Báo cáo đồ án mạng máy tính

# Khái quát chung về đồ án

Xài ngôn ngữ python phiên bản mấy, xài thư viện gì với mục đích gì

Sử dụng thư viện ngoài nào

# Mô tả giao diện

Cách dùng, khởi động server, client

Giao diện server

Chức năng, chuyện j sẽ xảy ra khi nhấn các nút, màn hình hiển thị j?

Giao diện client

3 trang, brower, thanh trạng thái, trang setting

Bla bla…

**File word báo cáo, mô tả giao diện, các hàm và thủ tục trong phần mềm**

# Mô tả hệ thống

## Sơ lược chung về hệ thống

Phần mềm sử dụng mô hình Client – Server để thực hiện việc giao tiếp và chia sẻ dữ liệu. Giao thức sử dụng ở tầng transport là TCP theo yêu cầu của đồ án. Để thực hiện việc cắt nhỏ file ra thành nhiều đoạn và download mỗi đoạn bằng những luồng riêng biệt, chúng em đã tham khảo cách thức vận hành của giao thức HTTP đến phiên bản 1.1. Đó là mở nhiều kết nối song song Client và Server, mỗi luồng sẽ đảm nhận gửi dữ liệu qua một kết nối.

Lí do chúng em chọn HTTP 1.1 là bởi những hiệu suất và tính năng mà nó mang lại, đồng thời cũng vì giao thức này tương đối dễ cài đặt so với những phiên bản sau của HTTP yêu cầu phải thực hiện ghép kênh chuyền, được chúng em xem là quá phức tạp để thực hiện.

### Những tính năng từ các giao thức mà chúng em đã lấy ý tưởng là:

**HTTP 1.0:**

* Header ở đầu những gói tin yêu cầu từ Client mô tả thông điệp yêu cầu, kích thước gói tin động. (HTTP 1.0)
* Header cho gói tin phản hồi yêu cầu từ Client mô tả kết quả sử lí yêu cầu cũng như dữ liệu theo yêu cầu.

**HTTP 1.1:**

* Byte serving: cho phép Client yêu cầu 1 đoạn dữ liệu từ một file lớn. Yêu cầu có thể là lấy 1 đoạn dữ liệu từ server hoặc tải lên 1 đoạn dữ liệu từ Client
* Duy trì kết nối cho đến khi Client yêu cầu đóng. Việc này giúp giảm thời gian phải thiết lập lại mỗi kết nối sau khi hoàn thành 1 yêu cầu.

**TFTP:**

* Opcode: dùng để mô tả yêu cầu từ Client. Dùng 1 byte để mô tả thay vì 1 chuỗi ký tự

**Polling Model**

* Direcroty Watcher: Server theo dõi cấu trúc thư mục chứa data và gửi lại cho các Client khi có sự thay đổi. Điều này giúp đồng bộ dữ liệu giữa Client và Server.

## Cách thức giao tiếp giữa Client và Server

A diagram of a server

Description automatically generated

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

Đối với Server thì sẽ có modules là request.py để giúp xử lí các yêu cầu gửi đến từ client. Việc xử lí các yêu cầu có bao gồm việc kiểm lỗi gửi lại thông điệp phản hồi cho Client.

## Cách thức Server đồng bộ dữ liệu với Client

A screenshot of a computer

Description automatically generated

Như đã nêu ở trên, hệ thống chúng em có bao gồm mô hình Polling để đồng bộ hóa dữ liệu. Server sẽ định kỳ kiểm tra thư mục của mình và gửi lại cấu trúc thư mục cho Client mỗi khi có sự thay đổi.

Cấu trúc gói tin cập nhật thư mục từ server là một JSON object chứa các thông tin như tên file, kích thước, ngày cập nhật, đường dẫn file và các quan hệ thể hiện các file nằm trong các thư mục con.

## Cách hệ thống thực hiện việc Upload

A screenshot of a phone

Description automatically generated

Để thực hiện việc upload, sẽ có 3 loại yêu cầu mà Client cần phải gọi với server trong cả quá trình.

Đầu tiên, đó là yêu cầu WRQ (write request), là yêu cầu đầu tiên trong việc thực hiện upload. Yêu cầu này sẽ bao gồm đường dẫn đích đến của file, cũng như là kích thước tệp sẽ tải lên. Do Client có cấu trúc thư mục của Server nên đường dẫn đích đến sẽ là đường dẫn của file nếu nó nằm trên Server. Việc cung cấp trước cho Server kích thước tệp tải lên là để server tạo trước 1 vùng nhớ chứa file trống có đuôi “.uploading”. File trống này là để chứa những dữ liệu gửi đến trong những yêu cầu sau.

Sau khi đã thực hiện WRQ, Client sẽ gửi đồng thời 1 loạt các DWRQ (data write request) qua nhiều kết nối khác nhau đã thiết lập trước đến server. Mỗi yêu cầu sẽ chứa thông tin bao gồm đường dẫn của tệp trên server, vị trí bắt đầu của dữ liệu, chiều dài gói tin và cuối cùng là đoạn dữ liệu trong tệp muốn upload.

Cuối cùng, Client cần phải gửi yêu cầu FWRQ (finish write request) để hoàn thành việc upload file. Việc này sẽ xóa đuôi .uploading của file và tạo file hoàn chỉnh, chỉnh thúc việc truyền dữ liệu đến Server.

