ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

MẠNG MÁY TÍNH (TN) (CO3094)

Bài tập lớn

Simple File-Sharing Application

Nhóm 1

Giảng viên hướng dẫn: Lê Bảo Khánh
Sinh viên: Huỳnh Nguyên Phúc - 2110451
Cao Minh Quân - 2112109
Trần Nguyễn Thái Bình - 2110051
Trương Hoàng Nguyên Vũ - 2112673

Email: phuc.huynhdaihocbk94@hcmut.edu.vn

Thành phố Hồ Chí Minh, tháng 10/2023

Mục lục

1 Danh sách thành viên & Phân công công việc

STT	Họ và tên	MSSV	Công việc	Phần trăm công việc
1	Huỳnh Nguyên Phúc	2110451	-	25%
2	Cao Minh Quân	2112109	-	25%
3	Trần Nguyễn Thái Bình	2110051	-	25%
4	Trương Hoàng Nguyên Vũ	2112673	-	25%

2 Bối cảnh

Trao đổi, chia sẻ tài liệu là một nhu cầu lớn và quan trọng trong thời đại công nghệ phát triển. Trong bối cảnh đất nước thực hiện chuyển đổi số, việc xây dựng các ứng dụng có khả năng tìm kiếm, trao đổi tài liệu một cách nhanh chóng, an toàn, tiện lợi sẽ giúp quá trình làm việc hiệu quả hơn. Một ứng dụng có khả năng tìm kiếm và chia sẻ file là rất đáng được đề cao và cần thiết.

Trong một tổ chức hoặc công ty nhiều phòng ban, một nhân viên A có thể yêu cầu cần một file về doanh thu để làm phân tích báo cáo trong cuộc họp, nhưng file này lại ở một máy nhân viên khác ở vị trí khá xa. Việc liên lạc sẽ tốn thời gian đáng kể để có thể có được file. Vì vậy, trong dự án này, nhóm sẽ xây dựng một ứng dụng File Sharing theo mô hình peer-to-peer phục vụ cho nhu cầu chia sẻ file cho các máy trong cùng một mạng Internet (mô hình các máy trong một công ty thường có xu hướng dùng chung một mạng).

3 Cơ sở lý thuyết

3.1 Kiến trúc peer-to-peer (P2P Architecture)

Trong kiến trúc P2P, ứng dụng khai thác khả năng giao tiếp trực tiếp giữa các cặp người dùng (host) không liên tục, được gọi là peers. Nhà cung cấp dịch vụ (Service Provider) không sở hữu các peers, mà thay vào đó là các thiết bị đầu cuối điều khiển bởi người dùng. Kiến trúc này rất ít (hoặc không) phụ thuộc vào một máy chủ chuyên dụng (dedicated server) trong data center.

Một trong những đặc điểm nổi bật của kiến trúc P2P là khả năng tự mở rộng của nó (self-scalability). Hơn nữa, kiến trúc P2P cũng có hiệu quả về chi phí nhờ tính không yêu cầu một kiến trúc hạ tầng đáng kể và băng thông cho server (trái ngược với mô hình Client-Server). Mặc dù vậy, các ứng dụng P2P cũng có những nhược điểm như các thách thức về độ bảo mật, hiệu suất và độ tin cậy do cấu trúc phân tán cao của kiến trúc.

3.2 Các giao thức được sử dụng

3.2.1 Các giao thức đã có

- TCP/IP (Transmission Control Protocol/Internet Protocol): là giao thức điều khiển để truyền nhận liên mạng. Đây là bộ giao thức có chức năng truyền và kết nối các thông tin giữa các thiết bị trong một mạng lưới Internet, trong đó:
  - TCP: Hoạt động ở tầng vận chuyển (Transport Layer) trong mô hình kiến trúc năm tầng hay mô hình OSI. Có chức năng lấy các gói tin từ các process và gửi cho tầng Network, cũng như nhận các gói tin từ tầng này và truyền về cho process tương ứng. TCP sử dụng port để quản lý các ứng dụng cùng truy cập đến TCP. Mỗi process khi muốn sử dụng TCP để giao tiếp với mạng đều phải được gắn với một port, và không có 2 process nào trên cùng một máy được sử dụng chung 1 port. TCP cũng có chức năng quản lý các thông tin khi được chia nhỏ để truyền tải qua Internet. Giao thức này sẽ tập hợp các thông tin này theo đúng thứ tự, đảm bảo truyền tải thông tin chính xác tới địa chỉ đến (reliable data transfer). Ngoài ra, TCP còn có khả năng kiểm tra lỗi bằng checksum và kiểm soát nghẽn.
  - IP: Hoạt động ở tầng mạng (Network Layer) trong mô hình kiến trúc năm tầng hay mô hình OSI. Có chức năng vận chuyển các gói tin trong mạng Internet. Mỗi máy sẽ

được gán một địa chỉ, gọi là địa chỉ IP. Địa chỉ này được sử dụng để truyền và định tuyến từng gói thông tin bên trong mạng Internet, qua các router, và đến được máy cuối cùng. Các gói tin từ tầng Transport của một máy được truyền đến tầng Network và được IP đưa đến tầng Transport của máy ở đầu bên kia.

- FTP (File Transfer Protocol): Đây là một giao thức truyền tải tập tin từ một máy tính đến máy tính khác. Giao thức này hoạt động ở tầng ứng dụng (Application Layer) trong mô hình kiến trúc năm tầng hay mô hình OSI. FTP sử dụng TCP làm giao thức ở tầng vận chuyển thay vì UDP bởi khả năng truyền dữ liệu đáng tin cậy của nó. TCP sẽ đảm bảo file được truyền qua mạng một cách đầy đủ và toàn vẹn. FTP là một giao thức hoạt động theo kiểu Server - Client. Máy cần tải file sẽ đóng vai trò là một Client, trong khi máy gửi file sẽ là một Server.

  FTP hoạt động bằng cách tạo 2 đường kết nối giữa server và client, một đường điều khiển (control connection) gắn với port 21 và một đường dữ liệu (data connection) gắn vào port 20. Các câu lệnh mà client muốn thực hiện sẽ được truyền đến đường kết nối ở port 21, còn dữ liệu về file mà client muốn gửi/nhận sẽ đi qua port 20.

