

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 (CO3093)

Bài tập lớn số 1

FILE-SHARING APPLICATION

Nhóm: 12 - Lớp: L09

GVHD:	Bùi Xuân Giang	
Sinh viên:	Đỗ Minh Đức	2113206
	Nguyễn Hoàng Kim	2111617
	Nguyễn Trương Phước Thọ	2114913
	Nguyễn Nhật Đăng	2110123



Mục lục

1	Specific functions of the file-sharing application	3
1.1	Client	3
1.1.1	Mô tả khái quát các function	3
1.1.2	Mô tả chi tiết các function	3
1.2	Server	6
1.2.1	Mô tả khái quát các function	6
1.2.2	Mô tả chi tiết các function	6
2	The communication protocols	8
2.1	HTTP (Hypertext Transfer Protocol)	8
2.2	Transmission Control Protocol (TCP)	8
2.3	Internet Protocol(IP)	9
3	Class Diagram	11
4	Validation and evaluation of actual result	11
4.1	Connect To The Server	11
4.2	Publish Files	12
4.3	Fetch Files	14



Members and Tasks

Họ tên	MSSV	Công việc	%
Nguyễn Hoàng Kim	2111617	Xây dựng UI, Viết báo cáo assignment 1	100
Đỗ Minh Đức	2113206	Xây dựng backend, Viết báo cáo assignment 1	100
Nguyễn Trương Phước Thọ	2114913	Assignment 2	100
Nguyễn Nhật Đăng	2110123	Assignment 2	100

1 Specific functions of the file-sharing application

1.1 Client

1.1.1 Mô tả khái quát các function

- **Tham gia vào mạng**

Chức năng này cho phép người dùng có thể tham gia vào một mạng để chia sẻ và tải file.

Người dùng cần cung cấp địa chỉ IP, port, hostname,... của thiết bị để server có thể quản lý người dùng.

- **Khai báo files**

Chức năng này cho phép người dùng chọn những file cá nhân mà họ muốn chia sẻ hoặc là hủy khai báo với server.

Người dùng sẽ chọn file có tên lname có trong thiết bị của người dùng và khai báo cho server với tên fname. Một fname có thể được khai báo bởi nhiều người dùng, do đó người nhận cần xác định người gửi khi yêu cầu tải file.

- **Xem files có sẵn**

Chức năng này cho phép người dùng xem danh sách các files của cá nhân đã khai báo và những file được quản lý bởi server.

- **Tìm files**

Chức năng này cho phép người dùng tìm kiếm các file cá nhân đã được khai báo và quản lý bởi server.

- **Tải files**

Chức năng này cho phép người dùng tìm và tải một số bản sao của file cá nhân đã được khai báo ở server và thêm nó vào kho lưu trữ cục bộ.

1.1.2 Mô tả chi tiết các function

- Phương thức `__init__(self, serverIP, serverPort, name, port)`: Phương thức dùng để khởi tạo các Instance variable

- Field `serverIP`: Địa chỉ IP của server mà peer kết nối.
- Field `serverPort`: Port của server mà peer kết nối.
- Field `name`: Tên của peer được khởi tạo.
- Field `port`: Port của peer được khởi tạo.

- Phương thức `setUp(self)`: Phương thức dùng để setup peer

- Kết nối tới server: Phương thức sẽ thử tạo một TCP/IP socket và kết nối với server.
 - + Nếu thất bại, client sẽ kết thúc hệ thống và kết thúc phương thức.