## Cách hệ thống thực hiện Download

A screenshot of a chat

Description automatically generated

Tương tự như việc upload, Client cũng sẽ gửi đồng loạt nhiều thông điệp đến Server qua nhiều kết nối. Mỗi thông điệp sẽ yêu cầu một đoạn dữ liệu từ tệp trên server. Tuy nhiên, khác với upload, Client sẽ lấy kích thước tệp từ JSON mô tả cấu trúc thư mục của server. JSON này đã được cập nhật qua việc polling mỗi khi có thay đổi. Do đó Download sẽ không có yêu cầu gửi lấy kích thước tệp từ server.

# Cấu trúc gói tin của các yêu cầu từ Client và phản hồi từ Server

## Cấu trúc gói tin phản hồi báo lỗi

A screenshot of a computer

Description automatically generated

Đối với tất cả các yêu cầu xử lí đến từ Client, Server sẽ có phản hồi. Với từng yêu cầu sẽ có cấu trúc gói tin phản hồi thực hiện yêu cầu thành công khác nhau. Nhưng đối với phản hồi báo lỗi thì chúng sẽ có cấu trúc như trên.

* Opcode: ERROR, thông báo lỗi
* Length: mô tả chiều dài của thông điệp báo lỗi
* Error message: thông điệp báo lỗi gửi đến từ server

## Cấu trúc các gói tin yêu cầu upload và phản hồi

### Write Request

A screenshot of a computer

Description automatically generated

Với gói tin yêu cầu:

* Opcode: WRQ
* Length: mô tả chiều dài của xâu chứa đường dẫn
* File path: Chứa đường dẫn đích đến của yêu cầu upload
* File size: kích thước tệp tin sẽ tải lên

Với gói tin phản hồi thành công:

* Opcode: SUCCESS

### Data Write Request

A screenshot of a computer

Description automatically generated

Với gói tin yêu cầu:

* Opcode: DWRQ
* Length: mô tả chiều dài của xâu chứa đường dẫn
* File path: Chứa đường dẫn của file trên server
* Data offset: mô tả vị trí bắt đầu của đoạn data trong file
* Data length: mô tả chiểu dài của trường Data tiếp theo
* Data: Dữ liệu của 1 đoạn file được upload

Với gói tin phản hồi thành công:

* Opcode: SUCCESS

### Finish Write Request

A screenshot of a computer

Description automatically generated

Với gói tin yêu cầu:

* Opcode: FWRQ
* Length: mô tả chiều dài của xâu chứa đường dẫn
* File path: Chứa đường dẫn của tệp đã hoàn thành upload trên server

Với gói tin phản hồi thành công:

* Opcode: SUCCESS

## Cấu trúc gói tin yêu cầu Download và phản hồi

A screenshot of a computer

Description automatically generated

Với gói tin yêu cầu:

* Opcode: DRRQ
* Length: mô tả chiều dài của xâu chứa đường dẫn
* File path: Chứa đường dẫn của file trên server
* Data offset: mô tả vị trí bắt đầu của đoạn data trong file
* Data length: mô tả chiểu dài của trường Data tiếp theo

Với gói tin phản hồi thành công:

* Opcode: SUCCESS
* Data: Đoạn dữ liệu từ tệp mà Client yêu cầu

## Cấu trúc gói tin yêu cầu xóa tệp, thư mục và phản hồi

A screenshot of a black screen

Description automatically generated

Với gói tin yêu cầu:

* Opcode: DRQ (Delete Request)
* Length: mô tả chiều dài của xâu chứa đường dẫn
* File/Folder path: Chứa đường dẫn của tệp hoặc thư mục muốn xóa trên Server

Với gói tin phản hồi thành công:

* Opcode: SUCCESS

## Cấu trúc gói tin yêu cầu tạo thư mục và phản hồi

A screenshot of a black screen

Description automatically generated

Với gói tin yêu cầu:

* Opcode: FRQ (Folder Request)
* Length: mô tả chiều dài của xâu chứa đường dẫn
* File/Folder path: Chứa đường dẫn của thư mục muốn tạo trên Server

Với gói tin phản hồi thành công:

* Opcode: SUCCESS

## Cấu trúc gói tin yêu cầu đăng ký theo dõi thư mục và phản hồi

A black and white screen with white text

Description automatically generated

Với gói tin yêu cầu:

* Opcode: DTRQ (Directory Request)

Với gói tin phản hồi thành công(được định kỳ gửi đi nếu có thay đổi bên Server):

* Opcode: SUCCESS
* Data length: kích thước của JSON mô tả thư mục server
* JSON data: dữ liệu JSON

# 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.

## 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.

## 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.

2 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

## 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 hổi 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 hổi 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.

## 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, 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
* 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 thức 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.

## 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.

# 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 luồn 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](https://en.m.wikipedia.org/wiki/Push_technology?fbclid=IwZXh0bgNhZW0CMTEAAR1s1PbMeWZNpoHRz-KkUBmM-jsgL1i_shgRISUzk09J1e4UHe3Bh8sezFM_aem_MVs9bau20MZ3tlW_wY0ehw)

Lập trình tkinter:

<https://docs.python.org/3/library/tkinter.html>

<https://pyinmyeye.blogspot.com/>

Các trang tham khảo cho các vấn đề chung:

<https://chatgpt.com/>

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>