ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

-----OO📖OO-----

BÁO CÁO ĐỒ ÁN
MẠNG MÁY TÍNH

Phần mềm download / upload file dùng kỹ thuật đa luồng

GVHD : Đỗ Hoàng Cường
: Huỳnh Thụy Bảo Trân
Lớp : 23CLC06
Sinh viên : Huỳnh Tuấn Phong – MSSV: 23127100
: Nguyễn Gia Huy – MSSV: 23127378
: Bùi Hải Long – MSSV: 23127409

TP Hồ Chí Minh, Tháng 8/2024

Mục lục

I. KHÁI QUÁT CHUNG VỀ ĐỒ ÁN

1. Link video demo:

https://youtu.be/pVa-UlBrsqQ

2. Ngôn ngữ lập trình:

Đồ án chỉ sử dụng ngôn ngữ lập trình Python 3.12 làm ngôn ngữ chính

3. Thư viện

a. Thư viện chuẩn

- Tkinter
- Socket
- Time
- DateTime
- Os
- Shutil
- Threading
- Queue
- Struct
- Json

b. Thư viện bên ngoài

- Custometkinter
- Pillow

c. Cài đặt

pip install -r requirements.txt

d. Chạy server

python server.py

e. Chạy client

python client.py

II. MÔ TẢ HỆ THỐNG

1. Chương trình download/upload file client-server dùng kỹ thuật đa luồng

Một chương trình download/upload file đa luồng yêu cầu client và server có khả năng truyền nhận dữ liệu file một cách đa luồng, ở đây client là bên gửi yêu cầu đầu tiên và server là bên cung cấp phản hồi.

Tuy nhiên để dễ dàng trong lúc sử dụng, hệ thống sẽ có thêm một số chức năng như tạo folder, xóa file, xóa folder, truyền thông tin các file hiện có.

2. Giao thức quản lý tập tin cơ bản

Tất nhiên để client và server có những khả năng trên thì server cần hiểu yêu cầu từ client và client hiểu phản hồi từ server. Vì vậy ở đây chúng ta sẽ định nghĩa một giao thức quản lý tập tin cơ bản.

a. Sơ lược về giao thức

Giao thức mà chúng ta định nghĩa sau đây sẽ sử dụng giao thức TCP ở tầng transport. Cách gửi và yêu cầu 1 đoạn dữ liệu đã được tham khảo từ chức năng Byte serving của

giao thức HTTP sẽ cho phép ta lập trình đa luồng dễ dàng hơn. Cách phân biệt mã lệnh bằng opcode được tham khảo từ giao thức TFTP (vì nó đơn giản) và cuối cùng cách đồng bộ hóa dữ liệu sẽ dùng cách giao tiếp Push technology thông qua một quá trình được gọi là publish–subscribe model nhằm giảm lượng dữ liệu trên đường truyền.

b. Quy ước mã lệnh

Ở đây ta sẽ liệt kê các mệnh lệnh mà client có thể yêu cầu từ server:

opcode	tên gọi
0	Read request (RRQ)
1	Write request (WRQ)
2	Data read request (DRRQ)
3	Data write request (DWRQ)
4	Finish write request (FWRQ)
5	Delete request (DRQ)
6	Directory request (DTRQ)
7	Create folder request (FRQ)

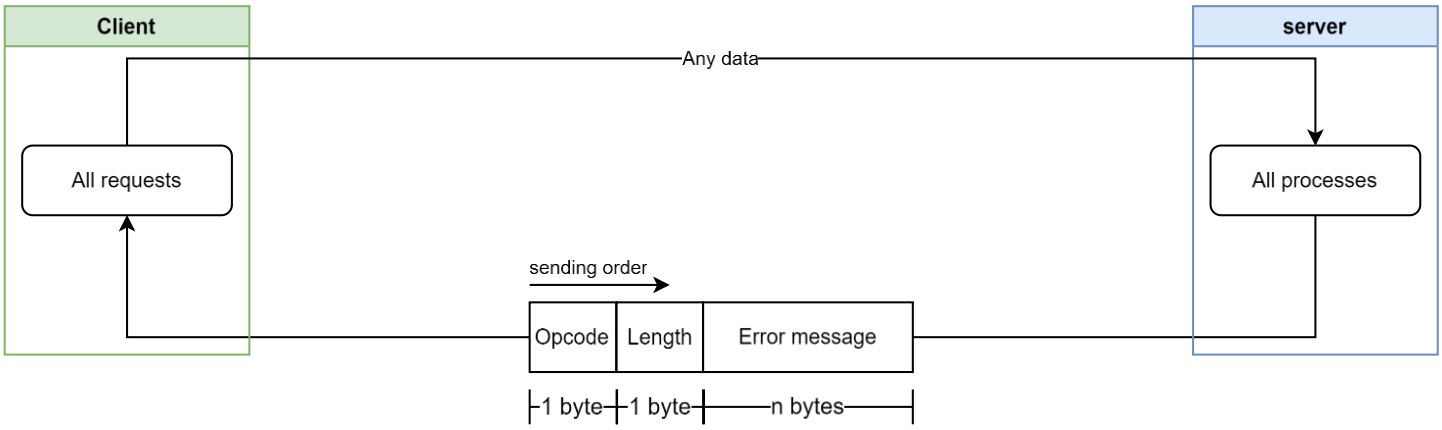
và kết quả client nhận từ server:

opcode	tên gọi
0	Error (ERROR)
1	Success (SUCCESS)

c. Quy ước chung của các lệnh

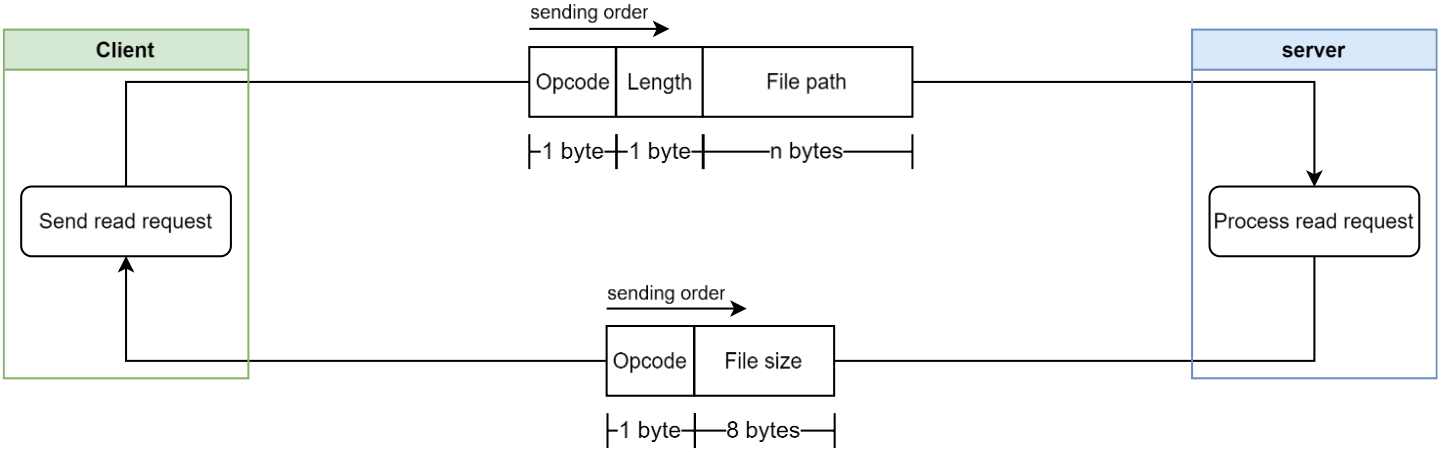
- Kích cỡ địa chỉ tối đa là 2^8 bytes = 255 bytes.
- Kích cỡ file tối đa là 2^(8*8) bytes ~ 18.3 exabytes.
- Khi một đoạn gửi đi từ client đến server có độ dài n bytes thì đoạn đứng trước mang giá trị số cho n.
- Server sẽ gửi cấu trúc giống nhau khi có lỗi.
- Số sẽ được gửi theo kiểu Big endian chuỗi ký tự (string) được gửi theo định dạng UTF-8

d. Trường hợp server báo lỗi



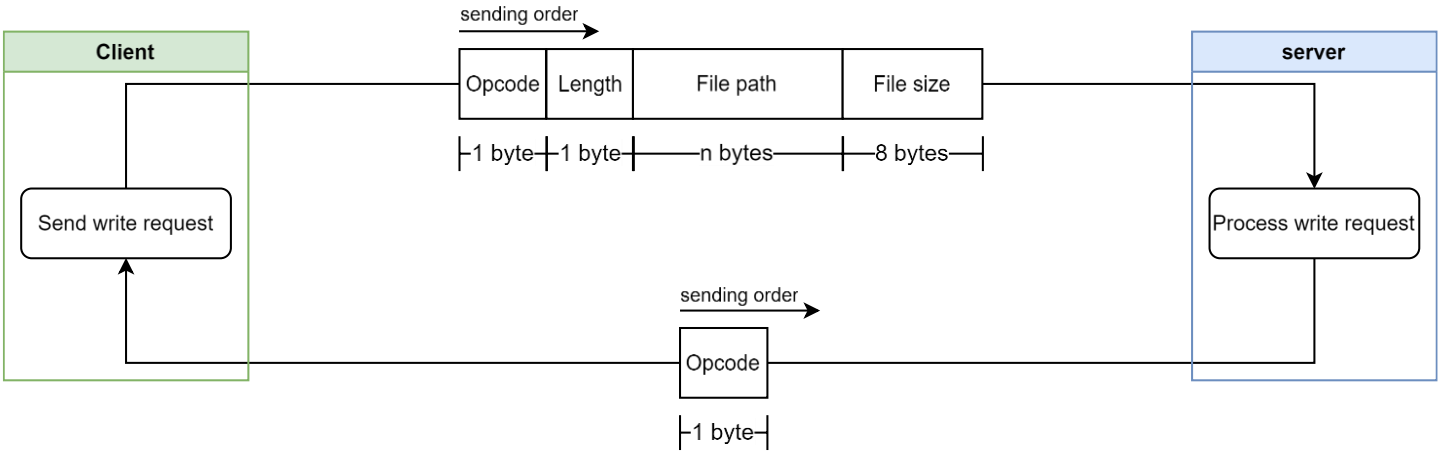
Ta đi vào trường hợp lỗi đầu tiên vì đây là cấu trúc sẽ được dùng cho mọi trường hợp lỗi server gửi về. Bất kể dữ liệu gửi đi server sẽ trả về đầu tiên là 1 byte opcode giá trị 0 nghĩa là kết quả lỗi so với bảng trên kèm theo 1 byte độ dài lỗi và n bytes ký tự thông điệp lỗi.

e. Lệnh read request (RRQ)



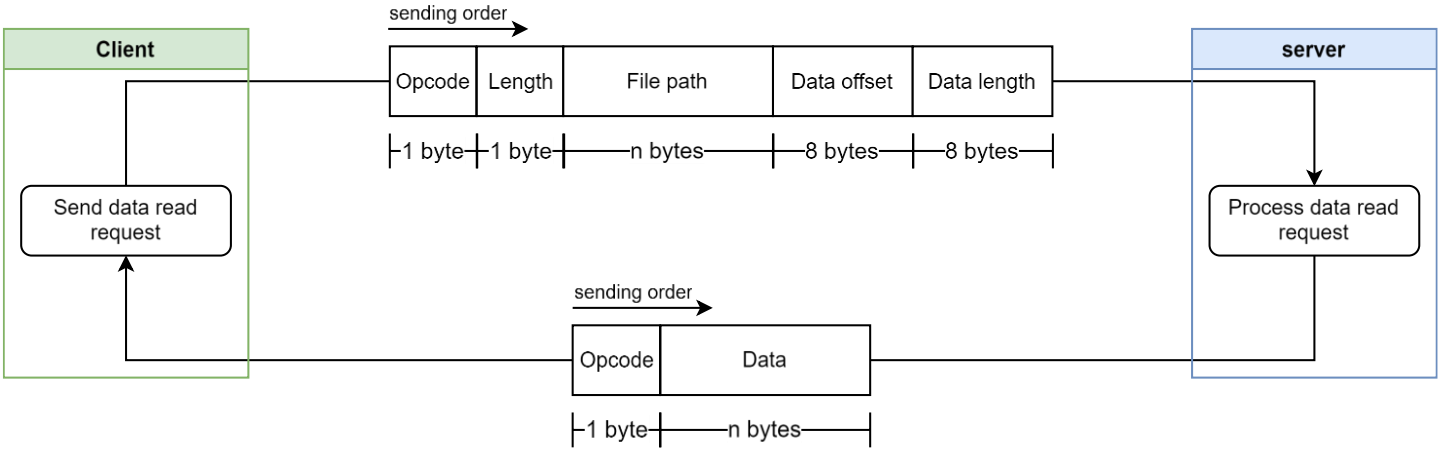
Đây là lệnh yêu cầu đọc file. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, cuối cùng là n bytes ký tự cho địa chỉ file trên server. Server sẽ trả lại 1 byte opcode và 8 bytes cho kích cỡ file trong database nếu không gặp lỗi. Lỗi có thể trả về ở đây là khi địa chỉ không hợp lệ.

f. Lệnh write request (WRQ)



