ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN

CƠ SỞ NGÀNH MẠNG

Đề tài:

Hệ điều hành: *Xây dựng chương trình đọc thông tin đĩa cứng với định dạng NTFS và FAT32.*

Lập trình mạng: *Tìm hiểu và sử dụng kỹ thuật lập trình với Socket xây dựng chương trình tính tiền dịch vụ Internet theo mô hình Client - Server.*

GIẢNG VIÊN HƯỚNG DẪN: ***Ths.Nguyễn Văn Nguyên***

SINH VIÊN THỰC HIỆN: ***Đặng Trung Nguyên***

MSSV: ***102180032***

Lớp: ***18T1***

Đà Nẵng, 11/2021

LỜI CẢM ƠN

Lời đầu tiên, em xin được gửi lời cảm ơn chân thành đến thầy Nguyễn Văn Nguyên đã hướng dẫn và giúp đỡ em những kiến thức, thông tin cần thiết trong suốt thời gian qua để có thể hoàn thành Đồ án Cơ sở ngành mạng.

Tuy nhiên, trong quá trình nghiên cứu và xây dựng đề tài không thể tránh khỏi những thiếu sót. Vì vậy, em rất mong sự góp ý từ thầy và hội đồng bảo vệ để đồ án này sẽ được phát triển cũng như hoàn thiện hơn.

Em xin chân thành cảm ơn.

Đặng Trung Nguyên

LỜI CAM ĐOAN

Tôi xin cam đoan:

1. Nội dung trong đồ ánnày là do tôi thực hiện dưới sự hướng dẫn trực tiếp của thầy Nguyễn Văn Nguyên.

2. Các tham khảo dùng trong đồ án đều được trích dẫn rõ ràng tên tác giả, tên công trình, thời gian, địa điểm công bố.

3.Nếu có những ghi chép không hợp lệ, vi phạm, tôi xin chịu hoàn toàn trách nhiệm.

Sinh viên thực hiện

Đặng Trung Nguyên

NHẬN XÉT CỦA NGƯỜI HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

NHẬN XÉT CỦA HỘI ĐỒNG DUYỆT ĐỒ ÁN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

MỤC LỤC 1
DANH SÁCH HÌNH ẢNH 4
DANH SÁCH BẢNG BIỂU 5
MỞ ĐẦU 6
PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH 7
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 7
1.1. Đĩa cứng: 7
1.1.1. Khái niệm: 7
1.1.2. Cấu tạo đĩa cứng: 7
- **Cụm đĩa:** 7
- **Cụm đầu đọc:** 9
- **Cụm mạch điện:** 11
- **Vỏ đĩa cứng:** 12

1.1.3. Phân vùng ổ cứng: 12
- **Khái niệm:** 12
- **Master Boot Record (MBR):** 13
- **GPT (Guid Partition Table):** 14

1.2. hệ thống File của hệ điều hành Windows: 15
1.2.1. Khái niệm: 15
1.2.2. Hệ thống file trong Windows: 15
1.3. Hệ thống FAT32: 15
1.3.1. Cấu trúc: 16
- **Partition Boot Sector:** 16
- **Bảng FAT** 16
- **Root Folder:** 16
- **Other file or folder** 16

1.4. Hệ thống NTFS: 16
1.4.1. Cấu trúc: 17
- **NTFS boot sector** 17
- **Master File Table (MFT)** 18

- **File System Storage** 18
- **Master File Table Copy** 19

1.5. So sánh giữa FAT32 và NTFS: **19**

CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 21

2.1. Phân tích: **21**

2.1.1. Phân tích yêu cầu: 21

2.1.2. Phân tích chức năng: 21

2.2. Thiết kế: **22**

2.2.1. Thiết kế chức năng: 22

- **getString():** 23
- **destroyListHardDrives():** 23
- **readHardDrives():** 23
- **Hàm getHardDrives():** 24
- **Hàm destroyListLogicalDrives():** 24
- **Hàm readLogicalDrives():** 24
- **Hàm getLogicalDrives():** 25

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 26

3.1. Triển khai: **26**

3.2. Kết quả triển khai: **26**

3.3. Đánh giá kết quả: **27**

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 28

1. Kết luận: **28**

1.1. Đạt được: 28

1.2. Khó khăn: 28

2. Hướng phát triển: **28**

PHẦN II: LẬP TRÌNH MẠNG 29

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT **29**

1.1. Mạng máy tính: **29**

1.1.1. Khái niệm: **29**

1.2. Mô hình Client – Server: **30**

1.2.1. Khái niệm: **30**

1.2.2. Đặc trưng của mô hình Client – Server: **31**

1.3. Giao thức TCP/IP: **31**

1.3.1. Giao thức IP: **32**

1.3.2. Giao thức TCP: 33
1.4. Socket trong Python: 34
1.4.1. Khái niệm Socket: 34
1.4.2. Lập trình Socket với TCP/IP: 34
1.4.3. Socket module trong Python: 36
CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 37
2.1. Phân tích: 37
2.1.1. Phân tích yêu cầu: 37
2.1.2. Phân tích chức năng: 37
2.2. Thiết kế: 37
2.2.1. Thiết kế cơ sở dữ liệu: 37
2.2.2. Thiết kế giao diện: 38
2.2.3. Thiết kế chức năng: 40
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 43
3.1. Triển khai: 43
3.2. Kết quả triển khai: 43
3.3. Đánh giá kết quả: 45
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 46
1. Kết luận: 46
1.1. Đạt được: 46
1.2. Khó khăn: 46
2. Hướng phát triển: 46
TÀI LIỆU THAM KHẢO 47
PHỤ LỤC 48

DANH SÁCH HÌNH ẢNH

Hình 1: Cấu tạo của ổ đĩa cứng 7
Hình 2: Cấu tạo đĩa từ. 8
Hình 3: Cụm đầu đọc 9
Hình 4: Cánh tay đòn. 10
Hình 5: Con trượt. 10
Hình 6: Đầu đọc. 11
Hình 7: Phân vùng. 12
Hình 8: Master Boot Record. 13
Hình 9: Guild Partition Table. 14
Hình 10: Cấu trúc FAT32. 16
Hình 11: Cấu trúc NTFS. 17
Hình 12: Master File Table. 18
Hình 13: Thông tin ổ cứng. 26
Hình 14: Thông tin phân vùng. 26
Hình 15: Mạng LAN. 29
Hình 16: Mạng WAN. 29
Hình 17: Mạng MAN. 30
Hình 18: Mô hình Client - Server. 30
Hình 19: Giao thức hoạt động bất đối xứng của Client - Server. 31
Hình 20: Mô hình TCP/IP. 31
Hình 21: IPv4. 32
Hình 22: IP header. 33
Hình 23: TCP header. 34
Hình 24: Client - Server Socket 35
Hình 25: Cơ sở dữ liệu account. 37
Hình 26: Giao diện chính Server. 38
Hình 27: Giao diện nạp tiền Server. 38
Hình 28: Giao diện tào tài khoản Server. 38
Hình 29: Giao diện đăng nhập Client. 39
Hình 30: Giao diện chính Client. 39
Hình 31: Giao diện đổi mật khẩu Client. 39
Hình 32: Giao diện sau khi đăng xuất Client. 40
Hình 33: Lỗi kết nối khi Server chưa hoạt động. 43
Hình 34: Chạy chương trình Server. 43
Hình 35: Server nạp tiền thành công. 43
Hình 36: Server tạo tài khoản thành công. 44
Hình 37: Client đăng nhập thành công. 44
Hình 38: Client thực hiện các dịch vụ Internet thành công. 44
Hình 39: Client đổi mật khẩu thành công. 45
Hình 40: Client sau khi đăng xuất. 45

DANH SÁCH BẢNG BIỂU

Bảng 1: Boot sector..... 17
Bảng 2: So sánh FAT32 và NTFS. 19
Bảng 3: Thông tin ổ đĩa cứng. 21
Bảng 4: Thông tin phân vùng...... 22
Bảng 5: Hàm của class HardDriveInfo. 23
Bảng 6: Hàm của class LogicalDriveInfo..... 23
Bảng 7: Phương thức Socket trong Python...... 36

MỞ ĐẦU

Nguyên lý hệ điều hành và Lập trình mạng là những môn học thiết yếu của sinh viên ngành công nghệ thông tin. Nắm vững các kiến thức đã học, trau dồi thông qua việc thực hành cũng như tìm đọc tư liệu tham khảo, từng bước làm quen với các công tác khoa học được giáo viên định hướng, từ đó hình thành hành vi nghiên cứu độc lập. Qua kết quả đồ án này, có thể đánh giá được công sức, thời gian cũng như năng lực của người thực hiện.

1. Tổng quan về đề tài:

-Hệ điều hành: Thiết kế xây dựng nên hệ thống quản lý bộ nhớ giúp theo dõi thông tin, các chỉ số và trạng thái của ổ đĩa cứng, phân vùng trên máy tính người dùng.

-Lập trình mạng: Với sự phát triển của mạng lưới internet ngày nay, nhiều dịch vụ internet đáp ứng người dùng cũng gia tăng không ngừng.Vì vậy, đề tài xây dựng chương trình tính tiền dịch vụ internet góp phần để đáp ứng nhu cầu sử dụng internet của người dùng lẫn việc thu phí truy cập của nhà dịch vụ.

2. Mục đích:

-Hệ điều hành: chương trình giúp người dùng nắm rõ các thông tin, thông số của ổ đĩa cứng trên máy tính mình đang sử dụng.

-Lập trình mạng, chương trình giúp người dùng truy cập mạng internet và sử dụng dịch vụ có tính phí.

3. Ý nghĩa:

Nắm rõ về cấu tạo, phân vùng của ổ đĩa cứng cũng như cách thức hoạt động của hệ thống file trong hệ điều hành Windows .

Nắm rõ về mô hình client-server, cách thức giao tiếp giữa client và server thông qua socket.

PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1. Đĩa cứng:

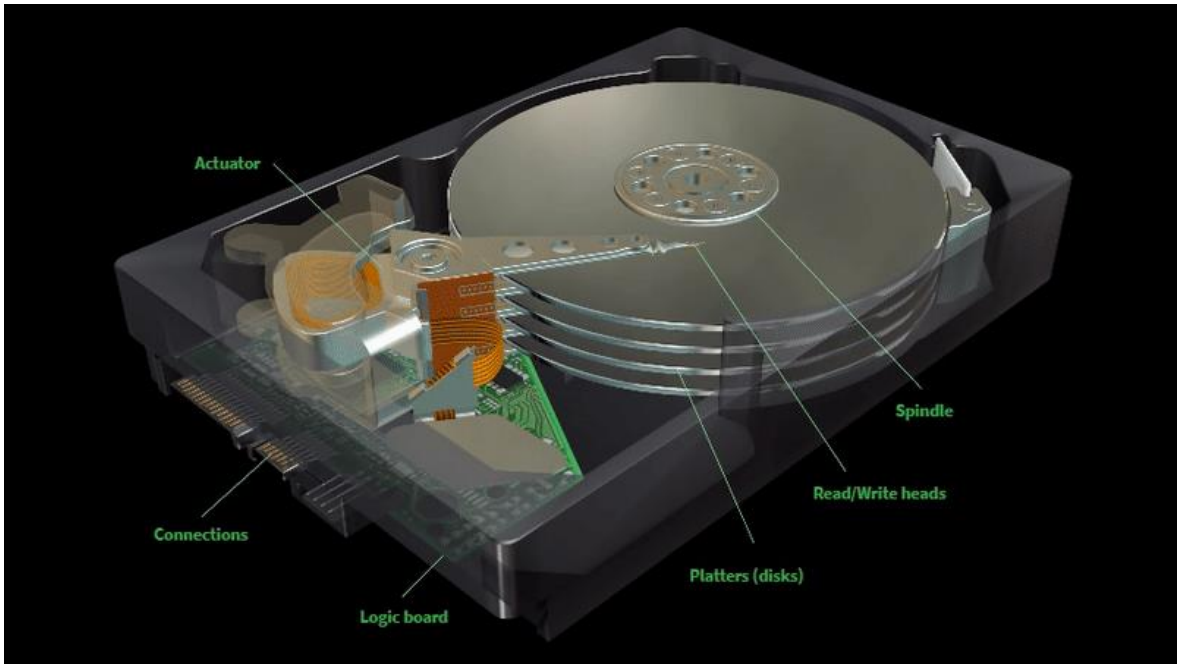
1.1.1. Khái niệm:

Đĩa cứng hay gọi đúng hơn là ổ đĩa cứng (tên tiếng anh: Hard Disk Drive, viết tắt HDD) là thiết bị dùng để lưu trữ dữ liệu trên bề mặt các tấm đĩa hình tròn phủ vật liệu từ tính.

Thường được biết đến như là một bộ phận rất quan trọng của máy tính với việc lưu trữ dữ liệu trong suốt quá trình sử dụng của người dùng: Các thao tác từ việc truy xuất đọc tài liệu office, tải file, thiết kế hình ảnh, các bản vẽ, biên tập video, cài đặt phần mềm, game… là các thao tác đọc/ ghi trên ổ đĩa cứng và những dữ liệu này được lưu trên đó.

1.1.2. Cấu tạo đĩa cứng:

Ổ đĩa cứng là một khối duy nhất, các đĩa cứng được lắp ráp cố định trong ổ ngay từ khi sản xuất nên không thể thay thế được các "đĩa cứng" như với cách hiểu như đối với ổ đĩa mềm hoặc ổ đĩa quang.



Hình 1: Cấu tạo của ổ đĩa cứng

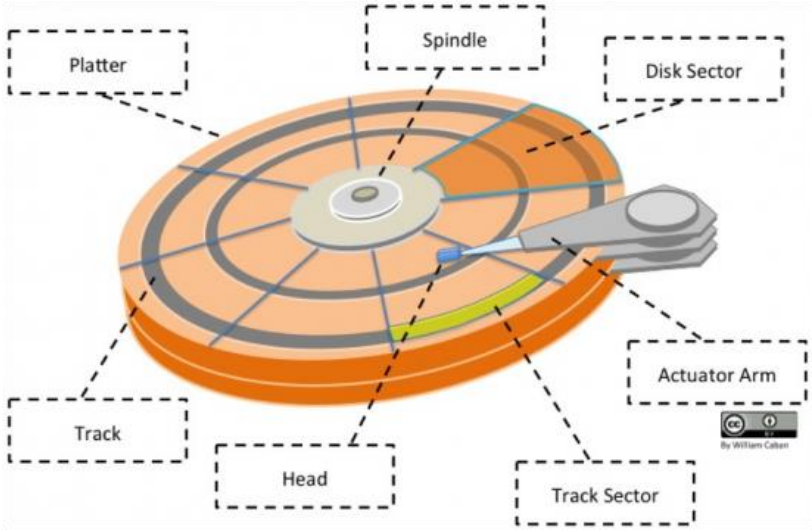
Ổ đĩa cứng gồm các thành phần, bộ phận chính sau:

- **Cụm đĩa:**

Bao gồm toàn bộ các đĩa, trục quay và động cơ.

Đĩa từ (Platter):

Đĩa từ cấu tạo bằng nhôm hoặc thuỷ tinh, trên bề mặt phủ một lớp vật liệu từ tính là nơi chứa dữ liệu. Mỗi đĩa từ có thể sử dụng hai mặt, đĩa cứng có thể có nhiều đĩa từ, chúng gắn song song, quay đồng trục, cùng tốc độ với nhau khi hoạt động.



Hình 2: Cấu tạo đĩa từ.

-Track: Trên một mặt làm việc của đĩa từ chia ra nhiều vòng tròn đồng tâm gọi là các track. Track có thể được hiểu đơn giản giống các rãnh ghi dữ liệu như các đĩa ghi nhạc nhưng sự cách biệt của các rãnh ghi này không có các gờ phân biệt và chúng là các vòng tròn đồng tâm chứ không nối tiếp nhau thành dạng xoắn trôn ốc như đĩa nhựa. Thông thường mỗi đĩa từ có từ 312 đến 2048 track.

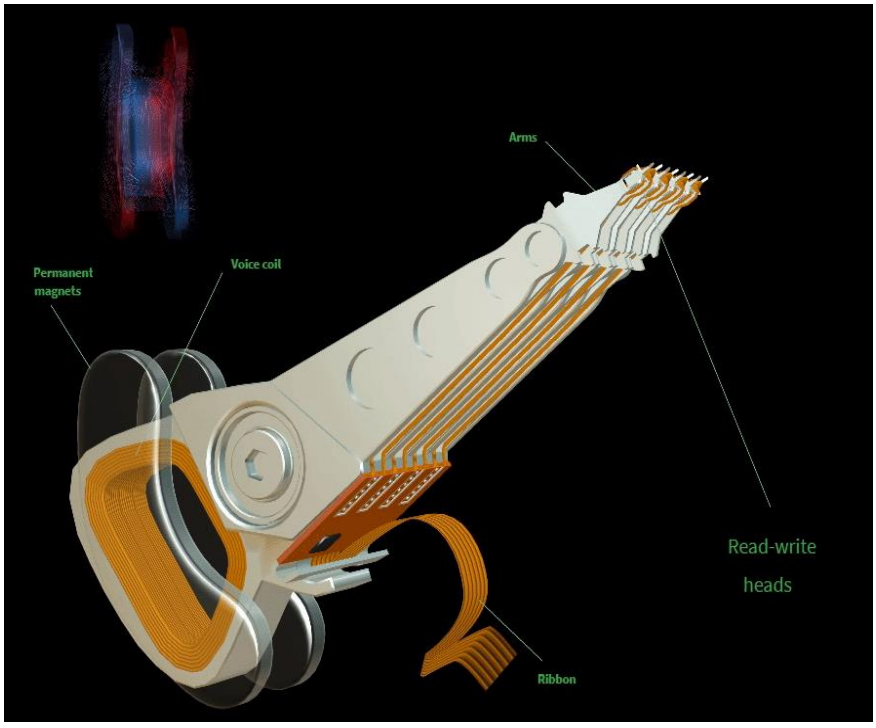
-Sector: Trên track chia thành những phần nhỏ bằng các đoạn hướng tâm gọi là các sector. Các sector là phần nhỏ cuối cùng được chia ra để chứa dữ liệu. Mỗi track đều chia thành một lượng sector nhất định. Nhưng vì các track bên ngoài bao giờ cũng lớn hơn các track phía trong (gần trục) nên càng vào sâu các track phía trong thì dung lượng mà một sector có thể chứa được càng thấp. Theo chuẩn thông thường thì một sector chứa dung lượng 512 byte.