- + Nếu thành công, gán giá trị của biến *endAllThread* thành *False*, thêm vào danh sách *listSocket* kết nối vừa khởi tạo thành công, tạo file JSON *data* gồm các trường *name*, *IP*, *port*, *action* với giá trị là *register* và *listFile*, sau đó gửi tới server.
- Sau đó, tạo một TCP/IP socket khác, gán IP và port.
 - + Nếu thất bại, client sẽ kết thúc hệ thống và kết thúc phương thức.
 - + Nếu thành công, thêm vào *listSocket* connection *peerSocket* và cho phép server chấp nhận kết nối.
- Tạo multi-thread: Tạo một thread để nhận phản hồi từ phía server, một thread để nhận phản hồi từ peer khác. Sau đó, chạy một vòng lặp nếu biến *endAllThread* có giá trị là *False*: Nếu có kết nối mới từ một peer khác, tạo một thread mới để nhận phản hồi.
- Phương thức *startPeer(self)*: Phương thức này tạo một thread với target là phương thức *setUp*, kiểm tra thư mục gốc đã tồn tại thư mục của peer hay chưa, nếu chưa thì khởi tạo thư mục, thêm thread vào danh sách *allThreads*, sau đó khởi động thread.
- Phương thức *listenRequest(self, conn)*: Phương thức này sẽ chạy một vòng lặp khi biến *endAllThread* có giá trị *False*. Khi đó một khối lệnh try except sẽ được thực thi:
 - Giải mã data nhận được từ biến *conn* và đưa dữ liệu từ dạng file JSON thành Python Object.
 - Nếu *jsonData["action"]* mang giá trị là *requestFile* thì gán biến *peerName* giá trị của *jsonData["name"]*, biến *fname* giá trị của *jsonData["fname"]*. Sau đó tạo một thread với target là phương thức *sendFile*, truyền vào phương thức *sendFile* các biến *conn*, *fname*, *peerName*, thêm vào danh sách *allThreads* và khởi động.
 - Nếu *jsonData["action"]* mang giá trị là *ping* thì tạo file JSON *mess* gồm các trường *IP*, *port*, *action*, sau đó gửi tới server.
 - Nếu có lỗi, lệnh *continue* sẽ được thực thi.
- Phương thức *listenResponse(self)*: Phương thức này sẽ chạy một vòng lặp khi biến *endAllThread* có giá trị *False*. Khi đó một khối lệnh try except sẽ được thực thi:
 - Giải mã data nhận được từ biến *connectSocket* và đưa dữ liệu từ dạng file JSON thành Python Object.
 - Nếu *jsonData["action"]* mang giá trị là *responseFile*, nếu *jsonData["status"]* mang giá trị là *enable* thì chạy phương thức *receiveFile* với các biến đầu vào là *jsonData["fname"]*, *jsonData["name"]*, *jsonData["IP"]*, *jsonData["port"]*, ngược lại, tạo file JSON *mess* gồm các trường *name* có giá trị *self.name*, *action* mang giá trị *fetch*, *IP* mang giá trị *self.IP*, *port* mang giá trị *self.port*, *statusRequest* mang giá trị *unsuccessful*, *fname* mang giá trị *jsonData["fname"]*. Sau đó đóng socket và gán giá trị bằng *None*.
 - Nếu có lỗi, lệnh *continue* sẽ được thực thi.

- Phương thức `sendFile(self, conn, fname, peerName)`: Phương thức này sẽ tìm *lname* trong *listFile* thông qua *fname* và tìm đường dẫn lưu trữ *lname* ở local, sau đó nếu tồn tại file ở local thì mã hoá thông tin thành một file JSON gồm các trường *name*, *IP*, *port*, *action* mang giá trị *responseFile*, *status* mang giá trị *enable*, *fname* để gửi đến server, mã hoá file và gửi file đến peer đích, ngược lại, mã hoá thông tin thành một file JSON với trường *status* mang giá trị *disable* để gửi đến server.
- Phương thức `receiveFile(self, fname, peerName, IP, port)`: Phương thức này sẽ kiểm tra tên file đã tồn tại trong local hay chưa, nếu đã tồn tại thì thêm số thứ tự vào tên file, sau đó giải mã file đã được mã hoá trước khi gửi đi từ peer chủ, lưu vào file và mã hoá thông tin thành một file JSON gồm các trường *name*, *IP*, *port*, *action* mang giá trị *responseFile*, *statusRequest* mang giá trị *successful*, *fname*, *connName*, *connIP*, *connPort* để gửi đến server.
- Phương thức `fetch(self, fname, hostname)` và `startConnection(self, IP, port, fname)`: Phương thức `fetch` sẽ tìm *hostname* trong danh sách *listPeerServer* để lấy *IP* và *port*. Nếu không tồn tại *IP* hoặc *port* hay *hostname* trùng với tên của peer, phương thức sẽ báo lỗi và kết thúc. Ngược lại, phương thức sẽ tạo một thread với target là phương thức `startConnection` và khởi động thread. Phương thức `startConnection` sẽ tạo một socket để kết nối với peer sở hữu file mà peer chủ muốn fetch, sau đó mã hoá thông tin thành một file JSON gồm các trường *name*, *action* mang giá trị *requestFile*, *fname* để gửi đến server.
- Phương thức `listenServer(self)`: Phương thức sẽ chạy một vòng lặp với điều kiện biến *endAllThread* mang giá trị *false*. Khi đó một khối lệnh try except sẽ được thực thi:
 - Nhận và giải mã data được gửi đến từ server. Nếu *action* mang giá trị *responseRegister*, *status* mang giá trị *successful* thì gán biến *ID* của peer giá trị *ID* trong file JSON, ngược lại phương thức sẽ kết thúc. Nếu *action* mang giá trị *responseListFile* thì gán *listFileServer* của peer giá trị của *listFile* trong file JSON. Nếu *action* mang giá trị *responseListPeer* thì gán *listPeerServer* của peer giá trị của *listPeer* trong file JSON.
 - Nếu có lỗi, lệnh continue sẽ được thực thi.
- Phương thức `requestListFile(self)`: Phương thức này sẽ mã hoá thông tin thành một file JSON gồm các trường *name*, *IP*, *port*, *action* mang giá trị *requestListFile* để gửi đến server.
- Phương thức `showFiles(self)`: Phương thức này sẽ in ra terminal các file đã được upload lên server được lưu trong *listFile* và các file chưa được upload lên server được lưu ở local.
- Phương thức `publish(self, lname, fname)`: Phương thức này sẽ kiểm tra xem một file đã được upload lên server hay chưa. Nếu đã upload lên server, phương thức sẽ kết thúc. Ngược lại, phương thức sẽ kiểm tra file có tồn tại ở local hay không, nếu có thì phương thức này sẽ thêm vào *listFile* tên file được lưu ở local và tên file sẽ lưu trên server, đồng thời mã hoá thông tin thành một file JSON gồm các trường *ID*, *action* mang giá trị *publishFile*, *fname* để gửi đến server, ngược lại phương thức sẽ kết thúc.