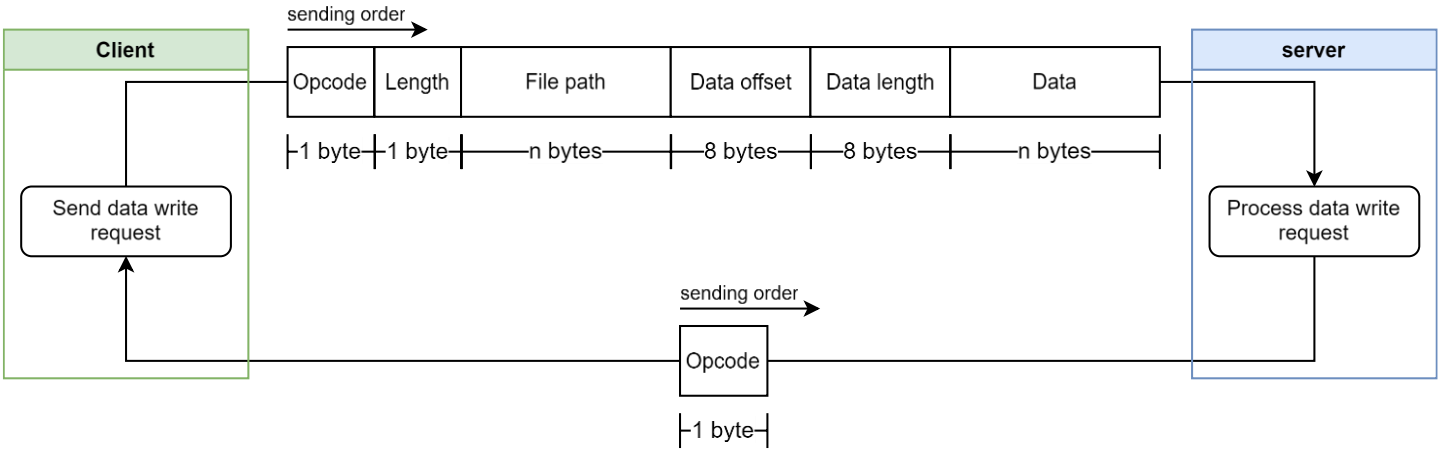
Đây là lệnh yêu cầu ghi file. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, n bytes ký tự cho địa chỉ file, cuối cùng là 8 bytes số cho kính cỡ file. Server sẽ tạo một file trong database với kích cỡ từ client và tên từ client đã gửi kèm ".uploading" sau đó trả lại 1 byte opcode nếu không gặp lỗi. Lỗi có thể trả về ở đây là địa chỉ không hợp lệ, server không đủ dung lượng, file đã đang được ghi (đã có file ".uploading").

g. Lệnh data read request (DRRQ)



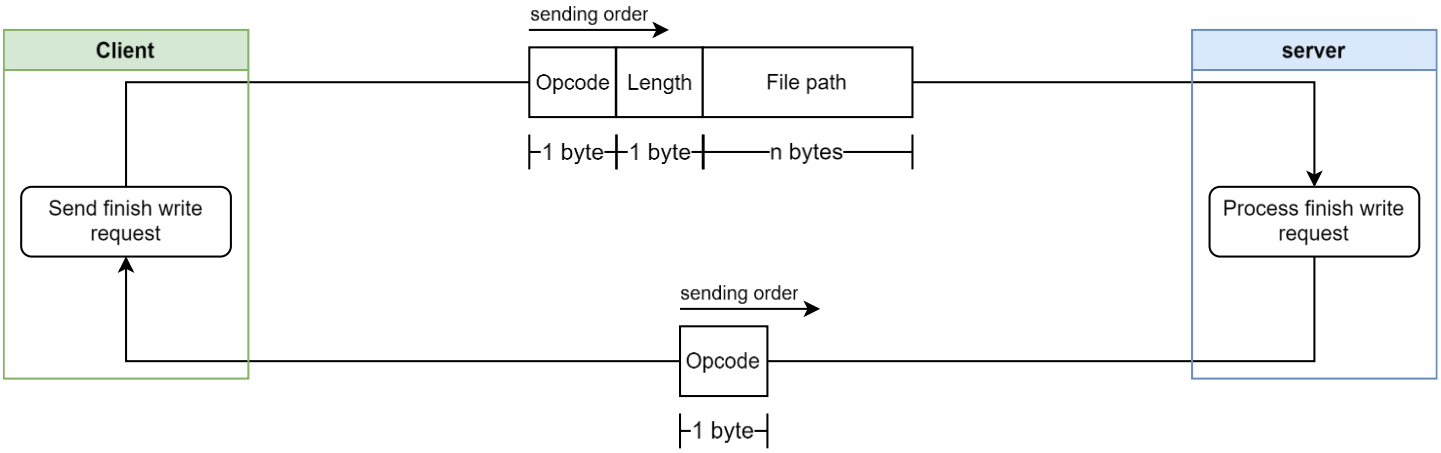
Đây là lệnh yêu cầu đọc dữ liệu vào file. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, n bytes ký tự cho địa chỉ file, 8 bytes số cho vị trí đọc dữ liệu, 8 bytes số cho độ dài dữ liệu cần đọc. Server sẽ gửi về 1 byte opcode, n bytes đoạn dữ liệu của file tại vị trí client yêu cầu nếu không gặp lỗi. Lỗi có thể trả về ở đây là địa chỉ không hợp lệ, vị trí đọc không hợp lệ, độ dài đọc không hợp lệ.

h. Lệnh data write request (DWRQ)



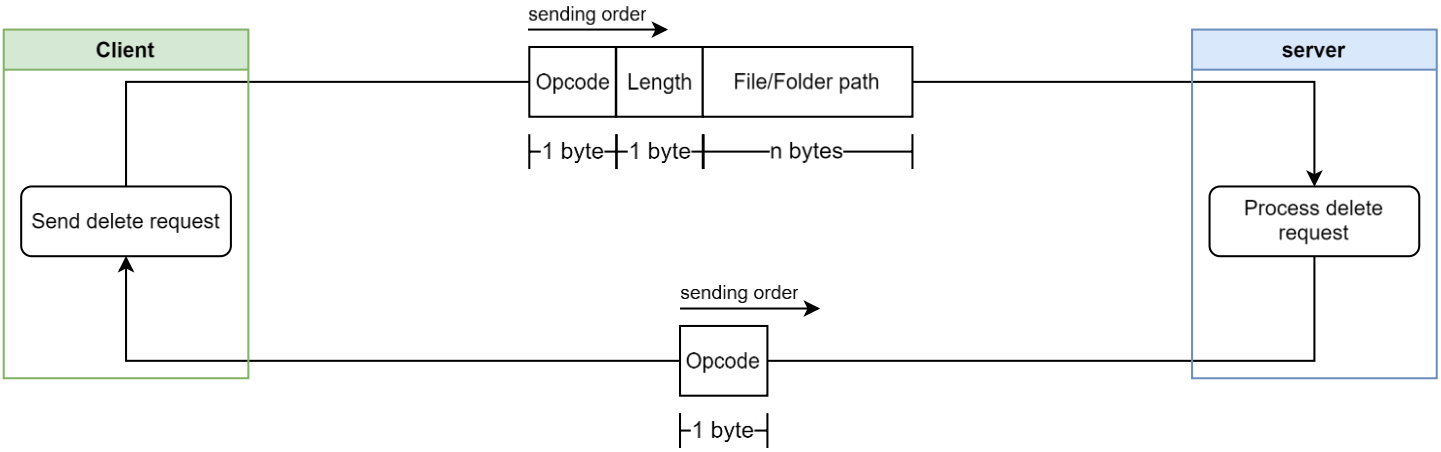
Đây là lệnh yêu cầu ghi dữ liệu vào file. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, n bytes ký tự cho địa chỉ file, 8 bytes số cho vị trí ghi dữ liệu, 8 bytes số cho độ dài dữ liệu cần ghi, n bytes cho đoạn dữ liệu. Server sẽ ghi đoạn dữ liệu nhận được vào vị trí trong file ".uploading" client yêu cầu sau đó gửi về 1 byte opcode nếu không gặp lỗi. Lỗi có thể trả về ở đây là địa chỉ không hợp lệ, vị trí ghi không hợp lệ, độ dài ghi không hợp lệ.

i. Lệnh finish write request (FWRQ)



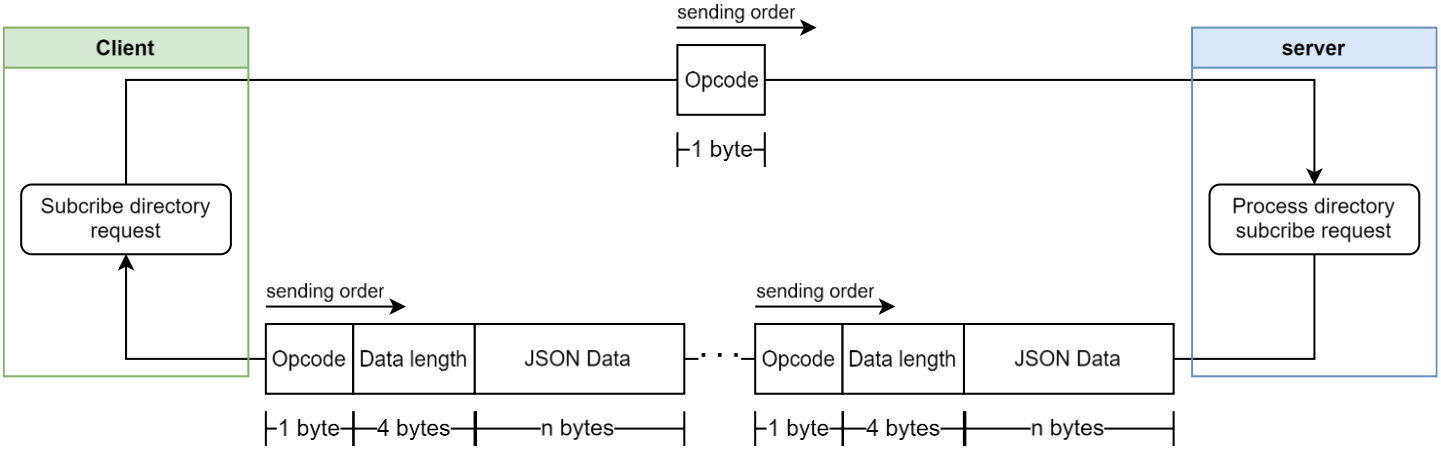
Đây là lệnh hoàn thành quá trình ghi file. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, n bytes ký tự cho địa chỉ file. Server sẽ cắt bỏ phần ".uploading" và file coi như sẽ không được ghi nữa sau đó gửi về 1 byte opcode nếu không gặp lỗi. Lỗi có thể trả về ở đây là địa chỉ không hợp lệ.

j. Lệnh delete request (DRQ)



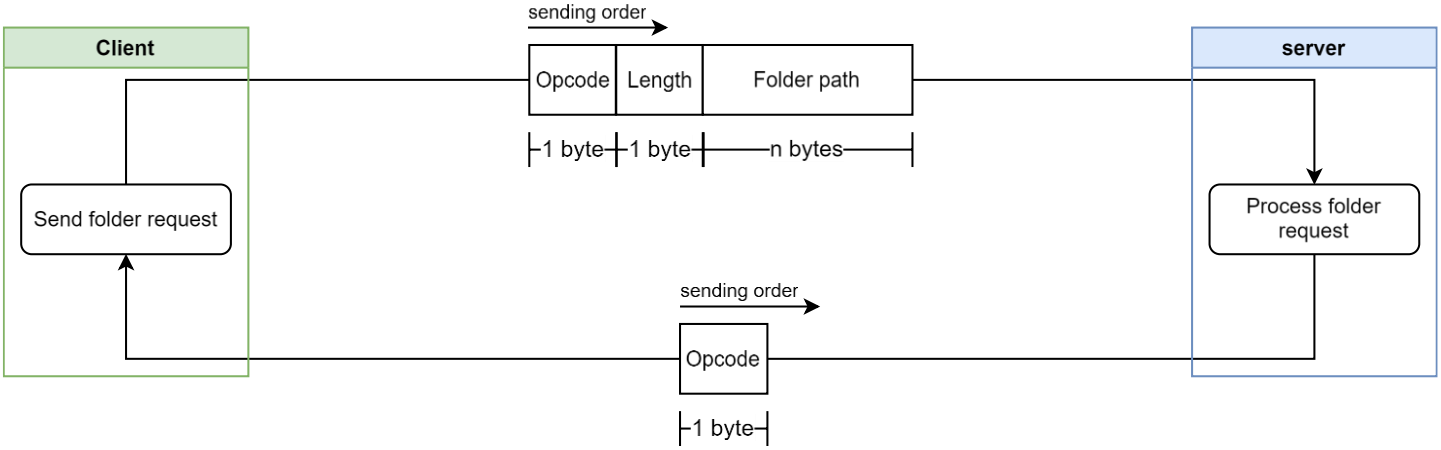
Đây là lệnh yêu cầu xóa file/folder. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, n bytes ký tự cho địa chỉ file/folder. Server sẽ xóa file/folder tại địa chỉ client yêu cầu và trả về 1 byte opcode nếu không gặp lỗi. Lệnh này không có lỗi trả về.

k. Lệnh directory request (DTRQ)