3.2.2 Giao thức tự định nghĩa

Để giao tiếp giữa centralized server và các client một cách hiệu quả, cũng như cung cấp thêm khả năng giao tiếp giữa các clients (các peers), nhóm đã xây dựng một giao thức truyền thông tin đơn giản, thuận tiện và dễ sử dụng. Giao thức này có tên là Brilliantly Verified and Quality Message Protocol (BVQMP). Đây là một giao thức ở tầng ứng dụng và sử dụng TCP/IP ở tầng bên dưới. Hai chương trình khác nhau sẽ giao tiếp với nhau bằng cách trao đổi các BVQMP messages. Sau đây là cấu trúc của một message:

Header Field	**Mục đích sử dụng**
HEADER	Xác định các tin nhắn phục vụ cho mục đích khác nhau
TYPE	Loại tin nhắn
INFO	Là phần dữ liệu (payload) của gói tin, phần này sẽ chứa các dữ liệu cần thiết ứng với mỗi loại HEADER và TYPE

Bảng 1: Cấu trúc của một BVQMP message

Các loại HEADER được định nghĩa như sau:

HEADER	Mục đích sử dụng
PING	Sử dụng cho lệnh `ping` của Server và cho các phản hồi `ping` của Client
DISCOVER	Sử dụng cho lệnh `discover` của Server và cho các phản hồi `discover` của Client
PUBLISH	Sử dụng cho lệnh `publish` của Client và cho các phản hồi `publish` của Server
FETCH	Sử dụng cho lệnh `fetch` của Client nhằm yêu cầu một danh sách các peer chứa file và cho các phản hồi `fetch` của Server
REGISTER	Sử dụng để Client đăng ký thông tin cho Server
LOG_IN	Sử dụng để Client đăng nhập vào mạng trao đổi file
LOG_OUT	Sử dụng để Client đăng xuất khỏi mạng
RETRIEVE	Sử dụng sau khi Client lấy danh sách các peer từ lệnh `fetch`. Sau khi có được danh sách này, Client sẽ chọn một peer bằng command line, sau đó giao tiếp trước với peer đó bằng RETRIEVE message trước khi sử dụng FTP để tải file

Bảng 2: Các loại HEADER của BVQMP Message

Các loại TYPE được định nghĩa như sau:

TYPE	Mục đích sử dụng
REQUEST	Message được gửi bởi bên yêu cầu
RESPONSE	Message phản hồi cho các REQUEST message

Bảng 3: Các loại TYPE của BVQMP Message

4 Phân tích yêu cầu

4.1 Mô tả hệ thống

4.1.1 Yêu cầu chức năng

- **Đăng ký và đăng nhập tài khoản người dùng:** Người dùng sẽ đăng ký tài khoản cá nhân với thông tin *username* và *password* cho Server và sẽ được Server lưu lại. Từ đó, ở các lần sau người dùng sử dụng *username* và *password* để đăng nhập vào hệ thống.

- **Lưu thông tin về client và danh sách file**: Máy chủ (server) phải lưu thông tin về client và danh sách các file được client thông báo qua lệnh `publish` trên hệ thống.
- **Thông báo về file vừa được thêm vào repository:** Client có thể thực hiện lệnh `publish lname fname` để thông báo cho máy chủ về tệp địa phương
- **Lấy một file cần thiết lưu về local repository:** Client có thể thực hiện lệnh fetch fname để yêu cầu máy chủ gửi về các clients đang giữ file đó và lấy được file đó về thư mục nội bộ.
- **Theo dõi danh sách file của một client**: Server có thể thực hiện lệnh `discover hostname` để khám phá các local file nằm trong resporitory của một client có tên *hostname*.
- **Kiểm tra client có online**: Máy chủ có thể thực hiện lệnh ping hostname để kiểm tra client có đang alive hay không.

4.1.2 Yêu cầu phi chức năng

- Đảm bảo tính bảo mật thông tin tài khoản của cá nhân client thông qua mã hóa và xác thực.
- Trong điều kiện mạng nhanh và tương đối ổn định, file được truyền đi đảm bảo thời gian tối đa là 2s đối với các file dưới 100MB.
- Tối đa 20 Clients có thể cùng tải một file từ thư mục local của một client tại một thời điểm.
- Hệ thống cho phép mỗi Client lưu trữ tối đa 30 File.
- Thời gian phản hồi từ Server ngắn, dưới 0.5s.

4.2 Mô tả các hàm của Class có sử dụng Protocol

4.2.1 Class Server

- `_init_(self, server_port)`
  - **Mô tả hàm:** Đây là constructor cho server. Hàm gán số port server sử dụng vào object để sử dụng cho các kết nối và chức năng.
  - **Giao thức sử dụng:** Không
- `start(self)`
  - **Mô tả hàm:** Sử dụng để tạo TCP Socket `server_socket` nhận kết nối từ client và đưa từng kết nối vào mỗi thread để xử lý.
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** Sau khi được khởi tạo, TCP Socket `server_socket` sử dụng hàm `bind()` để kết nối server với số port được lựa chọn.
- `close(self)`
  - **Mô tả hàm:** Hàm được sử dụng để đóng kết nối server và xóa mọi thông tin về địa chỉ IP tương ứng của các client.

- **Giao thức sử dụng:** Transmission Control Protocol
- **Giao thức hoạt động:** TCP Socket `server_socket` sử dụng hàm `close()` để đóng kết nối.

- `listen(self)`
  - **Mô tả hàm:** Tạo mỗi thread riêng cho mỗi kết nối từ client để xử lý. Từng thread sẽ sử dụng hàm `handle_client` để xử lý request của từng kết nối.
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** TCP Socket từ Server sẽ sử dụng hàm `accept()` để chấp nhận kết nối từ một client và lưu vào một biến `client_socket`. Sau đó, sẽ tạo thread để và hàm `handle_client` để xử lý yêu cầu gửi lên từ kết nối `client_socket`.

- `handle_client(self, client_socket, hostname, address)`
  - **Mô tả hàm:** Đây là hàm để nhận các request từ từng kết nối client_socket của một client. Hàm này được từng thread trong server sử dụng để xử lý request.
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** TCP Client Socket (`client_socket`) sử dụng hàm `recv()` `.decode()` để có thể nhận requested message gửi từ client trong thread.

- `publish(self, client_socket, hostname, message)`
  - **Mô tả hàm:** Hàm sử dụng để xử lý đáp lại `publish` request từ client. Response message sẽ là OK nếu tên file (nằm trong `message`) chưa tồn tại trong repository của client. Ngược lại, response message sẽ là DUPLICATE
  - **Giao thức sử dụng:** Không

- `ping(self, hostname):`
  - **Mô tả hàm:** Hàm được server sử dụng để kiểm tra kết nối máy hostname có đang alive hay không
  - **Giao thức sử dụng:** Internet Control Message Protocol (ICMP)
  - **Giao thức hoạt động:** Server sẽ tìm TCP Client Socket tương ứng của client tên `hostname`. Sau đó, socket sẽ sử dụng hàm `send` để gửi tin nhắn về cho client kiểm tra client có còn online hay không. Nếu server nhận lại tin nhắn phản hồi tự động từ client, thì xác nhận client vẫn đang online.