- Phương thức `deletePublishFile(self, fname)`: Phương thức này sẽ tìm kiếm tên file trong `listFile`, sau đó nếu người dùng xác nhận xoá, phương thức này sẽ xoá tên file trong `listFile`, đồng thời mã hoá thông tin thành một file JSON gồm các trường `ID`, `action` mang giá trị `deletePublishFile`, `fname` để gửi đến server.
- Phương thức `requestListPeer(self, fname)`: Phương thức này sẽ mã hoá thông tin thành một file JSON gồm các trường `action` mang giá trị `requestListPeer`, `fname` để gửi đến server.
- Phương thức `endSystem(self)`: Phương thức này sẽ đóng kết nối của peer với server, đồng thời kiểm tra các socket và thread tồn tại để đóng và xoá khỏi hệ thống.

1.2 Server

1.2.1 Mô tả khái quát các function

- **Xem files của người dùng**
Chức năng này để liệt kê ra các files của một người dùng cụ thể khi có yêu cầu.
- **Kiểm tra người dùng**
Chức năng này cho phép server kiểm tra thông tin của một người dùng như: địa chỉ ip, port, tình trạng kết nối, số file chia sẻ trong mạng,...
- **Tìm kiếm files**
Chức năng này để tìm kiếm file được yêu cầu bởi người dùng.
Khi người dùng yêu cầu tải file với tên `fname`, server sẽ trả lại người nhận thông tin địa chỉ ip, port của những người dùng khác đang lưu trữ file được khai báo với tên `fname`.

1.2.2 Mô tả chi tiết các function

- Phương thức `__init__(self, port)`: Phương thức dùng để khởi tạo các Instance variable
 - Field `port`: Port của server được khởi tạo.
- Phương thức `startServer(self)`: Phương thức này tạo một thread với target là phương thức `listenMessage`, thêm thread vào danh sách `allThreads`, sau đó khởi động thread.
- Phương thức `listenMessage(self)`: Phương thức này sẽ thực thi một khối lệnh try except:
 - Phương thức sẽ thử tạo một TCP/IP socket, sau đó gán giá trị IP và port của server vào socket. Tiếp theo, gán giá trị của biến `endAllThread` là `False`, thêm socket vào `listSocket` và chạy socket. Phương thức sẽ chạy một vòng lặp với điều kiện biến `endAllThread` mang giá trị `false`. Khi ấy, một khối lệnh try except sẽ được thực thi:
 - + Nếu nhận được kết nối, tạo một thread mới để nhận phản hồi.
 - + Nếu thất bại, lệnh break sẽ được thực thi.
 - Nếu thất bại, server sẽ kết thúc hệ thống và kết thúc phương thức.

- Phương thức `receiveMessage(self, conn)`: Phương thức này sẽ chạy một vòng lặp khi biến `endAllThread` có giá trị `False`. Khi đó một khối lệnh try except sẽ được thực thi:
 - Giải mã data nhận được từ biến `connectSocket` và đưa dữ liệu từ dạng file JSON thành Python Object.
 - Nếu `jsonData["action"]` mang giá trị là `fetch` thì in ra màn hình `UNSUCCESS` và `SUCCESS` tương ứng với `jsonData["status"]` mang giá trị `unsuccessful` và `successful`.
 - Nếu `jsonData["action"]` mang giá trị là `register` thì chạy phương thức `handleRegister`.
 - Nếu `jsonData["action"]` mang giá trị là `publishFile` thì chạy phương thức `handlePublish`.
 - Nếu `jsonData["action"]` mang giá trị là `deletePublishFile` thì chạy phương thức `handleDelete`.
 - Nếu `jsonData["action"]` mang giá trị là `requestListFile` thì chạy phương thức `sendListFile`.
 - Nếu `jsonData["action"]` mang giá trị là `requestListPeer` thì chạy phương thức `sendListPeer`.
 - Nếu `jsonData["action"]` mang giá trị là `leaveNetwork` thì chạy phương thức `handleLeave`,
 - Nếu thất bại, lệnh `continue` sẽ được thực thi.
- Phương thức `handleRegister(self, conn, jsonData)`: Phương thức này sẽ kiểm tra tên, IP, port của peer đã tồn tại trong server hay chưa, nếu đã tồn tại thì kết thúc phương thức, ngược lại, thêm thông tin của peer vào `jsonPeerDatas`, tăng số chỉ ID của peer thêm 1, mã hoá thông tin thành một file JSON gồm các trường `ID`, `action`, `status` và gửi đến client.
- Phương thức `handlePublish(self, jsonData)`: Phương thức này sẽ chạy một vòng lặp để tìm vị trí data của peer trên server, sau đó thêm tên của file được publish vào `listFile` của peer và `listFile` của server nếu chưa tồn tại.
- Phương thức `handleDelete(self, jsonData)`: Phương thức này sẽ chạy một vòng lặp để tìm vị trí data của peer trên server, sau đó xoá tên của file được delete ra khỏi `listFile` của peer và `listFile` của server nếu không còn tồn tại bất kì file nào có tên trùng với tên của file được delete.
- Phương thức `sendListPeer(self, conn, fname)`: Phương thức này sẽ chạy một vòng lặp với mỗi peer trong `jsonPeerDatas`, thêm vào biến `listPeer` thông tin gồm các trường `name`, `ID`, `IP`, `port`, mã hoá thông tin thành một file JSON gồm các trường `action`, `listPeer` và gửi đến client.
- Phương thức `sendListFile(self, conn)`: Phương thức này sẽ mã hoá thông tin thành một file JSON gồm các trường `action`, `listFile` và gửi đến client.
- Phương thức `ping(self, hostname)`: Phương thức này sẽ chạy một vòng lặp với mỗi peer trong `jsonPeerDatas`:
 - Nếu tồn tại peer trong server, gán giá trị IP và port của peer, sau đó thử tạo một socket TCP/IP và kết nối tới peer theo địa chỉ IP và port trên. Nếu thất bại thì kết thúc phương thức, ngược lại mã hoá thông tin thành một file JSON gồm trường `action` và gửi tới client và nhận file JSON từ client gửi đến.

