

Trường Đại học Khoa học tự nhiên - ĐHQG HCM



fit@hcmus

Khoa Công nghệ thông tin

BÁO CÁO ĐỒ ÁN

Mạng máy tính - Lập trình socket

Xây dựng chương trình truyền tải file
sử dụng TCP và UDP ở tầng Transport

Ngày 19 tháng 12 năm 2024

Lớp 23CTT3

NHÓM 1A

23120233 - Nguyễn Lê Hữu Điền

23120262 - Tống Dương Thái Hoà

23120264 - Nguyễn Phúc Hoàng

Giảng viên hướng dẫn

Ths. Huỳnh Thụy Bảo Trân

Ths. Chung Thụy Linh

Mục lục

1	Giới thiệu	2
1.1	Thông tin thành viên	3
1.2	Phân chia công việc	4
2	Sử dụng TCP tại tầng Transport	5
2.1	Minh hoạ kịch bản giao tiếp TCP bằng sơ đồ khối	6
2.2	Giao thức trao đổi giữa Client và Server	7
2.3	Cấu trúc thông điệp	9
2.4	Kiểu dữ liệu thông điệp	10
2.5	Tổ chức dữ liệu	11
3	Sử dụng UDP tại tầng Transport	14
3.1	Minh hoạ kịch bản giao tiếp UDP bằng sơ đồ khối	15
3.2	Giao thức trao đổi giữa Client và Server	16
3.3	Cấu trúc thông điệp	18
3.4	Kiểu dữ liệu thông	19
3.5	Tổ chức dữ liệu	21
4	Môi trường lập trình	23
4.1	Tổng quan mã nguồn	23
4.2	Các thư viện và module sử dụng trong mã nguồn	23
5	Hướng dẫn sử dụng các tính năng	24
5.1	Cài đặt Python và VSCode	24
5.2	Sử dụng các chức năng	26
	Tài liệu tham khảo	32

1 Giới thiệu

Trong thời đại công nghệ thông tin bùng nổ, việc trao đổi dữ liệu qua mạng đóng vai trò quan trọng trong nhiều lĩnh vực, từ giao tiếp hằng ngày đến quản lý và vận hành các hệ thống phức tạp. Một trong những phương thức phổ biến nhất để thực hiện việc trao đổi dữ liệu là sử dụng giao thức TCP và UDP, hai giao thức truyền tải quan trọng tại tầng Transport của mô hình TCP/IP.

Đề án này được thực hiện với mục tiêu nghiên cứu và triển khai hệ thống truyền tải file từ Server đến Client thông qua việc sử dụng hai giao thức TCP và UDP kèm cơ chế Truyền tin cậy RDT (Reliable Data Transfer). Hệ thống không chỉ đảm bảo tính chính xác, toàn vẹn của dữ liệu mà còn tối ưu hiệu suất truyền tải, đồng thời cho phép quản lý nhiều kết nối client một cách hiệu quả.

Dưới sự hướng dẫn tận tình của **Ths. Huỳnh Thụy Bảo Trân** và **Ths. Chung Thùy Linh**, nhóm chúng tôi đã hoàn thành đề án với các tính năng chính như *quản lý danh sách file trên server, tải file theo từng phần (part), và ghép nối dữ liệu tải về một cách hoàn chỉnh trên client*. Báo cáo này sẽ trình bày chi tiết *Thông tin, bảng phân công nhóm, Kịch bản giao tiếp giữa các chương trình (Giao thức trao đổi giữa client và server, cấu trúc thông điệp, kiểu dữ liệu của thông điệp, cách tổ chức cơ sở dữ liệu), Môi trường lập trình và các framework, module hỗ trợ, Hướng dẫn sử dụng các tính năng chương trình*.

Chúng tôi xin gửi lời cảm ơn chân thành đến những người bạn, người anh, người chị vì sự hỗ trợ quý báu trong suốt quá trình thực hiện đề án, cũng như đến các hai giảng viên hướng dẫn đã cung cấp nền tảng kiến thức vững chắc giúp chúng tôi hoàn thành tốt nhiệm vụ này. Trong quá trình làm đề án, nhóm chúng tôi vừa học kiến thức mới cũng như lần đầu được tiếp xúc với lập trình socket nên không thể thiếu những sai sót và hạn chế hy vọng giảng viên có thể giúp đỡ chúng tôi nhận ra và cải thiện. Xin cảm ơn.

1.1 Thông tin thành viên

STT	MSSV	Họ và tên	Email
1	23120233	Nguyễn Lê Hữu Điền	gaolua217@gmail.com
2	23120262	Tống Dương Thái Hoà	tdthoa.hry@gmail.com
3	23120264	Nguyễn Phúc Hoàng	nphuchoang.itus@gmail.com

Bảng 1: Thông tin thành viên

1.2 Phân chia công việc

STT	Yêu cầu	Mức độ hoàn thành	Thực hiện
TCP			
1.1	Ý tưởng Giao thức truyền file TCP (kịch bản giao tiếp, hành động sau khi nhận được tín hiệu,...)	100%	Phúc Hoàng (50%) Thái Hoà (50%)
1.2	Lập trình Server hàm xử lý Client (Gửi list file, nhận thông điệp tải, chia luồng để truyền từng phần,...)	100%	Thái Hoà
1.3	Lập trình Client hàm xử lý Server (Nhận list file, gửi thông điệp tải, chia luồng để yêu cầu tải từng phần,...)	100%	Thái Hoà
UDP			
2.1	Ý tưởng giao thức truyền file UDP (Kịch bản giao tiếp, hành động sau khi nhận được tín hiệu, cơ chế rdt,...)	100%	Hữu Điền (50%) Phúc Hoàng (50%)
2.2	Lập trình Server hàm xử lý Client (Gửi list file, Nhận các cờ thông điệp, chia luồng để truyền từng phần, cơ chế rdt sau khi nhận tín hiệu,...)	100%	Phúc Hoàng
2.3	Lập trình Client hàm xử lý Server (Nhận list file, gửi cờ thông điệp tải, chia luồng để yêu cầu tải từng phần, cơ chế rdt sau khi nhận tín hiệu,...)	100%	Phúc Hoàng
CÁC HÀM CHỨC NĂNG			
3.1	Hàm quét các file có trên server	100%	Hữu Điền
3.2	Hàm chuyển đổi kích thước file từ bytes sang KB, MB, hoặc GB phù hợp.	100%	Hữu Điền
3.3	Hàm đóng server (Ctrl+C, clear socket,...)	100%	Thái Hoà
3.4	Hàm quét lại input.txt sau mỗi 5s	100%	Thái Hoà
3.5	Hàm hiển thị thanh tiến độ	100%	Thái Hoà
3.6	Hàm merge chunk	100%	Phúc Hoàng
BÁO CÁO			
4.1	Giới thiệu	100%	Hữu Điền
4.2	TCP	100%	Thái Hoà
4.3	UDP	100%	Phúc Hoàng
4.4	Môi trường lập trình	100%	Thái Hoà
4.5	Hướng dẫn các chức năng	100%	Phúc Hoàng
4.6	Format lại báo cáo	100%	Phúc Hoàng