- `discover(self, hostname)`
  - **Mô tả hàm:** Hàm được sử dụng bởi server để truy xuất danh sách các file đang nằm trong local repository của client tên hostname.
  - **Giao thức sử dụng:** Transmission Control Protocol (TCP).
  - **Giao thức hoạt động:** Khi kết nối với server thông qua TCP, client sẽ gửi cho server một danh sách các file đang có trong repository của client thông qua TCP. Ở lần đầu sử dụng hàm này, server sẽ lưu trữ danh sách này theo `hostname` của client. Từ đó, các lần sau khi gọi hàm này, server sẽ cập nhật lại danh sách này mà không cần khởi tạo thêm danh sách mới.

- `register(self, client_socket, message)`

  - Mô tả hàm: Đây là hàm phản hồi cho request `register` từ client gửi lên. Response Message là OK nếu tên hostname đăng ký là hợp lệ và thông tin tài khoản sẽ được lưu vào một file json để quản lý. Ngược lại, nếu tên hostname đăng ký đã có trong danh sách đã đăng ký thì message sẽ là DUPLICATE
  - **Giao thức sử dụng:** Không

- `login(self, client_socket, address, message)`
  - **Mô tả hàm:** Đây là hàm để phản hồi lại yêu cầu `login` của client và cập nhật mapping giữa hostname và IP Address khi đăng nhập. Nếu thông tin đăng nhập là hợp lệ, thì gửi response message là OK. Response message là PASSWORD mật khẩu đăng nhập là sai (tên hostname đúng). Response message là AUTHENTIC nếu các file trên repo của máy đang đăng nhập với hostname này không trùng với danh sách file của hostname đó được server lưu trên máy.
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** TCP Socket `client_socket` kết nối giữa client và server sẽ sử dụng hàm `send()` để gửi tin nhắn về lại cho client.

- `logout(self, client_socket, hostname)`
  - **Mô tả hàm:** Hàm để response lại yêu cầu `logout` từ client. Hàm sẽ xóa địa chỉ ip tương ứng với hostname ra khỏi dictionary. Response message gửi cho client là OK.
  - **Giao thức sử dụng:** Transmission Control Protocol

- `fetch(self, client_socket, hostname, message)`
  - **Mô tả hàm:** Hàm được sử dụng để phản hồi lại lệnh `Fetch` từ client. Response message trả về danh sách các host đang alive và có file yêu cầu trong repo.
  - **Giao thức sử dụng:** Transmission Control Protocol

- `search(self, fname)`
  - **Mô tả hàm:** Hàm được sử dụng để tìm các hostname sở hữu
  - **Giao thức sử dụng:** Không.

- `check_active(self, hostname)`
  - **Mô tả hàm:** Hàm được sử dụng để kiểm tra một client (định danh qua hostname) có đang alive hay không.
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** Hàm khởi tạo một TCP Socket `client_socket` kết nối với client cần kết nối, sau đó gửi tin nhắn PING thông qua socket đó. Nếu nhận được tin nhắn phản hồi thông qua hàm `recv()` thì trả về TRUE. Ngược lại, trả về FALSE.

- `check_authentic(self, hostname)`
  - **Mô tả hàm:** Hàm được sử dụng để kiểm tra danh sách file trên repo của client có giống với danh sách file trên client lưu trên server hay không.
  - **Giao thức sử dụng:** Transmission Control Protocol

- `send(client_socket, message: Message)`

- **Mô tả hàm:** Hàm được sử dụng để gửi tin nhắn tới một client cụ thể thông qua socket `client_socket`.
- **Giao thức sử dụng:** Transmission Control Protocol.
- **Giao thức hoạt động:** TCP Socket `client_socket` sử dụng hàm `send()` để gửi tin nhắn về cho client.

- `run(self, opcode, hostname)`
  - **Mô tả hàm:** Hàm được sử dụng để chạy các tác vụ (ping, discover,...) của server.
  - **Giao thức sử dụng:** Không.

4.2.2 Class FTPServerSide

Class này được dùng để tạo một FTP Server instance. Class kế thừa class Thread trong module threading để chạy trên một thread khác.

- `__init__(self, host_ip, check_cache)`
  - **Mô tả hàm:** Khởi tạo một số thuộc tính cho FTP server
  - **Giao thức sử dụng:** Không
- `run(self)`
  - **Mô tả hàm:** Khởi chạy FTP server.
  - **Giao thức sử dụng:** File Transfer Protocol
- `stop(self)`
  - **Mô tả hàm:** Đóng kết nối của FTP Server
  - **Giao thức sử dụng:** File Transfer Protocol

4.2.3 Class Client

- `__init__(self, server_host, server_port, client_hostname, client_password)`
  - **Mô tả hàm:** Constructor cho client khởi tạo các biến và socket cần thiết cho class để hiện thực chức năng. Hàm sẽ lưu các thông tin về IP của server `(server_host)`, số port của server `(server_port)`, tên đăng nhập `(client_hostname)` và mật khẩu `(client_password)`, cùng nhiều thuộc tính phục vụ cho Client side.
  - **Giao thức sử dụng:** Transmission Control Protocol (TCP), File Transfer Protocol (FTP).
  - **Giao thức hoạt động:** Hàm sẽ khởi tạo TCP Socket cho Client (`listen_socket`), gắn với IP của máy đó và port 5001 để lắng nghe các tin nhắn từ server và các peer khác. Đồng thời một FTP socket cũng được tạo ở port 21 phục vụ cho việc gửi file.
- `run(self)`
  - **Mô tả hàm:** Hàm được sử dụng để khởi tạo một socket nghe ở port 5001 và một FTP server ở port 21. Socket port 5001 dùng để nhận các gói tin từ server (khi server dùng lệnh ping, discover) hoặc từ các peer khác trong mạng (dùng để liên lạc trước khi gửi file). FTP server port 21 dùng để truyền file cho các máy khác. Hàm này được chạy trong `__init__`, sau khi khởi tạo các thuộc tính cần thiết.

- **Giao thức sử dụng:** Transmission Control Protocol (TCP), File Transfer Protocol (FTP).
- **Giao thức hoạt động:** TCP Socket `listen_socket` sử dụng hàm `bind()` để gắn vào client host và client port (5001). Sau đó, tạo một thread đảm nhận sử dụng hàm `listen` của lớp Client lắng nghe thông điệp server gửi về. Ngoài ra, hàm cũng thiết lập 1 FTP Server instance `(ftp_server)` nghe ở port 21 để nhận các yêu cầu tải file.

- `stop(self)`
  - **Mô tả hàm:** Hàm được sử dụng để đóng port 5001 và 21 (được bật từ hàm `run()`).
  - **Giao thức sử dụng:** Transmission Control Protocol (TCP), File Transfer Protocol (FTP).
  - **Giao thức hoạt động:** Đối tượng `listen_socket` (TCP Socket của client) sử dụng hàm `close()` để đóng kết nối. Đối tượng `ftp_server` sử dụng hàm `stop()` để tắt FTP server và đóng kết nối.