-Cylinder: Tập hợp các track cùng bán kính (cùng số hiệu trên) ở các mặt đĩa khác nhau gọi là các cylinder. Nói một các chính xác hơn: khi đầu đọc/ghi đầu tiên làm việc tại một track nào thì tập hợp toàn bộ các track trên các bề mặt đĩa còn lại mà các đầu đọc còn lại đang làm việc tại đó gọi là cylinder. Trên một ổ đĩa cứng có thể có nhiều cylinder bởi có nhiều track trên mỗi mặt đĩa từ.

-Cluster: Cluster (liên cung) là một nhóm gồm 2, 4 hoặc 6sector liên tiếp nhau tạo thành một cluster. Kích thước của cluster thường là bội số kích thước một sector. Các cluster được đánh địa chỉ bắt đầu từ 0. Số sector trên một cluster phụ thuộc vào từng loại đĩa. Một số hệ điều hành cho phép người sử dụng quy định số sector trên một cluster. Các hệ điều hành thường tổ chức lưu trữ dữ liệu, nội dung các tập tin trên đĩa theo từng cluster. Trên bề mặt đĩa cũng tồn tại các bad cluster, đó là các cluster có chứa bad sector.

-Partition: Partition (phân khu) là một tập các sector liền kề trên một đĩa. Mỗi partition có một bảng partition hoặc một cơ sở dữ liệu quản lý đĩa riêng, dùng để lưu trữ sector đầu tiên, kích thước và các đặc tính khác của partition.

- **Cụm đầu đọc:**



Hình 3: Cụm đầu đọc

a. Nam châm vĩnh cửu (permanent magnet):

Mỗi nam châm vĩnh cửu (vật liệu từ tính tự nhiên) có một cực "bắc" và một cực "nam", với đặc tính cực bắc hút cực nam và ngược lại. Nam châm này có thể nâng một vật nặng gấp 1.300 lần khối lượng của nó, do đó không nên đặt ngón tay vào giữa thanh nam châm với thép hoặc với một thanh nam châm khác.

b. Cuộn dây di động (voice coil):

Là một bộ phận của HSA (head stack assembly - HSA), cuộn dây di động và nam châm vĩnh cửu tạo thành cơ cấu truyền động (actuator), đây là thiết bị làm cho đầu từ di chuyển. Điện đi qua cuộn dây tạo thành nam châm điện (từ trường được tạo bởi dòng điện). Chiều của dòng điện trong cuộn dây làm thay đổi hướng của cực bắc nam của nam châm điện, khiến cho cuộn dây di động dịch chuyển lại gần hoặc ra xa cực bắc và nam của nam châm vĩnh cửu. Cường độ và khoảng thời gian của dòng điện sẽ xác định việc cuộn dây di chuyển nhanh như thế nào và xa bao nhiêu.

c. Cánh tay đòn (arm):

HSA có một ổ trục chính xác để đầu từ di chuyển tốt và mượt mà. Phần lớn nhất của HSA được làm bằng nhôm, có tên gọi là cánh tay đòn.

Cánh tay đòn có nhiệm vụ di chuyển theo phương song song với các đĩa từ ở một khoảng cách nhất định, dịch chuyển và định vị chính xác đầu đọc tại các vị trí từ mép đĩa đến vùng phía trong của đĩa (phía trục quay). Khối khớp đầu từ (head gimbal assembly - HGA) là một trong hai khối phụ của khối đầu từ, cùng với HSA.

Hình 4: Cánh tay đòn.

Con trượt (slider): Vật nhỏ màu đen ở cuối của HGA được gọi là con trượt.

Hình 5: Con trượt.

d. Đầu đọc (head):

Đầu đọc/ghi dữ liệu trong đĩa cứng (head) có công dụng đọc dữ liệu dưới dạng từ hoá trên bề mặt đĩa từ hoặc từ hoá lên các mặt đĩa khi ghi dữ liệu.

Hình 6: Đầu đọc.

Các phần tử đọc/ghi thực sự nằm ở phía cuối của con trượt và chúng rất nhỏ bé, chỉ có thể được nhìn thấy qua kính hiển vi. Đầu đọc trong đĩa cứng có công dụng đọc dữ liệu dưới dạng từ hóa trên bề mặt đĩa từ hoặc từ hóa lên các bề mặt đĩa khi ghi dữ liệu. Số đầu đọc ghi luôn bằng với số mặt hoạt động được của các đĩa cứng, có nghĩa chúng nhỏ hơn hoặc bằng hai lần số đĩa (nhỏ hơn trong trường hợp ví dụ hai đĩa nhưng chỉ sử dụng ba mặt).

- **Cụm mạch điện:**

a. Mạch điều khiển:

Có nhiệm vụ điều khiển động cơ đồng trục, điều khiển sự di chuyển của cần di chuyển đầu đọc để đảm bảo đến đúng vị trí trên bề mặt đĩa.

b. Mạch xử lý dữ liệu:

Mạch xử lý dữ liệu dùng để xử lý những dữ liệu đọc/ghi của ổ đĩa cứng.

c. Bộ nhớ đệm (cache hoặc buffer):

Bộ nhớ đệm là nơi tạm lưu dữ liệu trong quá trình đọc/ghi dữ liệu. Dữ liệu trên bộ nhớ đệm sẽ mất đi khi ổ đĩa cứng ngừng được cấp điện. Bộ đệm của ổ cứng sử dụng một phần của RAM để lưu trữ thông tin thường xuyên được các ứng dụng truy nhập. Chính việc lưu trữ thông tin này trên RAM, bộ đệm đã giúp tốc độ truy xuất dữ liệu nhanh hơn, kéo dài tuổi thọ của ổ cứng.

d. Các cầu đấu thiết bị:

Các cầu đấu thiết bị thiết đặt chế độ làm việc của ổ đĩa cứng: lựa chọn chế độ làm việc của ổ đĩa cứng (SATA 150 hoặc SATA 300) hay thứ tự trên các kênh giao tiếp IDE (master hay slave hoặc tự lựa chọn), lựa chọn các thông số làm việc khác.

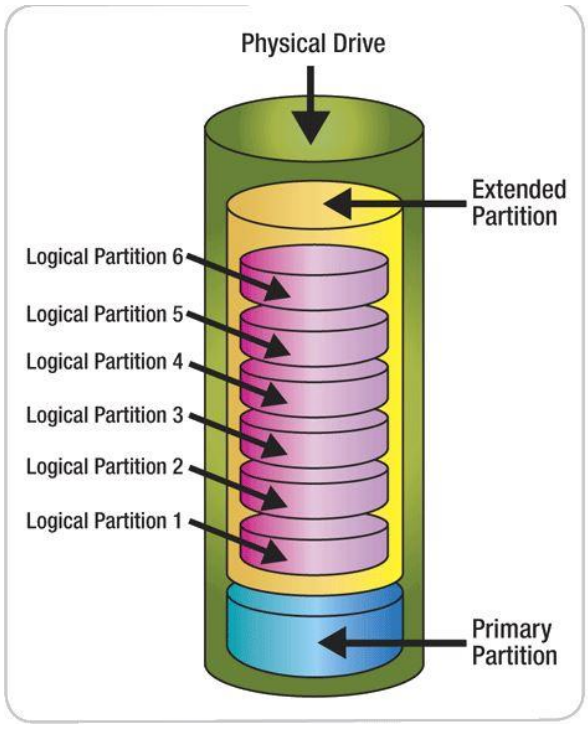
- **Vỏ đĩa cứng:**

Vỏ ổ đĩa cứng có chức năng chính nhằm định vị các linh kiện và đảm bảo độ kín khít để không cho phép bụi được lọt vào bên trong của ổ đĩa cứng. Ngoài ra, vỏ đĩa còn có tác dụng chịu đựng sự va chạm (ở mức độ thấp) để bảo vệ ổ đĩa cứng.

1.1.3. Phân vùng ổ cứng:

- **Khái niệm:**

Phân vùng ổ cứng là việc tạo ra một hoặc nhiều phân vùng trên ổ cứng, để mỗi phân vùng có thể quản lý riêng biệt, còn là tập hợp các vùng nhớ dữ liệu trên các cylinder gần nhau với dung lượng theo thiết lập người dùng. Có 2 loại phân vùng: phân vùng chính (Primary Partitons) và phân vùng mở rộng (Extended Partitions).



Hình 7: Phân vùng.

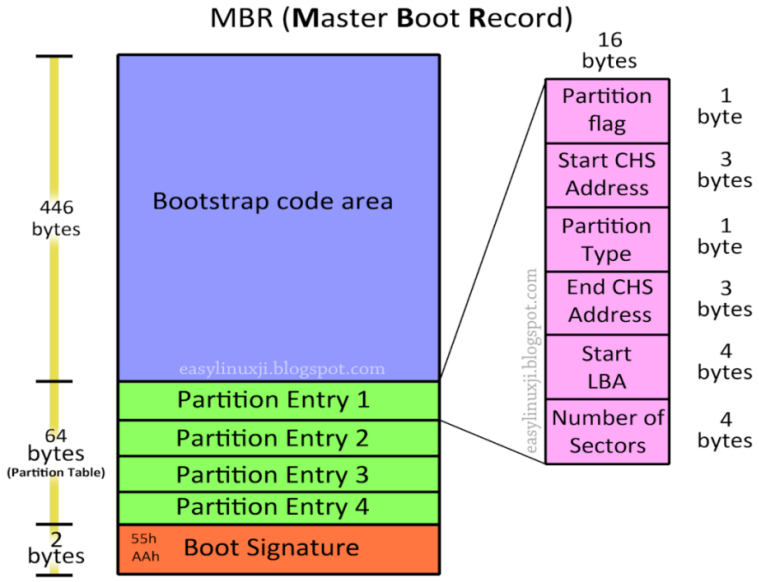
Phân vùng chính (Primary Partitons) là vùng thường dùng để cài đặt hệ điều hành. Mỗi Primary partition có thể cài đặt một hệ điều hành. Điều này cho phép tách biệt các hệ điều hành với những dữ liệu khác. Một ổ đĩa khi phân vùng có tối đa 4 primary partition. Cũng có thể chia thành 3 primary partition và 1 extented partiton. Trên ổ đĩa cứng phải có ít nhất một phân vùng chính. Phân vùng mở rộng

để mở rộng phạm vi sử dụng đĩa cứng. Chỉ có thể tạo một phân vùng mở rộng duy nhất nhưng có thể chia phân vùng mở rộng thành nhiều phân vùng logical.

Để hệ điều hành biết được cấu trúc của đĩa cứng khi chia ổ cứng thành nhiều phân vùng thì lấy thông tin từ MBR (Master Boot Record) và GPT (Guid Partition Table). Chúng là hai cấu trúc khác nhau, cả hai đều đóng vai trò quan trọng và cung cấp thông tin của các phân vùng trên ổ đĩa cứng.

- **Master Boot Record (MBR):**

MBR là một chuẩn quản lý phân vùng cũ trên ổ cứng, và nó vẫn được sử dụng một cách rộng rãi. MBR cư trú tại đầu các ổ cứng và chứa các thông tin tổ chức các phân vùng trên thiết bị lưu trữ. Ngoài ra, MBR còn chứa mã thực thi nó có thể quét các phân vùng active chứa hệ điều hành và tải lên trong quá trình khởi động.



Hình 8: Master Boot Record.

Trên một ổ đĩa MBR, chúng ta chỉ có thể chia thành 4 phân vùng primary. Để tạo ra nhiều phân vùng hơn, bạn phải thiết lập phân vùng thứ 4 thành dạng extended bạn sẽ chia phân vùng này thành nhiều sub-patitions hay còn (gọi là các logical driver) bên trong nó. MBR dùng 32 bit để ghi nhận phân vùng, mỗi phân vùng vì thế chỉ có thể mở rộng tối đa là 2 TB dung lượng.

Ngoài ra, trong khi MBR ban đầu chỉ hỗ trợ bốn phân vùng, các phiên bản mới hơn có thể hỗ trợ tới mười sáu phân vùng. Tuy nhiên, tất cả các bản ghi khởi động chính được giới hạn ở 512 byte, có nghĩa là chúng chỉ có thể giải quyết tối đa hai terabyte dữ liệu. Do đó, các đĩa được định dạng bằng MBR bị giới hạn ở 2TB dung lượng đĩa có thể sử dụng.

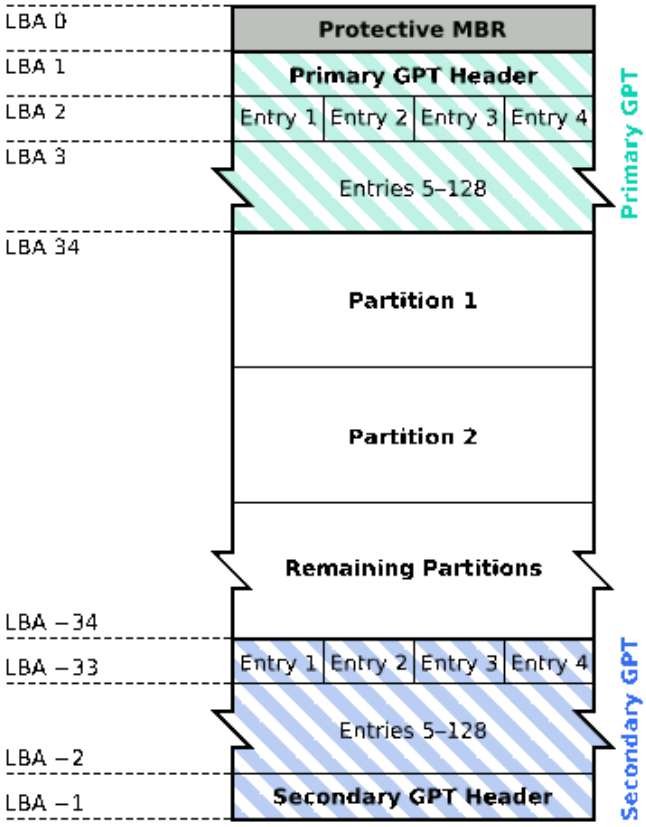
Tuy nhiên, hầu hết các ổ cứng hiện nay đang sử dụng bảng phân vùng GUID (GPT), bảng này tương thích với nhiều hệ điều hành và không có giới hạn 2TB.

- **GPT (Guid Partition Table):**

GPT là tiêu chuẩn mới cho việc phân chia phân vùng của ổ cứng. nó được xây dựng trên Globally Unique Identifiers (GUID) để xác định các phân vùng ổ cứng một phần của tiêu chuẩn UEFI. Điều này có nghĩa là hệ thống dựa trên tiêu chuẩn của UEFI phải dùng nó. Với GPT, có thể tạo không giới hạng số phân vùng trên ổ cứng, mặc dù thường được giới hạn trong 128 phân vùng bởi các hệ điều hành.

Không giống như MBR bị giới hạn mỗi patition chỉ có 2TB, mỗi phân vùng của GPT có thể chứa 2^64 block in length (trong trường hợp dùng 64 bit), tương đương 9.44ZB với một block 512-byte. Với windows thì nó được giới hạn 256TB.

Theo lược đồ miêu tả, có thể thấy 1 primary GPT ở đầu và 1 secondary GPT ở cuối. Đây là những gì làm cho GPT hữu ích hơn MBR. GPT lưu trữ và backup header và parttion table vào cuối ổ đĩa để nó có thể được phục hồi nếu primary table bị hỏng. Nó cũng thực hiện checksum bằng CRC32 để phát hiện lỗi và những chỗ hỏng của header và patition table.



Hình 9: Guild Partition Table.

1.2. hệ thống File của hệ điều hành Windows:

1.2.1. Khái niệm:

Trong điện toán, khái niệm hệ thống tập tin hoặc hệ thống tệp (file system) được dùng để chỉ các phương pháp và cấu trúc dữ liệu mà một hệ điều hành sử dụng để theo dõi các tập tin trên ổ đĩa hoặc các phân vùng. Hệ thống tập tin cũng kiểm soát cách lưu trữ và truy xuất dữ liệu.

Không có hệ thống tập tin, thông tin được lưu trong các phương tiện lưu trữ sẽ là một khối dữ liệu lớn mà không có cách nào để biết nơi một phần thông tin dừng lại và phần tiếp theo bắt đầu. Bằng cách tách dữ liệu thành từng mảnh và đặt tên cho mỗi mảnh, thông tin dễ dàng được phân tách và xác định. Lấy tên từ cách đặt tên cho các hệ thống quản lý thông tin dùng giấy, mỗi nhóm dữ liệu được gọi là "tập tin". Cấu trúc và quy tắc logic được sử dụng để quản lý các nhóm thông tin và tên của chúng được gọi là "hệ thống tập tin".

1.2.2. Hệ thống file trong Windows:

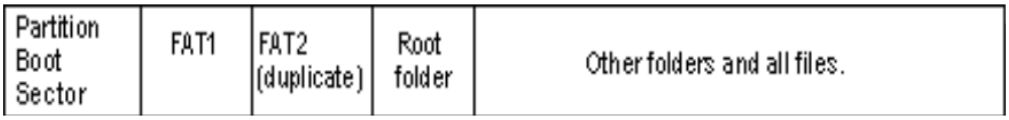
Các hệ thống file trong windows gồm 2 hệ thống chính là FAT và NTFS. Windows sử dụng một ký tự ổ đĩa trừu tượng ở cấp độ người dùng để phân biệt một đĩa hoặc phân vùng khác. Ví dụ: đường dẫn C:\WINDOWS đại diện cho một thư mục WINDOWS trên phân vùng được đại diện bởi chữ C. Ổ C: thường được sử dụng nhiều nhất cho phân vùng ổ đĩa cứng chính, trên đó Windows thường được cài đặt và từ đó khởi động.