- Nếu không tồn tại peer, kết thúc phương thức.
- Phương thức `handleLeave(self, conn, ID)`: Phương thức này sẽ chạy một vòng lặp với mỗi peer trong `jsonPeerDatas` để tìm peer trong server, sau đó copy `listFile` của peer, xoá thông tin của peer ra khỏi server, xoá tên các file nếu không còn bất kì file nào trong server trùng tên với file của peer, sau đó đóng kết nối.
- Phương thức `endSystem(self)`: Phương thức này sẽ gán biến `endAllThread` giá trị `False`, sau đó đóng tất cả socket, xoá tất cả socket và thread.

2 The communication protocols

2.1 HTTP (Hypertext Transfer Protocol)

- Giao thức HTTP (Hypertext Transfer Protocol) là một giao thức truyền tải dữ liệu phổ biến được sử dụng trên Internet. Trong bài tập lớn này, HTTP sẽ được ứng dụng dựa trên mô hình peer-to-peer.
- Một số chức năng chính mà HTTP có thể được sử dụng trong bài tập lớn lần này là:

- **Xác định danh sách các files từ Server**

Mỗi khi một client muốn biết danh sách các tệp mà một client khác đang chia sẻ, nó có thể gửi yêu cầu HTTP GET đến server.

HTTP Request từ Client: `GET /files/<client_id>`

HTTP Response từ Server: Trả về danh sách các tệp của client cần truy vấn.

- **Tải xuống files từ một Client khác**

Khi một Client muốn tải một file từ một Client khác, nó có thể yêu cầu file đó thông qua HTTP. Client chứa file sẽ trở thành một Server tạm thời để tải xuống.

HTTP Request từ Client: `GET /files/<shared_filename>`

HTTP Response từ Client chứa file: Trả về nội dung của file hoặc HTTP 404 Not Found nếu file không tồn tại.

- **Khai báo files lên Server**

Khi một Client muốn chia sẻ một file, nó có thể gửi yêu cầu HTTP POST chứa file đến Server.

HTTP Request từ Client: `POST /publish`

HTTP Response từ Server: Trả về xác nhận hoặc thông báo lỗi nếu có.

2.2 Transmission Control Protocol (TCP)

- Giao thức TCP (Transmission Control Protocol) là một trong những thành phần chính của giao thức TCP/IP stack. TCP đóng vai trò quan trọng trong việc thiết lập và duy trì các kết nối mạng ổn định giữa Server và các Client, cũng như giữa các Client khi chúng tải xuống tệp từ nhau.
- Trong bài tập lớn này, TCP có chức năng chính như sau:

– **Đảm bảo truyền tải dữ liệu ổn định**

TCP đảm bảo việc truyền tải dữ liệu một cách đáng tin cậy. Điều này rất quan trọng khi chuyển thông tin về files và yêu cầu tải files từ Server đến Client hoặc giữa các Client.

– **Xác nhận nhận dữ liệu**

TCP sử dụng cơ chế xác nhận ACK (acknowledgment) để đảm bảo rằng dữ liệu đã được gửi từ một bên đã được nhận đúng và hoàn chỉnh bởi bên nhận. Điều này giúp đảm bảo tính toàn vẹn của dữ liệu khi truyền tải.

– **Xử lý đa nhiệm**

TCP cho phép Server xử lý nhiều kết nối từ các Client cùng lúc, cho phép xử lý nhiều yêu cầu tải Files từ nhiều Client khác nhau đồng thời.

– **Tự động kết nối lại**

Nếu kết nối giữa Server và Client bị gián đoạn, TCP có khả năng tự động kết nối lại mà không yêu cầu sự can thiệp từ phía người dùng.