- `register(self)`
  - **Mô tả hàm:** Hàm được sử dụng để đăng ký thông tin cho server với `username` (cũng là `hostname`) và `password` để đăng nhập.
  - **Giao thức sử dụng:** Transmission Control Protocol (TCP).
  - **Giao thức hoạt động:** Khi kết nối với server lần đầu, client gửi cho server thông tin về hostname và địa chỉ IP thông qua TCP. Các thông tin này sẽ được server lưu trữ vào danh sách. Client cũng tạo `username` duy nhất của riêng mình và `password` để đăng nhập lại vào server.

- `log_in(self)`
  - **Mô tả hàm:** Hàm được client sử dụng để đăng nhập vào server với `username` riêng biệt với từng client và `password`.
  - **Giao thức sử dụng:** Transmission Control Protocol (TCP).
  - **Giao thức hoạt động:** Client gửi yêu cầu đăng nhập đến server bằng `username` và `password` và chờ đợi server phản hồi.

- `log_out(self)`
  - **Mô tả hàm:** Hàm được client sử dụng để đăng xuất khỏi server.
  - **Giao thức sử dụng:** Transmission Control Protocol (TCP).
  - **Giao thức hoạt động:** Client gửi yêu cầu đăng xuất lên server.

- `listen(self)`
  - **Mô tả hàm:** Hàm được sử dụng để nhận các thông điệp từ client hoặc server gửi về
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** TCP Socket `listen_socket` của client sử dụng hàm `listen()` để bắt đầu chờ xác nhận kết nối. Sau đó, socket sử dụng hàm `accept()` để chấp nhận kết nối và lưu socket kết nối đó vào `recv_socket`. Socket `recv_socket` sẽ được đưa vào một thread để xử lý.

- `handle_incoming_connection(self, recv_socket, src_addr)`

  - **Mô tả hàm:** Hàm được sử dụng để xử lý tin nhắn nhận được từ socket kết nối với client hoặc server. Nếu header của tin nhắn là PING, tức là lệnh `ping` từ server thì phản hồi lại tin nhắn bằng hàm `reply_ping_mesage()`. Nếu header tin nhắn là `PUBLISH`, sử dụng hàm `respond_publish()` để phản hồi tin nhắn. Nếu header tin nhắn là `RETRIEVE`, đây là một yêu cầu lấy file từ client khác, sử dụng hàm
  - **Giao thức sử dụng:** Transmission Control Protocol.
  - **Giao thức hoạt động:** TCP Socket `recv_socket` kết nối giữa client và bên còn lại (client hoặc server) sẽ sử dụng hàm `recv().decode()` để giải mã tin nhắn và lưu vào biến `message`.

- `send_message(self, message:Message, sock:socket.socket)`
  - **Mô tả hàm:** Hàm được sử dụng để gửi message được mã hóa tới server
  - **Giao thức sử dụng:** Transmission Control Protocol
  - **Giao thức hoạt động:** Nếu tham số truyền vào cho `existed_socket` không có sẵn, thì hàm tạo sẵn một socket `tmp_socket` để kết nối với địa chỉ và số port của host. Sau đó, socket (`tmp_sock` hoặc `existed_socket`) sử dụng hàm `sendall` để gửi tin nhắn cho server.

- `preprocess_file_transfer(self)`
  - **Mô tả hàm:** Hàm được sử dụng để tiền xử lý (preprocess) trước khi gửi file giữa hai client.
  - **Giao thức sử dụng**: Không

- `reply_ping_message(self, message, sock)`
  - **Mô tả hàm:** Hàm được sử sử dụng để phản hồi tin nhắn `ping` từ server.
  - **Giao thức sử dụng:** Không

- `reply_discover_message(self, message, sock)`
  - **Mô tả hàm:** Hàm được sử dụng để phản hồi tin nhắn `discover` từ server.
  - **Giao thức sử dụng:** Không

- `initiate_ftp_server(self)`
  - **Mô tả hàm:** Cấp phát và khởi tạo FTP server để các client khác có thể tải file từ máy này.
  - **Giao thức sử dụng:** File Transfer Protocol (FTP)

- `stop_ftp_server(self)`
  - **Mô tả hàm:** Đóng FTP server để các client khác không tải file nữa. Hàm này được sử dụng khi thoát khỏi chương trình.
  - **Giao thức sử dụng:** File Transfer Protocol (FTP)

- `retrieve(self, fname, host)`
  - **Mô tả hàm:** Khởi tạo kết nối FTP tới host server
  - **Giao thức sử dụng:** File Transfer Protocol (FTP)

- **Giao thức hoạt động:**

- `publish(self, lname, fname)`
  - **Mô tả hàm:** Hàm được sử dụng để yêu cầu muốn publish file và thông báo cho server sử dụng hàm `send_message()`. Hàm trả về TRUE nếu publish thành công, FALSE trong trường hợp ngược lại.
  - **Giao thức sử dụng:** Không
- `__request_server_(self, fname)`
  - **Mô tả hàm:** Hàm được sử dụng để gửi yêu cầu lên server danh sách các host đang sở hữu file tên `fname`
  - **Giao thức sử dụng:** Không
- `fetch(self, name)`
  - **Mô tả hàm:** Hàm được sử dụng để fetch một bản copy của file tên `name` về local repository.
  - **Giao thức sử dụng:** Không
- `_check_cached_(self, fname)`
  - **Mô tả hàm:** Hàm được sử dụng để thêm file `fname` vào bộ nhớ cache của directory
  - **Giao thức sử dụng:** Không

4.3 Mô tả các hàm của Class không sử dụng Protocol

Các hàm dưới đây không sử dụng Protocol nào nên sẽ chỉ mô tả chức năng của hàm

4.3.1 Class Message

- `_init_(self, header, type, info, json_string=None)`: Constructor của Message. Một Message bao gồm `header`, `type`, `info`
- `get_header(self)`: Trả về header của Message.
- `get_type(self)`: Trả về giá trị type của Message.
- `get_info(self)`: Trả về giá trị info của Message.
- `get_packet(self)`: Trả về toàn bộ thông tin của một Message gồm `header`, `type`, và `info`

4.3.2 Class Server_App

- `_init_(self)`: Constructor của `Server_App`.
- `trigger(self, frame)`: Hàm được sử dụng để chuyển hướng trang `frame`.
- `check_login(self, username_entry, password_entry)`: Hàm được sử dụng để kiểm tra thông tin đăng nhập của người dùng được gửi đến có đúng hay không.
- `sign_in(self)`: Hàm hiện thực interface Đăng nhập của ứng dụng.