1.3. Hệ thống FAT32:

FAT (File Allocation Table) – Bảng cấp phát tập tin: là hệ thống tệp mặc định được sử dụng trong các phiên bản Windows cũ hơn (trước Windows XP). Tuy nhiên, FAT có thể được sử dụng với các đĩa mềm và các phiên bản Windows cũ hơn (đối với các hệ thống đa khởi động). FAT có được tên của nó do sử dụng một loại cơ sở dữ liệu đặc biệt gọi là Bảng phân bổ tệp. Mỗi cụm trên đĩa có một mục tương ứng trên bảng. FAT ban đầu được sử dụng với DOS và ba phiên bản của nó là FAT12, FAT16 và FAT32. Số bit được sử dụng để xác định một cụm là số được sử dụng làm hậu tố trong tên.

FAT32 được xem là phiên bản mở rộng của FAT16, giới thiệu trong phiên bản Windows 95 Service Pack 2 (OSR 2). Do sử dụng không gian địa chỉ 32 bit nên FAT32 hỗ trợ nhiều cụm trên một phân vùng hơn, do vậy không gian đĩa cứng được tận dụng nhiều hơn. Ngoài ra, FAT32 có khả năng hỗ trợ kích thước của phân vùng từ 2GB lên 2TB và chiều dài tối đa của tên tập tin được mở rộng lên đến 255 ký tự.

1.3.1. Cấu trúc:



Hình 10: Cấu trúc FAT32.

- **Partition Boot Sector:**

Partition Boot Sector hay còn được gọi là Volume Boot Record, chứa bảng tham số đĩa bao gồm thông tin về cấu hình đĩa, kích thước,…, và loại hệ điều hành được cài đặt. Mã lệnh khởi động mồi bắt đầu cho hệ điều hành cũng được lưu ở đây.

Đây là một sector đặc biệt nằm ở đầu mỗi partition đĩa. Đây là nơi mà các boot virus sẽ hiệu chỉnh lại nội dung. Để cấm việc hiệu chỉnh sector này bởi các ứng dụng (chủ yếu là virus), thường BIOS của các máy đời mới đều có chức năng bảo vệ boot sector, bất kỳ ứng dụng nào muốn hiệu chỉnh nội dung đĩa đều phải nhờ BIOS làm và BIOS sẽ kiểm tra, nếu sector bị hiệu chỉnh là boot sector thì nó sẽ hiển thị thông báo là bạn đã trình bày để người dùng viết và quyết định. Có thể cho phép/cấm chức năng bảo vệ này của BIOS bằng cách vào BIOS Setup rồi thay đổi theo yêu cầu.

- **Bảng FAT**

FAT1, FAT2 là các bảng cấp phát và định vị file, thông tin chỉ mục giúp hệ điều hành có thể truy xuất chính xác đến file. Bảng FAT là sự ánh xạ của toàn bộ các cluster trên ổ đĩa, tuy nhiên FAT chỉ lưu thông tin về vị trí các cluster trên ổ cứng mà không lưu dữ liệu. Đồng thời qua bảng thông tin này hệ điều hành cũng xác định được dung lượng còn trống trên đĩa hoặc đánh dấu các vị trí BAD trên đĩa.

- **Root Folder:**

Root Folder hay còn được gọi là Root Directory (Bảng thư mục gốc) giống như bảng thư mục của một cuốn sách, lưu trữ thông tin liên quan đến file hoặc thư mục như tên, ngày giờ tạo lập, thuộc tính file hoặc thư mục. Những thông tin này có thể được thay đổi mỗi khi có file được tạo ra hoặc được sửa đổi sau đó.

- **Other file or folder**

Nơi lưu trữ thông tin chính của các file hoặc các thư mục con.

1.4. Hệ thống NTFS:

Vào cuối những năm 1990, Microsoft đã tạo ra một hệ thống file mới không dựa trên nền tảng hệ thống FAT là NTFS - hệ thống file công nghệ mới. NTFS có quan điểm mới rạch ròi không dựa trên nền tảng hệ thống FAT cũ. NTFS đã tiếp quản từ FAT làm hệ thống tệp mặc định bắt đầu từ Windows XP. Do đó, Windows NT 4.0, Windows 2000, Windows XP, máy chủ Windows .NET và máy trạm Windows sử dụng NTFS làm hệ thống tệp ưa thích của họ. Và cũng từ đó đến nay, NTFS đã có nhiều cải tiến để phù hợp với dung lượng lưu trữ lớn và yêu cầu bảo mật ngày càng cao.

NTFS có kiến trúc tổ chức dữ liệu hoàn toàn khác nhau. Về cơ bản, Microsoft đã phát triển NTFS để cạnh tranh với UNIX, bằng cách thay thế FAT đơn giản hơn nhiều. Phân vùng FAT có thể dễ dàng chuyển đổi thành phân vùng NTFS mà không mất dữ liệu. NTFS hỗ trợ các tính năng như lập chỉ mục, theo dõi hạn ngạch, mã hóa, nén và sửa chữa các điểm.

1.4.1. Cấu trúc:

Hình 11: Cấu trúc NTFS.

- **NTFS boot sector**

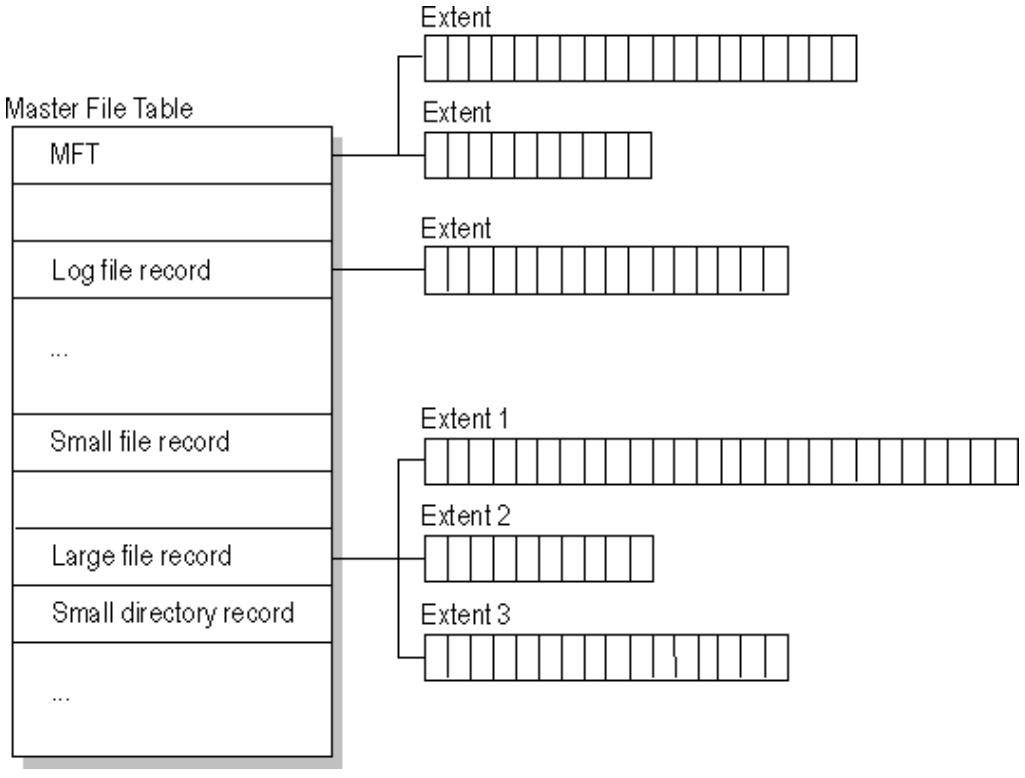
Bảng boot sector (hay còn gọi là Volume Boot Record) trên ổ đĩa NTFS được mô tả khi ổ đĩa được định dạng NTFS. Khi bạn định dạng ổ đĩa NTFS, thì chương trình Format định 16 sector đầu tiên cho boot sector và phần chứa lệnh thực thi.

Offset	Length	Description
0x00	3 bytes	Jmp (jump instructions)
0x03	8 bytes	FSName
0x0B	5 bytes	MusbeZero
0x10	4 bytes	Identifier
0x14	2 bytes	Length (of FSRS)
0x16	2 bytres	Checksum (of FSRS)
0x18	8 bytes	Sectors in volume
0x20	4 bytes	Bytes per sector
0x24	4 bytes	Sectors per cluster
0x28	1 bytes	File system major version
0x29	1 bytes	File system minor version
0x2A	14 bytes	Unknown
0x38	8 bytes	Volume Serial Number

Bảng 1: Boot sector.

- **Master File Table (MFT)**

Mỗi file trong hệ thống NTFS được đại diện bởi một bản ghi nằm trong một file đặc biệt gọi là master file table (MFT). Khi ta định dạng ổ đĩa với NTFS, Windows sẽ tạo ra cho ta MFT và dữ liệu cần có trên partition. MFT lưu trữ thông tin đòi hỏi truy cập dữ liệu từ phân vùng NTFS. MFT là một cơ sở dữ liệu quan hệ, nó bao gồm có những dòng và cột của File thuộc tính. Nó chứa đựng một vài entry cho những file trong ổ đĩa NTFS, bao gồm MFT và chính nó. NTFS dành riêng 16 bản ghi đầu tiên cho những thông tin đặc biệt. Bản ghi đầu tiên mô tả chính MFT đó, theo sau là một bản sao của nó. Nếu bản ghi đầu tiên bị lỗi thì NTFS sẽ đọc bản ghi thứ hai để tìm MFT mirror file, file mà bản ghi đầu tiên của nó giống với bản ghi đầu tiên trong MFT. Vị trí của những segment dữ liệu cho MFT và bản sao MFT được ghi trong boot sector. Bản sao của boot sector nằm ở giữa đĩa. MFT xác định vùng trống cho mỗi bản ghi file và những thuộc tính của file được ghi trên vùng trống này. Các file có dung lượng nhỏ hoặc là các thư mục (thường là 1500 bytes hoặc nhỏ hơn), như là những file được minh họa trong bảng số liệu trên, có thể được chứa toàn bộ trong bản ghi MFT.



Hình 12: Master File Table.

- **File System Storage**

Hệ thống tệp NTFS xem từng tệp (hoặc thư mục) dưới dạng một tập hợp các thuộc tính tệp. Các yếu tố như tên của tệp, thông tin bảo mật của nó và thậm chí dữ

liệu của nó, là tất cả các thuộc tính tệp. Mỗi thuộc tính là được xác định bằng mã loại thuộc tính và tùy chọn một tên thuộc tính.

- **Master File Table Copy**

Để chứa các bản sao của các bản ghi khóa liên quan đến việc khôi phục dữ liệu. Trong trường hợp có sự cố với bản gốc, để khôi phục lại bản gốc thì phần này là bắt buộc có.

1.5. So sánh giữa FAT32 và NTFS:

Cơ sở để so sánh	FAT32	NTFS
Căn bản	Cấu trúc đơn giản	Cấu trúc phức tạp
Số lượng ký tự tối đa được hỗ trợ trong một tên tệp	83	255
Kích thước tệp tối đa	4GB	16TB
Mã hóa	Không cung cấp	Cung cấp
Bảo vệ	Dạng kết nối	Địa phương và mạng
Chuyển đổi	Được phép	Không cho phép
Chịu lỗi	Không có quy định cho khả năng chịu lỗi.	Tự động khắc phục sự cố
Khả năng tương thích với các hệ điều hành	Phiên bản windows cũ- Win 95/98 / 2K / 2K3 / XP	Các phiên bản mới hơn - Giành NT / 2K / XP / Vista / 7
Danh sách điều khiển truy cập	Không	Vâng
Dung lượng đĩa người dùng	Không	Vâng
Nhật ký và nhật ký kênh	Vắng mặt	Cung cấp nhật ký để theo dõi các hoạt động trước đó.
Hiệu suất	Tốt	Tốt hơn so với FAT32
Liên kết cứng và mềm	Không có mặt	Chứa đựng
Tốc độ truy cập	Ít tương đối	Hơn
Nén	Không cung cấp nén.	Hỗ trợ nén file.
Khả năng chịu lỗi	Thấp	Cao

Bảng 2: So sánh FAT32 và NTFS.

Sự khác biệt chính giữa 2 hệ thống là:

-FAT32 đơn giản trong khi cấu trúc NTFS khá phức tạp.

-NTFS có thể hỗ trợ kích thước tệp và âm lượng lớn hơn cùng với tên tệp lớn so với hệ thống tệp FAT32.

-FAT32 không cung cấp mã hóa và bảo mật nhiều trong khi NTFS được kích hoạt với bảo mật và mã hóa.

-Khá dễ dàng để chuyển đổi một hệ thống tệp FAT thành một hệ thống khác mà không mất dữ liệu. Ngược lại, chuyển đổi NTFS rất khó đạt được.

-Hiệu năng NTFS tương đối tốt hơn so với FAT32 vì nó cũng cung cấp khả năng chịu lỗi.

-Các tập tin được truy cập nhanh hơn trong trường hợp NTFS. Ngược lại, FAT32 chậm hơn NTFS.

-NTFS truyền các tính năng như ghi nhật ký và nén, không được cung cấp bởi FAT32.

=> Nhận xét: Giữa các hệ thống tệp FAT32 và NTFS, hệ thống tệp NTFS là công nghệ mới hơn cung cấp nhiều tính năng hơn so với FAT32 như độ tin cậy, bảo mật và kiểm soát truy cập, hiệu quả lưu trữ, kích thước và tên tệp nâng cao. Mặc dù vậy, FAT32 vẫn được sử dụng vì tính tương thích của nó.

CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1. Phân tích:

2.1.1. Phân tích yêu cầu:

Xây dựng chương trình thực hiện các chức năng: Đưa ra màn hình Ổ đĩa, tên ổ đĩa số serial, trạng thái, định dạng, dung lượng, còn trống, số byte/sector, số sector/cluster.

2.1.2. Phân tích chức năng:

Từ yêu cầu trên, bám sát theo những thông tin cần hiển thị trên màn hình sẽ có những phần chính sau:

-Đọc thông tin của các ổ đĩa cứng và hiển thị qua giao diện:

Thông tin	**Mô tả**
productId	Tên và mã của ổ đĩa cứng
serialNumber	Số serial
driveType	Định dạng
bytesPerSector	Số bytes trên mỗi sector
sectorsPerTrack	Số sectors trên mỗi track
tracksPerCylinder	Số tracks trên mỗi cylinder
diskSize	Dung lượng của ổ đĩa cứng
cylinders	Tổng số cylinder trong ổ đĩa cứng

Bảng 3: Thông tin ổ đĩa cứng.

-Đọc thông tin của các phân vùng và hiển thị qua giao diện:

Thông tin	**Mô tả**
pathName	Đường dẫn.
volumeName	Tên phân vùng.
fileSystemName	Định dạng phân vùng.
type	Kiểu phân vùng

bytesPerSector	Số bytes trên mỗi sector
sectorsPerCluster	Số sectors trên mỗi cluster
freeClusters	Số cluster chưa sử dụng
allClusters	Tổng số cluster – dung lượng phân vùng
size	Tổng dung lượng của phân vùng
sizeFree	Dung lượng chưa sử dụng của phân vùng
sizeUsed	Dung lượng đang dùng của phân vùng

Bảng 4: Thông tin phân vùng.

2.2. Thiết kế:

2.2.1. Thiết kế chức năng:

Với mỗi bảng thông tin trên thì tạo các cấu trúc tương ứng để chứa đựng thông tin đó. Tạo cấu trúc HardDrive để chứa các thông tin của ổ đĩa cứng và LogicalDrive để chứa các thông tin của phân vùng.

```
struct HardDrive{
	string productId, serialNumber;
	unsigned long bytesPerSector, sectorsPerTrack,
tracksPerCylinder;
	long long diskSize, cylinders;
	string driveType;
};

struct LogicalDrive{
	string pathName, volumeName, fileSystemName;
	unsigned long bytesPerSector, sectorsPerCluster,
freeClusters, allClusters;
	long long size, sizeFree, sizeUsed;
	string type;
	};
```

Vì có thể nhiều ổ đĩa cứng, cũng như có thể có nhiều phân vùng khác nữa nên cần được lưu những cấu trúc trên vào 2 danh sách:

```
typedef vector<HardDrive*> ListHardDrive;
typedef vector<LogicalDrive*> ListLogicalDrive;
```

Sau khi đã khởi tạo cấu trúc cũng như định nghĩa danh sách lưu trữ xong, tiếp theo khởi tạo 2 class để thực hiện các chức năng tạo dựng, lấy thông tin lưu vào cấu trúc rồi lưu vào danh sách, hủy và xóa danh sách. Tạo class HardDriveInfo và class LogicalDriveInfo lần lượt tương ứng với thao tác với thông tin đĩa cứng và thao tác với thông tin phân vùng. Tạo hàm cho class.

-Với class HardDriveInfo:

getString()	Lấy thông tin từ bộ nhớ đệm về dưới dạng String.
destroyListHardDrives()	Giải phóng danh sách chứa thông tin ổ đĩa cứng.
readHardDrives()	Đọc thông tin của các ổ đĩa cứng.
getHardDrives()	Lấy danh sách chứa thông tin ổ đĩa cứng.

Bảng 5: Hàm của class HardDriveInfo.

-Với class LogicalDriveInfo:

destroyListLogicalDrives()	Giải phóng danh sách chứa thông tin phân vùng.
readLogicalDrives()	Đọc thông tin của các phân vùng.
getLogicalDrives()	Lấy danh sách chứa thông tin các phân vùng.

Bảng 6: Hàm của class LogicalDriveInfo

- **getString():**

Hàm này thực hiện chức năng phân tích các thông tin lấy được ở buffer thành các thông tin cần thiết như tên nhà sản xuất, mã sản phẩm, số hiệu...

- **destroyListHardDrives():**

Hàm này thực hiện chức năng giải phóng danh sách chứa thông tin ổ cứng với việc kiểm tra nếu danh sách không rỗng thì sẽ đẩy phần tử cuối danh sách ra.

- **readHardDrives():**

Hàm này thực hiện chức năng đọc thông tin của ổ cứng. Với mỗi thông tin của mỗi ổ cứng đọc được sẽ tiến hành thêm vào danh sách các ổ cứng. Ta sẽ sử dụng API của WIN để hỗ trợ trong hàm này. Bao gồm:

-Hàm CreateFile(): tạo hoặc mở một file hay thiết bị IO.