Bảng 2: Bảng Phân chia công việc

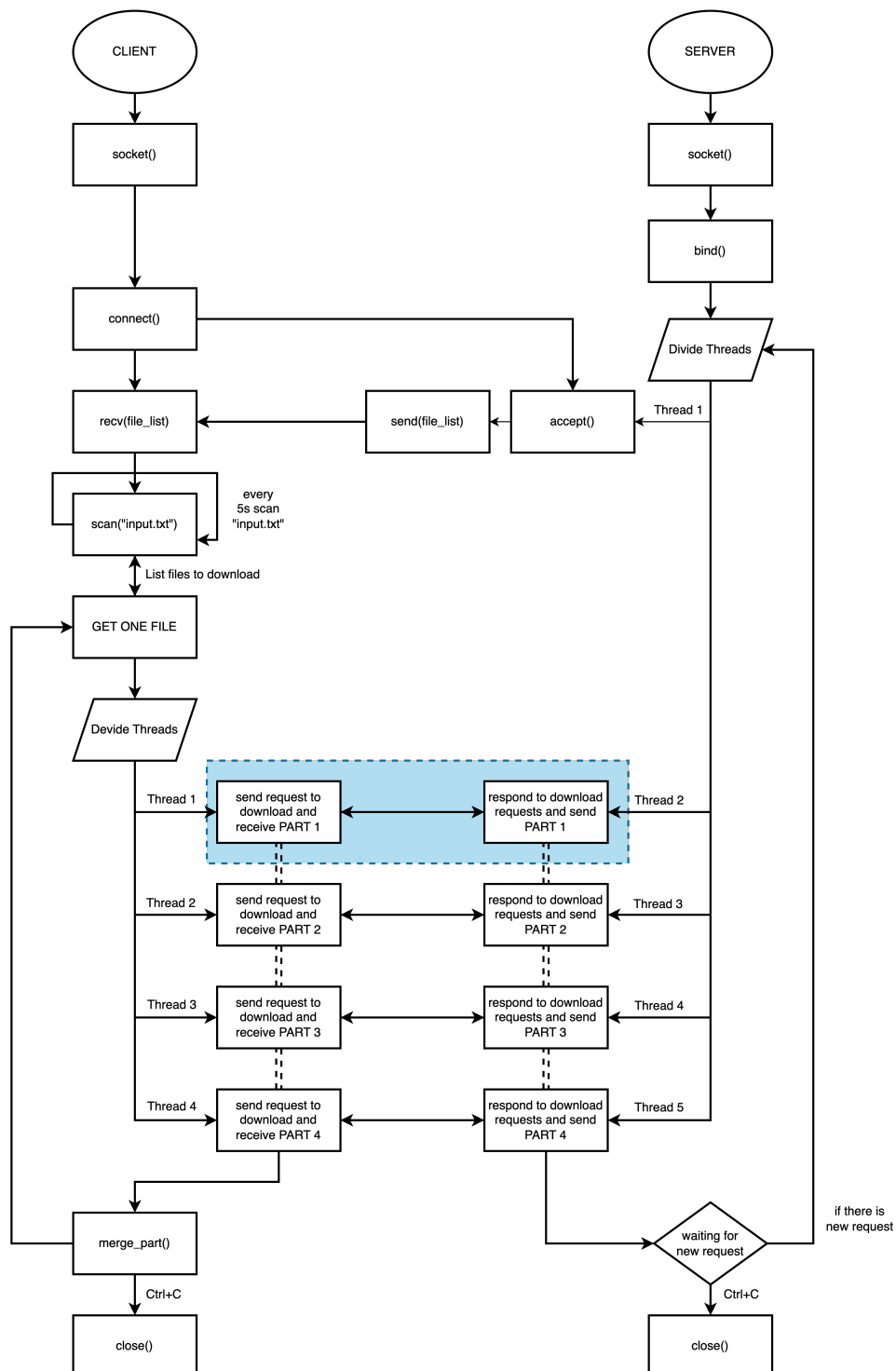
2 Sử dụng TCP tại tầng Transport

Trước khi đi vào chi tiết, mã nguồn của nhóm chúng tôi được cài đặt các hàm gửi và nhận như sau:

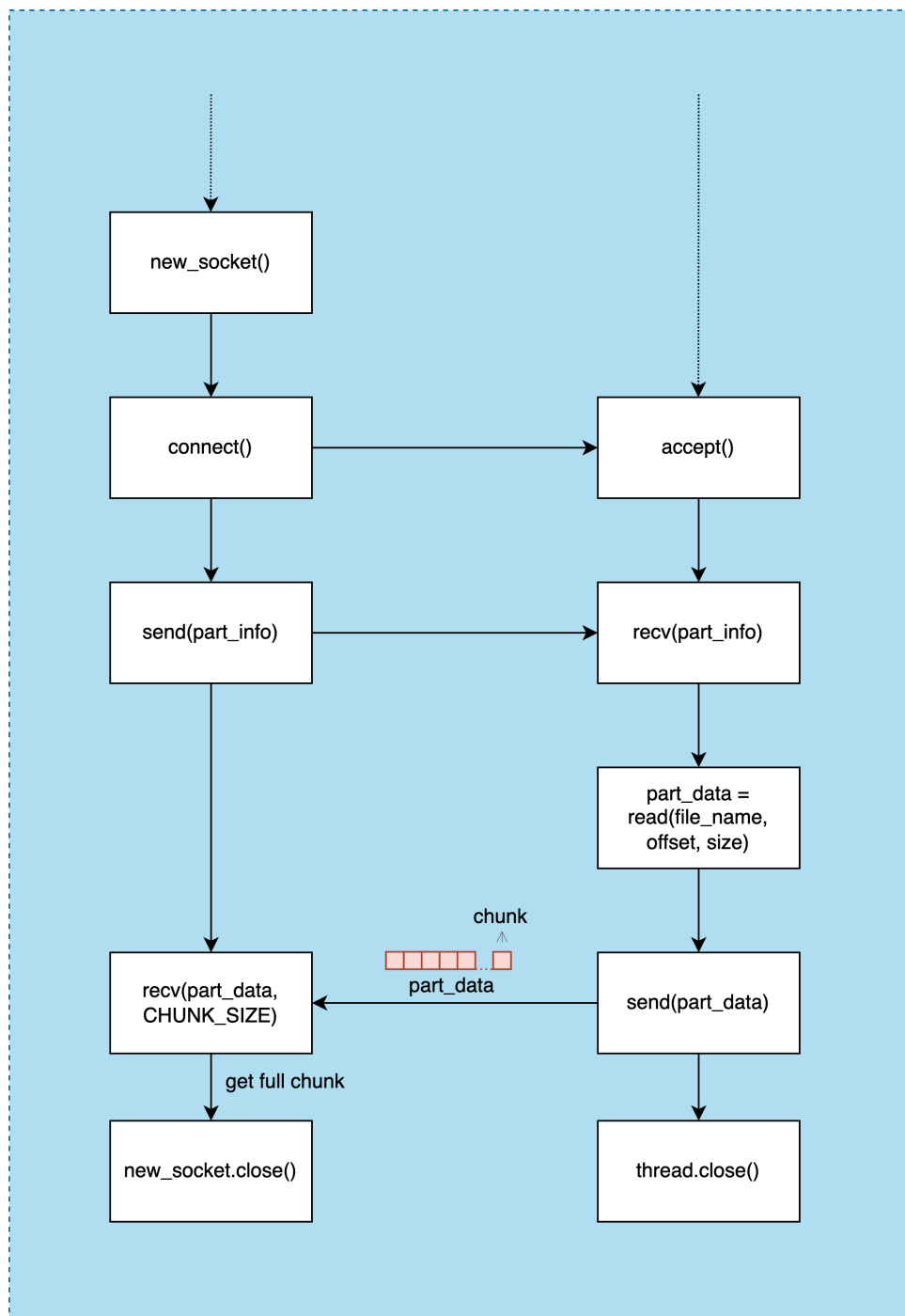
- **Bước 1:** Lưu trữ kích thước thông điệp vào một biến số nguyên 4 bytes.
- **Bước 2:** Gửi cho bên còn lại kích thước thông điệp trước. Bên nhận sẽ "chỉ" nhận 4 bytes để biết kích thước thông điệp mình sắp nhận có bao nhiêu bytes.
- **Bước 3:** Bên gửi thông điệp tiếp tục gửi dữ liệu thực sự qua cho bên nhận, khi này bên nhận đã biết được kích thước thông điệp mình sắp nhận và nhận đúng số bytes đã được chỉ định trước đó.

Để giải thích đơn giản và ngắn gọn hơn, chúng tôi sẽ không nhắc đến các bước gửi kích thước trước rồi gửi thông điệp sau mà gộp chung thành gửi thông điệp.