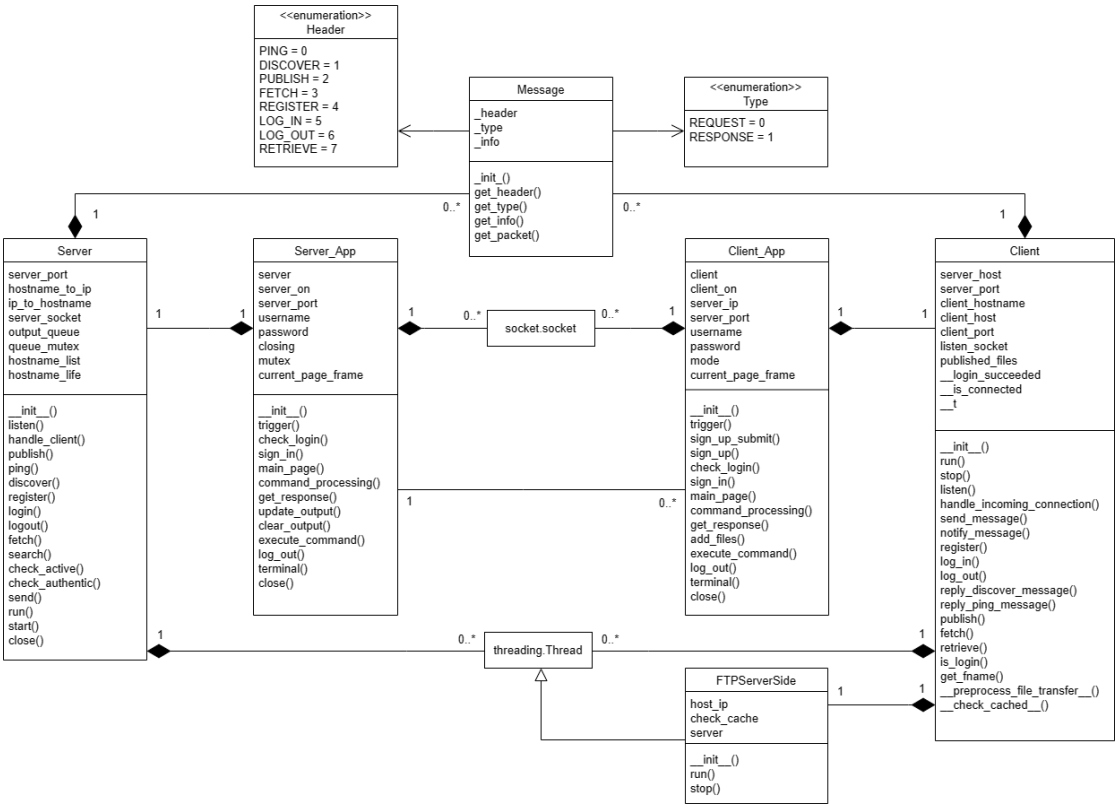
- `main_page(self)`: Hàm hiện thực interface Trang chủ của ứng dụng.
- `command_processing(self, command)`: Hàm kiểm tra lệnh nhập vào cmd có đúng format hay không. Hàm trả về `TRUE` nếu đúng format và `FALSE` nếu ngược lại.
- `get_response(self, command)`: Hàm được sử dụng để giải quyết các request từ client và trả về response message tương ứng cho request đó.
- `update_output(self, server_output)`: Hàm để cập nhật hiện ra trên màn hình hiển thị thông tin khi chạy xong một lệnh nào đó hoặc request nào đó từ client.
- `clear_output(self)`: hàm để xóa output hiển thị thông tin các response message khi chạy request của client.
- `execute_command(self, input_field, output_field)`: Trigger để thực thi lệnh nhập vào bởi người quản lý server và hiển thị kết quả trên terminal.
- `log_out(self)`: Trigger để log out ứng dụng của server.
- `terminal(self)`: Hiện thực interface terminal của ứng dụng cho server.
- `close(self)`: Hàm join các thread và đóng server khi Server đóng ứng dụng.

4.3.3 Class Client_App

- `_init_(self)`: Constructor của Client_App
- `trigger(self, frame)`: Hàm được sử dụng để điều hướng giữa các trang `frame`
- `sign_up_submit(self, username_entry, password_entry)`: Hàm được sử dụng để xử lý thông tin đăng kí của người dùng sau khi người dùng submit và điều hướng trang.
- `sign_up(self)`: Hàm hiện thực interface đăng ký của ứng dụng bên Client.
- `check_login(self, username_entry, password_entry)`: Hàm dùng để xác thực việc đăng nhập có hợp lệ hay không.
- `sign_in(self)`: Hàm hiện thực interface đăng nhập của ứng dụng bên phía Client.
- `main_page(self)`: Hàm hiện thực interface của trang chủ ứng dụng bên phía Client.
- `command_processing(self, command)`: Hàm trả về `TRUE` nếu lệnh nhập vào từ phía Client đúng format. Trả về `FALSE` trong trường hợp ngược lại.
- `get_response(self, command)`: Hàm được sử dụng để nhận response mesage từ server.
- `add_files(self, fname, list_files)`: Hàm được dùng để hiển thị thêm tên file mới được thêm vào repository.
- `execute_command(self, input_field, ouput_field, list_files)`: Trigger để thực thi các lệnh nhập vào của Client và hiển thị kết quả ra terminal.
- `log_out(self)`: Trigger để log out ứng dụng của Client.
- `terminal(self)`: Hàm hiện thực interface màn hình command line bên phía Client.
- `close(self)`: Hàm được sử dụng đóng kết nối khi Client tắt ứng dụng.

4.4 Class Diagram

Class Diagram cho toàn bộ hệ thống:



Hình 1: Class Diagram cho toàn bộ ứng dụng

5 A Summative Evaluation of Results Achieved

5.1 Về lập trình

Thông qua bài tập lớn "P2P File Sharing Application" này, nhóm đã có thể:

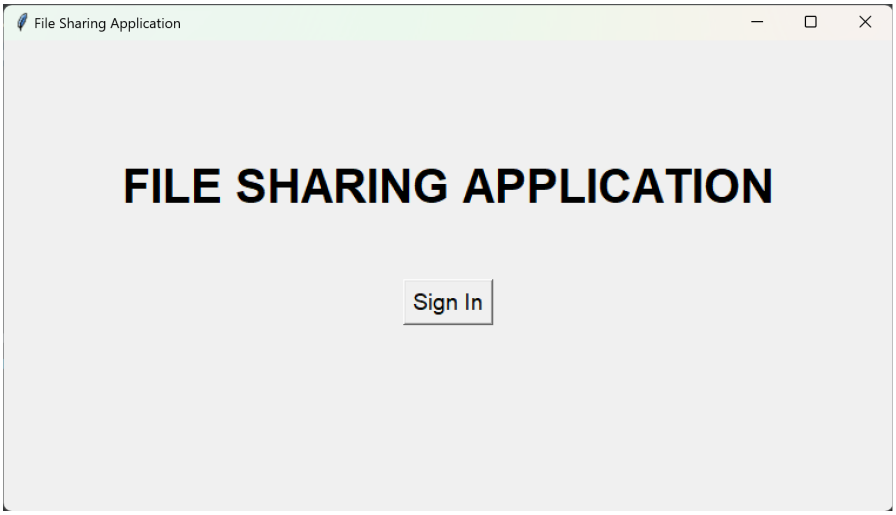
- Thiết kế được GUI cho người dùng (client và cả server) để có thể thực hiện các lệnh cơ bản thông qua thư viện tkinter của Python.
- Thiết kế một giao thức và cấu trúc yêu cầu và phản hồi giữa Client và Server.