-Hàm DeviceIoControl(): Gửi mã thiết bị tương ứng để xác định thao tác thực hiện tương ứng với từng thiết bị.

-Struct STORAGE_PROPERTY_QUERY: Cho biết các thuộc tính của thiết bị lưu trữ hoặc bộ điều hợp để truy xuất khi bộ đệm đầu vào được chuyển tới mã điều khiển IOCTL_STORAGE_QUERY_PROPERTY.

-Struct STORAGE_DEVICE_DESCRIPTOR: Được sử dụng cùng với mã điều khiển IOCTL_STORAGE_QUERY_PROPERTY để truy xuất dữ liệu bộ mô tả thiết bị lưu trữ cho một thiết bị.

-Struct DISK_GEOMETRY_EX: mô tả hình dạng mở rộng của thiết bị đĩa và phương tiện.

- **Hàm getHardDrives():**

Hàm này thực hiện việc trả về danh sách thông tin ổ cứng, dùng để hiển thị ở giao diện.

- **Hàm destroyListLogicalDrives():**

Hàm này thực hiện chức năng giải phóng danh sách chứa thông tin phân vùng với việc kiểm tra nếu danh sách không rỗng thì sẽ đẩy phần tử cuối danh sách ra.

- **Hàm readLogicalDrives():**

Hàm này thực hiện chức năng đọc thông tin của Logical Partition. Với mỗi thông tin của mỗi phân vùng đọc được sẽ tiến hành thêm vào danh sách các phân vùng. Ta sẽ sử dụng API của WIN để hỗ trợ trong hàm này. Bao gồm:

-Hàm GetLogicalDriveStrings(): Trả về buffer cho chuỗi các giá trị drive hợp lệ của hệ thống.

-Hàm GetDriveType(): Lấy thông tin kiểu thiết bị như Unknown, No Root Directory, Removable Disk, Local Disk, Network Drive, Compact Disc, RAM Disk.

-Hàm GetDiskFreeSpace(): Trả về thông tin xác định của phân vùng bao gồm: số sectors trên mỗi cluster, số bytes trên mỗi sector, số cluster chưa dùng, số clusters tổng.

-Hàm GetVolumeInformation(): Truy vấn thông tin file hệ thống và các thông tin liên quan đến thư mục gốc. Để xác định khi xử lý các thông tin này ta dùng hàm **GetVolumeInformationByHandleW()**.

- **Hàm getLogicalDrives():**

Hàm này thực hiện việc trả về danh sách thông tin ổ cứng, dùng để hiển thị ở giao diện.

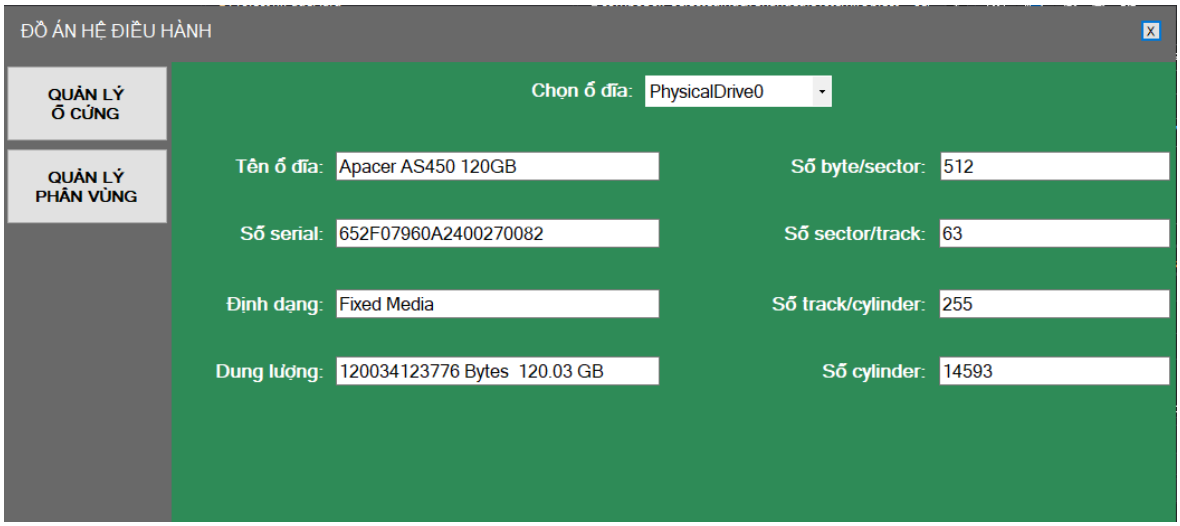
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

3.1. Triển khai:

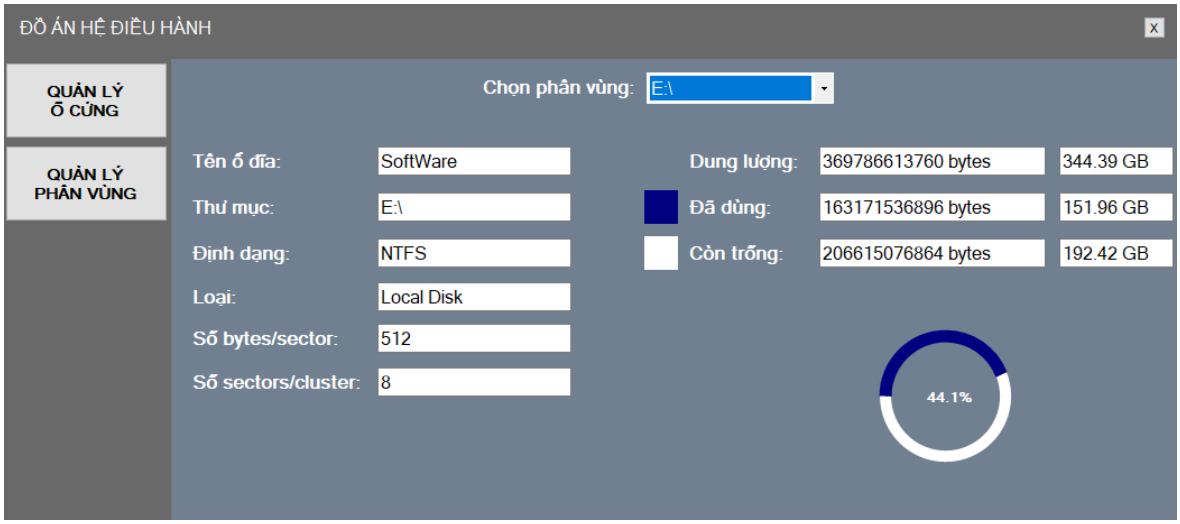
Lựa chọn ngôn ngữ và IDE: Chương trình được code bằng ngôn ngữ C++ trên Visual Studio 2019, sử dụng UI Windows Form C++/CLI để thiết lập giao diện.

Microsoft Visual C++ (còn được gọi là MSVC) là một môi trường phát triển tích hợp (IDE) được sử dụng để tạo các ứng dụng Windows trong các ngôn ngữ lập trình C, C++ và C++ / CLI. Nó chứa các công cụ cho việc phát triển và gỡ lỗi mã nguồn C++, đặc biệt là các mã nguồn viết cho Microsoft Windows API, DirectX API, và Microsoft.NET Framework.

3.2. Kết quả triển khai:



Hình 13: Thông tin ổ cứng.



Hình 14: Thông tin phân vùng.

Chương trình có 2 chức năng chính:

-Quản lý ổ cứng: Hiển thị danh sách các ổ cứng hiện thời, người sử dụng sẽ chọn ổ cứng cần xem và các thông tin sẽ được hiển thị ngay bên dưới để dễ dàng xem được.

-Quản lý phân vùng: Hiển thị danh sách các phân vùng hiện có của máy tính, người dùng sẽ chọn phân vùng cần xem và các thông tin sẽ được hiển thị ở dưới.

3.3. Đánh giá kết quả:

Qua kết quả triển khai trên, có thể đánh giá chương trình đã đáp ứng được yêu cầu của đề tài cho phép người dùng xem một số thông tin của các ổ cứng và thông tin của các phân vùng logic được định dạng theo kiểu NTFS hoặc FAT32.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận:

1.1. Đạt được:

Qua bài báo cáo, đã tìm hiểu được về cách thức hoạt động của ổ đĩa cứng, cũng như rõ hơn về hệ thống quản lý tệp tin của hệ điều hành MS Windows, cách thức hoạt động quản lý của hệ thống tệp và các định dạng NTFS, FAT32.

Có thể tiếp thu được cách thức cài đặt chương trình lấy thông tin ổ cứng và phân vùng trên máy tính, thu thập được nhiều kinh nghiệm hơn trong việc tạo dựng project với ngôn ngữ C++ trên môi trường phát triển tích hợp Visual Studio.

1.2. Khó khăn:

Tuy nhiên vẫn tồn tại một số hạn chế:

-Chưa cho thấy được trạng thái của ổ đĩa cứng.

-Chưa cho thấy được điểm bắt đầu và kết thúc của phân vùng đó trên ổ đĩa.

-Chưa cho thấy được phân vùng thuộc một hay nhiều ổ đĩa và ngược lại, ổ đĩa có một hay nhiều phân vùng.

2. Hướng phát triển:

Tiếp tục tìm hiểu và nghiên cứu sâu hơn về phần cơ sở lý thuyết để hoàn thiện và dễ dàng truyền đạt đến người đọc.

Tiếp tục xây dựng thêm một số tính năng như: điểm bắt đầu và kết thúc của mỗi phân vùng, số lượng phân vùng của một ổ đĩa, phân vùng nào thuộc ổ đĩa nào.

PHẦN II: LẬP TRÌNH MẠNG

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

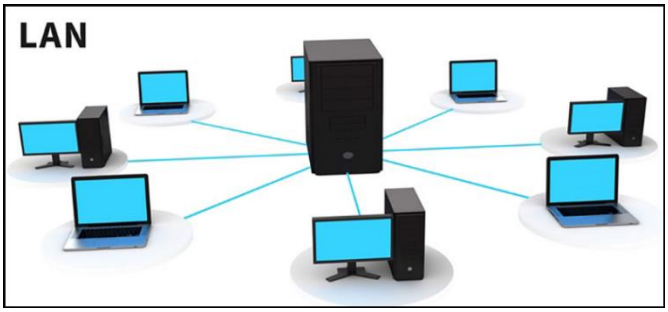
1.1. Mạng máy tính:

1.1.1. Khái niệm:

Mạng máy tính là tập hợp các máy tính hoặc các thiết bị được nối với nhau bởi các đường truyền vật lý và theo một kiến trúc nào đó.

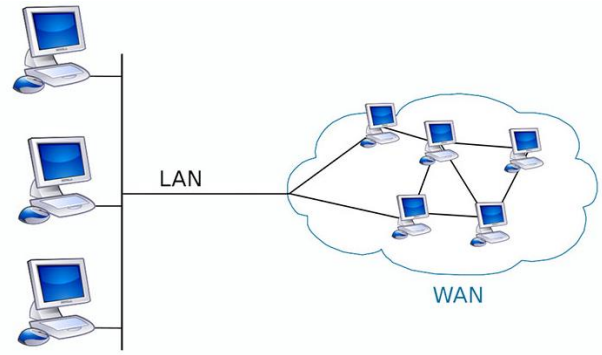
Chúng ta có thể phân loại mạng theo qui mô của nó:

Mạng LAN (Local Area Network): mạng cục bộ kết nối các nút trên một phạm vi giới hạn. Phạm vi này có thể là một công ty hay một tòa nhà.



Hình 15: Mạng LAN.

Mạng WAN (Wide Area Network): mạng diện rộng là nhiều mạng LAN kết nối với nhau tạo thành mạng WAN.



Hình 16: Mạng WAN.

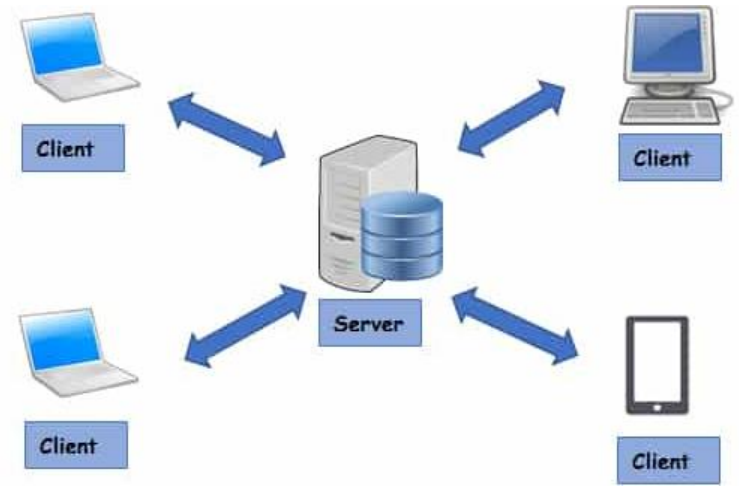
Mạng MAN (Metropolitan Area Network): tương tự như mạng WAN, cũng kết nối nhiều mạng LAN. Tuy nhiên, một mạng MAN có phạm vi là một thành phố hay một đô thị nhỏ. MAN sử dụng các mạng tốc độ cao để kết nối các mạng LAN của trường học, chính phủ, công ty… bằng cách sử dụng các liên kết nhanh tới từng điểm như cáp quang.

Hình 17: Mạng MAN.

1.2. Mô hình Client – Server:

1.2.1. Khái niệm:

Mô hình Client – Server (máy khách – máy chủ) là mô hình tổ chức trao đổi thông tin trong đó mô tả cách mà các máy tính có thể giao tiếp với nhau theo một phương thức nhất định. Phương thức này là một chiến lược tổ chức phân cấp mà trong đó có một máy tính đặc biệt phục vụ các yêu cầu về lưu trữ, xử lý, tính toán tất cả các máy trên mạng. Kiểu tổ chức tổng quát của mô hình này là một mạng LAN được thiết lập từ nhiều máy tính khác nhau, trong đó một máy tính gọi là Server. Một chương trình Client chạy từ bất kỳ máy tính nào trong mạng cũng có thể gởi yêu cầu của mình đến Server, khi Server nhận được các yêu cầu này thì nó sẽ thực hiện và gởi kết quả về cho Client.



Hình 18: Mô hình Client - Server.

Có nhiều mô hình được sử dụng trong các chương trình mạng, nhưng mô hình Client – Server là mô hình chuẩn. Một Server là một quá trình, quá trình này chờ sự liên kết từ một Client. Một phiên làm việc điển hình của mô hình này như sau:

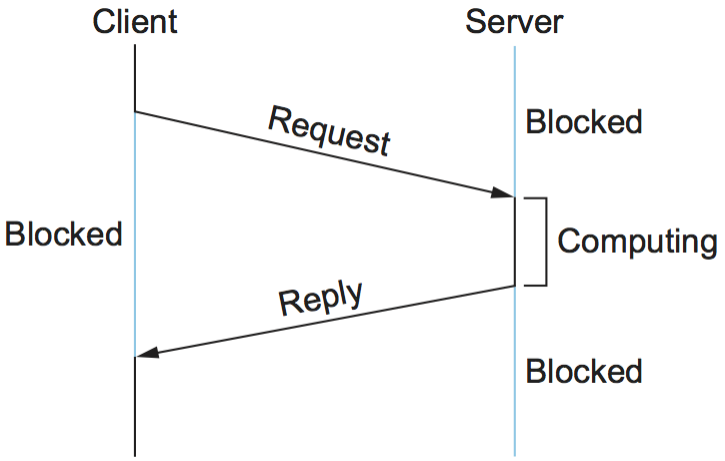
-Client gởi yêu cầu qua mạng đến Server để yêu cầu một số dạng dịch vụ.

-Server sẽ chờ đợi yêu cầu của Client và phản hồi lại kết quả.

Lợi ích của mô hình này là nó có thể làm việc trên bất cứ mạng máy tính nào có hỗ trợ giao thức truyền thông chuẩn cụ thể ở đây là giao thức TCP/IP.

1.2.2. Đặc trưng của mô hình Client – Server:

Hoạt động theo kiểu giao thức bất đối xứng. Thể hiện quan hệ một chiều giữa các Client và một Server. Client bắt đầu phiên làm việc bằng cách yêu cầu dịch vụ, Server sẵn sàng chờ các yêu cầu từ Client và phản hồi.

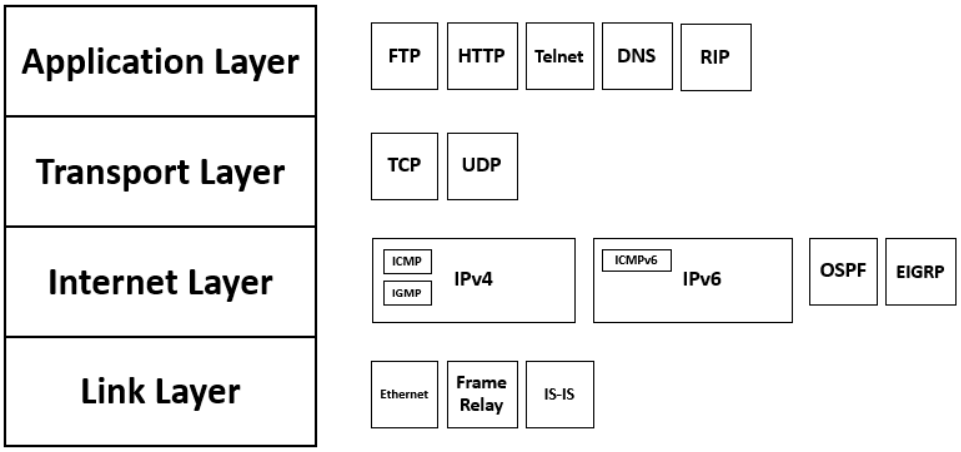


Hình 19: Giao thức hoạt động bất đối xứng của Client - Server.

Chia sẻ tài nguyên: Một Server có thể chia sẻ tài nguyên cho nhiều Client cùng một lúc. Server cũng có khả năng điều phối truy nhập các Client đến các tài nguyên dùng chung.