2.1 Minh hoạ kịch bản giao tiếp TCP bằng sơ đồ khối



Hình 1: Kịch bản giao tiếp của UDP thể hiện bằng sơ đồ khối



Hình 2: Kịch bản giao tiếp của từng thread trong TCP gửi và nhận từng phần

2.2 Giao thức trao đổi giữa Client và Server

Giao thức được sử dụng là giao thức **TCP** (*Transmission Control Protocol*) ở tầng Transport dựa trên socket. Đây là một giao thức hướng kết nối, đảm bảo truyền dữ liệu tin cậy và theo thứ tự. Điều này rất quan trọng trong truyền tải file, nơi mà việc mất hoặc xáo trộn dữ liệu có thể làm hỏng file. Giao thức ứng dụng mà nhóm chúng tôi thiết kế riêng để ứng dụng giao thức TCP trong việc truyền dữ liệu tin cậy. Sau đây, chúng tôi sẽ phân tích chi tiết hơn từng giai đoạn:

Bước 1: Khởi tạo kết nối (*TCP Handshake*) - Đây là bước đầu tiên, sử dụng cơ chế bắt tay ba bước (*three-way handshake*) của TCP

- **SYN (*Synchronize*)**: Client gửi một gói tin SYN đến Server. Gói tin này yêu cầu thiết lập kết nối và chứa một số thứ tự ngẫu nhiên (*sequence number*) được sử dụng để theo dõi các gói tin.
- **SYN-ACK (*Synchronize-Acknowledge*)**: Server nhận gói tin SYN, phản hồi bằng gói tin SYN-ACK. Gói tin này xác nhận yêu cầu kết nối từ Client, cũng chứa một số thứ tự riêng và số ACK (*acknowledgment number*) bằng số thứ tự của Client cộng thêm 1.
- **ACK (*Acknowledge*)**: Client nhận gói tin SYN-ACK, gửi lại gói tin ACK. Gói tin này xác nhận việc nhận gói tin SYN-ACK từ Server. Số ACK trong gói tin này bằng số thứ tự của Server cộng thêm 1.

Sau bước này, kết nối TCP được thiết lập và dữ liệu có thể được truyền giữa Client và Server.

Bước 2: Trao đổi danh sách file (*JSON over TCP*) - Sau khi kết nối TCP được thiết lập, Server sẽ gửi danh sách các file có sẵn để tải cho Client, đính kèm là header chứa kích thước của danh sách đó. Dữ liệu này được định dạng JSON và được mã hóa bằng UTF-8

- **Định dạng JSON**: Dữ liệu được cấu trúc dưới dạng key-value pairs, với key là tên file và value là kích thước file (*tính bằng byte*).
Ví dụ: "File1.zip": 5242880, "File2.zip": 10485760, "File3.zip": 20971520, ...
- **UTF-8 Encoding**: Việc sử dụng UTF-8 đảm bảo rằng tên file có thể chứa các ký tự Unicode (Ví dụ: *tiếng Việt có dấu*).
- **Truyền qua TCP**: Chuỗi JSON được chia thành các gói tin nhỏ hơn nếu cần và được gửi qua kết nối TCP. Client nhận các gói tin này, ghép chúng lại thành chuỗi JSON hoàn chỉnh và sau đó phân tích (*parse*) chuỗi JSON để lấy thông tin về các file.

Bước 3: Yêu cầu tải file: Client gửi yêu cầu tải file đến Server dưới dạng một chuỗi văn bản có cấu trúc `filename|offset|size` và được đính kèm một header chứa kích thước của chuỗi vào phía trước.

- **filename**: Tên của file mà Client muốn tải.
- **offset**: Vị trí byte bắt đầu tải (tính từ đầu file). Điều này cho phép Client tải các phần khác nhau của file một cách song song.
- **size**: Số lượng byte mà Client muốn tải từ vị trí offset.
Ví dụ: `File1.zip|0|1048576` (yêu cầu tải 1MB đầu tiên của File1.zip).
- **Phân tích cú pháp (*Parsing*)**: Server nhận chuỗi này và sử dụng dấu gạch đứng (|) để phân tách các thành phần filename, offset và size. Sau đó, offset và size được chuyển đổi sang kiểu số nguyên cho đúng định dạng.

Bước 4: Truyền dữ liệu file (*Raw Bytes over TCP*): Sau khi nhận yêu cầu tải file từ Client, Server sẽ thực hiện các bước sau:

- Mở file được yêu cầu trong thư mục `server_files`.
- Di chuyển đến vị trí byte offset bằng hàm `seek()`.
- Đọc `size` byte từ file.
- Gửi dữ liệu đã đọc (dưới dạng byte stream) cho Client qua kết nối TCP.

Client nhận dữ liệu này và ghi vào file cục bộ. Việc tải file được chia thành các phần (chunks) và có thể được thực hiện song song bằng cách sử dụng nhiều luồng (threads).

Bước 5: Đóng kết nối (*TCP Connection Termination*): Sau khi Client đã nhận đủ dữ liệu cho một phần (*chunk*), kết nối socket tương ứng với phần đó sẽ được đóng. Quá trình này sử dụng cơ chế bắt tay bốn bước (*four-way handshake*) của TCP:

- **Request FIN (*Finish*):** Một trong hai bên gửi gói tin FIN để báo hiệu kết thúc việc truyền dữ liệu.
- **Request ACK (*Acknowledge*):** Bên còn lại nhận gói tin FIN và gửi lại gói tin ACK để xác nhận.
- **Response FIN:** Bên còn lại cũng gửi gói tin FIN để báo hiệu kết thúc kết nối từ phía mình.
- **Response ACK:** Bên gửi FIN ban đầu nhận gói tin FIN và gửi lại gói tin ACK để hoàn tất quá trình đóng kết nối.

2.3 Cấu trúc thông điệp

Trong hệ thống tải file Client/Server sử dụng giao thức TCP, chúng ta có một số loại thông điệp chính được trao đổi giữa Client và server, mỗi loại có cấu trúc riêng. Ngoài ra, Server cũng sử dụng một dạng "cơ sở dữ liệu" đơn giản để lưu trữ thông tin về các file có sẵn.

2.3.1 Cấu trúc thông điệp trao đổi danh sách file

Thông điệp này được gửi từ Server đến Client ngay sau khi kết nối TCP được thiết lập, với mục đích cung cấp danh sách các file có sẵn để tải.

- **Định dạng:** JSON (*JavaScript Object Notation*), một định dạng dữ liệu dạng văn bản với các cặp key-value. Lý do vì sao nhóm chúng tôi lại chọn JSON bởi vì đây là định dạng dễ đọc, dễ hiểu, và được hỗ trợ bởi hầu hết các ngôn ngữ lập trình, ngoài ra có thể thêm thông tin khác mà không ảnh hưởng đến chương trình.
- **Cấu trúc cụ thể:** Một đối tượng JSON (object) với:
 - **Key:** Tên file (string). Ví dụ: `"File1.zip"`, `"image.jpg"`.
 - **Value:** Kích thước file tính bằng byte (số nguyên). Ví dụ: `1048576` (1MB), `5242880` (5MB).
- **Ví dụ:**

```
{  
    "File1.zip": 5242880,  
    "File2.zip": 10485760,  
    "File3.zip": 20971520,  
    "image.jpg": 2097152,  
    "document.pdf": 1572864  
}
```

2.3.2 Cấu trúc thông điệp yêu cầu tải file

Thông điệp này được gửi từ Client đến Server để yêu cầu tải một phần (*part*) của file.

- **Định dạng:** Chuỗi văn bản phân tách bằng dấu hai chấm (|).
- **Cấu trúc cụ thể:** filename|offset|size
 - filename: Tên file (string). Ví dụ: "File1.zip".
 - offset: Vị trí byte bắt đầu tải (integer). Ví dụ: 0, 1048576.
 - size: Số lượng byte cần tải (integer). Ví dụ: 1048576.
- **Ví dụ:** File1.zip|1048576|2097152 có nghĩa là tải 2MB của file từ vị trí byte offset 1048576 (1MB kể từ đầu offset 0) của File1.zip.

2.3.3 Cấu trúc dữ liệu file được truyền tải

Server sẽ đọc file bằng phương thức đọc dữ liệu nhị phân (đối với python là sử dụng hàm `read(file_name, 'rb')`) và truyền đi cho Client bằng kiểu dữ liệu byte. Client nhận dữ liệu và thực hiện ghi vào file tương ứng được bằng phương thức ghi dữ liệu nhị phân (đối với python là sử dụng hàm `read(file_name, 'wb')`).

2.4 Kiểu dữ liệu thông điệp

Kiểu dữ liệu được sử dụng cho từng loại thông điệp của giao thức truyền tải file là yếu tố quan trọng đảm bảo hiệu suất và tính tương thích. Dưới đây là những phân phân tích chi tiết:

2.4.1 Kiểu dữ liệu của thông điệp trao đổi danh sách file

- **Dữ liệu ban đầu trên Server:**
 - Server sử dụng một dictionary Python (`self.file_data` trong `server.py`, lưu dưới dạng *key – value*).
 - **Key (Tên file):** Kiểu `string`. Ví dụ: "File1.zip".
 - **Value (Kích thước file):** Kiểu `int`. Ví dụ: 5242880.
- **Dữ liệu được gửi qua mạng:**

- Dictionary được chuyển thành chuỗi JSON bằng `json.dumps()`.
- **Kiểu dữ liệu trên đường truyền:** Chuỗi JSON được mã hóa UTF-8.