Đây là lệnh đồng bộ hóa dữ liệu thông qua mô hình publish–subscribe. Client sẽ gửi 1 byte opcode nhằm đang ký (subcribe) kết nối hiện tại cho quá trình đồng bộ. Server sẽ gửi ngay lặp tức 1 byte opcode, 4 bytes cho độ dài dữ liệu, n bytes dữ liệu database định dạng JSON sau khi client gửi opcode, sau đó kết nối này vẫn giữ nguyên trạng thái và mỗi khi database có thay đổi server sẽ tự động xuất bản (publish) dữ liệu như trên về client mà không cần yêu cầu từ client nếu kết nối TCP vẫn được mở. Lệnh này không có lỗi trả về.

l. Lệnh create folder request (FRQ)

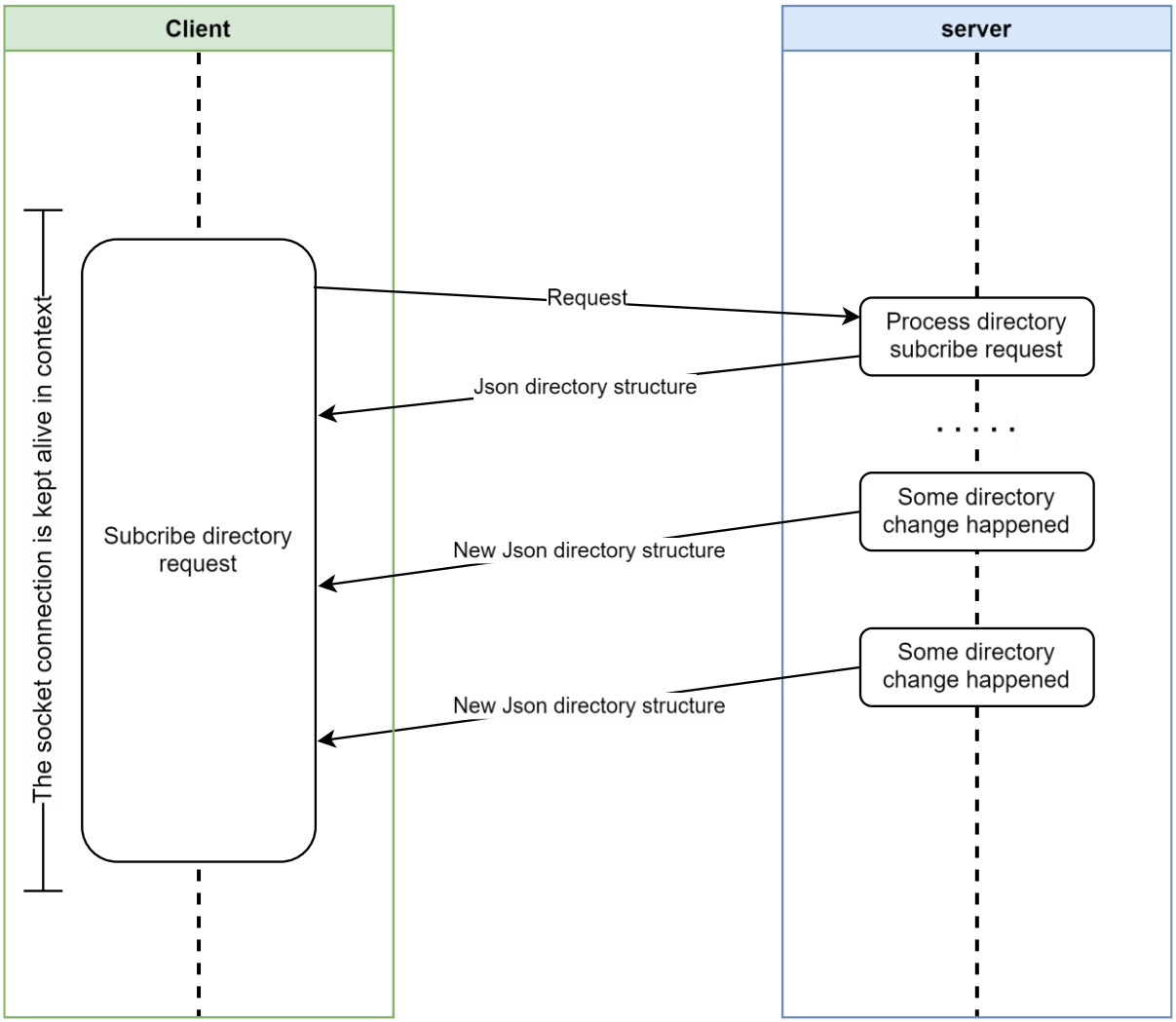


Đây là lệnh tạo folder. Client sẽ gửi 1 byte số cho opcode, 1 byte số cho kích cỡ địa chỉ, n bytes ký tự cho địa chỉ folder. Server sẽ tạo folder với địa chỉ client yêu cầu và gửi 1 byte opcode nếu không gặp lỗi. Lỗi có thể trả về ở đây là địa chỉ không hợp lệ, không thể tạo folder.

3. Sử dụng giao thức quản lý tập tin cơ bản

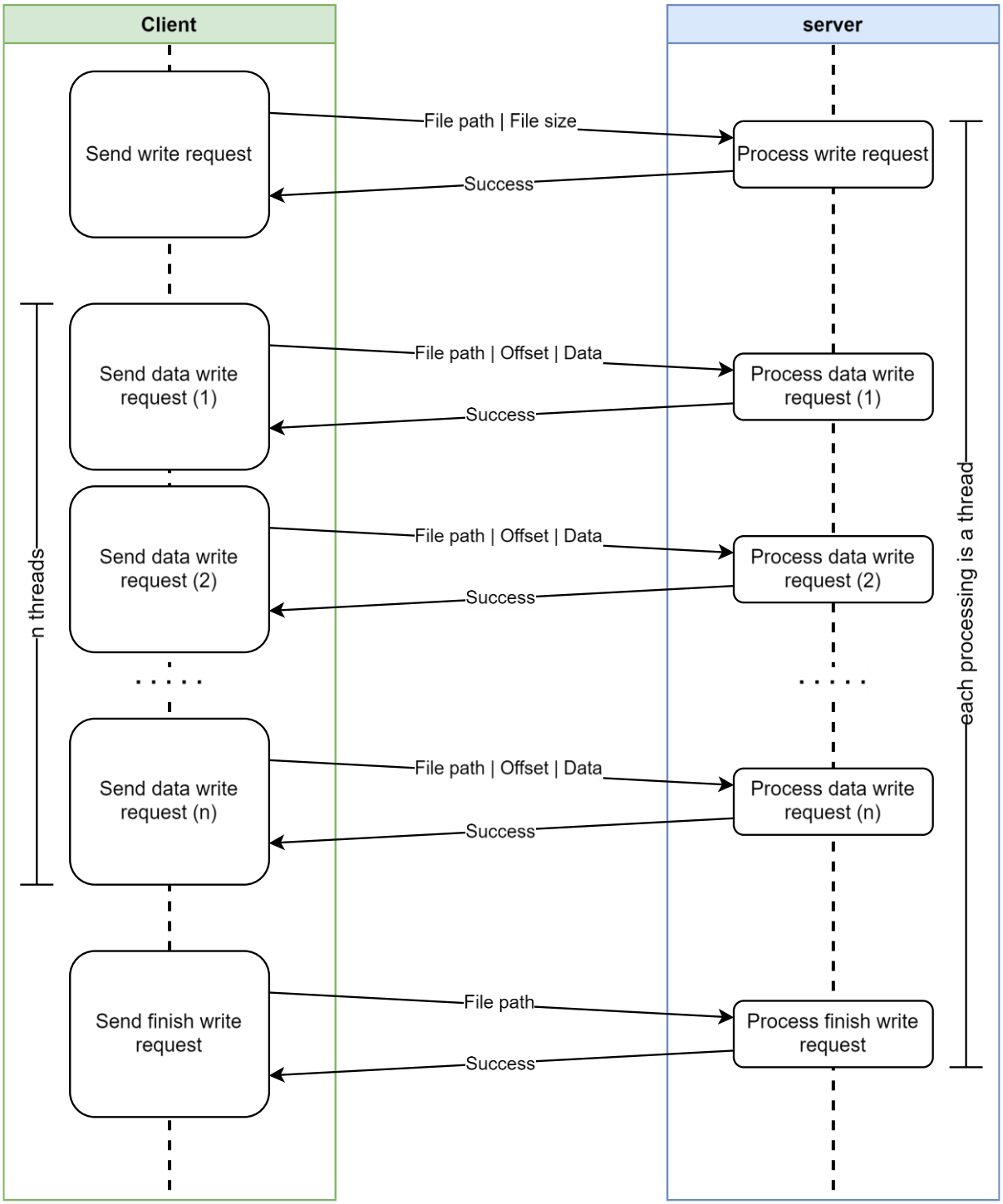
Với những lệnh đã có ở trên ta sẽ dễ dàng lập trình hệ thống download/upload file client-server dùng kỹ thuật đa luồng.

a. Quy trình đồng bộ dữ liệu



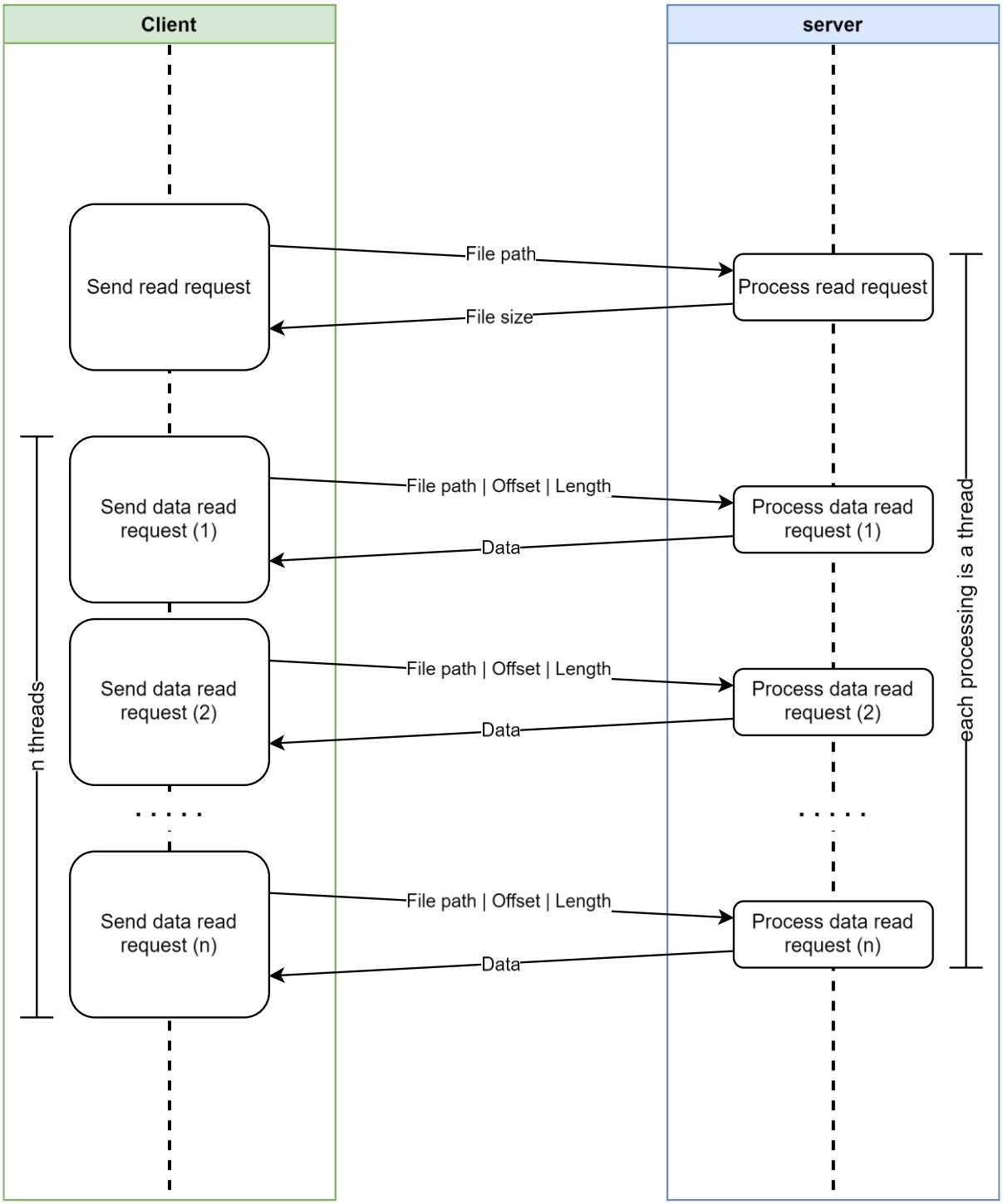
Chức năng này không cần thiết đối với một quá trình download/upload thông thường dùng giao thức mà ta đã định nghĩa nhưng đối với chương trình dùng để quản lý tập tin thư mục thì chức năng này là cần thiết và vì hầu hết chương trình có chức năng download/upload hiện nay là để quản lý tập tin thư mục nên chúng ta sẽ thêm vào chương trình chức năng này.