1.3. Giao thức TCP/IP:

TCP/IP là tên chung cho một tập hợp hơn 100 giao thức được sử dụng để kết nối các máy tính vào mạng, trong đó hai giao thức chính là TCP (Transmission Control Protocol) và IP (Internet Protocol).

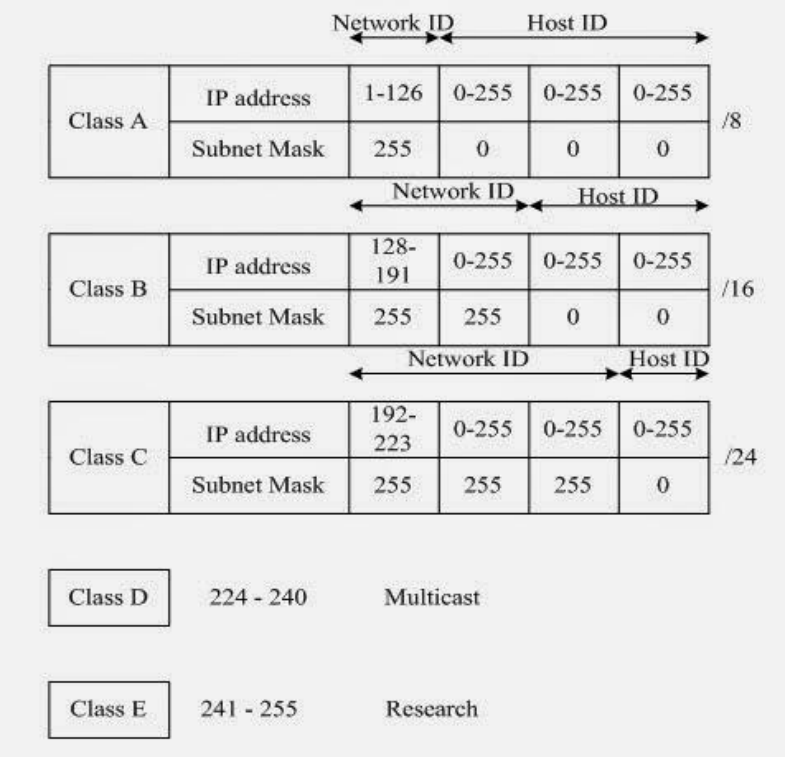


Hình 20: Mô hình TCP/IP.

1.3.1. Giao thức IP:

Internet Protocol (tiếng Anh, viết tắt: IP, có nghĩa là Giao thức Internet) là một giao thức hướng dữ liệu được sử dụng bởi các máy chủ nguồn và đích để truyền dữ liệu trong một liên mạng chuyển mạch gói.

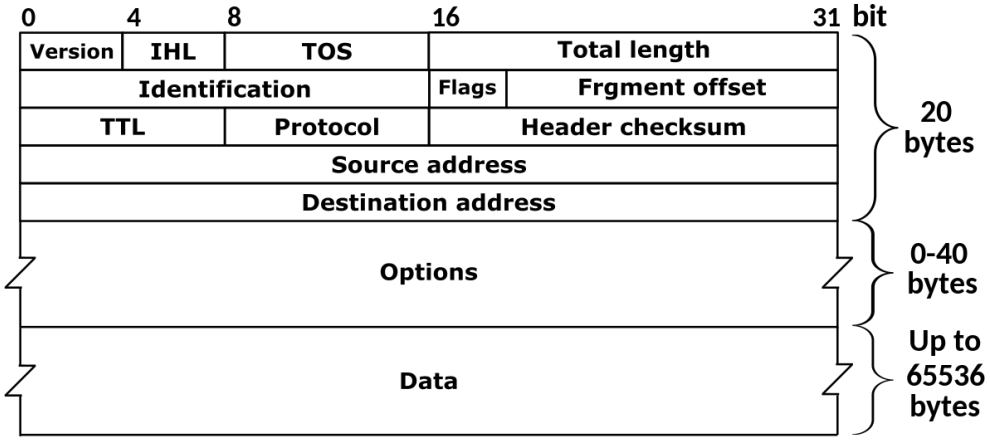
Dữ liệu trong một liên mạng IP được gửi theo các khối được gọi là các gói (packet hoặc datagram). Cụ thể, IP không cần thiết lập các đường truyền trước khi một máy chủ gửi các gói tin cho một máy khác mà trước đó nó chưa từng liên lạc với. IP cung cấp một dịch vụ gửi dữ liệu không đảm bảo. Gói dữ liệu có thể đến nơi mà không còn nguyên vẹn, nó có thể đến không theo thứ tự (so với các gói khác được gửi giữa hai máy nguồn và đích đó), nó có thể bị trùng lặp hoặc bị mất hoàn toàn. Nếu một phần mềm ứng dụng cần được bảo đảm, nó có thể được cung cấp từ nơi khác, thường từ các giao thức giao vận nằm phía trên IP.



Hình 21: IPv4.

Địa chỉ IP được chia thành 4 số giới hạn từ 0 - 255. Mỗi số được lưu bởi 1 byte - > IP có kích thước là 4byte, được chia thành các lớp địa chỉ. Có 3 lớp là A, B, và C. Nếu ở lớp A, ta sẽ có thể có 16 triệu địa chỉ, ở lớp B có 65536 địa chỉ.

Trên Internet thì địa chỉ IP của mỗi người là duy nhất và nó sẽ đại diện cho chính người đó, địa chỉ IP được sử dụng bởi các máy tính khác nhau để nhận biết các máy tính kết nối giữa chúng.



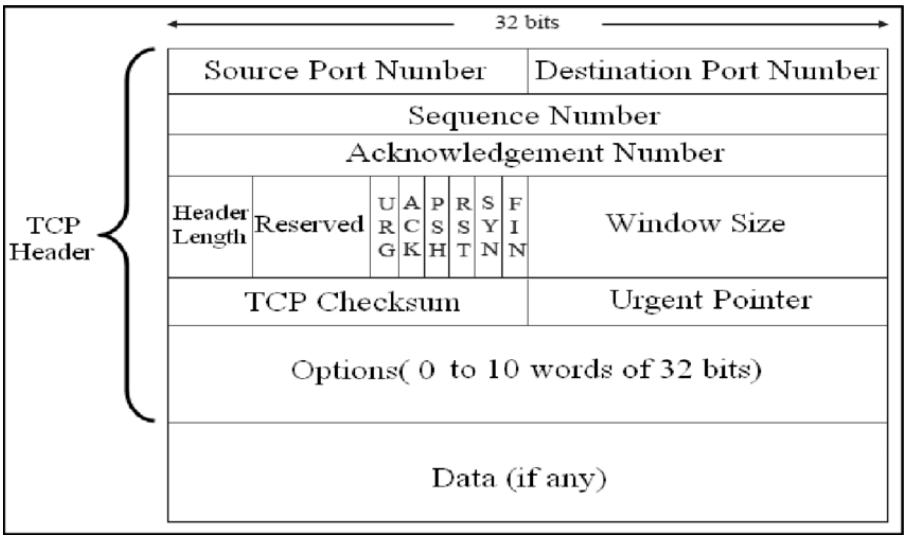
Hình 22: IP header.

1.3.2. Giao thức TCP:

TCP (Transmission Control Protocol - "Giao thức điều khiển truyền vận") là một trong các giao thức cốt lõi của bộ giao thức TCP/IP. Sử dụng TCP, các ứng dụng trên các máy chủ được nối mạng có thể tạo các "kết nối" với nhau, mà qua đó chúng có thể trao đổi dữ liệu hoặc các gói tin. Giao thức này đảm bảo chuyển giao dữ liệu tới nơi nhận một cách đáng tin cậy và đúng thứ tự. TCP còn phân biệt giữa dữ liệu của nhiều ứng dụng (chẳng hạn, dịch vụ Web và dịch vụ thư điện tử) đồng thời chạy trên cùng một máy chủ. TCP hỗ trợ nhiều giao thức ứng dụng phổ biến nhất trên Internet và các ứng dụng kết quả, trong đó có WWW, thư điện tử và Secure Shell.

Các ứng dụng gửi các dòng gồm các byte 8-bit tới TCP để chuyển qua mạng. TCP phân chia dòng byte này thành các đoạn (segment) có kích thước thích hợp (thường được quyết định dựa theo kích thước của đơn vị truyền dẫn tối đa (MTU) của tầng liên kết dữ liệu của mạng mà máy tính đang nằm trong đó). Sau đó, TCP chuyển các gói tin thu được tới giao thức IP để gửi nó qua một liên mạng tới mô đun TCP tại máy tính đích.

TCP kiểm tra để đảm bảo không có gói tin nào bị thất lạc bằng cách gán cho mỗi gói tin một "số thứ tự" (sequence number). Số thứ tự này còn được sử dụng để đảm bảo dữ liệu được trao cho ứng dụng đích theo đúng thứ tự. Mô đun TCP tại đầu kia gửi lại "tin báo nhận" (acknowledgement) cho các gói tin đã nhận được thành công; một "đồng hồ" (timer) tại nơi gửi sẽ báo time-out nếu không nhận được tin báo nhận trong khoảng thời gian bằng một round-trip time (RTT), và dữ liệu (được coi là bị thất lạc) sẽ được gửi lại. TCP sử dụng checksum (giá trị kiểm tra) để xem có byte nào bị hỏng trong quá trình truyền hay không; giá trị này được tính toán cho mỗi khối dữ liệu tại nơi gửi trước khi nó được gửi, và được kiểm tra tại nơi nhận.



Hình 23: TCP header.

1.4. Socket trong Python:

1.4.1. Khái niệm Socket:

Một Socket là một end-point của liên kết giữa hai ứng dụng. Socket cho phép giao tiếp trong 1 hoặc giữa những tiến trình trên cùng 1 hoặc nhiều máy với nhau.

Trong hệ thống mạng có rất nhiều ứng dụng bao gồm chương trình khách và chương trình chủ ở 2 hệ cuối khác nhau. Sau khi được kích hoạt, một tiến trình khách và chủ được tạo và mục đích của Socket được sử dụng là để giúp 2 tiến trình này có thể truyền thông với nhau dễ dàng.

Socket được chia chủ yếu thành 2 loại:

Stream Socket: Dựa trên giao thức TCP, chỉ thực hiện trên 2 tiến trình đã thiết lập kết nối. Giao thức này đảm bảo dữ liệu truyền đến nơi nhận 1 cách tin cậy và đúng tuần tự. Còn được gọi là Socket hướng kết nối.

Datagram Socket: Dựa trên giao thức UDP, không yêu cầu thiết lập kết nối. Giao thức này không đảm bảo dữ liệu truyền đến nơi nhận 1 cách tin cậy và toàn vẹn. Còn được gọi là Socket hướng không kết nối.

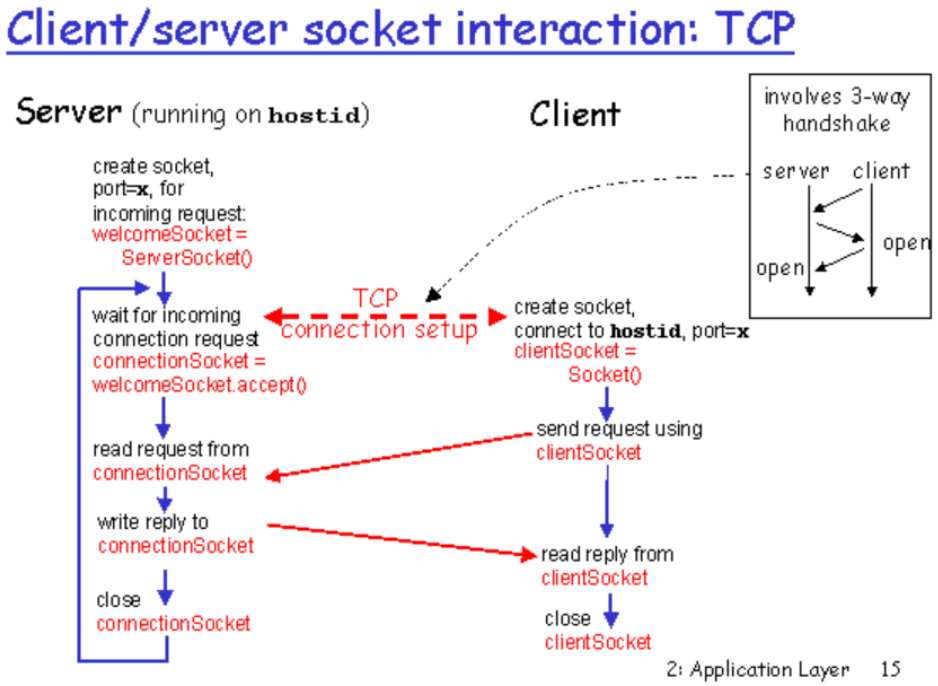
1.4.2. Lập trình Socket với TCP/IP:

Các tiến trình muốn truyền thông với nhau sẽ gửi thông điệp thông qua các Socket. Socket là cánh cửa của tiến trình ứng dụng và giao thức tầng transport (TCP). Để có thể tương tác với nhau và máy chủ có thể nhận liên lạc từ máy khách thì máy chủ phải luôn sẵn sàng. Điều này có 2 nghĩa:

-Thứ nhất, giống như trường hợp của UDP, một tiến trình của máy chủ phải được chạy trước khi máy khách khởi tạo liên lạc đến.

-Thứ hai, chương trình chủ phải tạo ra 1 Socket để sẵn sàng chấp nhận kết nối từ tiến trình khách. Khi tiến trình chủ đã chạy, lúc này tiến trình khách sẽ tạo ra 1 Socket TCP để có thể kết nối đến máy chủ. Trong khi máy khách đang tạo TCP Socket, nó sẽ đặc tả địa chỉ IP, số cổng của tiến trình chủ. Khi Socket của tiến trình khách vừa được tạo, TCP trên máy khách sẽ tiến hành thực hiện quá trình bắt tay 3 bước và thiết lập kết nối TCP tới máy chủ.

Trong quá trình bắt tay 3 bước, khi tiến trình chủ nhận thấy tiến trình khách, nó sẽ tự tạo ra 1 Socket mới chỉ dành riêng cho tiến trình khách đó. Khi được máy khách gõ cửa, chương trình kích hoạt với phương thức accept(). Cuối quá trình bắt tay 3 bước, kết nối TCP tồn tại giữa Socket của máy khách và Socket của máy chủ.



Hình 24: Client - Server Socket

Một Socket có thể thực hiện bảy thao tác cơ bản: Kết nối với một máy ở xa (ví dụ, chuẩn bị để gửi và nhận dữ liệu); Gửi dữ liệu; Nhận dữ liệu; Ngắt liên kết; Gán cổng; Nghe dữ liệu đến; Chấp nhận liên kết từ các máy ở xa trên cổng đã được gán.

Lớp Socket của Python được sử dụng bởi cả client và server, có các phương thức tương ứng với bốn thao tác đầu tiên. Ba thao tác cuối chỉ cần cho server để chờ các client liên kết với chúng. Các thao tác này được cài đặt bởi module Socket.

Mỗi khi liên kết được thiết lập, các host ở xa nhận các luồng vào và luồng ra từ Socket, và sử dụng các luồng này để gửi dữ liệu cho nhau. Kiểu liên kết này được

gọi là song công (full-duplex)-các host có thể nhận và gửi dữ liệu đồng thời. Ý nghĩa của dữ liệu phụ thuộc vào giao thức. Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết. Một số giao thức, như HTTP, đòi hỏi mỗi liên kết phải bị đóng sau mỗi khi yêu cầu được phục vụ. Các giao thức khác, chẳng hạn FTP, cho phép nhiều yêu cầu được xử lý trong một liên kết đơn.

1.4.3. Socket module trong Python:

Python cung cấp module Socket giúp chúng ta dễ dàng thực hiện kết nối client server để giao tiếp với nhau.

-Để có thể sử dụng được trước tiên ta phải import module Socket vào chương trình: import socket

-Tạo một Socket:

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

Tham số đầu tiên là Address Family: kiểu thiết lập kết nối. Python hỗ trợ 3 dạng:

AF_INET: Ipv4

AF_INET6: Ipv6

AF_UNIX

Tham số thứ hai là Socket Type: cách thiết lập giao thức

SOCK_STREAM: TCP

SOCK_DGRAM: UDP

-Một số phương thức được sử dụng:

s.bind((HOST, PORT))	Đăng ký tên cho Socket, ràng buộc địa chỉ vào Socket
s.listen(2)	Cho Socket đang lắng nghe tới tối đa 2 kết nối
client, addr = s.accept()	Khi một client gõ cửa, server chấp nhận kết nối và 1 Socket mới được tạo ra. Client và server bây giờ đã có thể truyền và nhận dữ liệu với nhau
data = client.recv(1024)	Nhận gói dữ liệu
str_data = data.decode("utf8")	Phân tích gói dữ liệu vừa nhận
s.sendall(bytes(msg, "utf8"))	Gửi dữ liệu thông qua giao thức TCP
s.close()	Đóng một kết nối

Bảng 7: Phương thức Socket trong Python.

CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1. Phân tích:

2.1.1. Phân tích yêu cầu:

Tìm hiểu và sử dụng kỹ thuật lập trình với Socket xây dựng chương trình tính tiền dịch vụ Internet theo mô hình Client - Server.

2.1.2. Phân tích chức năng:

Với yêu cầu trên, chương trình Client và Server sẽ được thực hiện qua các chức năng chính sau:

Client sẽ yêu cầu kết nối đến Server khi được khởi tạo. Client gồm 2 phần:

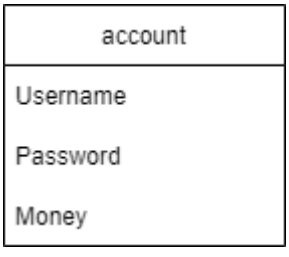
Phần đăng nhập: gửi thông tin người dùng và mật khẩu đến Server để kiểm tra và đợi chờ phản hồi, nếu tài khoản tồn tài trong cơ sở dữ liệu và không đang trong phiên làm việc thì Server sẽ hồi đáp việc xác thực tài khoản để truy cập vào dịch vụ Internet.

Phần trang chủ: Sau khi đã truy cập vào, người dùng có thể tra cứu thông tin qua mạng và có thể phiên dịch chuỗi từ hay văn bảng sang tiếng Việt. Ngoài ra còn hiển thị số tiền dịch vụ Internet mà người dùng đã nạp vào và sẽ phải trả khi đang sử dụng.