5.2 Về kết quả

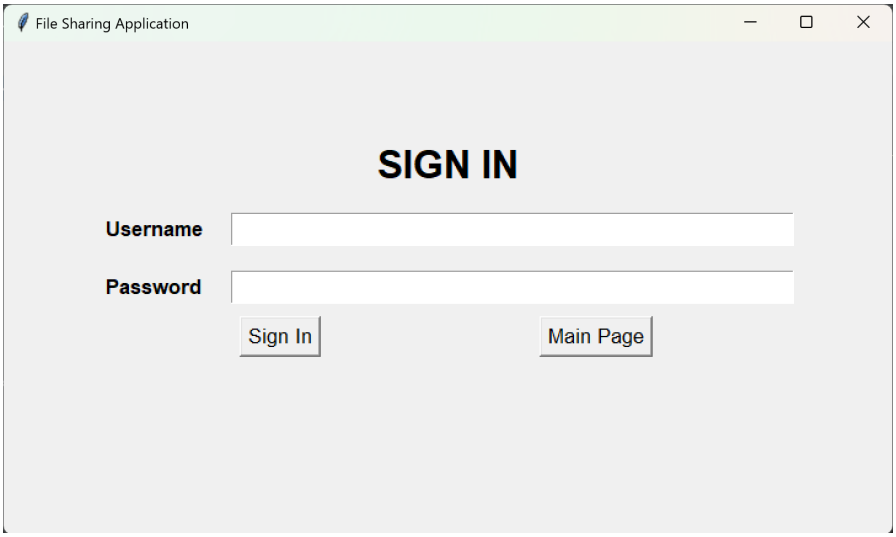
- Khởi động và đăng nhập vào Server. Server nhận được yêu cầu của Client gửi đến, quá trình trích xuất yêu cầu tại Server hoạt động và hiển thị thông tin về các request và trạng thái phản hồi ra như mong muốn.
- Client nhận được file (bất kỳ định dạng) từ một Client khác gửi về khi thực hiện kết nối yêu cầu file.

5.3 User manual

- Bước 1: Khởi động Server hoạt động:
  - Khởi động Server UI : chạy Terminal trong thư mục chứa file `server.py` và `server_ui.py`. Gõ lệnh `python server_ui.py "server_port"`
    (a) Trong đó `"server_port"` là một số bất kỳ lớn hơn 1024.
    (b) Ví dụ: `python server_ui.py 5001`
  - Sau khi khởi động, màn hình bắt đầu úng dụng bên phía Server như hình dưới ta nhấn vào nút Sign In để đăng nhập. Ở giao diện đăng nhập, ta có thể nhấn nút Main Page nếu muốn quay lại Trang chủ. Màn hình đăng nhập ở hình 3.

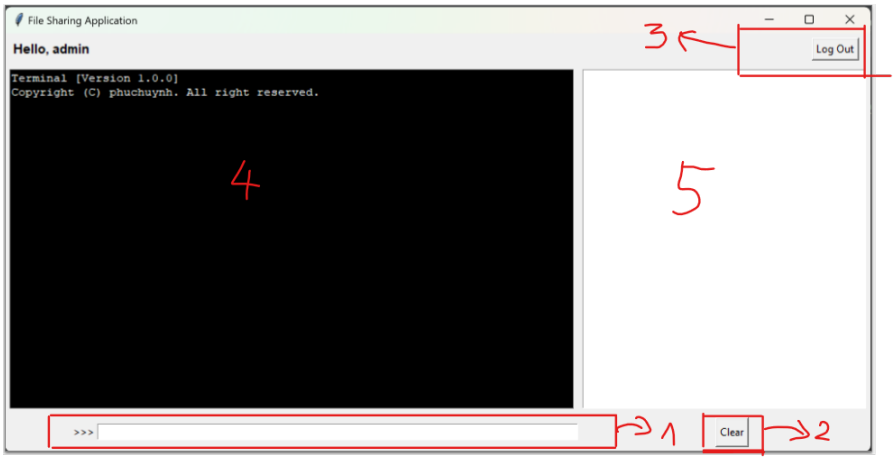


Hình 2: Main Page của Server khi khởi động ứng dụng



Hình 3: Giao diện đăng nhập của Server

- Sau khi đăng nhập và được xác thực hợp lệ, giao diện hoạt động được biểu diễn ở hình 4

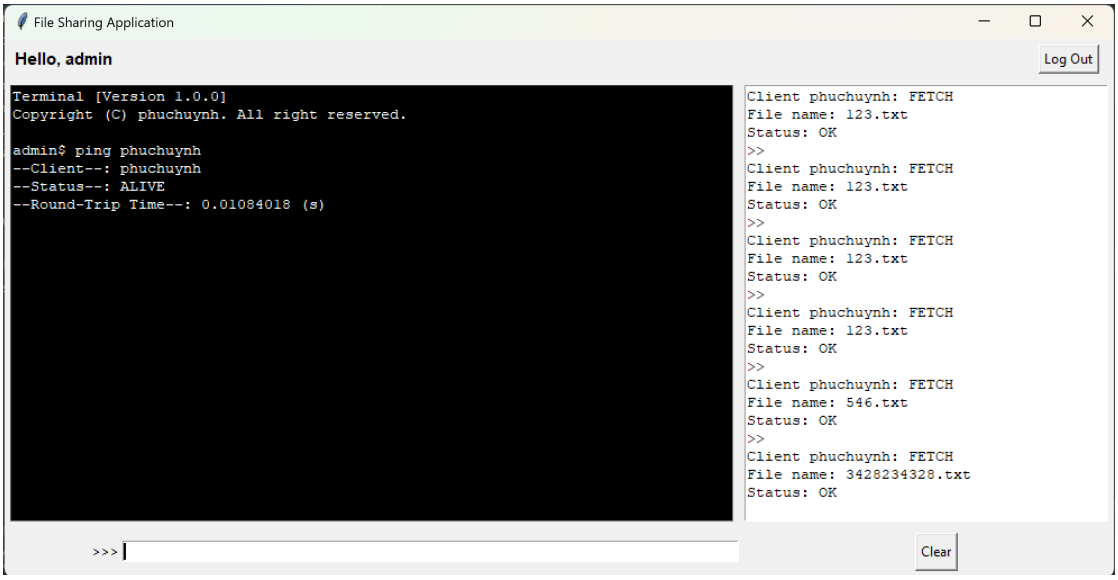


Hình 4: Giao diện hoạt động của Server

Dựa vào chú thích đánh số trong hình 4, giao diện hoạt động bao gồm:

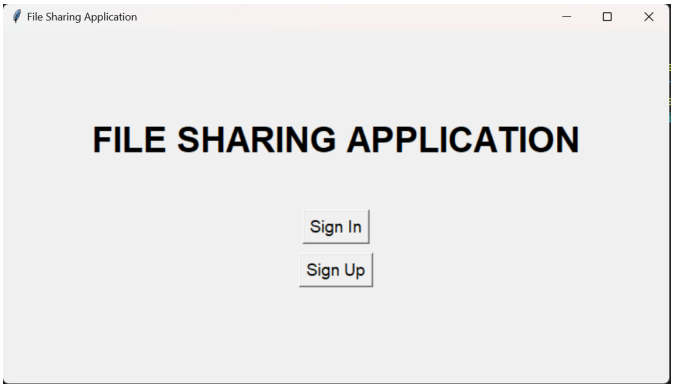
1) Vùng nhập lệnh vào ứng dụng của server.
2) Nút Clear để xóa màn hình hiển thị theo format các lệnh request từ phía client và trạng thái phản hồi (chú thích số 5)
3) Nút Log Out dùng để đăng xuất và tắt Server.
4) Màn hình terminal hiển thị các lệnh nhập vào và phản hồi của server.
5) Màn hình hiển thị theo format các lệnh request từ phía client và trạng thái phản hồi.

Tới đây, Server có thể nhận các request từ client và thực hiện các lệnh cơ bản của server. Hình 5 dưới đây có các output sau khi thực hiện các lệnh và nhận request từ client, giúp hình dung được cách hoạt động.

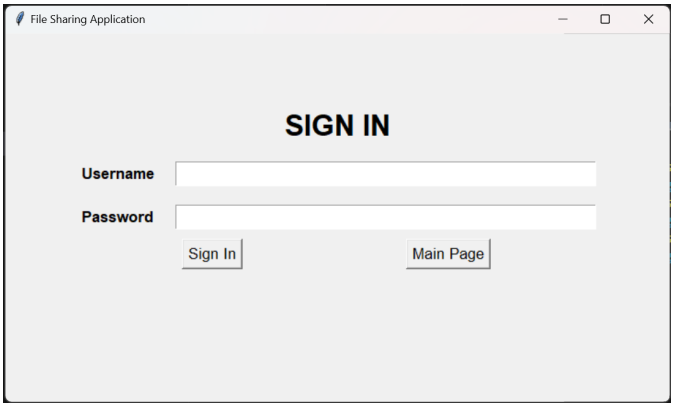


Hình 5: Màn hình output của server sau khi nhận các request và thực hiện các lệnh

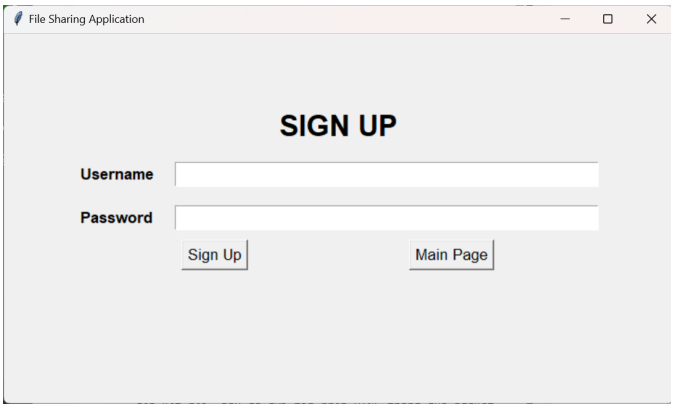
- Bước 2: Khởi động Client UI và hoạt động:
  1. Khởi động Client UI: chạy Terminal trong thư mục chứa file `client.py` và `client_ui.py`. Gõ lệnh `python client_ui.py "client_ip" "server_port"`
     (a) Trong đó `"client_ip"` là địa chỉ IP của client, `"server_port"`.
     (b) Ví dụ: `python client_ui.py 192.168.41.4 5000`
  2. Sau khi khởi động, màn hình bắt đầu úng dụng bên phía Client như hình 6 dưới ta nhấn vào nút Sign In nếu muốn đăng nhập hoặc Sign Up nếu muốn đăng ký. Hình 7 là giao diện đăng nhập và hình 8 là giao diện đăng ký.



