----
1        | directory     | 2024-03-12  | 21:44:06    |           | /..
2        | archive       | 2024-03-12  | 21:44:06    | 4         | AAAAA.TXT
```

- In ra cây thư mục bằng lệnh số *7*:
  - In cây thư mục của ổ đĩa được chọn lúc đầu

```
Input command: 7
F:
├─HCMUS.TXT
├─C
├─A
│  └─TEST
│      └─AAAAA.TXT
├─B
│  └─ABCBA.TXT
└─22127129
    ├─HW
    │  ├─EX2.PNG
    │  └─EX1.PNG
    └─SLIDE
        ├─Chap1_Introduction (1).pdf
        ├─Chap1_Data_Representation_MOOC.pdf
        ├─Report-1.pdf
        └─Project 1 - Quan ly he thong tap tin.docx.doc
├─WPSettings.dat
├─EDP
│  └─Recovery
└─IndexerVolumeGuid
```

- Thoát chương trình với lệnh số *8:*

```
Input command: 8
Program exitted! Thanks for using!
```

6 Nguồn tham khảo

- FAT32 - Documentation
- Understanding FAT32
- FAT32 - Feature
- LeGiaCongBLog - NTFS
- NTFS - Documentation
- Microsoft Learn - NTFS
- ChatGPT: model gpt-3.5 & gpt-4
- StackOverflow