2.3 Internet Protocol(IP)

- Giao thức Internet Protocol (IP) chủ yếu được sử dụng để xác định các địa chỉ mạng và địa chỉ host trên mạng Internet hoặc trong một mạng cục bộ (LAN).

- Trong bài tập lớn này, IP có chức năng chính như sau:

– **Xác định địa chỉ Server và Client**

Mỗi máy tính (bao gồm Server và client) trên mạng LAN hoặc Internet đều có một địa chỉ IP duy nhất.

Chức năng: Địa chỉ IP được sử dụng để xác định Server và Client trong quá trình gửi và nhận yêu cầu, đáp ứng.

– **Định tuyến giao thức**

IP quản lý việc định tuyến dữ liệu từ nguồn đến đích trên mạng.

Chức năng: IP giúp xác định tuyến đường dữ liệu từ Server đến client hoặc từ Client này đến Client khác, đảm bảo dữ liệu được định tuyến đúng đắn.

– **Phân đoạn và ghép dữ liệu**

IP có thể phân đoạn dữ liệu thành các gói nhỏ để truyền tải trên mạng, sau đó ghép chúng lại thành dữ liệu ban đầu tại đích.

Chức năng: Phân đoạn dữ liệu giúp truyền tải dữ liệu lớn một cách hiệu quả và đáng tin cậy giữa các Server và Client.

– **Xác định các mạng cục bộ**

Địa chỉ IP có thể cho biết nếu các Server và Client nằm trong cùng một mạng cục bộ hoặc nếu cần qua các router để truy cập các mạng khác.

Chức năng: Điều này quan trọng khi các Server và Client nằm trong các mạng LAN khác nhau và cần giao tiếp thông qua router.

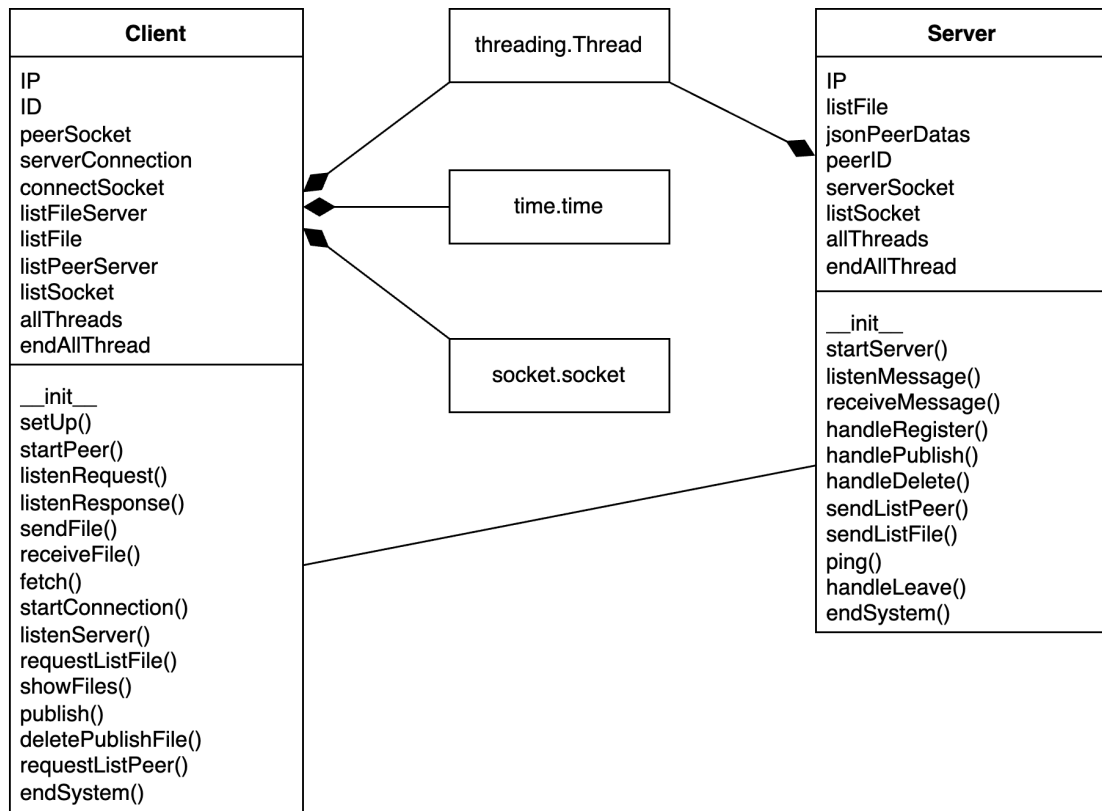


– **Phân giải tên miền (Domain Names) thành địa chỉ IP**

Sử dụng dịch vụ DNS (Domain Name System) để chuyển đổi tên miền (ví dụ: `www.example.com`) thành địa chỉ IP.

Chức năng: Điều này cho phép người dùng sử dụng tên miền thay vì nhớ các địa chỉ IP phức tạp khi kết nối với các máy chủ và client.