b. Quy trình upload file lên server



Để upload file lên server, client cần biết địa chỉ trong database muốn đặt file và địa chỉ này có thể được quy ước, được định sẵn hoặc được biết từ quá trình đồng bộ dữ liệu. Quá trình ghi dữ liệu vào file là bước mà ta có thể sử dụng kỹ thuật đa luồng chia file và gửi từng đoạn riêng biệt qua nhiều kết nối tất nhiên quá trình này hoàn toàn có thể chạy trên 1 luồng tùy nhu cầu người dùng. Việc gửi lệnh hoàn thành quá trình ghi là cần thiết để server biết file này sẽ không còn được ghi trong tương lai. Trong chương trình ta sẽ lấy địa chỉ từ quá trình đồng bộ dữ liệu và số luồng cùng độ dài đoạn dữ liệu là do người dùng chọn.

c. Quy trình download file xuống client



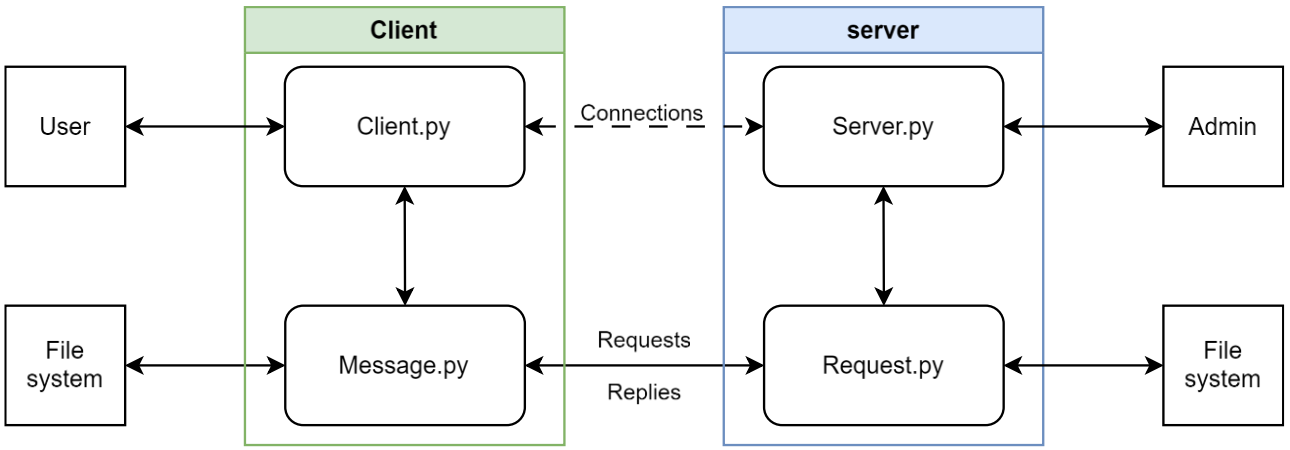
Để download file xuống client, client cần biết địa chỉ và kích thước file trên server, địa chỉ này có thể được quy ước, được định sẵn hoặc được biết từ quá trình đồng bộ dữ liệu, kích thước file có thể được lấy từ lệnh read request hoặc quá trình đồng bộ dữ liệu. Quá trình đọc dữ liệu vào file là bước mà ta có thể sử dụng kỹ thuật đa luồng đọc từng đoạn riêng biệt qua nhiều kết nối tất nhiên quá trình này hoàn toàn có thể chạy trên 1 luồng tùy nhu cầu người dùng. Trong chương trình ta sẽ lấy địa chỉ và kích thước từ quá trình đồng bộ dữ liệu nên không cần lệnh read request và số luồng cùng độ dài đoạn dữ liệu là do người dùng chọn.

d. Một số quy trình đơn giản

Để xóa file/folder, tạo folder là những quy trình đơn giản chỉ cần gọi đúng lệnh tương ứng.

Quy trình upload folder là sự kết hợp giữa quy trình tạo folder và upload file.

4. Cấu trúc code của chương trình



Để thực hiện việc giao tiếp giữa Client và Server, Client có modules Message.py gồm những hàm và thủ tục giúp ứng dụng bên Client đóng gói những yêu cầu, dữ liệu cần gửi cũng như giúp nhận dạng thông điệp phản hồi từ Server. Những thông điệp từ Server giúp báo hiệu yêu cầu đã được xử lí hay đã xảy ra lỗi. Đối với Server có modules là request.py giúp xử lí các yêu cầu gửi từ client. Việc xử lí các yêu cầu bao gồm việc kiểm lỗi, gửi lại thông điệp phản hồi cho Client.

Giao thức được định nghĩa ở trên sẽ được lập trình vào Message.py và Request.py .

Giao diện người dùng sẽ được lập trình vào Client.py và Server.py .

III. MÔ TẢ GIAO DIỆN

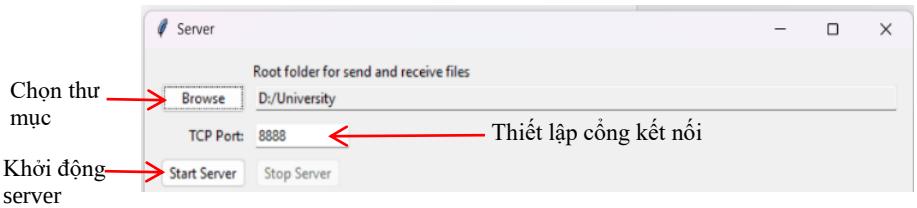
5. Giao diện sever

a. Tính năng

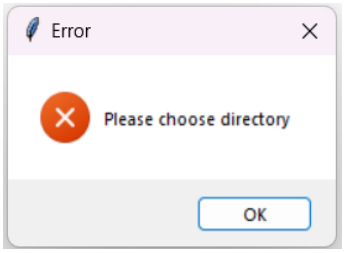
- Tùy chọn thư mục chứa dữ liệu
- Thay đổi Port
- Mở và đóng kết nối
- Hiển thị thông tin kết nối với các client

b. Mô tả các đối tượng

- *Thiết đặt ban đầu:* Sau khi chạy chương trình server.py sẽ hiện cửa sổ
  - ✓ *Bước 1:* Chọn Browse để chuyển đến cửa sổ Select folder
  - ✓ *Bước 2:* Chọn thư mục chứa tài nguyên
  - ✓ *Bước 3:* Thiết lập Port kết nối nếu cần (mặt định là 8888)
  - *Bước 4:* Chọn Start Server để bắt đầu kết nối



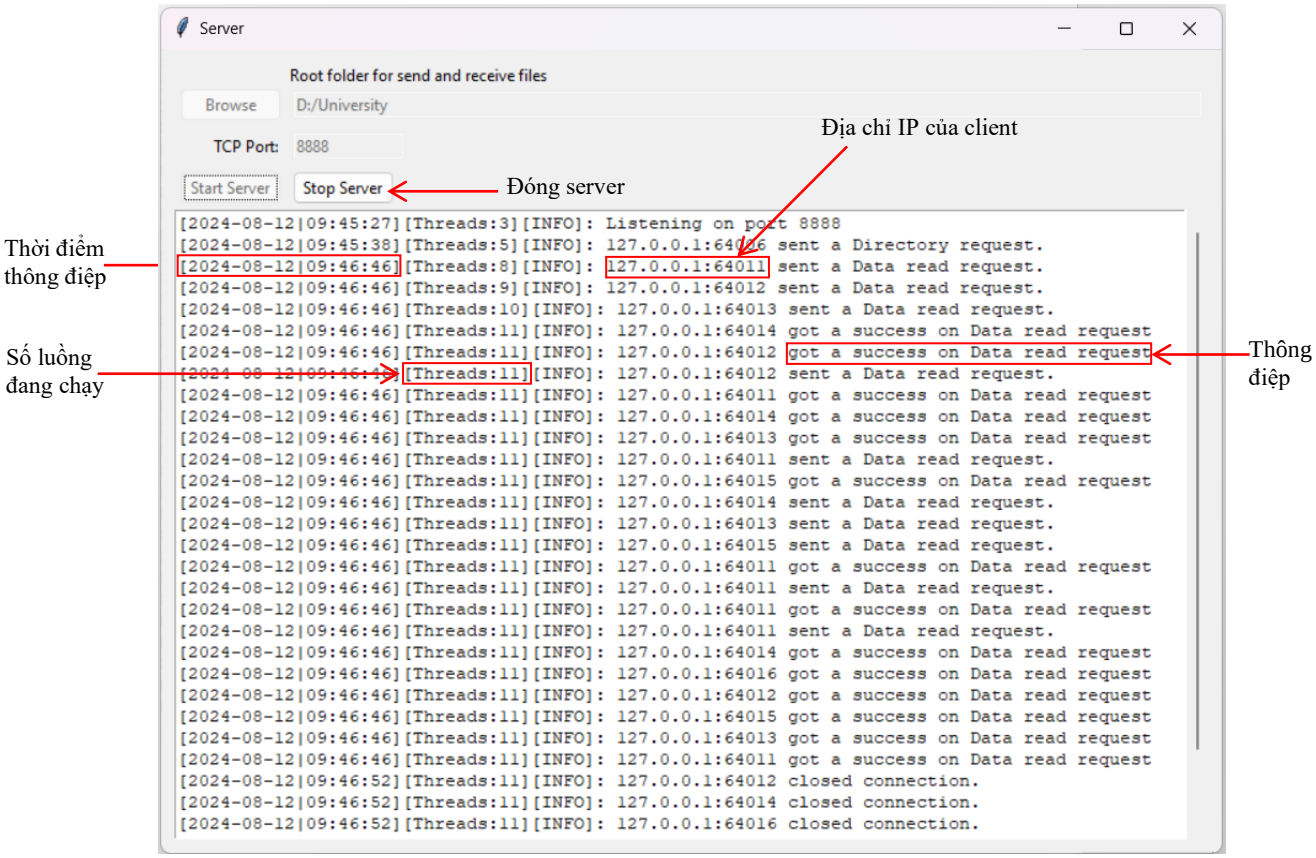
***Hình 1**: Thiết đặt kết nối*



***Hình 2**: Lỗi chưa chọn thư mục*

- ❖ *Lưu ý:* Nếu chọn Start Sever trong khi chưa chọn thư mục tài nguyên thì sẽ có thông báo lỗi như ***Hình 2***

– Theo dõi các kết nối: Sau khi server được khởi động, thông tin về các kết nối sẽ hiển thị bên dưới. Khi cần đóng server, admin nhấn Stop Server



Hình 3: *Thông tin các kết nối*

6. Giao diện client

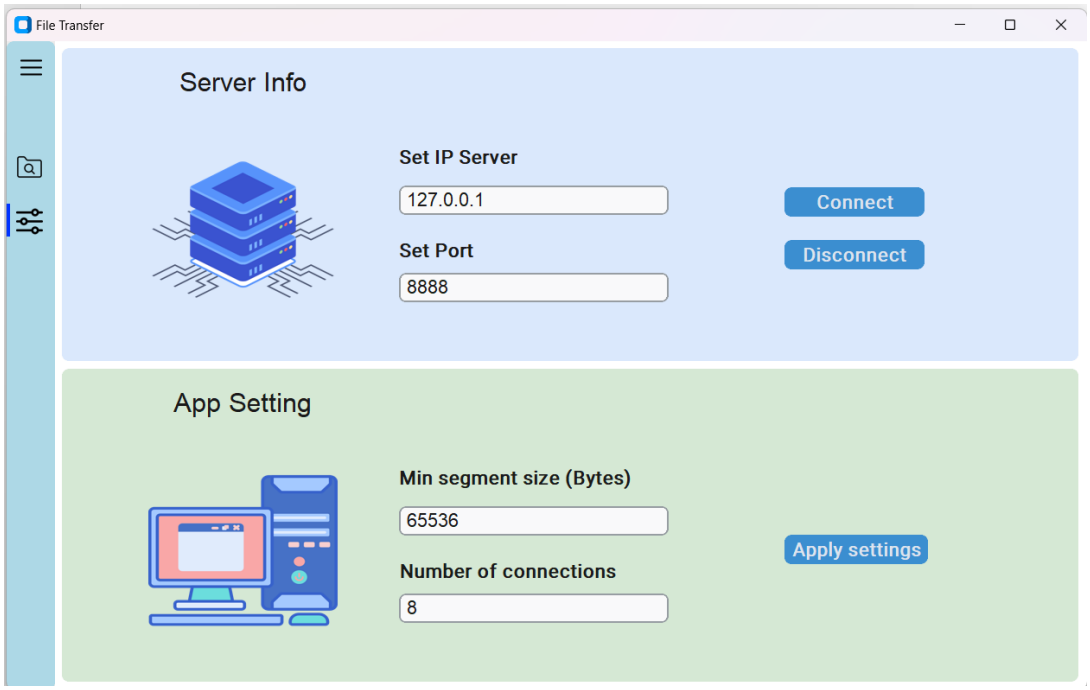
a. Tính năng

- ✓ Upload, download file
- ✓ Upload, download folder
- ✓ Xóa file, folder
- ✓ Tạo folder mới
- ✓ Tìm kiếm file, folder
- ✓ Xem tiến trình upload, download
- ✓ Xem thông tin của server đang kết nối
- ✓ Xem thông tin của thiết bị
- ✓ Tùy chọn server để kết nối
- ✓ Khởi tạo kết nối và ngắt kết nối

b. Mô tả đối tượng

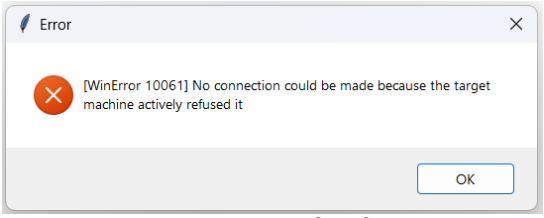
❖ **Trang Setting:** Khi mở ứng dụng lên, trang Setting sẽ hiện ra cho phép người dùng tùy chọn server muốn kết nối bằng cách thay đổi địa chỉ IP và PORT