- **Dữ liệu được nhận tại Client:**

- Client sử dụng `json.loads()` để chuyển chuỗi JSON thành dictionary Python.
- **Kiểu dữ liệu sau khi phân tích:** Giống như trên Server là key sử dụng kiểu dữ liệu string và value sử dụng kiểu dữ liệu int.

2.4.2 Kiểu dữ liệu của thông điệp yêu cầu tải file

- **Dữ liệu được gửi từ Client:** Chuỗi theo định dạng `filename|offset|size`.

- `filename`: Kiểu string.
- `offset`: Kiểu int (chuyển thành string để ghép).
- `size`: Kiểu int (chuyển thành string để ghép).

- **Dữ liệu được gửi qua mạng:** Chuỗi string mã hóa UTF-8.

- **Dữ liệu được nhận tại Server:**

- Server sử dụng `split('|')` để tách chuỗi thành một danh sách các string.
- Chuyển đổi `offset` và `size` từ string sang int.

2.4.3 Kiểu dữ liệu của dữ liệu file được truyền tải

Tất cả dữ liệu liên quan tới file gốc (file mà server chuyển đi và client nhận được) đều ở dạng **bytes**

2.5 Tổ chức dữ liệu

Việc tổ chức dữ liệu trong hệ thống tải file bằng TCP của chúng tôi khá đơn giản để phục vụ mục đích cơ bản của chương trình. Chúng tôi sẽ phân tích cách dữ liệu được tổ chức ở cả phía Server và Client, hoàn toàn dựa trên logic đã được triển khai trong code.

2.5.1 Tổ chức dữ liệu phía Server

Server chịu trách nhiệm chính trong việc quản lý danh sách các file sẵn sàng để tải và cung cấp dữ liệu file theo yêu cầu của client. Cách tổ chức dữ liệu của server được chia thành hai phần:

Danh sách file (Được lưu ở file text `data.txt` và dictionary `self.file_data`)

- **File `data.txt`:**

- Đây là nguồn dữ liệu ban đầu. File này nằm trong cùng thư mục với script server.

- Mỗi dòng trong file chứa thông tin về một file theo định dạng: `file_name file_size`.

Ví dụ:

```
File1.zip 4,88GB
File2.zip 10MB
image.jpg 204,79KB
```

- **Dictionary `self.file_data`:**

- Khi server khởi động, nó sẽ đọc nội dung của `data.txt`. Dữ liệu từ file này được chuyển đổi và lưu trữ trong một dictionary Python có tên `self.file_data`.

- **Cấu trúc:**

- * **Key:** Tên file (kiểu `string`).
- * **Value:** Kích thước file (kiểu `int`).

Ví dụ:

```
{
    "File1.zip": 5242880928,
    "File2.zip": 10485760,
    "image.jpg": 209715,
}
```

Việc sử dụng dictionary giúp server truy cập kích thước file theo tên file một cách nhanh chóng.

Lưu trữ file thực tế

- Các file thực tế được lưu trữ trong thư mục `server_files`. Thư mục này **phải tồn tại** trong cùng thư mục với script server. Nếu thư mục này không tồn tại, server sẽ gặp lỗi.
- Server **lưu dữ liệu** của các file này, bao gồm tên và kích thước (kích thước dưới dạng **bytes** sẽ được ghi vào file `data.txt` dưới dạng đã được chuyển đổi rút gọn).

2.5.2 Tổ chức dữ liệu phía Client

Client chịu trách nhiệm quản lý danh sách các file cần tải và lưu trữ dữ liệu đã tải về.

Danh sách file cần tải (`input.txt` và `set self.downloaded_files`)

- **File `input.txt`:**

- File này nằm trong cùng thư mục với script client.
- Mỗi dòng trong file chứa tên của một file cần tải.

– Ví dụ:



- **Set `self.downloaded_files`:**

- Client sử dụng một set Python có tên `self.downloaded_files` để theo dõi các file đã được tải thành công.
- Set là một cấu trúc dữ liệu lưu trữ các phần tử **duy nhất** và cho phép kiểm tra sự tồn tại của một phần tử rất nhanh.

Lưu trữ dữ liệu đã tải về (thư mục Downloads và các file .part)

- Client tạo (nếu chưa tồn tại) một thư mục có tên **downloads** trong cùng thư mục với script client. Đây là nơi các file đã tải về được lưu trữ.
- Để hỗ trợ tải file song song (với 4 luồng), client chia mỗi file thành 4 phần bằng nhau (trừ phần cuối cùng có thể nhỏ hơn). Mỗi phần được lưu tạm thời vào một file riêng biệt với định dạng `tên_file.partN` và lưu vào một thư mục **bin**.
Ví dụ: Các phần của `File1.zip` sẽ được lưu vào `File1.zip.part0`, `File1.zip.part1`, `File1.zip.part2`, và `File1.zip.part3`.
- Sau khi tất cả các phần của một file đã được tải xuống thành công, client sẽ ghép các phần này lại thành file hoàn chỉnh và lưu vào thư mục **downloads** với tên file gốc. Sau đó, các file **.part** sẽ bị xóa.

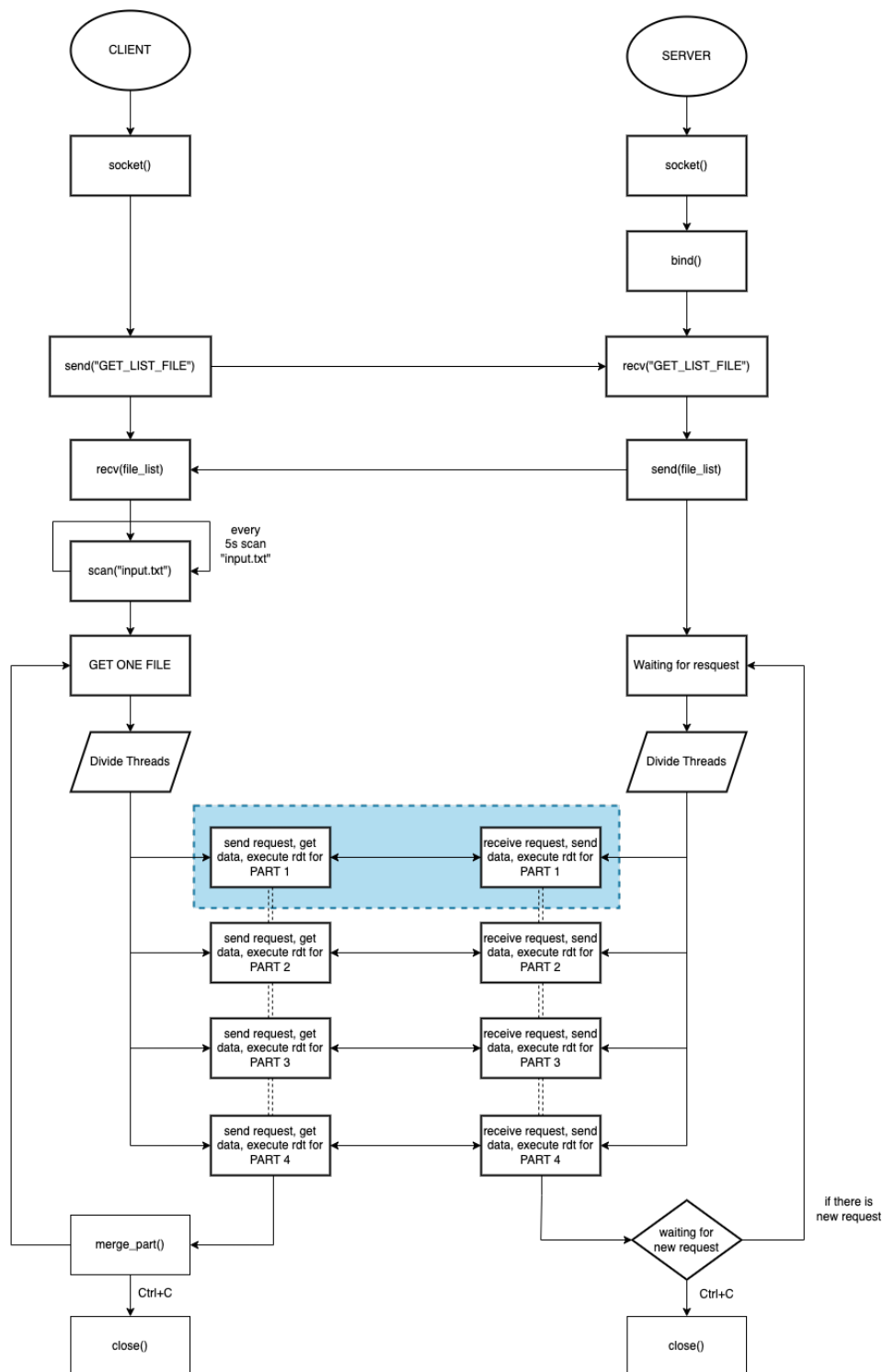
3 Sử dụng UDP tại tầng Transport

Tương tự phần TCP, trước khi đi vào chi tiết, mã nguồn của nhóm chúng tôi được cài đặt các hàm gửi và nhận như sau:

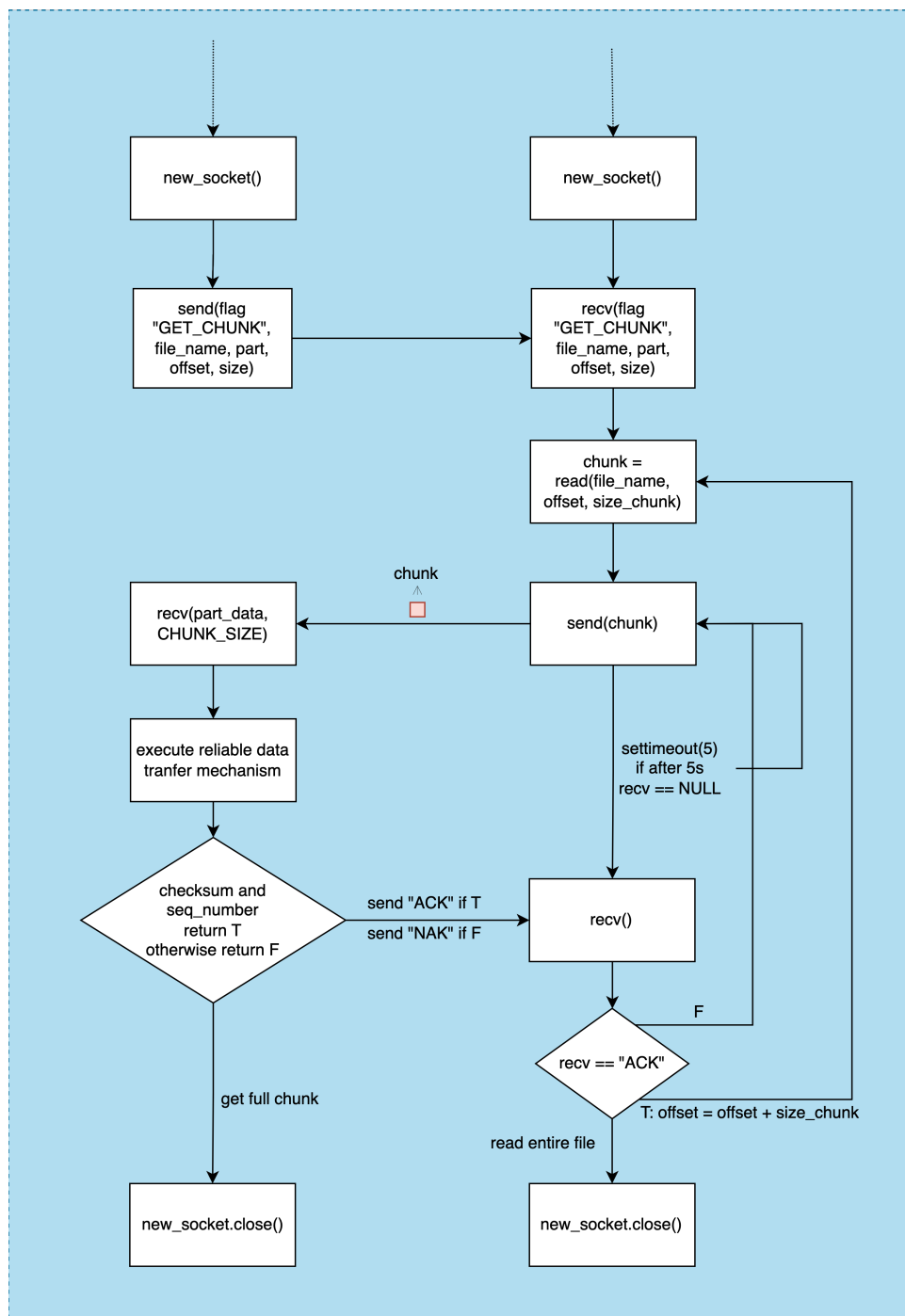
- **Bước 1:** Lưu trữ kích thước thông điệp vào một biến số nguyên 4 bytes.
- **Bước 2:** Gửi cho bên còn lại kích thước thông điệp trước. Bên nhận sẽ "chỉ" nhận 4 bytes để biết kích thước thông điệp mình sắp nhận có bao nhiêu bytes.
- **Bước 3:** Bên gửi thông điệp tiếp tục gửi dữ liệu thực sự qua cho bên nhận, khi này bên nhận đã biết được kích thước thông điệp mình sắp nhận và nhận đúng số bytes đã được chỉ định trước đó.

Với mục đích cài đặt tối ưu chương trình sao cho nhận đúng kích thước của thông điệp, nhóm chúng tôi đã thực hiện các cài đặt trên. Để giải thích đơn giản và ngắn gọn hơn, chúng tôi sẽ không nhắc đến các bước gửi kích thước trước rồi gửi thông điệp sau mà gộp chung thành gửi thông điệp.

3.1 Minh hoạ kịch bản giao tiếp UDP bằng sơ đồ khối



Hình 3: Kịch bản giao tiếp thể hiện bằng sơ đồ khối



Hình 4: Kịch bản giao tiếp của từng thread trong UDP gửi và nhận từng phần

3.2 Giao thức trao đổi giữa Client và Server

Giao thức được sử dụng là giao thức **UDP** (*User Datagram Protocol*) ở tầng Transport dựa trên socket. Đây là một giao thức không kết nối, không đảm bảo truyền dữ liệu tin cậy hoặc theo thứ tự. Do đó, giao thức ứng dụng mà chúng tôi thiết kế riêng cho trong việc truyền tải file bằng UDP nhằm đảm bảo truyền dữ liệu tin cậy (*Reliable Data Transfer - RDT*). Chi tiết các bước trao đổi như sau:

Bước 1: Khởi tạo socket (*UDP Socket Initialization*):

- **Client và Server khởi tạo:** Cả Client và Server tạo socket UDP để trao đổi dữ liệu.
- **Địa chỉ kết nối:** Client cần biết địa chỉ IP và cổng của Server để gửi dữ liệu. Ngược lại, Server có thể xác định địa chỉ IP và cổng của Client từ gói tin nhận được.

Bước 2: Yêu cầu danh sách file (*GET_FILE_LIST*):

- Client gửi thông điệp "GET_FILE_LIST" đến Server.
- Server nhận được thông điệp, quét thư mục `server_files`, tạo danh sách file hiện có (bao gồm tên và kích thước), mã hóa danh sách này dưới dạng JSON, và gửi lại Client.

Bước 3: Yêu cầu tải dữ liệu (*GET_CHUNK*): Sau khi nhận được danh sách file, Client quyết định file nào cần tải và chia file đó thành các phần (*chunks*). Với mỗi phần, Client gửi yêu cầu GET_CHUNK:

- **Định dạng thông điệp:** "GET_CHUNK|filename|offset|size|part_number"
- **Giải thích các thành phần:**
 - `filename`: Tên file cần tải.
 - `offset`: Vị trí bắt đầu của phần (tính bằng byte).
 - `size`: Kích thước của phần (tính bằng byte).
 - `part_number`: Số thứ tự của phần.

Client sử dụng một socket riêng biệt để tải từng phần, cho phép thực hiện tải đồng thời (*parallel threads*).