3 Class Diagram

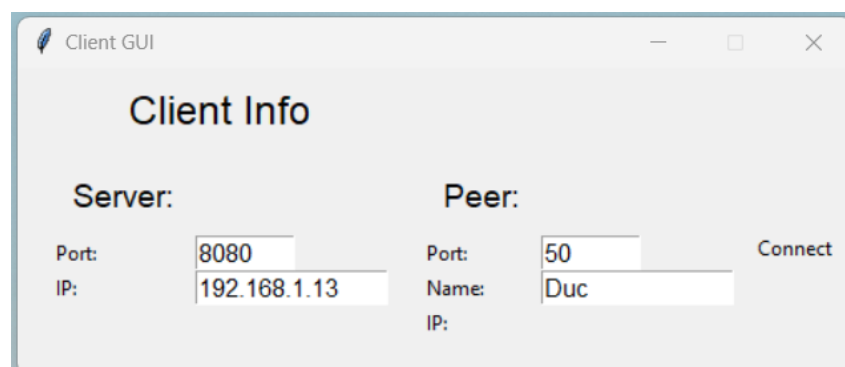


Hình 3.1 Class diagram

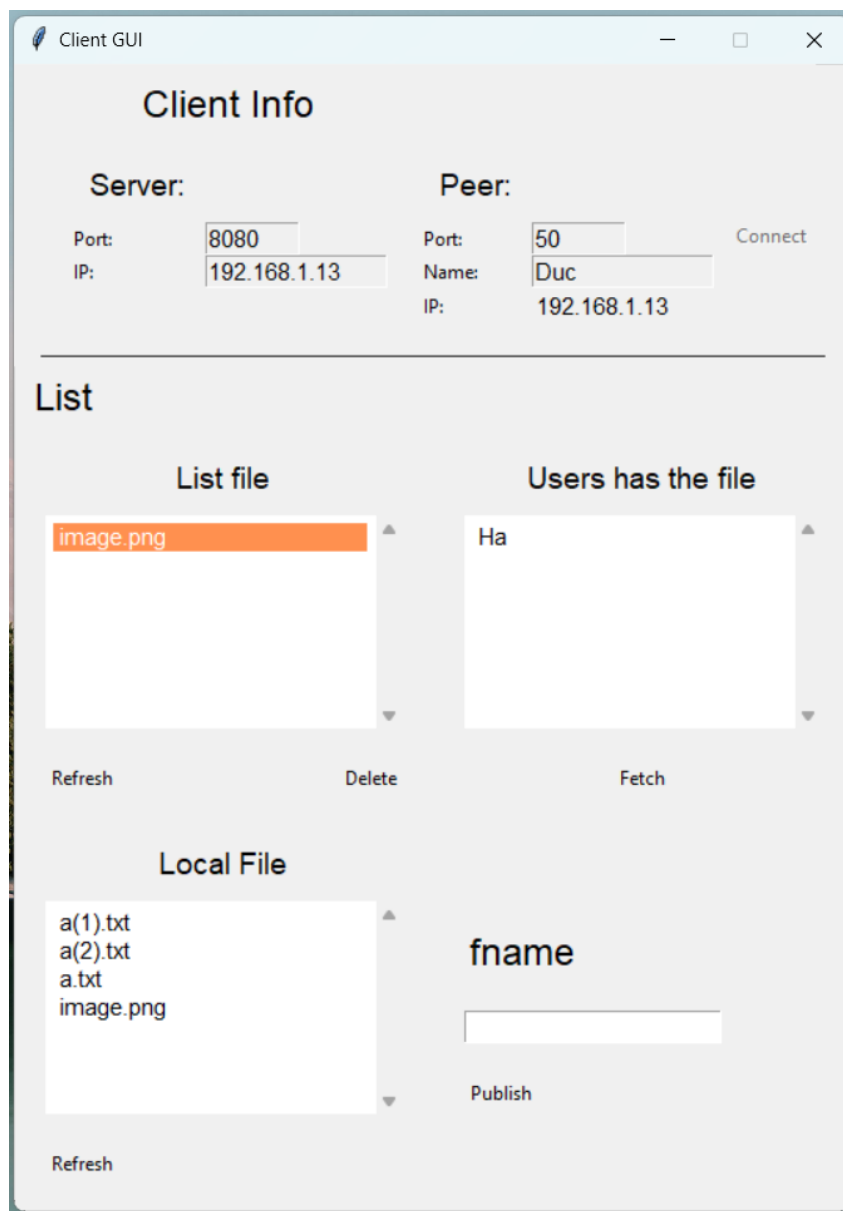
4 Validation and evaluation of actual result

4.1 Connect To The Server

- **Connect:** Đầu tiên chúng ta sẽ kết nối tới Server bằng cách nhập địa chỉ IP và Port của Server cùng với Port và Name chúng ta sẽ dùng để có thể giao tiếp với những người trong cùng một Server.