Server được xây dựng theo kiểu đa luồng đa tuyến để chấp nhận kết nối từ nhiều Client. Với mỗi Client sẽ thực hiện việc lắng nghe và hồi đáp lời gọi riêng với Client đó. Server còn thực hiện việc tạo tài khoản cho Client, nạp tiền cho Client.

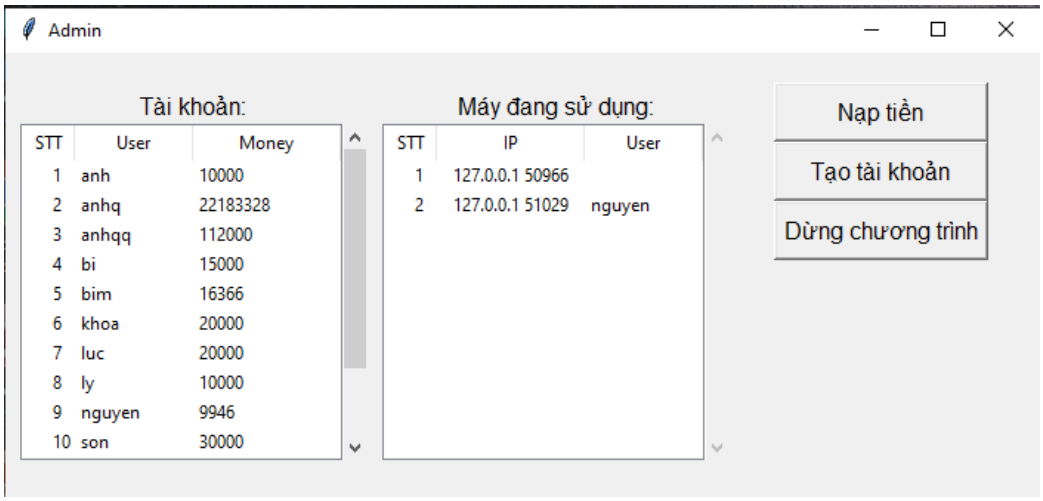
2.2. Thiết kế:

2.2.1. Thiết kế cơ sở dữ liệu:

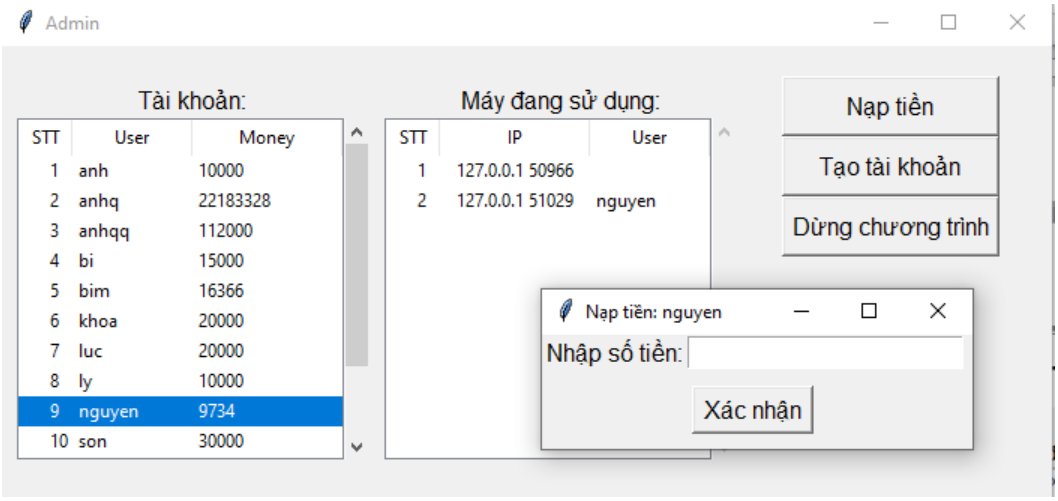


Hình 25: Cơ sở dữ liệu account.

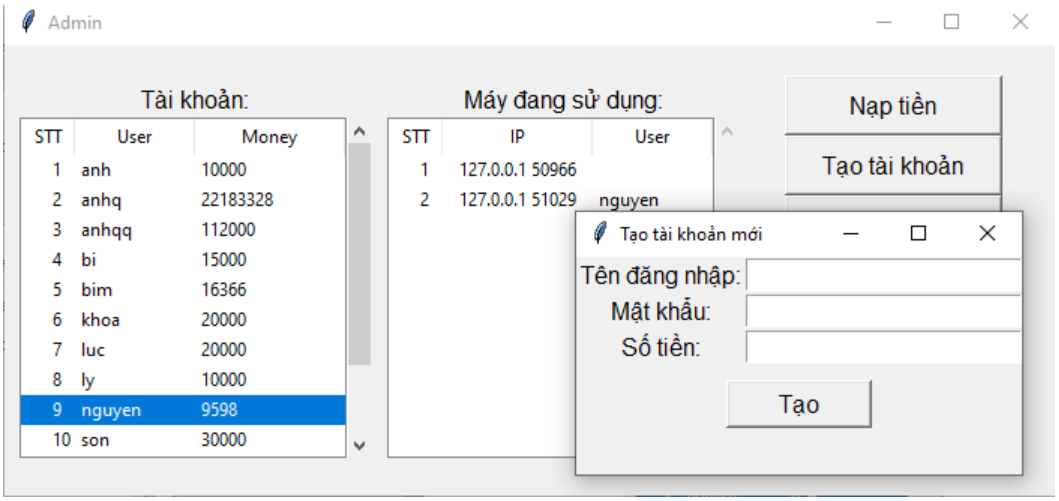
2.2.2. Thiết kế giao diện:



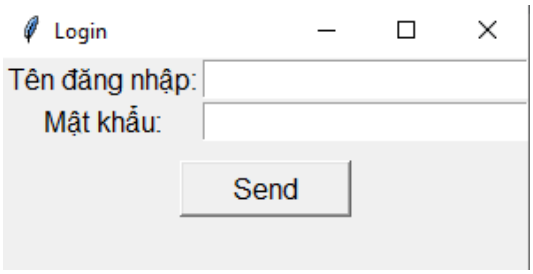
Hình 26: Giao diện chính Server.



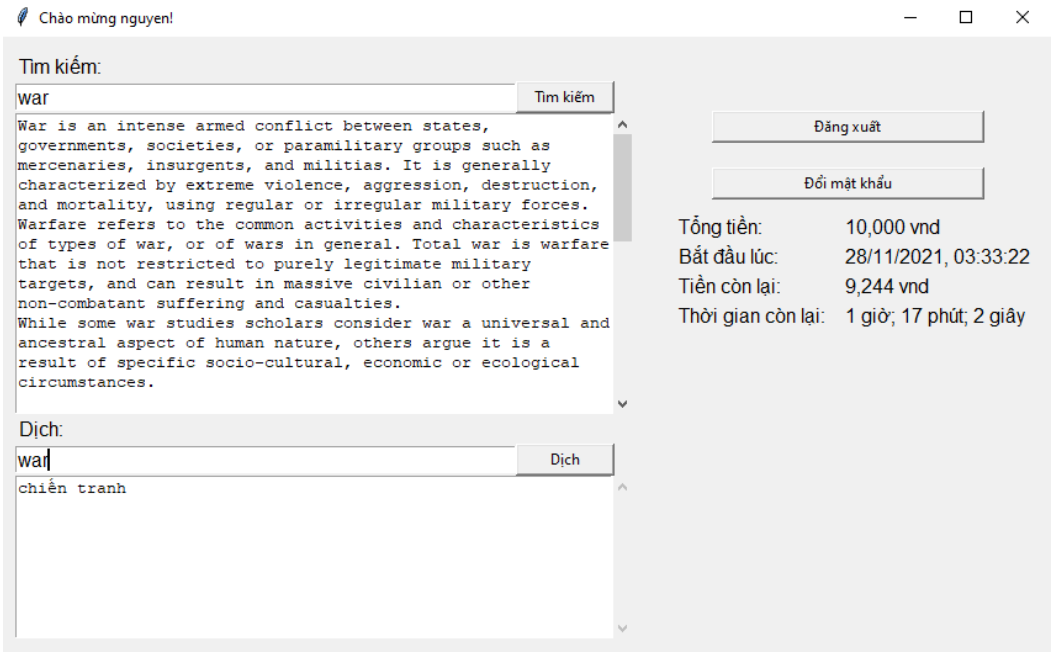
Hình 27: Giao diện nạp tiền Server.



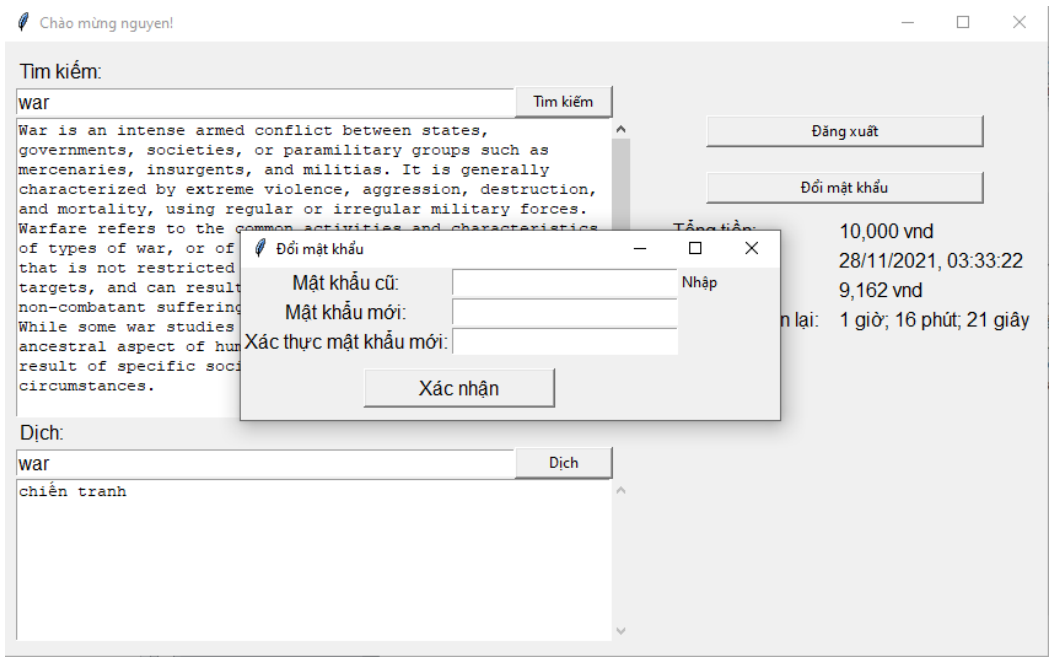
Hình 28: Giao diện tào tài khoản Server.



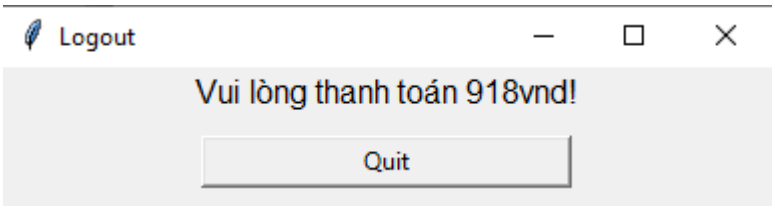
Hình 29: Giao diện đăng nhập Client.



Hình 30: Giao diện chính Client.



Hình 31: Giao diện đổi mật khẩu Client.



Hình 32: Giao diện sau khi đăng xuất Client.

-Tài khoản: hiển thị danh sách các tài khoản đã được đăng ký (không trùng tên nhau).

-Máy đang sử dụng: hiển thị danh sách các Client đang kết nối đến Server và tên người dùng nếu Client đó được người dùng đăng nhập vào bằng tài khoản cá nhân.

-Nạp tiền: dùng để thực hiện việc nạp tiền vào tài khoản của người dùng qua việc chọn tên tài khoản cần nạp trong danh sách **Tài khoản** hoặc tài khoản đang sử dụng bên danh sách **Máy đang sử dụng**.

-Nhập số tiền: nhập số tiền để cộng thêm vào tài khoản đang chọn.

Tạo tài khoản: dùng để tạo tài khoản mới cho người dùng, cần nhập đầy đủ thông tin vào 3 vùng: **Tên đăng nhập**, **Mật khẩu**, **Số tiền**.

-Dừng chương trình: dừng chương trình Server, ngắt kết nối đến các Client.

-Tên đăng nhập, mật khẩu: nhập vào thông tin tài khoản cá nhân người dùng để truy cập sử dụng dịch vụ Internet.

-Send: thực hiện việc đăng nhập.

-Tìm kếm: Tra cứu thông tin trên dịch vụ Internet với chuỗi được nhập ở thanh nhập dữ liệu bên cạnh.

-Dịch: Dịch từ chuỗi được nhập ở thanh nhập dữ liệu bên cạnh sang tiếng Việt.

-Đổi mật khẩu: thực hiện việc đổi mật khẩu của người dùng với 3 dữ liệu đầu vào: **Mật khẩu cũ**, **Mật khẩu mới**, **Xác thực mật khẩu mới**.

-Đăng xuất: thoát khỏi dịch vụ Internet.

2.2.3. Thiết kế chức năng:

-Xây dựng chương trình Client (sử dụng socket.socket()):

+Mở một socket nối kết đến Server đã biết địa chỉ IP ("localhost") và số hiệu Port (5000).

+Tạo vòng lặp cho việc đăng nhập, điều kiện thoát vòng lặp sẽ là dữ liệu xác nhận đăng nhập thành công từ Server. Bên trong vòng lặp là đưa dữ liệu tài khoản được nhập từ giao diện lên Server.

+Sau khi đăng nhập, tạo luồng xử lý việc gửi dữ liệu GET lên Server mỗi 1 giây liên tục.

+Tạo luồng xử lý việc nhận dữ liệu Server trả về liên tục, dữ liệu này sẽ bao gồm: tên tài khoản, mật khẩu, tiền còn lại trong tài khoản, tiền được cộng thêm, chuỗi dữ liệu muốn tra cứu thông tin, chuỗi dữ liệu muốn dịch. Sau khi nhận dữ liệu sẽ hiển thị lên giao diện.

+Hàm xử lý việc gửi dữ liệu tra cứu thông tin được người dùng nhập vào lên Server.

+Hàm xử lý việc gửi dữ liệu cần dịch sang tiếng Việt được người dùng nhập vào lên Server.

+Hàm xử lý việc đổi mật khẩu: cần xác thực mật khẩu cũ trùng khớp, gửi lên Server dữ liệu PASSWORD cùng dữ liệu mật khẩu mới để đổi mật khẩu.

+Hàm xử lý việc đăng xuất: đăng xuất tài khoản khỏi dịch vụ Internet bằng cách gửi dữ liệu DISCONNECT lên Server để đăng xuất tài khoản, ngưng việc trừ tiền tài khoản.

-Xây dựng chương trình Server (sử dụng socket.socket()):

+Đăng ký Server IP ("localhost") và Port (5000) và thực hiện việc lắng nghe yêu cầu kết nối từ Client.

+Với mỗi yêu cầu kết nối từ Client tạo luồng mới xử lý Client vừa kết nối, thêm địa chỉ Client này vào danh sách máy đang sử dụng.

+Bên trong luồng xử lý Client vừa tạo, khởi tạo 1 vòng lặp lớn chứa 2 vòng lặp xử lý lời gọi từ Client:

- Vòng lặp đầu tiên xử lý việc đăng nhập tài khoản của Client: điều kiện thoát khi Client gửi lên đúng tài khoản đã đăng ký, tài khoản đó đang không được sử dụng cũng như còn tiền trong tài khoản. Sau khi thoát khỏi vòng lặp đầu tiên, lưu tài khoản vừa truy cập dịch vụ vào danh sách các tài khoản đang sử dụng. Tiếp theo tạo hàm xử lý việc trừ tiền

tài khoản đó mỗi 1 giây liên tục khi sử dụng dịch vụ Internet, luồng này sẽ ngắt khi tài khoản đăng xuất hoặc hết tiền.

- Vòng lặp thứ hai xử lý các dữ liệu nhận được từ Client như GET, PASSWORD, DISCONNECT, SEARCH, TRANSLATE để hồi đáp lại cho Client dữ liệu tương ứng với lời gọi.

+Tạo hàm thực hiện việc thêm tiền vào tài khoản người dùng.

+Tạo hàm thực hiện việc thêm tài khoản mới.

+Khi nhận dữ liệu PASSWORD thực hiện việc đổi mật khẩu mới được Client gửi lên.

+Khi nhận dữ liệu SEARCH tạo luồng mới và sử dụng thư viện wikipedia để tra cứu thông tin vừa được nhận từ Client rồi gửi trả lại cho Client đó.

+Khi nhận dữ liệu TRANSLATE tạo luồng mới và sử dụng thư viện googletrans với class Translator để dịch thông tin vừa được nhận từ Client rồi gửi trả lại cho Client đó.

+Khi nhận dữ liệu DISCONNECT thì thực hiện việc xóa tài khoản khỏi danh sách tài khoản đang sử dụng, thoát khỏi vòng lặp nhỏ thứ 2 và quay trở lại vòng lặp 1 – xử lý đăng nhập.

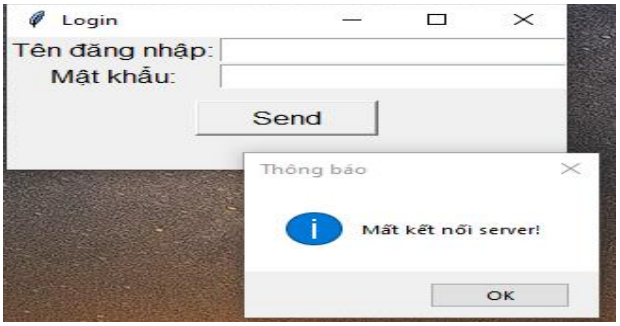
+Cập nhật danh sách tài khoản từ cơ sở dữ liệu liên tục để hiển thị số tiền của người dùng và cập nhật danh sách các máy đang sử dụng từ danh sách máy đang sử dụng kèm theo tài khoản trên máy đó.