Bước 4: Phản hồi yêu cầu chunk (*Chunk Data Response*): Server nhận được thông điệp GET_CHUNK mở ra kết nối thread song song kết nối tương ứng với các luồng mà Client gửi yêu cầu qua và xử lý:

- Mở file tương ứng và đọc dữ liệu từ vị trí `offset` với kích thước `size`.
- Chia dữ liệu thành các chunk nhỏ (tối đa `BUFFER_SIZE` byte mỗi chunk).
- Gửi từng chunk dữ liệu kèm thông tin kiểm tra lỗi (checksum, `seq_number`) cho Client.
- Sử dụng timeout để chờ phản hồi từ Client (ACK hoặc NAK) cho từng chunk (Nếu quá timeout mà Server không nhận được phải hồi thì sẽ gửi lại gói tin).

Bước 5: Nhận và xác nhận chunk dữ liệu (*ACK/NAK Mechanism*): Client xử lý từng chunk dữ liệu nhận được từ Server:

- **Kiểm tra checksum:** Xác minh tính toàn vẹn của dữ liệu. Nó tính một giá trị tóm tắt (summary value) đại diện cho nội dung dữ liệu gốc. Nếu dữ liệu bị thay đổi (do lỗi trong truyền dẫn hoặc lưu trữ), checksum của dữ liệu thay đổi sẽ không khớp với checksum ban đầu, giúp phát hiện lỗi.
- **Kiểm tra `seq_number`:** Kiểm tra số thứ tự của gói tin, đảm bảo các gói tin được truyền đi đủ và đúng thứ tự.
- **Phản hồi:** Gửi ACK nếu chunk hợp lệ hoặc NAK nếu chunk bị lỗi.

- **Lưu dữ liệu:** Chunk hợp lệ được lưu tạm thời vào file cục bộ, nếu có gói tin trùng sẽ thực hiện xoá trùng lặp gói tin đó.

Nếu Server nhận được NAK, nó sẽ gửi lại chunk tương ứng.

Bước 6: Ghép các phần (*File Reassembly*): Sau khi nhận đủ các chunk của một phần, Client lưu chúng thành một file tạm. Khi tất cả các phần được tải xong, các file tạm này sẽ được ghép lại thành file hoàn chỉnh.

Bước 7: Hoàn tất truyền dữ liệu (*Completion*): Sau khi tất cả dữ liệu được truyền thành công, Client và Server đóng các socket UDP tương ứng.

Tóm lại, giao thức sử dụng **UDP** để truyền dữ liệu, kết hợp với một cơ chế tự định nghĩa đảm bảo truyền tin cậy thông qua việc phân mảnh dữ liệu thành các chunk nhỏ và sử dụng phản hồi ACK/NAK.

3.3 Cấu trúc thông điệp

Cấu trúc thông điệp trong giao thức UDP tự định nghĩa cơ chế RDT có sử dụng các thông điệp ở dạng flag và gửi chunk dữ liệu, dưới đây là các loại thông điệp được sử dụng:

Loại 1: Thông điệp GET_FILE_LIST - "Yêu cầu danh sách file":

- **Cấu trúc:** Một chuỗi ký tự đơn giản: "GET_FILE_LIST".
- **Ý nghĩa:** Báo hiệu cho server rằng client muốn nhận danh sách các file hiện có trên server.

Loại 2: Thông điệp JSON File List - "Danh sách file"(từ Server gửi về Client):

- **Cấu trúc:** Chuỗi JSON chứa một dictionary Python, trong đó:
 - **Key:** Tên file (kiểu `string`).
 - **Value:** Kích thước file (kiểu `int`, đơn vị byte).
- **Ví dụ:** `{"file1.txt": 1024, "file2.zip": 2048, "image.jpg": 5120}`
- **Ý nghĩa:** Cung cấp cho client danh sách các file có thể tải, bao gồm thông tin tên và kích thước.

Loại 3: Thông điệp GET_CHUNK - "Yêu cầu tải chunk":

- **Cấu trúc:** Chuỗi ký tự có định dạng:
`"GET_CHUNK|file_name|offset_part|size_part|part_number"`.
 - **file_name:** Tên file cần tải (kiểu `string`).
 - **offset_part:** Vị trí byte bắt đầu của phần (kiểu `int`).
 - **size_part:** Kích thước của phần (kiểu `int`).
 - **part_number:** Số thứ tự của phần (kiểu `int`, bắt đầu từ 0).
- **Ý nghĩa:** Yêu cầu server gửi một phần cụ thể của file.
- **Ví dụ:** `"GET_CHUNK|my_document.pdf|0|10240|0"` có nghĩa là "Tải cho tôi chunk của phần đầu tiên file document.pdf từ offset 0 với kích thước 10240 bytes".

Loại 4: Thông điệp "Chunk dữ liệu"(từ Server gửi về Client):

- **Cấu trúc:** Dữ liệu nhị phân được đóng gói bằng hàm `struct.pack()` với định dạng: `"!I I I {len(data)}s"`. Trong đó:
 - I là số nguyên không dấu 4 byte gồm 2 loại
 - `part_number`: Số thứ tự của phần.
 - `seq_number`: Số thứ tự của chunk trong phần.
 - `checksum`: Giá trị kiểm tra lỗi của chunk dữ liệu.
 - `{len(data)}s`: Chuỗi byte chứa dữ liệu của chunk.
- **Ý nghĩa:** Gửi dữ liệu thực tế của một chunk file, kèm metadata để đảm bảo truyền tin cậy.

Loại 5: Thông điệp ACK/NAK - "Xác nhận/Từ chối":

- **Cấu trúc:**
 - ACK: `"ACK-part_number_seq_number"`.
 - NAK: `"NAK-part_number_seq_number"`.
- **Ví dụ:** `"ACK-1_10"`, `"NAK-0_5"`.
- **Ý nghĩa:**
 - ACK: Client thông báo đã nhận thành công chunk có số thứ tự `seq_number` trong phần `part_number`.
 - NAK: Client thông báo chunk bị lỗi hoặc không nhận được, yêu cầu server gửi lại.

3.4 Kiểu dữ liệu thông

Tương tự phần TCP, Kiểu dữ liệu thông điệp sử dụng cho giao thức ứng dụng dưới đây kết hợp UDP cũng sử dụng những kiểu dữ liệu cơ bản. Dưới đây là chi tiết kiểu dữ liệu của từng loại thông điệp:

3.4.1 Thông điệp "Yêu cầu danh sách file"(GET_FILE_LIST):

- **Kiểu dữ liệu:** `string` (chuỗi ký tự).
- **Encoding:** UTF-8.
- **Giải thích:** Đây là một chuỗi văn bản đơn giản được mã hóa bằng UTF-8.

3.4.2 Thông điệp "Danh sách file"(từ Server về Client):

- **Kiểu dữ liệu:** `string` (chuỗi ký tự).
- **Định dạng:** JSON.
- **Encoding:** UTF-8.
- **Ví dụ:** `{"file1.txt": 1024, "file2.zip": 2048, "image.jpg": 5120}`.
- **Giải thích:** Server sử dụng JSON (JavaScript Object Notation) để biểu diễn danh sách file.

3.4.3 Thông điệp "Yêu cầu tải chunk"(GET_CHUNK):

- **Kiểu dữ liệu:** `string` (chuỗi ký tự).
- **Định dạng:** Chuỗi được phân tách bằng dấu `'|'`.
- **Encoding:** UTF-8.
- **Ví dụ:** `"GET_CHUNK|my_document.pdf|0|10240|0"`.
- **Giải thích:** Thông điệp này được cấu trúc thành các phần, mỗi phần được phân tách bằng dấu gạch đứng (`'|'`):
 - `GET_CHUNK`: Chuỗi cố định xác định loại thông điệp.
 - `file_name`: Tên file (kiểu `string`).
 - `offset_part`: Offset của phần (kiểu `int`, được chuyển thành `string`).
 - `size_part`: Kích thước của phần (kiểu `int`, được chuyển thành `string`).
 - `part_number`: Số thứ tự của phần (kiểu `int`, được chuyển thành `string`).

3.4.4 Thông điệp "Chunk dữ liệu"(từ Server về Client):

- **Kiểu dữ liệu:** `bytes` (dữ liệu nhị phân).
- **Định dạng:** Đóng gói bằng `struct.pack()` với format string `"!I I I {len(data)}s"`:
 - `!`: Thứ tự theo Network byte (big-endian).
 - `I`: Số nguyên không dấu (4 bytes) - `part_number`, `seq_number`.
 - `{len(data)}s`: Chuỗi byte chứa dữ liệu chunk thực tế.
- **Ví dụ:** Giả sử `data = b"0123456789"`, `part_number = 1`, `seq_number = 5`, `checksum = 45`, gói tin sẽ có dạng: `[0, 0, 0, 1, 0, 0, 0, 5, 45, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57]`.