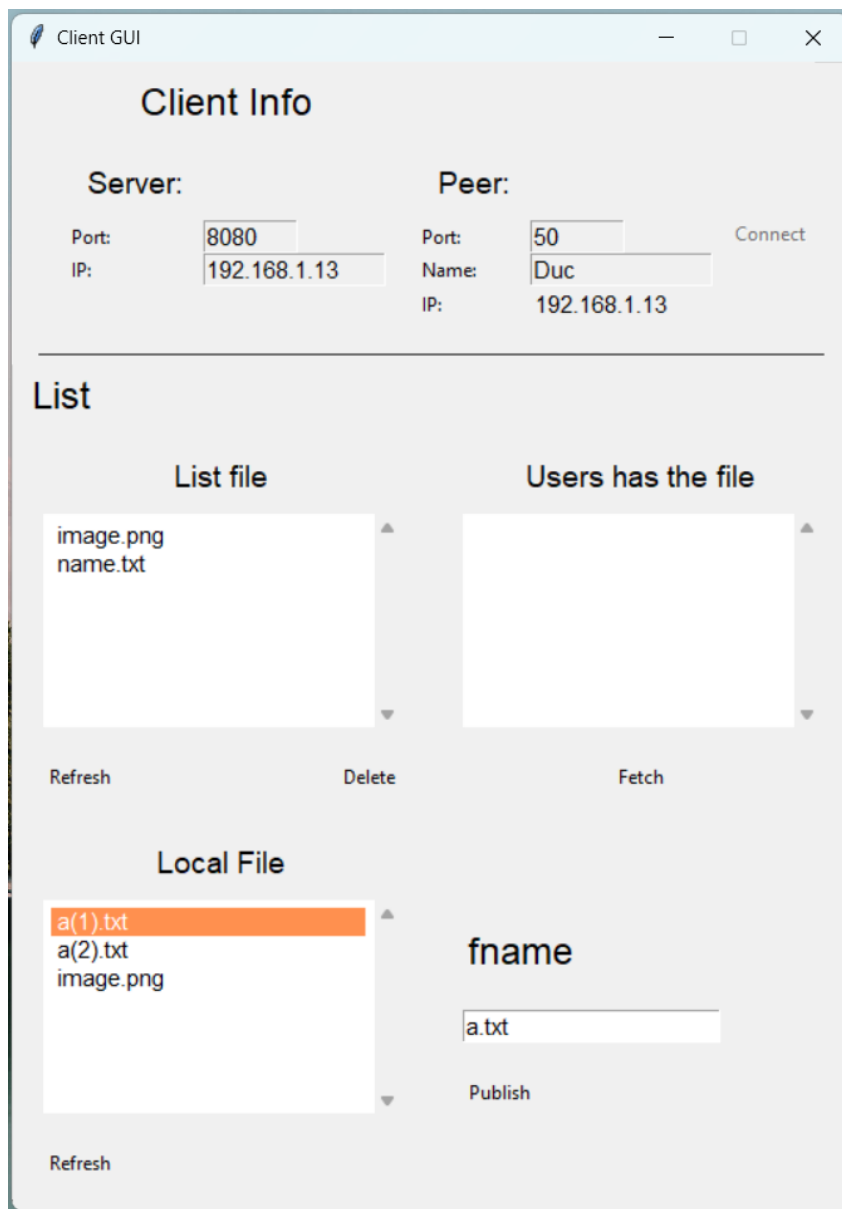
- Sau khi connect thành công tới Server, ta sẽ nhận được giao diện ứng dụng như sau:



- Giao diện của ứng dụng bao gồm:
 - List File: bao gồm các files đã được người dùng chia sẻ lên Server.
 - Users Has The File: là những người sở hữu file mà ta đã chọn ở List File.
 - Local File: gồm những files ở local repository của mình những chưa được publish lên Server.
 - Fname: là tên mà mình muốn đặt cho file ở Local File sẽ được publish lên Server.