- *Set IP Server và Set Port:* Thiết đặt sever cần kết nối. Người dùng có thể thay đổi nếu muốn kết nối đến server khác hợp lệ (server mặt định: IP – 127.0.0.1; Port - 8888)
- *Nút Connect và Disconnect:* Kết nối và ngắt kết nối, sau khi chọn server →Nhấn Connect để bắt đầu kết nối
- Người dùng có thể tùy chọn số lượng kết nối và kích thước của segment bằng cách nhập giá trị tương ứng vào ô *Number of connections* và *egment size* sau đó nhấn Apply setting



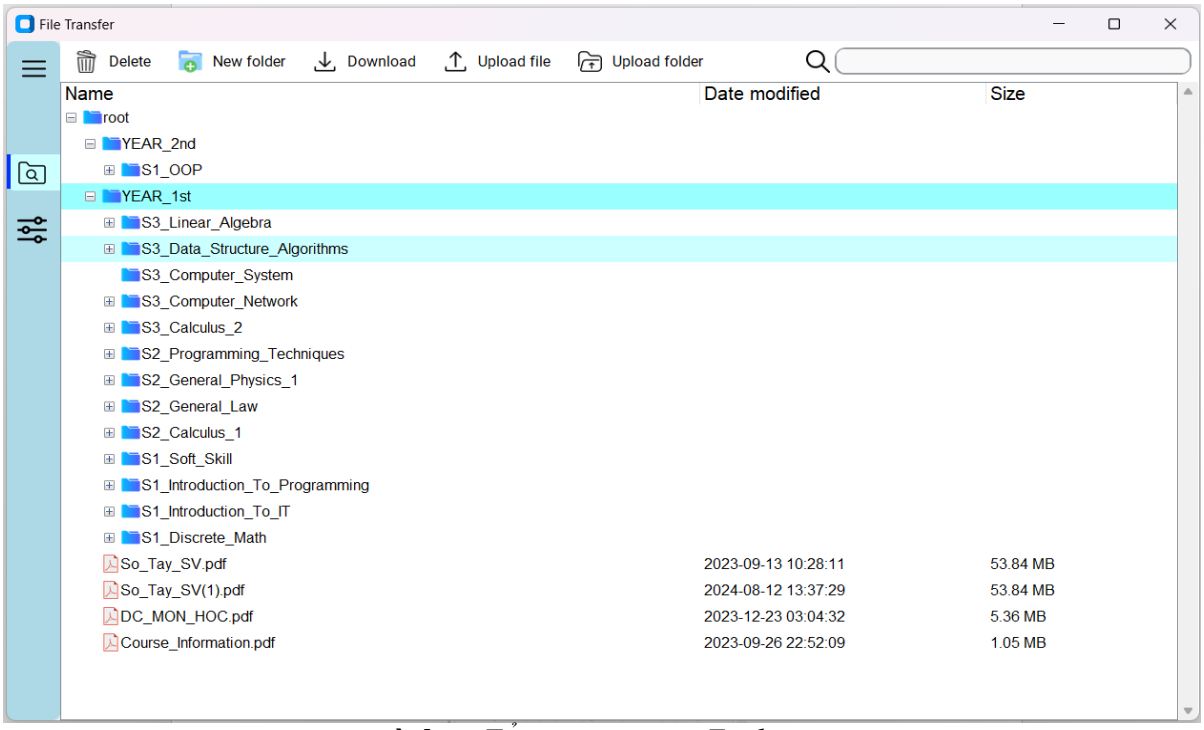
Hình 4: *Tổng quan trang Setting*

❖ ***Lưu ý:*** Nếu ứng dụng kết nối đến một server không hợp lệ hoặc server kết nối đang ngừng hoạt động thì sẽ hiện thông báo như ***Hình 5***



Hình 5: *Lỗi kết nối*

❖ **Trang Explorer:** Sau khi kết nối, người dùng chuyển đến trang Explorer để thao tác với tài nguyên của server

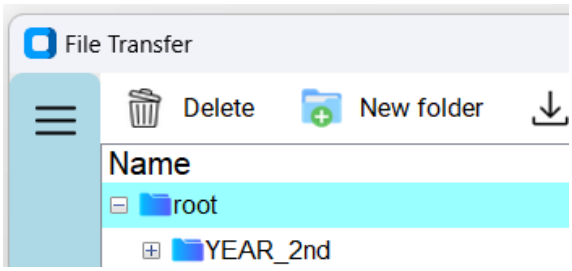


***Hình 6**: Tổng quan trang Explorer*

– *Thanh công cụ gồm có:*

✓ **Delete:** Khi cần xóa dữ liệu trên server:

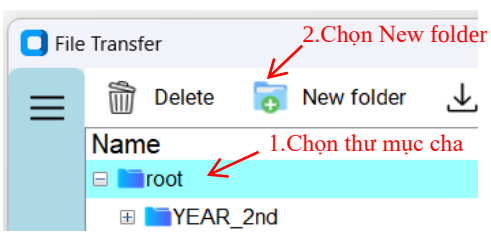
- *Bước 1:* Chọn dữ liệu cần xóa
- *Bước 2:* Chọn vào nút Delete



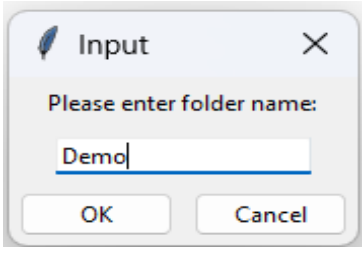
***Hình 7:** Xóa dữ liệu*

✓ ***New folder:*** Nếu người dùng muốn tạo một folder mới trên server

- *Bước 1:* Chọn thư mục cha
- *Bước 2:* Chọn New folder, chương trình sẽ mở hộp thoại Input như ***Hình 9***
- *Bước 3:* Nhập tên thư mục cần tạo trên hộp thoại và nhấn Ok



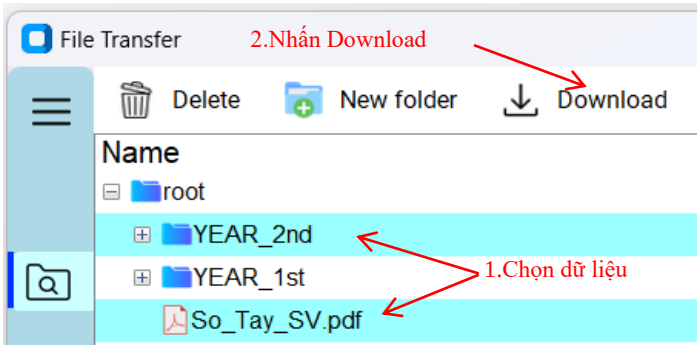
***Hình 8:** Tạo Folder*



***Hình 9:** Hộp thoại Input*

✓ ***Download*:** Khi cần tải file/folder về máy:

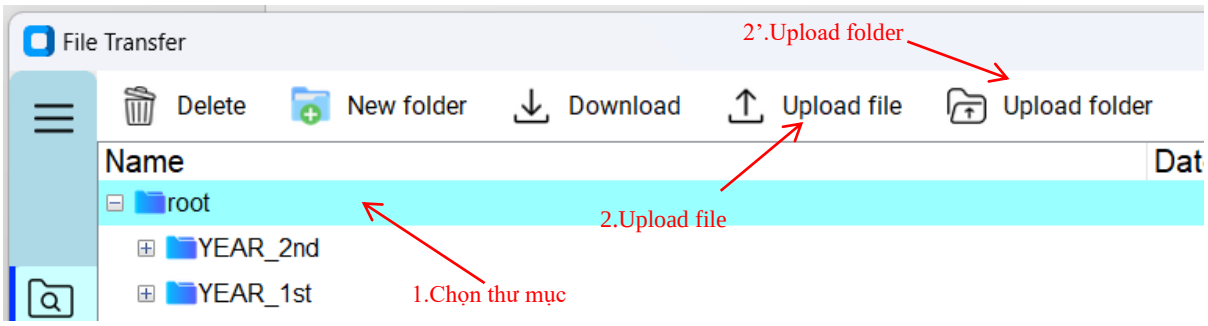
- *Bước 1:* Chọn tất cả file/folder cần tải
- *Bước 2:* Chọn Download, chương trình chuyển đến cửa sổ của file explorer trên thiết bị
- *Bước 3:* Người dùng chọn vị trí lưu dữ liệu và chọn Select folder, file sẽ tải về thiết bị ở vị trí đó. Người dùng có thể xem quá trình tải file trên hộp thoại Process như ***Hình 12***



Hình 10: *Download*

✓ ***Upload file/folder:*** Khi cần Upload dữ liệu

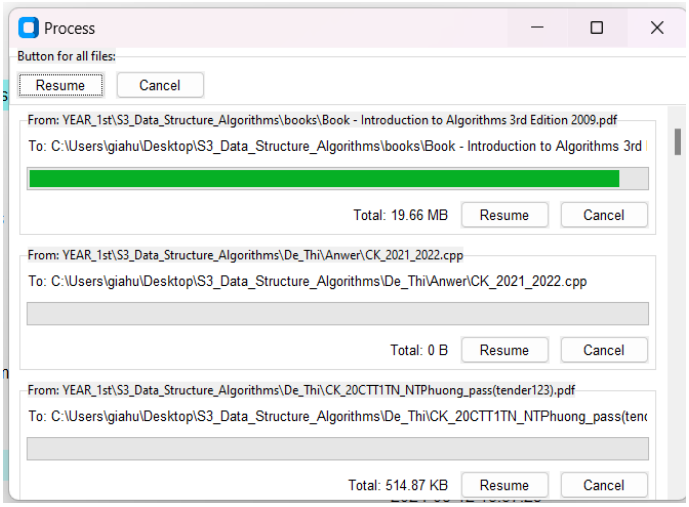
- *Bước 1:* Chọn thư mục trên server cần tải dữ liệu lên
- *Bước 2:* Chọn Upload file để tải lên file hoặc Upload folder để tải lên thư mục, chương trình sẽ chuyển người dùng đến file explorer trên thiết bị để người dùng chọn dữ liệu cần upload
- *Bước 3*: Người dùng chọn dữ liệu cần upload và nhấn Open. Người dùng theo dõi quá trình Upload trên hộp thoại Process như ***Hình 12***



Hình 11: *Upload*

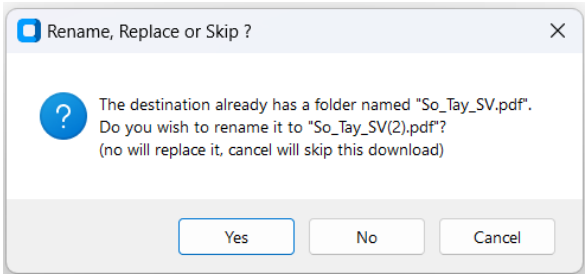
Hộp thoại Proces: Người dùng có thể theo dõi quá trình upload và download dữ liệu

- Chọn Pause để tạm dừng thao tác
- Chọn Resume để tiếp tục thao tác
- Chọn Cancel để hủy thao tác



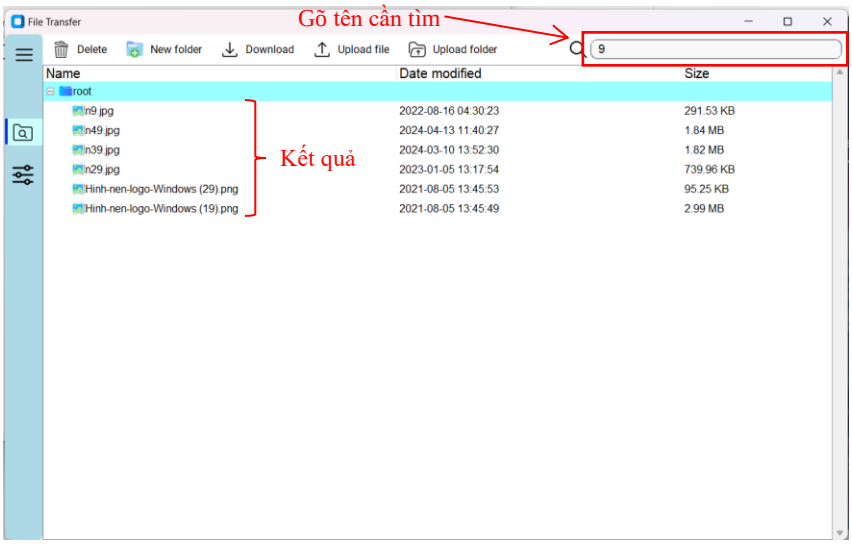
Hình 12. *Hộp thoại Process*

❖ ***Lưu ý:*** Trong trường hợp file Upload/Download trùng tên với các file hiện có ứng dụng sẽ hiển thị cửa sổ thông báo cho phép người dùng chọn: Yes – Đổi tên file; No – Ghi đè lên file cũ; Cancel – Bỏ qua thao tác