3.4.5 Thông điệp "Xác nhận/Từ chối"(ACK/NAK):

- **Kiểu dữ liệu:** `string` (chuỗi ký tự).
- **Định dạng:**
 - `ACK`: `"ACK-{part_number}_{seq_number}"`.
 - `NAK`: `"NAK-{part_number}_{seq_number}"`.
- **Encoding:** UTF-8.
- **Ví dụ:** `"ACK-1_10"`, `"NAK-0_5"`.
- **Giải thích:** Đây là các thông điệp ngắn gọn để báo hiệu nhận thành công (ACK) hoặc thất bại (NAK) của chunk.

3.5 Tổ chức dữ liệu

Việc tổ chức dữ liệu đóng vai trò quan trọng trong việc quản lý thông tin và đảm bảo hiệu suất của ứng dụng. Dưới đây là chi tiết cách dữ liệu được tổ chức ở cả phía server và client:

3.5.1 Tổ chức dữ liệu phía Server

Server chịu trách nhiệm quản lý danh sách các file sẵn sàng để tải và cung cấp dữ liệu file theo yêu cầu của client. Dữ liệu được tổ chức như sau:

- **self.available_files (dictionary):** Đây là cấu trúc dữ liệu chính để quản lý thông tin về các file trên server.

- **Key:** Tên file (kiểu `string`). Ví dụ: `"file1.txt"`, `"image.jpg"`.
- **Value:** Kích thước file (kiểu `int`, số byte). Ví dụ: 1024, 5120.

Dictionary này được khởi tạo bằng hàm `self.scan_available_files()`. Hàm này thực hiện các công việc sau:

- Đọc thông tin từ các file trong thư mục `server_files`.
- Với mỗi file, lấy tên và kích thước.
- Lưu thông tin vào dictionary `self.available_files`.
- Ghi thông tin ra file `data.txt` (để log lại thông tin).

- **Ví dụ:** Nếu thư mục `server_files` chứa hai file `file1.txt` (kích thước 1024 byte) và `image.jpg` (kích thước 5120 byte), thì `self.available_files` sẽ có dạng:

```
{ "file1.txt": 1024, "image.jpg": 5120 }
```

- **Lưu trữ file thực tế:** Các file thực tế được lưu trữ trong thư mục `server_files`. Server chỉ lưu trữ tên và kích thước file trong `self.available_files`, không quản lý metadata (dữ liệu về dữ liệu) chi tiết khác của các file này.

3.5.2 Tổ chức dữ liệu phía Client

Client chịu trách nhiệm quản lý danh sách các file cần tải, thông tin về các file có sẵn trên Server và dữ liệu đã tải về. Dữ liệu được tổ chức như sau:

- **self.available_files (dictionary):** Lưu trữ thông tin về các file có sẵn trên server mà Client nhận được từ server trong thông điệp "Danh sách file". Cấu trúc tương tự như `self.available_files` trên Server:

- **Key:** Tên file (kiểu `string`).
- **Value:** Kích thước file (kiểu `int`).

- **self.downloaded_files (set):** Lưu trữ tên của các file đã được tải xuống thành công. Set được sử dụng vì nó đảm bảo mỗi file chỉ được lưu một lần, giúp tránh việc tải lại các file đã tải.
- **Lưu trữ dữ liệu đã tải về:**
 - **File tạm thời (các file .part):** Trong quá trình tải, client chia mỗi file thành 4 phần và lưu tạm thời mỗi phần vào một file riêng biệt với định dạng `file_name.part[number]` trong thư mục `downloads`. Ví dụ: `file1.txt.part0`, `file1.txt.part1`, v.v.
 - **File hoàn chỉnh:** Sau khi tất cả các phần của một file được tải xong, client sẽ ghép các file `.part` này lại thành file hoàn chỉnh và lưu vào thư mục `downloads` với tên file gốc. Sau đó, các file `.part` sẽ bị xóa.

4 Môi trường lập trình

Chúng tôi sẽ tổng hợp và trình bày chi tiết về "Môi trường lập trình" được sử dụng, bao gồm ngôn ngữ Python, IDE VSCode, quản lý mã nguồn bằng Git/GitHub, và các thư viện/module.

4.1 Tổng quan mã nguồn

Đồ án thực hành “Lập trình socket” được phát triển bằng ngôn ngữ lập trình Python phiên bản 3.12.4 được xây dựng trên cả hệ điều hành Windows 11 và MacOS Sequoia.

Trong quá trình phát triển, chúng tôi đã sử dụng môi trường Text Editor Visual Studio Code.

Để quản lý và chia sẻ mã nguồn, chúng tôi đã sử dụng GitHub. Đây là một nền tảng lưu trữ mã nguồn phổ biến, cho phép chúng tôi cộng tác với các thành viên khác và theo dõi các thay đổi được thực hiện trên mã nguồn.

4.2 Các thư viện và module sử dụng trong mã nguồn

Dựa trên code phần 2, các thư viện và module chính được sử dụng là:

Thư viện	Mô tả
<code>socket</code>	Module cốt lõi cho lập trình mạng trong Python. Hỗ trợ cả TCP và UDP. Được sử dụng để tạo socket UDP, gửi và nhận dữ liệu.
<code>threading</code>	Module hỗ trợ lập trình đa luồng. Được sử dụng để xử lý nhiều kết nối đồng thời (ở server) và tải nhiều phần của file đồng thời (ở client).
<code>os</code>	Module cung cấp các hàm tương tác với hệ điều hành. Dùng để xử lý đường dẫn file, kiểm tra sự tồn tại của file/thư mục, lấy kích thước file.
<code>json</code>	Module hỗ trợ làm việc với định dạng JSON. Dùng để mã hóa danh sách file thành chuỗi JSON để gửi từ server cho client.
<code>time</code>	Module cung cấp các hàm liên quan đến thời gian. Dùng để tạm dừng chương trình (<code>time.sleep()</code>) và xử lý timeout cho socket.
<code>struct</code>	Module hỗ trợ làm việc với dữ liệu nhị phân. Dùng để đóng gói và giải mã dữ liệu chunk với định dạng cụ thể, giúp tối ưu hiệu năng truyền dữ liệu.
<code>logging</code>	Module hỗ trợ ghi log (nhật ký hoạt động) của chương trình. Giúp debug và theo dõi hoạt động của ứng dụng.
<code>signal</code>	Module xử lý các tín hiệu hệ thống, ví dụ như <code>SIGINT</code> (Ctrl+C) để tắt chương trình một cách an toàn.

Bảng 3: Các thư viện và module sử dụng trong Python

5 Hướng dẫn sử dụng các tính năng

Các tính năng cho người dùng Client của hai giao thức TCP và UDP là như nhau, hướng tới mục đích là có thể download file một cách toàn vẹn dữ liệu từ Server. Dưới đây là hướng dẫn sử dụng các tính năng của chương trình:

5.1 Cài đặt Python và VSCode

Cài đặt Python

- **Đối với Windows:**

1. Truy cập trang web chính thức của Python: <https://www.python.org/downloads/windows/>.
2. Tải phiên bản Python 3 về máy.
3. Chạy file cài đặt đã tải về.
4. **Quan trọng:** Đánh dấu chọn vào ô "Add Python to PATH" trong quá trình cài đặt. Điều này giúp bạn có thể chạy Python từ command prompt hoặc terminal.
5. Hoàn tất cài đặt.