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

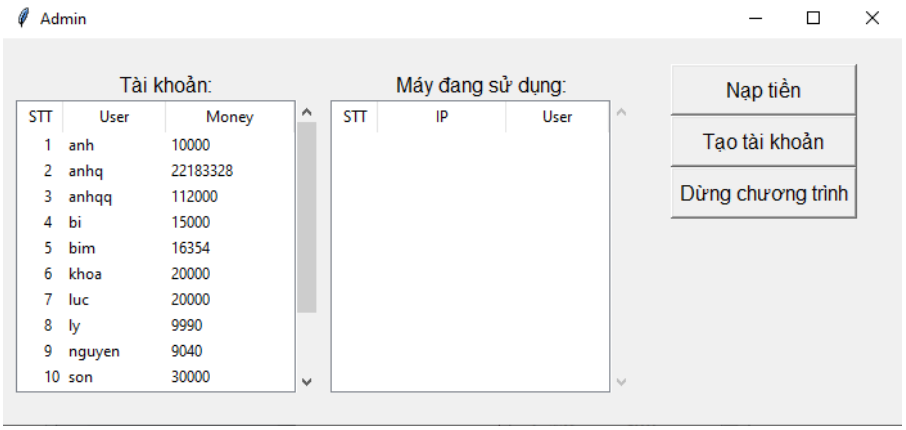
3.1. Triển khai:

Lựa chọn ngôn ngữ và IDE: Chương trình được code bằng ngôn ngữ Python (Python version 3) trên Visual Studio Code (Microsoft VC Code) sử dụng UI Tkinter để thiết lập giao diện.

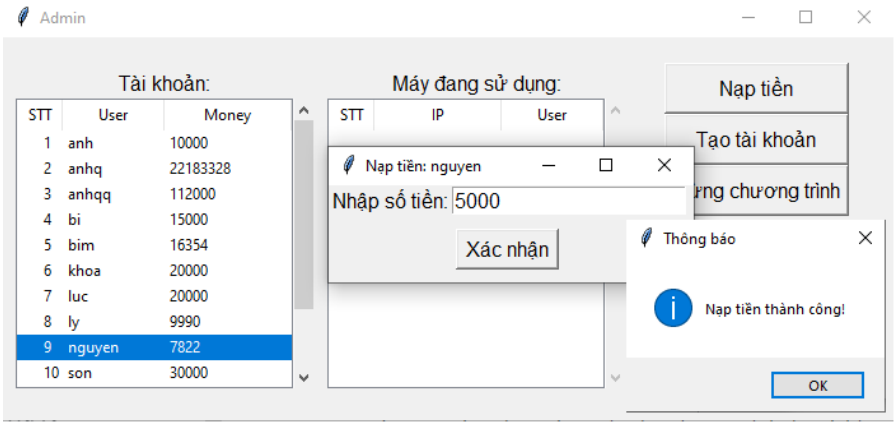
3.2. Kết quả triển khai:



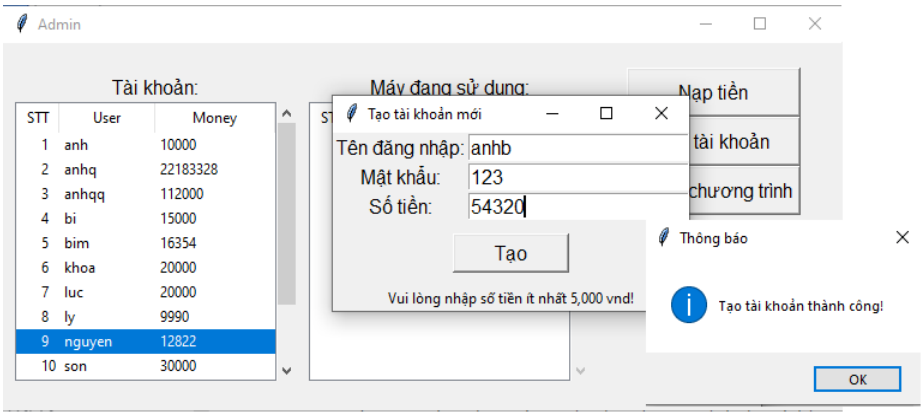
Hình 33: Lỗi kết nối khi Server chưa hoạt động.



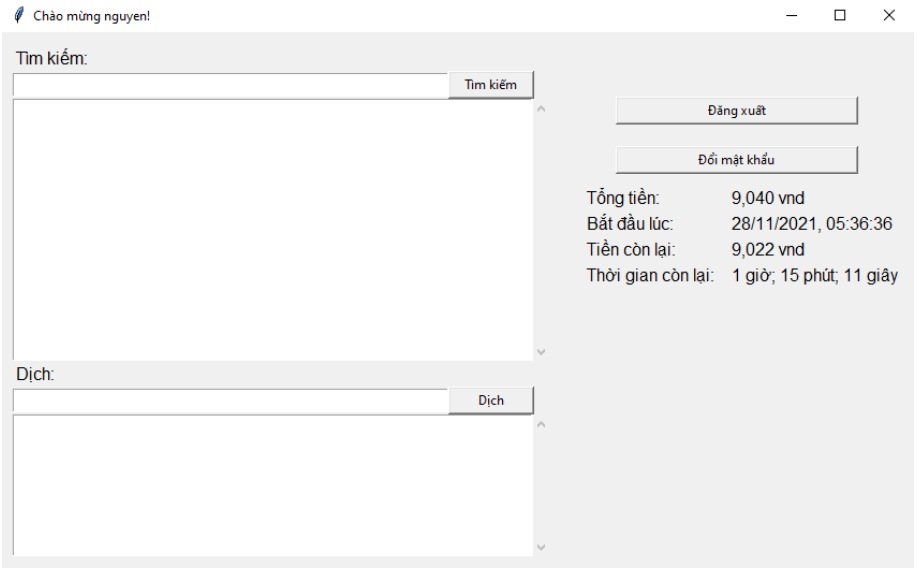
Hình 34: Chạy chương trình Server.



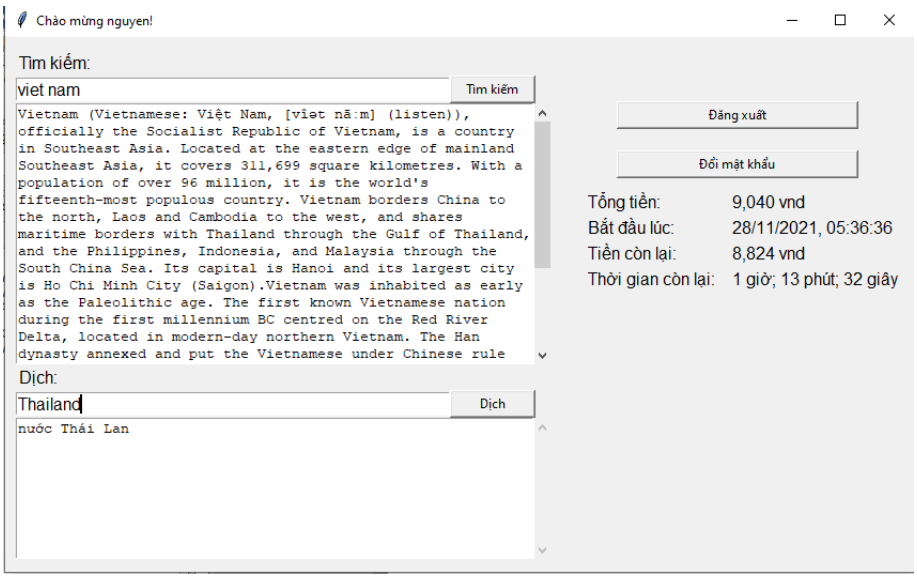
Hình 35: Server nạp tiền thành công.



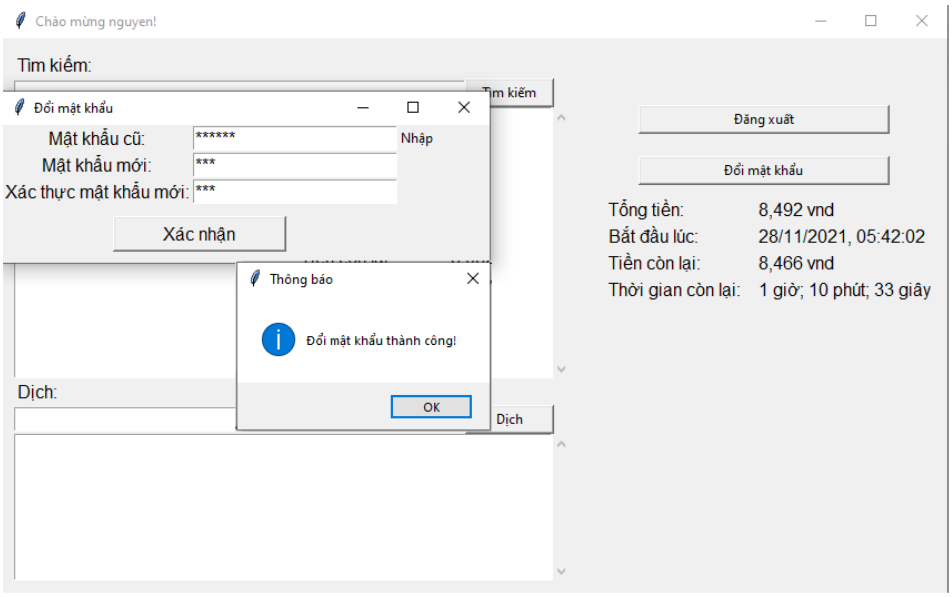
Hình 36: Server tạo tài khoản thành công.



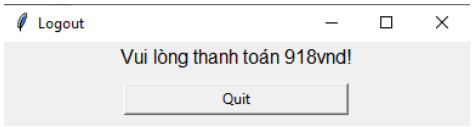
Hình 37: Client đăng nhập thành công.



Hình 38: Client thực hiện các dịch vụ Internet thành công.



Hình 39: Client đổi mật khẩu thành công.



Hình 40: Client sau khi đăng xuất.

3.3. Đánh giá kết quả:

Đã kết nối thành công, xử lý được đa số lỗi từ chương trình. Đáp ứng được yêu cầu tính tiền dịch vụ Internet.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận:

1.1. Đạt được:

-Hoàn thành việc mô phỏng mô hình Client-Server trong mạng LAN.

-Thực hiện được yêu cầu tính tiền dịch vụ Internet của Client-Server.

-Thực hiện được việc kiểm tra kết nối giữa Client-Server và xử lý lỗi mất kết nối.

1.2. Khó khăn:

Tuy nhiên vẫn tồn tại một số hạn chế:

-Dịch vụ Internet đơn giản, sử dụng thư viện có sẵn để tra cứu thông tin và dịch ngôn ngữ.

-Giao diện không bắt mắt.

Chương trình code chưa tối ưu.

2. Hướng phát triển:

-Tối ưu code hơn.

-Thêm tính năng chat giữa Client với Server, khi tin nhắn Client gửi đến sẽ hiển thị cửa sổ chat bên Server và ngược lại.

-Chỉnh sửa giao diện phù hợp hơn, dễ sử dụng hơn cho người dùng.

TÀI LIỆU THAM KHẢO

Tài liệu Hệ điều hành:

1. Cấu tạo và nguyên lý làm việc của ổ cứng (ảnh động 3D) (cuudulieutransang.vn)
2. WI-11_Articulo Final_Jose M. Rodriguez.pdf (prcrepository.org)
3. https://lazytrick.wordpress.com/2015/12/27/khai-quat-ve-fat/
4. NTFS – Wikipedia tiếng Việt
5. Microsoft Windows – Wikipedia tiếng Việt
6. Nguyen_ly_he_dieu_hanh.pdf (mientayvn.com)
7. Winioctl.h header - Win32 apps | Microsoft Docs

Tài liệu Lập trình mạng:

1. Mai Văn Hà, Bài giảng Lập Trình Mạng, Khoa CNTT
2. Microsoft PowerPoint - csb051-python.pptx (nuk.edu.tw)
3. Lập Trình Socket Với TCP/IP Trong Python (codelearn.io)

PHỤ LỤC

Hệ điều hành:

Disk_Manager.h

```
#ifndef Disk_Manager_h_
#define Disk_Manager_h_
#include <vector>
#include <string>
using std::vector;
using std::string;

struct HardDrive{
	string productId, serialNumber;
	unsigned long bytesPerSector, sectorsPerTrack, tracksPerCylinder;
	long long diskSize, cylinders;
	string driveType;
};

struct LogicalDrive{
	string pathName, volumeName, fileSystemName;
	unsigned long bytesPerSector, sectorsPerCluster, freeClusters, allClusters;
	long long size, sizeFree, sizeUsed;
	string type;
};

typedef vector<HardDrive*> ListHardDrive;
typedef vector<LogicalDrive*> ListLogicalDrive;
class HardDriveInfo{
private:
	ListHardDrive listHardDrive;
	char* getString(const char* str, int pos, char* buf);
	void destroyListHardDrives();
public:
	int readHardDrives();
	ListHardDrive getHardDrives();
	HardDriveInfo();
	virtual ~HardDriveInfo();
};
class LogicalDriveInfo{
private:
	ListLogicalDrive listLogicalDrive;
	void destroyListLogicalDrives();
public:
	int readLogicalDrives();
	ListLogicalDrive getLogicalDrives();
	LogicalDriveInfo();
	virtual ~LogicalDriveInfo();
};

#endif
```