Hình 13: *Cửa sổ thông báo trường hợp trùng tên*

✓ ***Thanh tìm kiếm***: Nhập tên file/folder cần tìm kiếm, kết quả tìm kiếm trả về trên thời gian thực

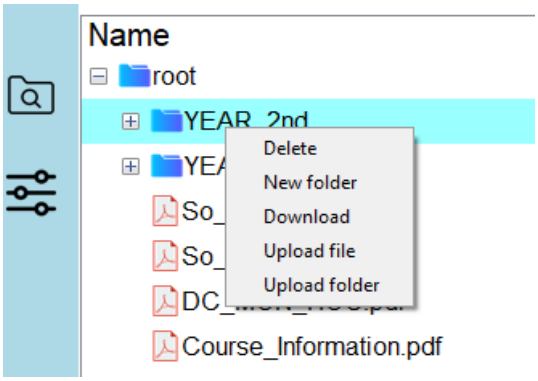


Hình 14: *Chức năng tìm kiếm*

– Vùng dữ liệu của server:

✓ Khi nhấn chuột phải và một đối tượng bất kỳ sẽ hiện thị danh sách các theo tác (upload file, upload folder, download, xóa, tạo thư mục) ***(Hình 15)*** để người dùng có thể thao tác nhanh

✓ Tại đây người dùng có thể xem các thông tin về tài nguyên trên server như: thời gian chỉnh sửa gần nhất, kích thước file, tên file/folder



Hình 15. *Thao tác nhanh*

IV. MÔ TẢ VÀ CHỨC NĂNG CỦA CÁC HÀM

Các mô tả sau đây sẽ là mô tả khái quát chi tiết cách các hàm, lớp và phương thức hoạt động và tương tác như nào qua cách đạt bằng lời. Việc đọc phần này là để hiểu rõ hơn chi tiết về cách hệ thống và phần mềm hoạt động như nào đến mức độ hiểu rõ cấu trúc hệ thống và không cần thiết để nắm khái quát ý tưởng và cách vận hành của phần mềm.

1. Các hàm trong server.py

Hàm *handle_incoming_connections():*

Hàm đảm nhiệm nhiệm vụ chấp nhận các kết nối đến từ Client. Khi Server chưa gọi lệnh dừng Server, hàm sẽ chấp nhận các kết nối đến từ Client và tạo một luồng mới để quản lí các yêu cầu của Client thông qua kết nối ấy. Luồng mới này sẽ chạy trên hàm handle_request() cho đến khi Client ngắt kết nối hoặc khi Server gọi lệnh dừng.

Khi Server gọi lệnh dừng, hàm sẽ ngắt kết nối với tất cả các socket mà Server đang quản lí.

Hàm *handle_request(sock, ip):*

Hàm quản lí việc thực hiện xử lí các yêu cầu đến từ Client. Tham số của hàm là socket và địa chỉ ip của kết nối. Các tham số này được truyền bởi hàm handle_incoming_connections() phía trên. Khi nhận một yêu cầu từ Client, hàm sẽ thực hiện đọc byte đầu tiên của yêu cầu để nhận biết loại yêu cầu của Client và gọi các hàm trong modules.request tưởng ứng để xử lí yêu cầu. Hàm sẽ chạy cho đến khi Client ngắt kết nối hay Server gọi lệnh dừng.

Hàm stop_server():

Hàm gọi dừng Server. Hàm này sẽ được gọi khi nhấn vào nút dừng Server trên màn hình Admin. Khi gọi, hàm sẽ bật sự kiện stop_event lên. Các hàm mà có điều kiện quan sát biến stop_event sẽ được dừng lại.

Hàm start_server():

Hàm khởi tạo Server. Hàm này được gọi khi nhấn nút bắt đầu Server trên màn hình Admin. Khi gọi, hàm sẽ tắt sự kiện stop_event. Sau đó tạo một socket IPv4, loại TCP và lắng nghe port theo lựa chọn của Admin (mặc định là 8888). Tiếp đó thì tạo một luồng để chấp nhận các kết nối đến từ Client

Hàm log_clear():

Hàm giao diện. Chức năng của hàm là xóa nội dung đang hiển thị trên cửa sổ log của màn hình Admin.

Hàm log(message):

Hàm giao diện. Chức năng của hàm là đẩy những xâu được truyền đến hàm lên cửa sổ log.

Hàm browse_directory():

Hàm Giao diện. Chức năng của hàm là cho phép Admin chọn folder trên máy sẽ dùng làm thư mục cho Server.

Hàm validate_input():

Hàm kiểm lỗi. Hàm này sẽ được gọi mỗi khi nhấn nút Start Server để kiểm tra đường dẫn thư mục và số port trước khi thực hiện bắt đầu server. Nếu có lỗi trong việc chọn đường dẫn hay port, một cửa sổ pop up sẽ hiện lên để thông báo với Admin.

2. Các hàm trong client.py

Các hàm trong FrontEnd đảm nhận nhiệm vụ quản lí các hàm và chức năng về mặc giao diện và tương tác với các modules process để quản lí các tiến trình, module message để giao tiếp với server.

Phần này sẽ chỉ chủ yếu mô tả các hàm chức năng do đó sẽ không đề cập nhiều đến các hàm về đặt giao diện.

Hàm *file_progress_ui(file_list, process):*

Hàm giao diện và quản lí, chức năng dùng để tạo cửa sổ hiển thị các thanh trạng thái thể hiện mức độ hoàn thành của các tiến trình xử lí Download/Upload. Cũng như là quản lí các chức năng thao tác với các tiến trình như tạm ngưng hay loại bỏ.

Đầu vào gồm file_list là danh sách file cần phải xử lí và process. Hàm này được gọi bởi các hàm xử lí tiến trình tốn nhiều thời gian như Download hay Upload.

Khi gọi, hàm sẽ gọi xuống module process để tạo một lớp giúp quản lí và thực hiện các tiến trình. Hàm sẽ dựa vào lớp quản lí này để thực hiện việc theo dõi tiến độ xử lí cũng như để gọi phương thức của lớp để thực hiện các thao tác tương tác của người dùng ở trên.

Sau đây là mô tả cho các hàm con của hàm, dùng cho việc định nghĩa hàm cho các nút cũng như là hàm để truyền cho lớp quản lý process

Hàm *update_progress(index, bytes):*

Hàm giao diện, chức năng dùng để cập nhật mức độ hoàn thành của một tiến trình xử lí của một file. Hàm này được dùng để truyền cho lớp quản lý tiến trình nhằm gọi trong lớp khi có một sự kiện xảy ra và cập nhật mức độ hoàn thành của thanh trạng thái.

Hàm *cancel(index):*

Hàm tương tác giao diện, được gọi khi nhấn nút Cancel của một tiến trình. Dùng để hủy bỏ một tiến trình được chọn. Hàm sẽ gọi đến phương thức remove_file của lớp quản lí để thực hiện việc hủy bỏ tiến trình và xóa thanh trạng thái của tiến trình.

Hàm *toggle_pause(index):*

Hàm tương tác giao diện, được gọi khi nhấn nút Pause/Resume. Hàm dùng để bật, tắt việc tạm dừng xử lí file. Khi gọi, hàm sẽ thực hiện chuyển tiếp qua lại giữa 2 trạng thái của tiến trình là tạm dừng hay tiếp tục xử lí.

Hàm *toggle_pause_all():*

Hàm tương tác giao diện, được gọi khi nhấn nút Pause All. Hàm dùng để bật, tắt việc tạm dừng xử lí của tất cả các tiến trình có trong hàng chờ.

Hàm *cancel_all_thread():*

Hàm hỗ trợ xử lí, được gọi bởi hàm cancel_all. Hàm dùng để tạo một luồng riêng dùng để gọi phương thức stop của lớp xử lí và thực hiện xóa các file chưa hoàn thiện trên máy Client.

Hàm *cancel_all():*

Hàm tương tác giao diện, được gọi khi nhấn nút Cancal_All. Hàm dùng để hủy tất cả các tiến trình đang xử lí hoặc đang trong hàng chờ. Khi gọi, hàm sẽ gọi cancel_all_thread để thực hiện việc hủy các tiến trình ở một luồng khác. Lí do cần tạo luồng riêng là vì thao tác xóa file có thể sẽ mất nhiều thời gian để xử lí.

Hàm *add_next_file():*

Hàm xử lí, dùng để thêm tiến trình tiếp theo trong danh sách vào hàng chờ xử lí của lớp quản lí tiến trình. Khi gọi, hàm sẽ thêm file vào hàng chờ xử lí cũng như là tạo thêm 1 thanh thể hiện mức độ hoàn thành của tiến trình.

Hàm *download():*

Hàm chức năng, dùng cho việc Download file hay folder. Hàm được gọi khi người dùng nhấn nút Download. Khi gọi, hàm sẽ thực hiện lấy các tệp tin và thư mục mà người dùng đã chọn và lọc chúng thông qua việc gọi hàm download_siever và tạo thành một danh sách các tệp tin từ chúng. Sau đó thực hiện truyền danh sách này cho file_progress_ui để xử lí cũng như là theo dõi việc download các file.

Hàm *download_siever(item, parent=download_dir):*

Hàm hỗ trợ, dùng để lọc xem có trùng đường dẫn hay không. Khi gọi, hàm sẽ thực hiện di chuyển qua các thư mục theo kiểu đệ quy và thêm các đường dẫn mới vào danh sách.

Trường hợp đường dẫn bị trùng thì sẽ hiện một cửa sổ hỏi người dùng việc nên xử lí thế nào đối với file bị trùng. Sau khi xử lí thì trả về danh sách các đường dẫn đã lọc

Hàm *upload_files():*

Hàm chức năng, dùng cho việc Upload file lên Server. Hàm được gọi khi người dùng nhấn nút Upload File. Khi gọi, hàm sẽ hiện cửa sổ thư mục trên máy để người dùng có thể chọn file để upload. Sau đó kiểm tra xem chúng đã có tồn tại trên Server hay chưa. Nếu có thì sẽ hiện cửa sổ để hỏi ý kiến của người dùng và cho chúng vào danh sách file cần tải. Nếu người dùng chọn “Overwrite” thì sẽ thêm file này vào danh sách file cần xóa.

Sau khi đã có danh sách file cần tải và cần xóa, hàm sẽ thực hiện gửi các yêu cầu xóa file với các file trong danh sách xóa. Rồi gọi file_progress_ui để truyền danh sách này vào để thực hiện việc tải và theo dõi quá trình.

Hàm *upload_folder():*

Hàm chức năng, dùng cho việc Upload thư mục lên Server. Hàm được gọi khi người dùng nhấn nút Upload Folder. Khi gọi, hàm sẽ hiện cửa sổ thư mục trên máy để người dùng có thể chọn folder để upload. Sau đó gọi upload_siever để lấy hết tất cả các file có trong thư mục đã chọn.

Từ danh sách các file yêu cầu Upload, hàm kiểm tra xem chúng đã có tồn tại trên Server hay chưa. Nếu có thì sẽ hiện cửa sổ để hỏi ý kiến của người dùng và cho chúng vào danh sách file cần tải. Nếu người dùng chọn "Overwrite" thì sẽ thêm file này vào danh sách file cần xóa.

Sau khi đã có danh sách file cần tải và cần xóa, hàm sẽ thực hiện gửi các yêu cầu xóa file với các file trong danh sách xóa. Rồi gọi file_progress_ui để truyền danh sách này vào để thực hiện việc tải và theo dõi quá trình.

Hàm *upload_siever(root_dir, current_path):*

Hàm hỗ trợ, dùng để lấy đường dẫn của tất cả các file có trong thư mục và trả về danh sách là các đường dẫn của các file ấy.

Hàm *flatten_directory():*

Hàm xử lí, dùng để tạo danh sách đường dẫn của tất cả các tệp hay thư mục đang có trong cây thư mục. Việc tạo danh sách này là để thuận tiện hơn trong việc truy cập vào đường dẫn của các file trong hàm

Hàm *normalize_directory(dir):*

Hàm xử lí, dùng để chuẩn hóa đường dẫn của tất cả các tệp hay thư mục hiện đang có trong file thư mục. Khi gọi, hàm sẽ đi qua cây thư mục và gọi os.path.normpath đối với từng đường dẫn trong cây.

Hàm *monitor_directory(msgr):*

Hàm xử lí, dùng để tạo một kết nối đến Server và thực hiện theo dõi thư mục của Server theo mô hình Publish – Subscribe để đồng bộ hóa dữ liệu giữa 2 bên. Hàm này được gọi mỗi khi ứng dụng kết nối đến Server.

Hàm *format_bytes(size):*

Hàm hỗ trợ, dùng để chuyển đổi đơn vị từ byte sang đơn vị lớn nhất có thể đổi với số byte cho vào bởi tham số size.

Hàm *update_directory(query=""):*

Hàm giao diện, dùng trong việc cập nhật lại cấu trúc cây thư mục của Server trên ứng dụng. Giá trị đầu vào query dùng cho việc lọc file, làm cho cây thư mục chỉ hiển thị những file có tên trùng với xâu ký tự lưu trong query. Việc lọc này được dùng trong việc tìm kiếm file theo tên mà người dùng nhập vào.