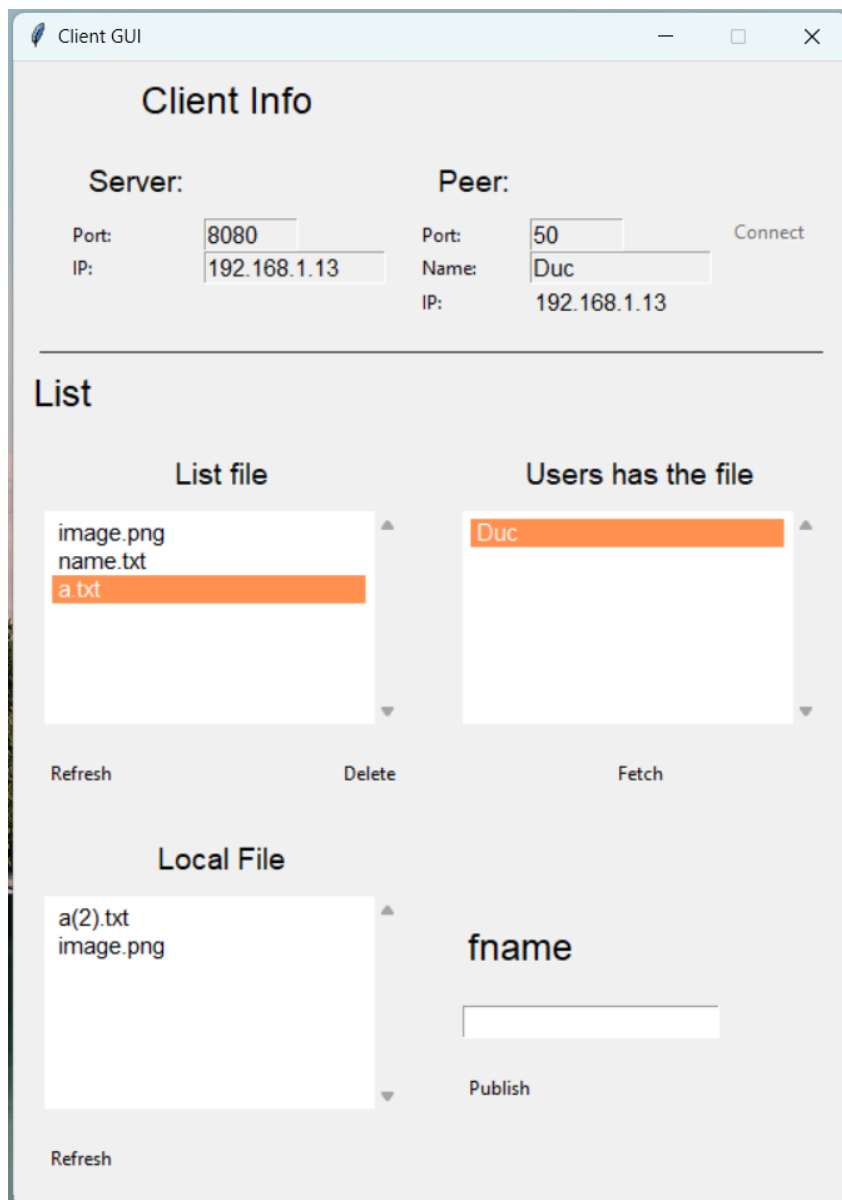
4.2 Publish Files

- Publish:** Chúng ta chọn file muốn Publish ở Local File sau đó nhập tên fname mà mình muốn đặt cho file đó khi nó được truyền thông tin lên Server.



The image shows a 'Client GUI' window with a light blue title bar. The main content area is divided into several sections. At the top is 'Client Info', which contains two columns: 'Server' and 'Peer'. The 'Server' column has input fields for 'Port' (8080) and 'IP' (192.168.1.13). The 'Peer' column has input fields for 'Port' (50), 'Name' (Duc), and 'IP' (192.168.1.13). A 'Connect' button is located to the right of the 'Peer' inputs. Below this is a 'List' section with two columns: 'List file' and 'Users has the file'. The 'List file' column contains a list box with 'image.png' and 'name.txt'. The 'Users has the file' column contains an empty list box. Below these list boxes are three buttons: 'Refresh', 'Delete', and 'Fetch'. At the bottom is a 'Local File' section with a list box containing 'a(1).txt', 'a(2).txt', and 'image.png'. To the right of this list box is a 'fname' label and an input field containing 'a.txt'. Below the input field is a 'Publish' button. At the bottom left of the 'Local File' section is a 'Refresh' button.

- **Kết quả:** Sau khi Publish file thành công file đó sẽ bị ẩn đi ở Local File và truyền fname lên Server.



The screenshot shows a 'Client GUI' window with the following sections:

- Client Info**:
 - Server:** Port: 8080, IP: 192.168.1.13
 - Peer:** Port: 50, Name: Duc, IP: 192.168.1.13
 - A 'Connect' button is located to the right of the Peer information.
- List**:
 - List file**: A list box containing 'image.png', 'name.txt', and 'a.txt'. 'a.txt' is selected.
 - Users has the file**: A list box containing 'Duc'. 'Duc' is selected.
 - Buttons: 'Refresh', 'Delete', and 'Fetch' are located below the list boxes.
- Local File**:
 - A list box containing 'a(2).txt' and 'image.png'.
 - fname**: A text input field.
 - A 'Publish' button is located below the 'fname' field.
 - A 'Refresh' button is located at the bottom left of this section.

4.3 Fetch Files

- **Fetch:** Để Fetch một file có thông tin trên Server ta chọn file ở List File sau đó chọn Peer chúng ta muốn Fetch về ở bên Users Has The File.

Client GUI

Client Info

Server:

Port: 8080
IP: 192.168.1.13

Peer:

Port: 50
Name: Duc
IP: 192.168.1.13

Connect

List

List file

image.png
name.txt
a.txt

RefreshDeleteFetch

Users has the file

Ha

Local File

a(2).txt
image.png

fname

Publish

Refresh

- Sau khi Fetch thành công chúng ta sẽ nhận được file và lưu nó tại Local File:

Client GUI

Client Info

Server:

Port: 8080
IP: 192.168.1.13

Peer:

Port: 50
Name: Duc
IP: 192.168.1.13

Connect

List

List file

image.png
name.txt
a.txt

RefreshDelete

Users has the file

Ha

Fetch

Local File

a(2).txt
image(1).png
image.png

Refresh

fname

Publish