Hình 6: Main Page của Server khi khởi động ứng dụng

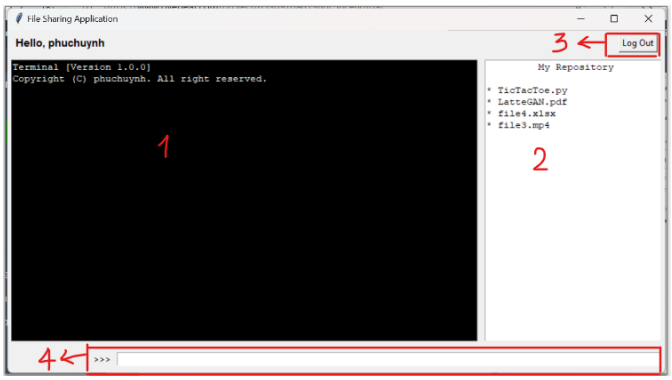


Hình 7: Giao diện đăng nhập bên phía Client



Hình 8: Giao diện đăng ký bên phía Client

3. Sau khi đăng nhập và được xác thực hợp lệ, giao diện hoạt động được biểu diễn như hình 9.

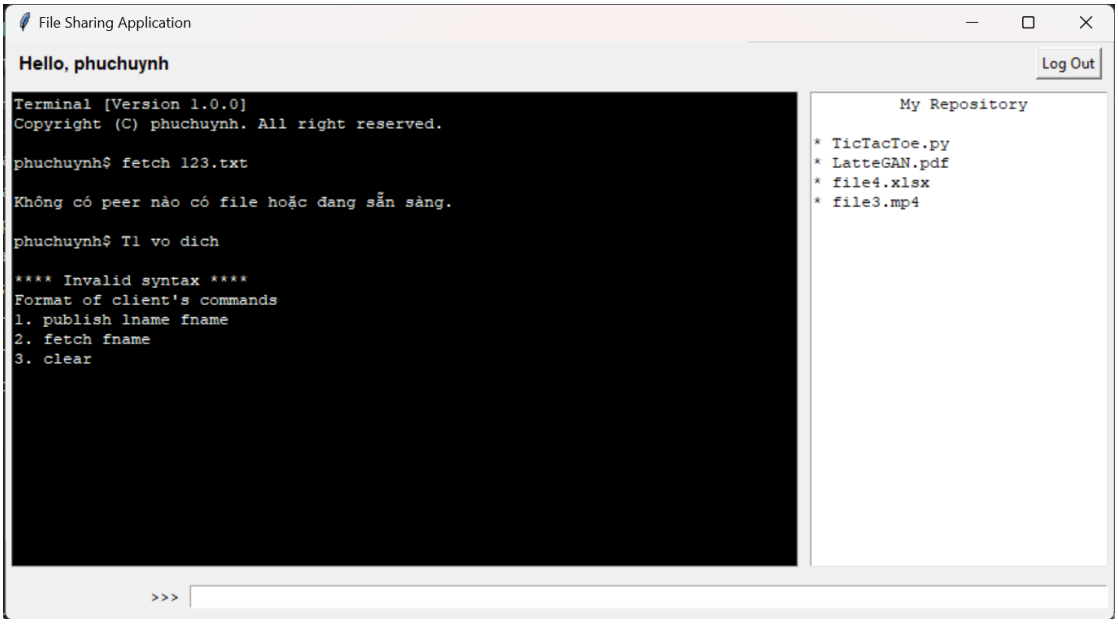


Hình 9: Chú thích giao diện hoạt động của Client

Dựa vào chú thích hình 9, giao diện hoạt động bao gồm:

1) Màn hình terminal hiển thị các lệnh nhập vào và phản hồi.
2) Màn hình hiển thị các file nằm trong repo của client.
3) Nút Log Out để đăng xuất khỏi ứng dụng.
4) Vùng nhập lệnh vào ứng dụng của client.

Tới đây, client có thể nhập các request gửi tới server.



Hình 10: Màn hình output của client sau khi nhập các request gửi lên server.

6 Evaluation of performance

Nhóm đã hiện thực hàm `retrieve(self, fname, host)` để client hiện thực việc kết nối với một peer đã chọn và thực hiện kết nối để download file.

- Nếu giao tiếp được với peers đã chọn, thực hiện kết nối FTP tới host server và lấy file.
- Ngược lại, trả về và chọn một peer khác.

Bên cạnh đó, trong hàm còn có hiện thực phần đếm thời gian kể từ khi thực hiện kết nối đến khi nhận file. Chi tiết phần hiện thực hàm bên dưới cũng như dung lượng file download. Thời gian tính tốc độ truyền file sẽ được tính theo công thức:

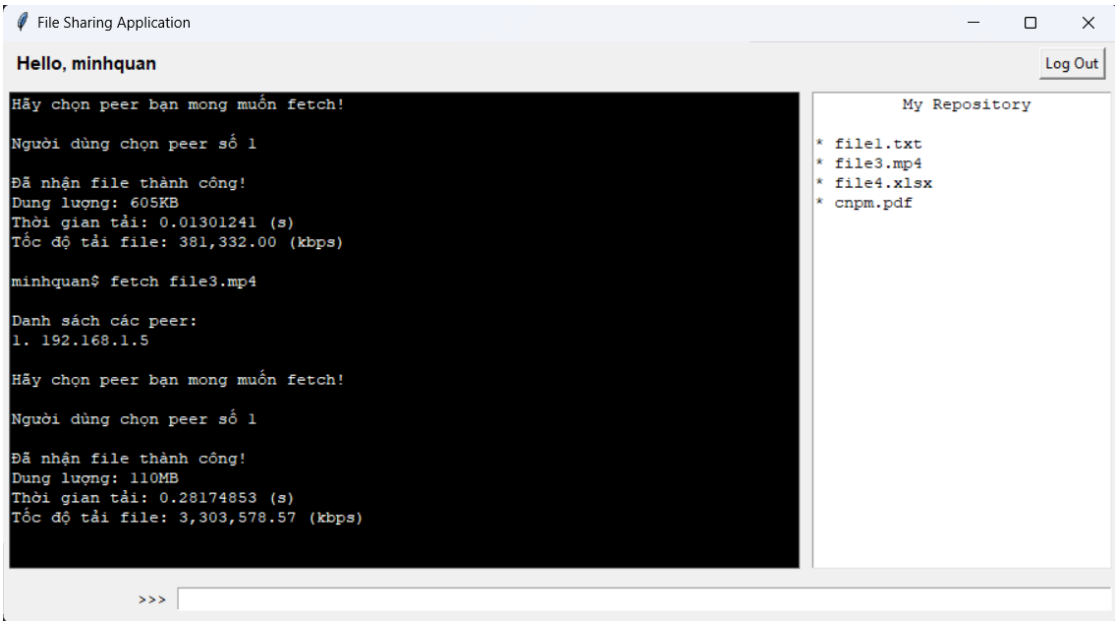
$$\texttt{download speed} = \frac{\texttt{file size}}{\texttt{transmission time}} \tag{1}$$

```
1     def retrieve(self, fname, host):
2
3         # Ask that host to download the file
4         request = Message(Header.RETRIEVE, Type.REQUEST, fname)
```

```
        tmp_sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
        tmp_sock.settimeout(5)
        try:
            tmp_sock.connect((host, self.client_port))
        except:
            return 'UNREACHABLE'

        # If request cannot be sent, return
        if not self.send_message(request, tmp_sock):
            tmp_sock.close()
            return 'UNREACHABLE'

        # Receive the response
        response = tmp_sock.recv(2048).decode()
        tmp_sock.close()

        # Process the response
        response = Message(None, None, None, response)
        result = response.get_info()
        # Check if it accepts or refuses to send the file. If
    DENIED, try other hosts
        if result == 'DENIED':
            return result

        # If it has accepted, proceed file transfering using FTP
        dest_dir, dest_file = "download/", fname
        # Handle non-existing download directory
        if not os.path.exists(dest_dir):
            os.mkdir(dest_dir)

        i = 0
        # Handle existing files
        while os.path.exists(dest_dir + dest_file):
            i += 1
            dest_file = f"Copy_{i}_" + fname

        # File transfer protocol begins
        ftp = FTP(host)
        ftp.login('mmt', 'hk231')

        start_time = time.time()
        with open(dest_dir + dest_file, 'wb') as fp:
            ftp.retrbinary(f'RETR {fname}', fp.write)
        end_time = time.time()

        ftp.quit()
        # File transfer protocol ends

```

```
52          # Publish file
53          self.publish(os.path.abspath(dest_dir + dest_file), fname)
54          retrieve_time = "{:,.8f}".format(end_time - start_time)
55          return ('OK', retrieve_time)
```

Dưới đây là ví dụ cho thời gian download một file:



Hình 11: Output fetch và download file