Hàm *validate_input():*

Hàm tương tác giao diện, dùng để kiểm tra và chuẩn hóa số Port mà người dùng nhập vào trước khi thực hiện kết nối.

Hàm *apply_setting():*

Hàm điều chỉnh, dùng để cập nhật các thông số mà người dùng đã chọn cho kích thước gói tin và số luồng kết nối sẽ mở trên hệ thống. Các tham số ấy sẽ được dựa vào khi hệ thống thực hiện việc Upload hay Download

Hàm *connect():*

Hàm thủ tục, dùng để kết nối đến Server. Hàm được gọi sau khi người dùng nhấn nút Connect

Hàm *disconnect():*

Hàm thủ tục, dùng để ngắt kết nối đến Server. Hàm được gọi sau khi người dùng nhấn nút Disconnect

Hàm *delete():*

Hàm chức năng, dùng để thực hiện xóa các file đã chọn trên ứng dụng. Hàm được gọi khi người dùng nhấn nút Delete.

Hàm *ask_string(title, prompt):*

Hàm hỗ trợ, dùng để hiện cửa sổ nhập tên và trả về tên đã nhập đó. Hàm này dùng để hỗ trợ hàm folder trong việc lấy đầu vào từ người dùng. Tham số đầu vào gồm 2 xâu ký tự, title là tham số cho tên của cửa sổ popup sẽ hiện và prompt là tham số cho nội dung sẽ hiển thị trong cửa sổ ấy.

Hàm *folder():*

Hàm chức năng, dùng để tạo folder mới trên Server. Hàm này được gọi khi người dùng nhấn nút New Folder. Khi gọi, hàm thực hiện lấy tên folder bằng cách gọi hàm ask_string và dùng tên đó để tạo folder.

Hàm *highlight_row(event):*

Hàm giao diện, dùng để làm nổi bật các dòng trên cây thư mục khi rê chuột tới.

Hàm *popup_menu(event):*

Hàm chức năng, dùng để hiển thị cửa sổ lựa chọn khi người dùng nhấn chuột phải vào các file trên cây thư mục

Hàm *search_dir(*args):*

Hàm giao diện, hàm dùng để bắt đầu thực hiện việc search trên cây thư mục mỗi khi người dùng gõ chữ vào thanh tìm kiếm. Việc tìm kiếm thư mục trên cây sẽ được thực hiện bằng cách gọi hàm update_directory

Hàm *find_image_file(filename):*

Hàm giao diện, chức năng dùng để tìm icon thích hợp để hiển thị file thông qua việc đọc đuôi file. Đầu vào filename là xâu chứa đường dẫn đến file.

3. Các hàm trong modules.request

Các hàm trong module này chủ yếu là các hàm được gọi từ hàm handle_request(sock, ip) để xử lí các yêu cầu gửi đến Server từ Client. Các hàm này có tham số là Socket và địa chỉ IP của máy Client gửi yêu cầu. Tham số sock nhận socket kết nối đến client, dùng để nhận tiếp những thông tin trong các gói tin yêu cầu và xử lí chúng. Đối với tham số ip thì nó chỉ dùng vào mục đích log và hiển thị trạng thái kết nối trên màn hình Server.

Ngoài ra thì trong đây sẽ có thêm vài hàm để hỗ trợ cho các thao tác của các hàm xử lí yêu cầu trên

Hàm *set_log_method(func):*

Hàm giao diện, thay đổi biến toàn cục log. Biến này được dùng để cài đặt hàm gọi trả về kết quả và hiện lên cửa sổ log bên Server

Hàm *set_server_data_path(path):*

Hàm cài đặt, thay đổi biến toàn cục SERVER_DATA_PATH. Biến này được dùng để lưu đường dẫn thư mục lưu trữ file của Server

Hàm get_path(sock):

Hàm dùng để lấy đường dẫn file từ các socket request, trả về xâu là đường dẫn lấy được hoặc trả về null string "" nếu đường dẫn không tồn tại hoặc đường dẫn là uplink

Hàm *send_error(sock, ip, msg):*

Hàm dùng để gửi gói tin phản hồi xảy ra lỗi đối với các yêu cầu đến từ Client. Tham số msg (message) là thông điệp lỗi muốn gửi đến Client.

Hàm *process_RRQ(sock, ip):*

Hàm dùng để xử lí yêu cầu bắt đầu đọc file của Client. Khi gọi, hàm sẽ thực hiện kiểm tra đường dẫn, đường dẫn có tồn tại hay không, nếu có, báo về kích thước tệp cho client, nếu không có thì báo lỗi.

Đây là hàm cũ và không còn dùng đến nữa do đã có mô hình Polling gửi cấu trúc thư mục của Server kèm thông tin kích thước file.

Hàm *process_WRQ(sock, ip):*

Hàm dùng để xử lí yêu cầu bắt đầu upload file từ Client. Khi gọi, hàm sẽ thực hiện kiểm tra đường dẫn và kiểm tra dung lượng còn lại có đủ để chứa file hay không.

Nếu thỏa các đều kiện trên, server sẽ tạo một file rỗng có đuôi ".uploading" với kích thước bằng với tệp sắp tải lên nhằm dùng cho việc nhận data từ các yêu cầu DWRQ của client. Nếu có file trùng tên thì hàm sẽ thực hiện đánh số file để file không trùng tên nữa Sau đó server gửi opcode báo đã thực hiện thành công.

Nếu không thỏa, lỗi xảy ra và server thực hiện việc gửi opcode báo lỗi kèm theo thông điệp lỗi

Hàm *process_DRRQ(sock, ip):*

Hàm dùng để xử lí yêu cầu đọc 1 đoạn dữ liệu từ Client. Sau khi nhận gói tin, server thực hiện kiểm tra đường dẫn và trả về các byte trong đoạn file dữ liệu mà client yêu cầu. Nếu có lỗi, server sẽ gửi error.

Hàm *process_DWRQ(sock, ip):*

Hàm dùng để xử lí yêu cầu viết 1 đoạn dữ liệu lên file của Server. Khi nhận gói tin, server thực hiện kiểm tra đường dẫn và ghi dữ liệu vào file .uploading. Sau đó gửi về lại cho client opcode báo đã thực hiện thành công yêu cầu.

Nếu có lỗi, server thực hiện gửi opcode lỗi kèm thông điệp báo lỗi.

Hàm *get_unique_filename(filename):*

Hàm hỗ trợ, dùng để đánh số phía sau tên file. Hàm sẽ thực hiện thử đánh số và tăng dần con số đó cho đến khi tên file không còn bị trùng nữa.

Hàm *process_FWRQ(sock, ip):*

Hàm dùng để xử lí yêu cầu hoàn tất quá trình Upload. Khi được gọi, hàm sẽ thực hiện kiểm tra đường dẫn. Sau đó xóa đuôi ".uploading" khỏi file để hoàn tất quá trình upload. Sau đó gửi opcode thành công về cho Client.

Nếu có lỗi, Server gửi opcode báo lỗi kèm thông điệp lỗi cho Client

Hàm *process_DRQ(sock, ip):*

Hàm xử lí yêu cầu xóa file với cấu trúc gói tin gồm chiều dài đường dẫn và đường dẫn đến file cần xóa.

Khi được gọi, hàm sẽ thực hiện việc kiểm tra đường dẫn và sau đó thực hiện việc xóa file/folder khỏi database. Rồi server gửi opcode cho client báo hiệu việc hoàn thành.

Nếu có lỗi, Server gửi opcode báo lỗi kèm thông điệp lỗi cho Client

Hàm *get_directory():*

Hàm hỗ trợ, dùng để lấy cấu trúc thư mục của Server. Hàm dùng để hỗ trợ cho moniter_directory().

Hàm *send_directory(sock, directory):*

Hàm dùng cho việc gửi lại JSON mô tả cấu trúc thư mục lại cho Client

Hàm *set_directory_refresh_rate(seconds):*

Hàm cài đặt, dùng để cập nhật giá trị biến toàn cục directory_refresh_rate được dùng trong việc canh thời gian kiểm tra thư mục định kỳ bên Server.

Hàm *set_stop_event(event):*

Hàm cài đặt, dùng để định dạng sự kiện mà hàm moniter_directory sẽ theo dõi cho việc dừng vòng lặp.

Hàm *monitor_directory():*

Hàm theo dõi thay đổi cấu trúc thư mục của Server. Hàm này sẽ được gọi bởi một luồng riêng bên Server. Khi được gọi, cứ sau mỗi khoảng thời gian theo biến toàn cục directory_refresh_rate thì hàm sẽ thực hiện lấy cấu trúc thư mục mới và so sánh với cấu trúc cũ. Nếu có thay đổi, hàm sẽ cập nhật 2 biến toàn cục directory_snapshot và directory_time.

Hai biến toàn cục directory_snapshot chứa cấu trúc thư mục của Server và directory_time chứa thời gian cập nhật đảm nhận vị trí lưu thông tin hiện tại của thư mục Server

Hàm *process_DTRQ(sock, ip):*

Hàm xử lí yêu cầu đăng ký theo dõi thư mục Server. Khi được gọi, socket yêu cầu lệnh này sẽ không thể thực hiện việc khác cho đến khi dừng việc theo dõi folder, điều này chỉ xảy ra khi Client dừng toàn bộ connection bên phía mình. Do đó socket yêu cầu lệnh này được dành riêng ra chỉ để theo dõi và cập nhật cấu trúc thư mục.

Khi gọi, hàm sẽ thực theo dõi 2 biến toàn cục directory_snapshot và directory_time. Nếu có thay đổi, hàm sẽ cập nhật theo 2 biến đó và thực hiện việc gửi dữ liệu chứa thông tin cấu trúc thư mục mới tới Client.

Hàm *process_FRQ(sock, ip):*

Hàm xử lí yêu cầu tạo folder từ client. Khi nhận yêu cầu này, Server thực hiện kiểm tra đường dẫn rồi tạo folder. Nếu thành công, gửi opcode báo hiệu thành công cho client. Ngược lại gửi opcode báo lỗi kèm thông điệp lỗi

4. Các hàm trong modules.message

Modules.message có chức năng cung cấp giao diện cho client.py có thể dễ dàng giao tiếp với Server hơn. Thông qua những thủ tục do modules.message cung cấp sẵn. Đối với chức năng của các hàm trong modules, chúng sẽ giúp đóng gói thông điệp yêu cầu từ Client và gửi đi cho Server. Đồng thời, chúng cũng giúp nhận phản hồi từ Server và trả về những thông báo phù hợp cho client.py xử lí

Các biến toàn cục

- Biến ***messengers***: là một danh sách (list) bao gồm các kết nối đã thiết lập với Server
- Biến ***lock***: là một cơ chế khóa, giúp kiểm soát việc truy cập vào biến messengers bởi các luồng khác nhau

Hàm *disconnect_all():*

Có chức năng đóng và ngắt hết tất cả mọi kết nối trong biến toàn cục messengers. Dùng khi Client thực hiện việc ngắt kết nối với Server hay đóng app.

Class *MessengerError (Exception):*

Là một lớp được kế thừa từ Exception, giúp định dạng các lỗi có thể xảy ra khi thực hiện việc giao tiếp với Server.

Class *Messenger:*

Là một lớp giúp khởi tạo và quản lí các socket. Có chức năng cung cấp giao diện đến các hàm và thủ tục giúp đóng gói và gửi các gói tin yêu cầu đến Server.

Các phương thức sau đây đều sẽ là phương thức của lớp Messenger

Phương thức khởi tạo *__init__(self, host, port):*

Các thông số nhận vào là host là địa chỉ IP của Server muốn kết nối đến và port là số port của Server. Khi khởi tạo một đối tượng của lớp Messenger, lớp sẽ tạo một socket TCP IPv4 và kết nối đến server có địa chỉ ip và số port lấy từ các tham số. Sau đó, hàm sẽ thực hiện thêm vào danh sách toàn cục messengers một tham chiếu đến chính lớp đối tượng đang khởi tạo

Phương thức báo lỗi *_raise_err(self):*

Phương thức dùng để báo lỗi đến client.py. Được gọi khi có lỗi xảy ra trong các giao thức khác của lớp Messenger. Khi gọi, phương thức sẽ báo về lỗi từ lớp MessengerError

Phương thức *send_RRQ(self, file_path):*

Dùng để đóng gói và truyền yêu cầu bắt đầu tiến trình Download đến Server. Tham số nhận vào là file_path, là đường dẫn đến file muốn đọc trên Server. Giá trị trả về là kích thước của file muốn đọc hoặc thông báo lỗi nếu như có lỗi trên Server.

Đây là một phương thức cũ không còn dùng đến nữa

Phương thức *send_WRQ(self, file_path, size):*

Phương thức dùng cho việc đóng gói và gửi yêu cầu bắt đầu tiến trình Upload đến Server. Tham số nhận vào gồm xâu đường dẫn đến đích đến của file trên Server và kích thước file. Phương thức không có giá trị trả về và sẽ báo lỗi nếu như Server gửi về phản hồi thông báo lỗi.

Phương thức *send_DRRQ(self, file_path, offset, length, local_file_path):*

Phương thức dùng cho việc đóng gói và gửi yêu cầu Download một đoạn tệp lên Server. Tham số nhận vào bao gồm:

- file_path: là đường dẫn đến tệp muốn đọc trên Server
- offset là vị trí bắt đầu lấy dữ liệu

- length là độ dài đoạn dữ liệu sẽ lấy
- local_file_path là đường dẫn đến file dùng để nhận dữ liệu về trên máy Client.