Disk_Manager.cpp

```
// Disk_Manager.cpp : Defines the entry point for the console application.
//

#include "Disk_Manager.h"
```

```
#include <Windows.h>
#include <iostream>
#include <map>
using namespace std;
#define  MAX_IDE_DRIVES  16
#define IOCTL_DISK_GET_DRIVE_GEOMETRY_EX CTL_CODE(IOCTL_DISK_BASE, 0x0028,
METHOD_BUFFERED, FILE_ANY_ACCESS)
std::map<int, string> mediaType = {
	{0,"Unknown"},                 // Format is unknown
	{1,"F5_1Pt2_512"},             // 5.25", 1.2MB,  512 bytes/sector
	{2,"F3_1Pt44_512"},            // 3.5",  1.44MB, 512 bytes/sector
	{3,"F3_2Pt88_512"},            // 3.5",  2.88MB, 512 bytes/sector
	{4,"F3_20Pt8_512"},            // 3.5",  20.8MB, 512 bytes/sector
	{5,"F3_720_512"},              // 3.5",  720KB,  512 bytes/sector
	{6,"F5_360_512"},              // 5.25", 360KB,  512 bytes/sector
	{7,"F5_320_512"},              // 5.25", 320KB,  512 bytes/sector
	{8,"F5_320_1024"},             // 5.25", 320KB,  1024 bytes/sector
	{9,"F5_180_512"},              // 5.25", 180KB,  512 bytes/sector
	{10,"F5_160_512"},              // 5.25", 160KB,  512 bytes/sector
	{11,"Removable Media"},         // Removable media other than floppy
	{12,"Fixed Media"},             // Fixed hard disk media
	{13,"F3_120M_512"},            // 3.5", 120M Floppy
	{14,"F3_640_512"},             // 3.5" ,  640KB,  512 bytes/sector
	{15,"F5_640_512"},             // 5.25",  640KB,  512 bytes/sector
	{16,"F5_720_512"},             // 5.25",  720KB,  512 bytes/sector
	{17,"F3_1Pt2_512"},            // 3.5" ,  1.2Mb,  512 bytes/sector
	{18,"F3_1Pt23_1024"},          // 3.5" ,  1.23Mb, 1024 bytes/sector
	{19,"F5_1Pt23_1024"},          // 5.25",  1.23MB, 1024 bytes/sector
	{20,"F3_128Mb_512"},           // 3.5" MO 128Mb   512 bytes/sector
	{21,"F3_230Mb_512"},           // 3.5" MO 230Mb   512 bytes/sector
	{22,"F8_256_128"},             // 8",     256KB,  128 bytes/sector
	{23,"F3_200Mb_512"},           // 3.5",   200M Floppy (HiFD)
	{24,"F3_240M_512"},            // 3.5",   240Mb Floppy (HiFD)
	{25,"F3_32M_512"}              // 3.5",   32Mb Floppy
};
std::map<int, string> description = {
	{ 0, "Unknown" },
	{ 1, "No Root Directory" },
	{ 2, "Removable Disk" },
	{3,"Local Disk"},
	{4,"Network Drive"},
	{5,"Compact Disc"},
	{6,"RAM Disk"}
};


HardDriveInfo::HardDriveInfo(void){
	this->readHardDrives();
}
LogicalDriveInfo::LogicalDriveInfo(void){
	this->readLogicalDrives();
}

int HardDriveInfo::readHardDrives(){
	this->destroyListHardDrives();
	int drive = 0;
	HardDrive* hardDrive;
	for (drive = 0; drive < MAX_IDE_DRIVES; drive++){
		HANDLE hPhysicalDrive = 0;
		char driveName[256];
		sprintf_s(driveName, "\\\\.\\PhysicalDrive%d", drive);
		WCHAR name[256];
```

```
MultiByteToWideChar(0, 0, driveName, 255, name, 256);

hPhysicalDrive = CreateFile((LPCWSTR)name, 0,
       FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
       OPEN_EXISTING, 0, NULL);

if (hPhysicalDrive != INVALID_HANDLE_VALUE){
       STORAGE_PROPERTY_QUERY query;
       DWORD cbBytesReturned = 0;
       char buffer[10000];
       memset((void*)&query, 0, sizeof(query));
       query.PropertyId = StorageDeviceProperty;
       query.QueryType = PropertyStandardQuery;
       memset(buffer, 0, sizeof(buffer));
       if (DeviceIoControl(hPhysicalDrive,
              IOCTL_STORAGE_QUERY_PROPERTY,
              &query,
              sizeof(query),
              &buffer,
              sizeof(buffer),
              &cbBytesReturned, NULL))
       {
              STORAGE_DEVICE_DESCRIPTOR* descrip =
              (STORAGE_DEVICE_DESCRIPTOR*)&buffer;
              char serialNumber[1000];
              char modelNumber[1000];
              getString(buffer, descrip->ProductIdOffset,
              modelNumber);
              getString(buffer, descrip->SerialNumberOffset,
              serialNumber);
              hardDrive = new HardDrive;
              hardDrive->productId = string(modelNumber);
              hardDrive->serialNumber = string(serialNumber);
              hardDrive->bytesPerSector = 0;
              hardDrive->cylinders = 0;
              hardDrive->diskSize = 0;
              hardDrive->tracksPerCylinder = 0;
              hardDrive->sectorsPerTrack = 0;
              hardDrive->driveType = mediaType[0];
              listHardDrive.push_back(hardDrive);

              memset(buffer, 0, sizeof(buffer));
              if (DeviceIoControl(hPhysicalDrive,
                     IOCTL_DISK_GET_DRIVE_GEOMETRY_EX,
                     NULL,
                     0,
                     &buffer,
                     sizeof(buffer),
                     &cbBytesReturned,
                     NULL))
              {
                     DISK_GEOMETRY_EX* geom =
                     (DISK_GEOMETRY_EX*)&buffer;
                     int fixed = (geom->Geometry.MediaType ==
                     FixedMedia);
                     __int64 size = geom->DiskSize.QuadPart;

                     hardDrive->bytesPerSector = geom->
                     Geometry.BytesPerSector;
                     hardDrive->cylinders = geom->
                     Geometry.Cylinders.QuadPart;
                     hardDrive->diskSize = geom->DiskSize.QuadPart;
```

```
                                    hardDrive->tracksPerCylinder = geom->
                                    Geometry.TracksPerCylinder;
                                    hardDrive->sectorsPerTrack = geom->
                                    Geometry.SectorsPerTrack;
                                    hardDrive->driveType = mediaType[geom->
                                    Geometry.MediaType];
                             }
                      }
                      CloseHandle(hPhysicalDrive);
               }
        }
        return drive;
}
int LogicalDriveInfo::readLogicalDrives(){
        this->destroyListLogicalDrives();
        const int lenght = 255;
        char buffer[lenght + 1];
        WCHAR names[256];
        ::GetLogicalDriveStrings(lenght, names);
        //Convert WCHAR* to char*
        for (int i = 0; i < 256; i++) {
               buffer[i] = (char)names[i];
        }
        char* s = buffer;
        while (*s)    {
               LogicalDrive* volumeInfo = new LogicalDrive;
               volumeInfo->pathName = s;
               listLogicalDrive.push_back(volumeInfo);
               s += strlen(s) + 1;
        }

        WCHAR VolumeNameBuffer[256], FileSystemBuffer[256];
        for (unsigned int i = 0; i < listLogicalDrive.size(); i++){
               memset(VolumeNameBuffer, 0, 256 * sizeof(WCHAR));
               memset(FileSystemBuffer, 0, 256 * sizeof(WCHAR));
               unsigned long bytesPerSector = 0;
               unsigned long sectorsPerCluster = 0;
               unsigned long freeClusters = 0;
               unsigned long allClusters = 0;
               unsigned long serialNumber = 0;
               unsigned long maximumComponentLength = 0;
               unsigned long fileSystemFlags = 0;
               int driverType;

               WCHAR rootPath[5];
               MultiByteToWideChar(0, 0, listLogicalDrive[i]->pathName.data(), 5,
rootPath, 5);

               driverType = ::GetDriveType((LPCWSTR)rootPath);
               ::GetDiskFreeSpace(
                      (LPCWSTR)rootPath,
                      &sectorsPerCluster,
                      &bytesPerSector,
                      &freeClusters,
                      &allClusters
               );
               ::GetVolumeInformation(
                      (LPCWSTR)rootPath,
                      (LPWSTR)VolumeNameBuffer,
                      256,
                      &serialNumber,
                      &maximumComponentLength,
```

```
                        &fileSystemFlags,
                        (LPWSTR)FileSystemBuffer,
                        256
                );
                //Convert WCHAR * to char *
                char volumName[256], fileSystem[256];
                for (int i = 0; i < 256; i++) {
                        volumName[i] = (char)VolumeNameBuffer[i];
                        fileSystem[i] = (char)FileSystemBuffer[i];
                }
                listLogicalDrive[i]->type = description[driverType];
                listLogicalDrive[i]->freeClusters = freeClusters;
                listLogicalDrive[i]->bytesPerSector = bytesPerSector;
                listLogicalDrive[i]->sectorsPerCluster = sectorsPerCluster;
                listLogicalDrive[i]->allClusters = allClusters;
                listLogicalDrive[i]->volumeName = string(volumName);
                listLogicalDrive[i]->fileSystemName = string(fileSystem);
                listLogicalDrive[i]->size = bytesPerSector * sectorsPerCluster * (long
long)allClusters;
                listLogicalDrive[i]->sizeFree = bytesPerSector * sectorsPerCluster *
(long long)freeClusters;
                listLogicalDrive[i]->sizeUsed = listLogicalDrive[i]->size -
listLogicalDrive[i]->sizeFree;
        }
        return this->listLogicalDrive.size();
}


char* HardDriveInfo::getString(const char* str, int pos, char* buf){
        buf[0] = 0;
        if (pos <= 0)
                return buf;

        int i = pos;
        int j = 1;
        int k = 0;
        while (j && str[i] != 0){
                char c = str[i];
                if (!isprint(c)){
                        j = 0;
                        break;
                }
                buf[k] = c;
                ++k;
                ++i;
        }

        if (!j)
                k = 0;

        buf[k] = 0;

        while (isspace(buf[0]) && buf[0] != 0)    {
                i = 0;
                while (buf[i] != 0)  {
                        buf[i] = buf[i + 1];
                        i++;
                }
        }
        if (strlen((buf)) > 1)
                while (isspace(buf[strlen(buf) - 1])){
                        buf[strlen(buf) - 1] = 0;
```

```
			}
		return buf;
}

void LogicalDriveInfo::destroyListLogicalDrives(){
		while (!listLogicalDrive.empty()) {
			delete listLogicalDrive.back();
			listLogicalDrive.pop_back();
		}
}

void HardDriveInfo::destroyListHardDrives(){
		while (!listHardDrive.empty()){
			delete listHardDrive.back();
			listHardDrive.pop_back();
		}
}


HardDriveInfo::~HardDriveInfo(){
		this->destroyListHardDrives();
}
LogicalDriveInfo::~LogicalDriveInfo(){
		this->destroyListLogicalDrives();
}

ListHardDrive HardDriveInfo::getHardDrives(){
		return this->listHardDrive;
}

ListLogicalDrive LogicalDriveInfo::getLogicalDrives(){
		return this->listLogicalDrive;
}
```

Lập trình mạng:

account_entity.py

```
class Account():
    def __init__(self, User,Password):
        self.User=User
        self.Password=Password
        self.Current=0
        self.Add=0
        self.Content=""
        self.Translate=""
        self.Login=True
        self.Mess=""
    #get
    def get_User(self):
        return self.User
    def get_Password(self):
        return self.Password
    def get_Current(self):
        return self.Current
    def get_Add(self):
```

```
        return self.Add
    def get_Content(self):
        return self.Content
    def get_Translate(self):
        return self.Translate
    def get_Login(self):
        return self.Login
    def get_Mess(self):
        return self.Mess
    #set
    def set_Password(self,password):
        self.Password=password
    def set_Current(self,current):
        self.Current=current
    def set_Add(self,add):
        self.Add=add
    def set_Content(self,content):
        self.Content=content
    def set_Translate(self,trans):
        self.Translate=trans
    def set_Login(self):
        self.Login=False
    def set_Mess(self,mess):
        self.Mess=mess
```

IS_server.py

```
import socket
import pickle
import threading
import time
import datetime
import os
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import wikipedia as wiki
from googletrans import Translator

import IS_mysql as sql
from IS_TEMP_MSG import *
from account_entity import Account

class CountDown:
  def __init__(self):
      self._running = True
  def terminate(self):
      self._running = False
```

```
  def run(self,userName):
      while self._running:
          currM=sql.viewMoney(userName)
          if currM>=2 and self._running:
              sql.subtractMoney(userName,round(SERVICE_VALUE/3.6))
              currM=sql.viewMoney(userName)
          else:
              break
          time.sleep(1)

class Search(threading.Thread):
  def __init__(self,sc,acc,receive):
      super().__init__()
      self.sc=sc
      self.acc=acc
      self.receive=receive
  def run(self):
      try:
          data= wiki.summary(self.receive,sentences=10)
      except wiki.PageError as e:
          data="Không tìm thấy '"+self.receive+"' !"
      self.acc.set_Content(data)
      print("dang search")
      self.acc.set_Current(sql.viewMoney(self.acc.get_User() ) )
      self.sc.sendall(pickle.dumps(self.acc) )
      time.sleep(0.5)
      self.acc.set_Content("")

class Translate(threading.Thread):
  def __init__(self,sc,acc,receive):
      super().__init__()
      self.sc=sc
      self.acc=acc
      self.receive=receive
  def run(self):
      print("dang search")
      translator = Translator()
      data = translator.translate(self.receive,dest='vi').text
      self.acc.set_Translate(data)
      self.acc.set_Current(sql.viewMoney(self.acc.get_User() ) )
      self.sc.sendall(pickle.dumps(self.acc) )
      time.sleep(0.5)
      self.acc.set_Translate("")

class Server(threading.Thread):
    def __init__(self, host, port):
        super().__init__()
        self.host = host
```

```
        self.port = port
        global accounts

    def run(self):
        print("[STARTING] Server is starting...")
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server.bind((self.host, self.port))
        server.listen(1)
        print("[LISTENING] Server is listening on {}".format(server.g
etsockname()))
        while True:
            sc, sockname = server.accept()
            server_socket = ServerSocket(sc, sockname, self)
            server_socket.start()
            accounts["{} {}".format(sockname[0],sockname[1])]=Account
("","")
            print('Ready to receive messages from', sc.getpeername())

class ServerSocket(threading.Thread):
  def __init__(self, sc, sockname, server):
    super().__init__()
    self.sc = sc
    self.sockname = sockname
    self.server = server
  def run(self):
    global accounts
    while True:#may client van con dang hoat dong
      print("reset")
      while True:#dang nhap
        try:
          recv=pickle.loads(self.sc.recv(SIZE))
          userName=recv[0]
          userPass=recv[1]
          if sql.existAcc(str(userName),str(userPass)):
            notYet=True
            for _ in accounts.values():
              if (userName == _.get_User()) and (userPass == _.get_Pa
ssword() ):
                self.sc.sendall(pickle.dumps(LOGIN_ALREADY))
                notYet=False
                print("already")
                break
            if notYet:
              self.sc.sendall(pickle.dumps(LOGIN_SUCCESS))
              begin = datetime.datetime.now()
              acc=Account(userName,userPass)
              acc.set_Current( int(sql.viewMoney(userName) ) )
```

```
                accounts["{} {}".format(self.sockname[0],self.sockname[
1])]=acc
                self.sc.sendall(pickle.dumps([SERVICE_VALUE,acc.get_Cur
rent(),begin.strftime("%d/%m/%Y, %H:%M:%S")]) )
                print(f"[ONLINE] {len(accounts)}")
                break# login = True
            else:
              self.sc.sendall(pickle.dumps(LOGIN_FAIL))
              print("fail")
          except socket.error:
            del accounts["{} {}".format(self.sockname[0],self.sockname[
1])]
            print(f"[DISCONNECTED] [{self.sockname}]")
            print(f"[ONLINE] {len(accounts)}")
            self.sc.close()
            return
      # print("sleep")
      time.sleep(0.1)
      cd = CountDown()
      t = threading.Thread(target = cd.run, args =(userName, ))
      t.start()
      while True:
        try:
          receive = pickle.loads(self.sc.recv(SIZE))
          acc.set_Current(sql.viewMoney(userName))
          if receive[0] == DISCONNECT_MSG:#dang xuat
            cd.terminate()
            acc.set_Login()
            self.sc.sendall(pickle.dumps(acc))
            accounts["{} {}".format(self.sockname[0],self.sockname[1])]
=Account("","")
            print(f"[DISCONNECTED] [{self.sockname}:'{userName}']")
            print(f"[ONLINE] {len(accounts)}")
            break
          elif receive[0] == PASSWORD_MSG:
            newPass = receive[1]
            sql.changePass(userName,newPass)
            acc.set_Password(newPass)
          elif receive[0] == SEARCH_MSG:
            search = Search(self.sc,acc,receive[1])
            search.start()
          elif receive[0] == TRANSLATE_MSG:
            translate=Translate(self.sc,acc,receive[1])
            translate.start()
          accounts["{} {}".format(self.sockname[0],self.sockname[1])]=a
cc
          self.sc.sendall(pickle.dumps(acc))
        except socket.error:
```

```
            cd.terminate()
            del accounts["{} {}".format(self.sockname[0],self.sockname[1]
)]
            print(f"[DISCONNECTED] [{self.sockname}:'{userName}']")
            print(f"[ONLINE] {len(accounts)}")
            self.sc.close()
            return
accounts={}
if __name__ == "__main__":
    server = Server(IP, PORT)
    server.start()
    # root=tk.Tk()
    # app=Home()#them phan giao dien
```

IS_client.py

```
import socket
import tkinter as tk
import pickle
import time
import os
import threading
from tkinter import messagebox
from IS_TEMP_MSG import *
passApp=False
def on_closing():
    os._exit(0)

class Login(tk.Tk):#dang nhap
  def __init__(self, client,noError):
    super().__init__()
    self.client=client
    self.draw()
    self.protocol ("WM_DELETE_WINDOW", on_closing)
    if not noError:
        messagebox.showinfo("Thông báo", "Mất kết nối server!")
    self.mainloop()
  def draw(self):#ve giao dien

  def login(self):
    if self.text_input_user.get()=="" or self.text_input_pass.get()==
"":
      self.loginError.config(text="Vui lòng nhập đầy đủ thông tin")
      return
    user=self.text_input_user.get()
    password=self.text_input_pass.get()
    try:
      self.client.sendall(pickle.dumps([user,password]))
```

```
      trial=pickle.loads(self.client.recv(SIZE))
      print(trial)
      if trial==LOGIN_SUCCESS:#dang nhap thanh cong
        self.text_input_user.delete(0, tk.END)
        self.text_input_pass.delete(0, tk.END)
        svValue,Money,begin = pickle.loads(self.client.recv(SIZE))
        print(begin,trial)
        self.destroy()
        app=Home(self.client,user,password,begin,Money,svValue)
        return
      elif trial==LOGIN_ALREADY:
        self.loginError.config(text="Tài khoản đang được sử dụng")
        return
      else:
        self.loginError.config(text="Sai tài khoản hoặc mật khẩu")
        return
    except socket.error:
      messagebox.showinfo("Thông báo", "Mất kết nối server!")

class Send: #chuc nang gui cua yeu cau
  def __init__(self):
    self._running = True
  def terminate(self):
    self._running = False
  def run(self,client):
    while self._running:
      try:
        client.send(pickle.dumps([GET_MSG]))
        time.sleep(1)
      except socket.error:
        break
def MtoT(a,svVaalue):# doi tien sang thoi gian
  hour=0
  minute=0
  sec=0
  a=int(a*3.6/svVaalue)
  if a>=3600:
    hour=a//3600
  if a-3600*hour>=60:
    minute=(a-(hour*3600))//60
  sec=a-3600*hour-60*minute
  return (hour,minute,sec)

class Home(tk.Tk):#giao dien chinh
    def __init__(self,client,user,password,begTime,money,value):
      super().__init__()
      self.client=client
      self.user = user
```

```
        self.pw = password
        self.begTime=begTime
        self.money=int(money)
        self.value=float(value)
        self.add=0
        self.keyword=None
        self.draw()
        self.protocol ("WM_DELETE_WINDOW", self.iconify)
        self.mainloop()
    def draw(self):
        #ve giao dien chinh
        self.send = Send()
        t2 = threading.Thread(target = self.send.run, args =(self.clien
t, ))
        t2.start()

        self.alive=True
        t3=threading.Thread(target = self.updateTime)
        t3.setDaemon(True)
        t3.start()
    def searching(self):
        if self.search.get()=="":
            return
        self.keyword=self.search.get()
        try:
            self.client.send(pickle.dumps([SEARCH_MSG,self.keyword]))
        except socket.error:
            pass
        time.sleep(0.5)

    def translating(self):
        if self.translate.get()=="":
            return
        try:
            self.client.send(pickle.dumps([TRANSLATE_MSG,self.translate.g
et()]))
            time.sleep(.5)
        except socket.error:
            pass
    def updateTime(self):
        while self.alive:
            try:
                account = pickle.loads(self.client.recv(SIZE*12))
                #nhan du lieu tu server va cap nhat giao dien
            except socket.error:
                self.send.terminate()
                self.passBtn['state']='disable'
                self.close_window()
```

```
                break
    def close_window(self):
      self.send.terminate()
      try:
        self.client.send(pickle.dumps([DISCONNECT_MSG]))
        self.destroy()
        money_use=int(self.money)+int(self.add)-self.currMoney
        Logout(self.client,money_use)
      except socket.error:
        messagebox.showinfo("Thông báo", "Mất kết nối server!")
        self.withdraw()
        money_use=int(self.money)+int(self.add)-self.currMoney
        Logout(self.client,money_use)

class Logout(tk.Tk):#giao dien dang xuat
  def __init__(self,client,used):
    super().__init__()
    self.client=client
    self.used = used
    self.draw()
    self.protocol ("WM_DELETE_WINDOW", on_closing)
    self.mainloop()
  def draw(self):
    self.title("Logout")
    self.label_welcome=tk.Label(self,text="Vui lòng thanh toán {}vnd!
".format(self.used),font=(18))
    self.label_welcome.grid(row=0,columnspan=2,sticky="ew")
    self.quitButton = tk.Button(self, text = 'Quit', width = 25, comm
and = self.close_windows)
    self.quitButton.grid(row=1,columnspan=2, pady=10,padx=100, sticky
="ew")
  def close_windows(self):
    self.destroy()
    app=Login(self.client,True)
    return

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client.connect(ADDR)
        print(f"[CONNECTED] Server ip: {IP}; Port:{PORT}")
        app=Login(client,True)
    except socket.error:
        app=Login(client,False)
        pass
if __name__ == '__main__':
    main()
```