Phương thức không có giá trị trả về và sẽ báo lỗi nếu như nhận phản hồi ERROR từ Server

Phương thức *send_DWRQ(self, file_path, offset, length, local_file_path):*

Phương thức dùng cho việc đóng gói và gửi yêu cầu Upload một đoạn tệp lên Server. Tham số nhận vào bao gồm:

- file_path: là đường dẫn đến vị trí đích đến của file trên Server
- offset: là vị trí bắt đầu ghi
- length: là độ dài của đoạn tệp tải lên
- local_file_path: là đường dẫn đến file cần Upload trên máy Client.

Phương thức không có giá trị trả về và sẽ báo lỗi nếu như nhận phản hồi ERROR từ Server

Phương thức *send_FWRQ(self, file_path):*

Phương thức dùng cho việc đóng gói và gửi yêu cầu hoàn thành tiến trình Upload lên Server. Tham số nhận vào là file_path là đường dẫn đến file đã hoàn tất việc Upload lên Server. Phương thức sẽ báo lỗi nếu phản hồi từ Server là ERROR

Phương thức *send_DRQ(self, file_path):*

Phương thức dùng cho việc đóng gói và gửi yêu cầu xóa tệp hoặc thư mục trên Server. Tham số nhận vào là file_path là đường dẫn đến tệp hay thư mục muốn xóa trên Server. Phương thức sẽ báo lỗi nếu phản hồi từ Server là ERROR

Phương thức *sub_DTRQ(self):*

Phương thức dùng cho việc gửi yêu cầu đăng ký theo dõi thư mục đến Server. Phương thức này để dùng để biến Messenger gửi và nhận phản hồi bình thường trở thành Messenger dùng cho việc theo dõi và nhận các thông tin cập nhật cấu trúc Server.

Phương thức *recv_DTRQ(self):*

Phương thức dùng để nhận các gói tin cập nhật thông tin thư mục gửi về từ Server. Phương thức này được dùng trong một vòng lặp chạy trong 1 luồng riêng. Kết quả trả về của hàm là cấu trúc thư mục trả về từ Server.

Phương thức *send_FRQ(self, path):*

Phương thức dùng để đóng gói và gửi gói tin tạo folder mới lên Server. Tham số nhận vào path là đường dẫn thư mục mới muốn tạo trên Server. Phương thức không có giá trị trả về và sẽ báo lỗi nếu nhận phản hồi ERROR từ Server.

Phương thức *shutdown(self):*

Phương thức dùng cho việc đóng việc truyền dữ liệu trên socket mà lớp Messenger quản lý. Dữ liệu có thể vẫn được nhận do các thông điệp chưa đến đích vẫn còn tồn động trên đường truyền.

Phương thức *close(self):*

Phương thức dùng cho việc ngắt kết nối và phá hủy socket mà lớp Messenger đang quản lí. Đồng thời phương thức này cũng sẽ xóa luôn cả tham chiếu đến nó trong danh sách toàn cục messengers.

Phương thức *__enter__(self) và __exit__(self)*

Là cặp phương thức dùng cho việc tự động ngắt kết nối và thực hiện đóng socket. Đây là phương thức tiện ích dùng cho việc lập trình trong python.

5. Các hàm trong modules.process

Modules.process có chức năng quản lí và vận hành các tiến trình Upload và Download của Client. Nó có nhiệm vụ chia cắt các tiến trình ra thành nhiều công việc nhỏ và xếp chúng vào hàng đợi. Sau đó tạo ra nhiều luồng nhỏ, mỗi luồng truy cập đến 1 kết nối để lần lượt xử lí các công việc trong hàng đợi. Modules này được tạo ra nhằm làm giao diện gọi các lệnh xử lí tiến trình Download và Upload tệp hay thư mục trở nên dễ dàng hơn.

Hàm *create_file(path, size):*

Là một hàm hỗ trợ trong modules này, có nhiệm vụ tạo một file rỗng nhằm nhận dữ liệu trả về từ Server. Tham số đầu vào gồm path là đường dẫn vị trí nơi tạo file và size là kích thước của file muốn tạo.

Class *DownloadManager:*

Là một lớp dùng cho việc quản lí các tiến trình Download file. Thành phần của nó sẽ bao gồm các luồng worker xử lí công việc, hàng đợi các công việc đã được chia nhỏ và danh sách các file của các tiến trình Download.

Các phương thức sau sẽ thuộc về lớp DownloadManager cho đến khi có định nghĩa về lớp khác.

Phương thức *__init__ (self, host, port, num_threads, min_seg, update=print):*

Là phương thức khởi tạo cho lớp DownloadManager. Tham số đầu vào bao gồm:

- host: là địa chỉ IPv4 của Server
- port: là số port của Server
- num_threads: số luồng worker mà lớp sẽ tạo để xử lí công việc
- min_seg: số bytes 1 segment
- update: là hàm dùng để thông báo hay cập nhật tiến trình thực hiện của các tiến trình

Khi được gọi, hàm sẽ khởi tạo số lượng các kết nối đến Server bằng với số luồng num_threads để xử lí các tiến trình Download có trong hàng đợi.

Phương thức *worker(self):*

Là phương thức dùng để các chạy các luồng worker, có nhiệm vụ xử lí các tiến trình đang có trong hàng đợi. Đối với mỗi tiến trình, phương thức sẽ thực hiện kiểm tra xem trạng thái của file có phải đang "removed" (khi nhấn nút cancel) hay "paused" (khi nhấn nút pause) không, nếu không thì sẽ thực hiện việc Download đoạn dữ liệu của tiến trình đang chạy. Khi đã chạy xong sẽ thực hiện việc kiểm tra xem liệu số byte down về có bằng kích thước của file không, nếu đã bằng thì sẽ thực hiện xóa đuôi ".downloading" và kết thúc quá trình tải file.

Đối với trường hợp file trong trạng thái "removed", phương thức sẽ bỏ qua tiến trình đang chạy và chuyển đến chạy tiến trình tiếp theo.

Đối với trường hợp file trong trạng thái "paused", phương thức sẽ đẩy tiến trình vào danh sách tiến trình chờ của file và thực hiện bỏ qua tiến trình.

Phương thức *add_file(self, server_path, size, client_path):*

Phương thức dùng để thêm một tiến trình vào hàng chờ. Tham số đầu vào của phương thức bao gồm:

- server_path: là đường dẫn của file cần tải trên server
- size: kích thước file
- client_path: đường dẫn file vị trí sẽ tải xuống máy Client

Khi gọi, phương thức sẽ thực hiện tạo một file rỗng có đuôi ".downloading" trên máy Client tại vị trí theo tham số đường dẫn client_path. Sau đó thực hiện tạo nhiều tiến trình nhỏ hơn và thêm vào hàng chờ xử lí. Mỗi tiến trình nhỏ sẽ đảm nhiệm việc Download một đoạn nhỏ của file.

Phương thức *add_file (self, file_list):*

Phương thức dùng để thêm một danh sách các file cần tải vào hàng chờ. Tham số đầu vào là file_list là danh sách các file cần Download. Khi gọi, phương thức đơn giản gọi add_file(self, server_path, size, client_path) cho mỗi file trong danh sách

Phương thức *start(self):*

Phương thức dùng để bắt đầu các luồng xử lí worker của lớp DownloadManager

Phương thức *pause_file(self, file_id):*

Phương thức dùng để tạm dừng quá trình Download một file. Tham số đầu vào là file_id, là số thứ tự đánh riêng cho mỗi tiến trình Download khi chạy ứng dụng. Khi gọi, phương thức sẽ bật cờ "paused" của file lên.

Phương thức *resume_file(self, file_id):*

Phương thức dùng để tiếp quá trình Download một file. Tham số đầu vào file_id là số thứ tự riêng của quá trình khi chạy ứng dụng. Khi gọi, hàm thực hiện tắt cờ hiệu "paused" ra khỏi quá trình và thực hiện đẩy các tiến trình nhỏ của quá trình vào lại hàng chờ xử lí.

Phương thức *remove_file(self, file_id):*

Phương thức dùng để hủy bỏ quá trình Download một file. Tham số đầu vào là số thứ tự riêng của quá trình. Khi gọi, phương thức bật cờ hiệu "removed" của quá trình lên

Phương thức *wait_for_completion(self):*

Phương thức dùng để chờ cho đến khi tất cả các tiến trình trong hàng chờ xử lí xong. Dùng cho việc khóa màn hình ứng dụng lại cho đến khi các tiến trình trong hàng chờ đã xử lí xong hết trước khi có thể thêm tiến trình mới vào xử lí.

Phương thức *stop(self):*

Phương thức dùng cho việc dừng toàn bộ các quá trình Download đang nằm trong hàng chờ xử lí. Khi gọi, phương thức sẽ gọi remove_file trên tất cả các file có trong danh sách các file cần Downloaf của lớp DownloadManager. Sau đó gọi wait_for_completion() để chờ cho các luồng worker thực hiện xong các công việc của mình. Sau đó phương thức sẽ xóa các file đang trong trạng thái, là các file có đuôi ".downloading"

Class *UploadManager:*

Là một lớp dùng cho việc quản lí các tiến trình Upload file. Thành phần của nó sẽ bao gồm các luồng worker xử lí công việc, hàng đợi các công việc đã được chia nhỏ và danh sách các file của các tiến trình Upload. Cấu trúc và các phương thức của nó sẽ có phần giống DownloadManager.

Các phương thức của lớp UploadManager cũng có các thuộc tính và cách hoạt động tương tự như của DownloadManager nên chúng em lựa chọn không mô tả lại.

Khác biệt duy nhất trong quá trình xử lí quá trình Upload file của lớp này là việc khi đã Upload xong một file, lớp sẽ thực hiện gửi thông điệp FWRQ đến Server để thông báo việc hoàn tất quá trình Upload file.

6. Các hàm trong modules.shared

Modules.shared có chức năng định nghĩa cho các tham số và biến toàn cục của toàn hệ thống cũng như việc định nghĩa các hàm sẽ dùng chung nhiều trong phần mềm. Ngoài ra nó còn chứa định nghĩa cho các tham số mặc định của hệ thống, ví dụ điển hình là host và port mặc định của server (local_host và port 8888). Lí do có nó là để code có phần dễ đọc và tự định nghĩa hơn.

Các constant bao gồm trong module

- Module sẽ bao gồm định nghĩa đánh số cho các opcode yêu cầu và phản hồi.

- Đối với opcode của các request (yêu cầu từ Client) sẽ đánh số tăng dần từ 0 đến 7 theo thứ tự sau: RRQ, WRQ, DRRQ, DWRQ, FWRQ, DRQ, DTRQ, FRQ
- Đối với opcode phản hồi từ Server sẽ đánh số 0 và 1 lần lượt cho các phản hồi ERROR và SUCCESS
- Ngoài ra modules cũng chứa các xâu định nghĩa cho các request phía trên, chúng được chứa trong một cấu trúc từ điển.

Hàm *recv_data(sock, file_path, offset, length):*

Hàm có chức năng nhận dữ liệu của một đoạn file trên socket và ghi vào đường dẫn cho bởi tham số. Các tham số của hàm gồm:

- sock: socket sẽ nhận data
- file_path: đường dẫn đến file sẽ nhận dữ liệu trên máy
- offset: vị trí bắt đầu ghi dữ liệu trong file
- length: chiều dài dữ liệu sẽ nhận từ socket

Hàm này được dùng bởi các hàm và phương thức khác nhằm đơn giản hóa việc nhận dữ liệu từ socket về file.

Hàm *recv_all(sock, n):*

Hàm có chức năng nhận dữ liệu truyền đến từ socket. Tham số đầu vào n là số lượng byte mà hàm này sẽ nhận. Khi gọi, hàm sẽ nhận n byte cho đến khi nhận đủ số byte. Khi hàm này không nhận đủ byte, nó sẽ báo lỗi. Khi đã nhận đủ byte, nó sẽ trả về dữ liệu byte mà hàm đã nhận được.

Hàm *get_unique_filename(filename, folder):*

Hàm hỗ trợ, dùng để sinh tên file không bị trùng tên trong cùng folder. Hàm sẽ thực hiện thử đánh số vào phía sau tên file và tăng dần con số đó cho đến khi tên file không còn bị trùng nữa. Hàm dùng trong trường hợp người dùng tiếp tục tải file lặp hay bị trùng tên.

V. TÀI LIỆU THAM KHẢO

Lập trình socket python:
https://docs.python.org/3/howto/sockets.html#socket-howto
https://docs.python.org/3/library/socket.html

Lập trình đa luồng python:
https://docs.python.org/3/library/threading.html

Lập trình hệ điều hành, cho việc quản lí file:
https://docs.python.org/3/library/os.html

Ý tưởng lấy theo các giao thức, mô hình có sẵn
https://en.wikipedia.org/wiki/HTTP
https://datatracker.ietf.org/doc/html/rfc1350
https://datatracker.ietf.org/doc/html/rfc959
https://stackoverflow.com/questions/93642/how-do-download-accelerators-work

Mô hình Polling
https://en.m.wikipedia.org/wiki/Push_technology

Lập trình tkinter:
https://docs.python.org/3/library/tkinter.html
https://pyinmyeye.blogspot.com/