- **Đối với macOS:**

1. Có hai cách để tải ngôn ngữ Python về macOS, đầu tiên là lên hẳn trang web chính chủ của Python và tải theo hướng dẫn trên đó, cách thứ hai là sử dụng Homebrew. Ở đây, chúng tôi sẽ hướng dẫn cài đặt Python bằng homebrew.
2. Cài đặt homebrew bằng lệnh sau trong Terminal:

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

3. Sau khi cài Homebrew, cài đặt Python bằng lệnh:

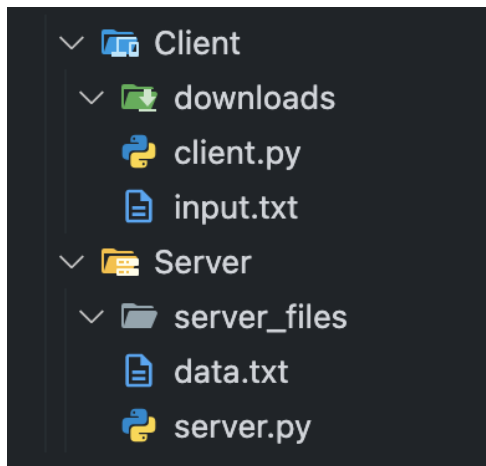
```
brew install python3
```

Cài đặt VSCode

1. Truy cập trang web chính thức của VSCode: <https://code.visualstudio.com/>.
2. Tải phiên bản phù hợp với hệ điều hành của bạn.
3. Chạy file cài đặt và làm theo hướng dẫn.
4. Sau khi cài đặt VSCode, cài đặt extension "Python" từ Microsoft trong VSCode. Mở VSCode, nhấn **Ctrl+Shift+X** (hoặc **Cmd+Shift+X** trên macOS), tìm kiếm "Python" và cài đặt.

Chuẩn bị mã nguồn

1. Chuẩn bị một thư mục riêng để lưu trữ toàn bộ mã nguồn (có thể lên github để clone repositories qua link sau: <https://github.com/nphoang-itus/socket.git>). Thư mục **BẮT BUỘC** phải đầy đủ các phần như hình dưới đây:



Hình 5: Cấu trúc mã nguồn

Thư mục Client:

- Thư mục `downloads`: Thư mục chứa các file đã download
- `client.py`: Mã nguồn chính của client
- `input.txt`: Danh sách file yêu cầu download từ server

Thư mục Server:

- Thư mục `server_files`: Lưu trữ file cho phép client download.
- `data.txt`: Thông tin các file có trong `server_files`.
- `server.py`: Mã nguồn chính của server

2. Biên dịch và chạy chương trình:

Để biên dịch và chạy chương trình ta sử dụng command prompt hoặc terminal, ta sử dụng python3, ứng với mỗi client và server sẽ mở trình biên dịch riêng và thực hiện các bước như nhau. Tuy nhiên, yêu cầu server phải được khởi động trước và chờ yêu cầu từ các client. Mô tả các bước như sau:

Bước 1: Sử dụng 2 thiết bị hoặc 2 tab terminal (hoặc command prompt) cho Client và Server. Cần di chuyển vào thư mục TCP/Client và TCP/Server (nếu như sử dụng giao thức TCP) hoặc UDP/Client và UDP/Server (nếu như sử dụng giao thức UDP) tương ứng bằng lệnh `cd`.

Bước 2: Nhập lệnh sau vào terminal hoặc command prompt:

Đối với Server: `python3 server.py`

Đối với Client: `python3 client.py`

Trong cú pháp trên:

- `python3`: Chỉ định trình biên dịch sử dụng python3 để dịch chương trình.
- `server.py` và `client.py` là các file thực thi chính.

Như vậy, chương trình đã được biên dịch và khởi động thành công, Theo dõi phần Sử dụng các chức năng của chương trình để tiếp tục.

5.2 Sử dụng các chức năng

LƯU Ý QUAN TRỌNG: Nếu như sử dụng hai máy tính (1 máy làm Client và 1 máy làm Server) thì phải đảm bảo được hai yếu tố sau:

- **Thứ nhất**, đảm bảo cả hai thiết bị nằm trong cùng subnet. Tắt AP Isolation trên router nếu sử dụng mạng WiFi bởi vì nhiều router hiện đại ngăn các thiết bị trong cùng một mạng giao tiếp trực tiếp với nhau để tăng cường bảo mật. Nếu tính năng này được bật, các gói tin giữa hai thiết bị sẽ bị router chặn, khiến kết nối socket thất bại.
- **Thứ hai**, nếu sử dụng chương trình truyền tải file bằng UDP phải đảm bảo rằng cả hai máy đều tắt firewall (tường lửa) bởi vì trong quá trình thực hiện đồ án, nhóm chúng tôi phát hiện khi chia Thread ở bên Server, mỗi Thread sẽ tạo một socket và gửi trực tiếp tới Thread của Client, khi này nếu như không tắt tường lửa do không có phương thức bind() đến port Thread tương ứng thì tường lửa sẽ thực hiện chặn các gói tin UDP không rõ nguồn gốc (khi dùng bind thì chương trình sẽ biết rõ gói tin này từ đâu tới nên sẽ hạn chế việc bị tường lửa chặn) dẫn đến file không thể tải khi có tường lửa. [1]

Hai vấn đề trên trong quá trình thực hiện đồ án nhóm chúng tôi đã phát hiện ra và đã tìm được phương án thích hợp để truyền tải dữ liệu là sử dụng một thiết bị trung tâm là chiếc điện thoại để phát mạng cho hai máy. Khi này hai thiết bị sẽ sử dụng máy điện thoại làm trung gian truyền tải dữ liệu do máy điện thoại lúc này đóng vai trò như một Access Point (AP), tạo ra một mạng riêng (thường là mạng con - subnet) và cung cấp địa chỉ IP cho các thiết bị kết nối thông qua DHCP (Dynamic Host Configuration Protocol), thêm nữa firewall trên điện thoại ít bảo mật nên các gói tin có thể dễ dàng truyền tải trên đường mạng do điện thoại làm trung tâm.

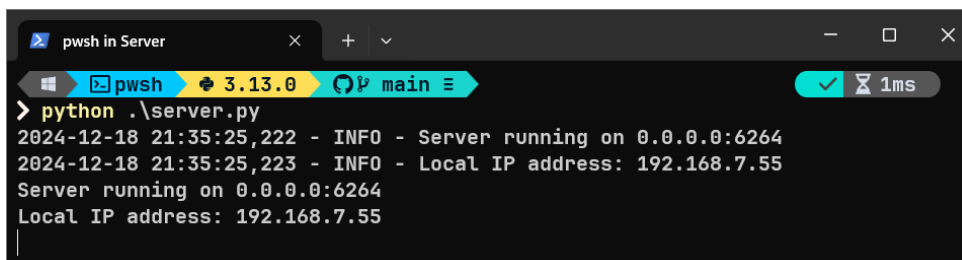
Còn nếu như test trên một thiết bị, vì chỉ truyền dữ liệu trong local nên máy không bị hạn chế bởi tường lửa, chính vì vậy dữ liệu có thể truyền tải dễ dàng mà không bị kiểm soát.

Dưới đây là các bước hướng dẫn sử dụng các tính năng của chương trình:

Bước 1: Khởi tạo kết nối giữa Server và Client

- **Server:** Server mở lời mời kết nối broadcast (Đến tất cả các máy Client có trong khu vực mạng) tại IP 0.0.0.0 với PORT 6264. Sau đó tự động lấy địa chỉ IP local của máy Server gắn vào local_ip.

Ví dụ hiển thị trên màn hình:



```
pwsh in Server
> python .\server.py
2024-12-18 21:35:25,222 - INFO - Server running on 0.0.0.0:6264
2024-12-18 21:35:25,223 - INFO - Local IP address: 192.168.7.55
Server running on 0.0.0.0:6264
Local IP address: 192.168.7.55
```

Hình 6: Server mở kết nối

```

pwsh in Server
> python .\server.py
2024-12-18 21:35:25,222 - INFO - Server running on 0.0.0.0:6264
2024-12-18 21:35:25,223 - INFO - Local IP address: 192.168.7.55
Server running on 0.0.0.0:6264
Local IP address: 192.168.7.55
New connection from ('192.168.7.55', 50235)
2024-12-18 21:39:37,166 - INFO - New connection from ('192.168.7.55', 50235)
2024-12-18 21:39:37,168 - INFO - New connection from ('192.168.7.55', 50235)
New connection from ('192.168.7.55', 50235)

```

Hình 7: Server khi kết nối thành công với Client

- **Client:** Trên màn hình Client, thực hiện nhập địa chỉ IP (nhập local_ip) và PORT của Server mà Client muốn kết nối.

Nếu như kết nối thành công thì Client sẽ hiển thị "Connected to server" và Server sẽ hiển thị "New connection from ('IP_CLIENT', PORT_CLIENT)". Nếu thất bại có thể là server chưa khởi động hoặc người dùng bên Client nhập sai IP hoặc PORT thì chương trình sẽ thực hiện cho nhập lại IP và PORT.

Sau khi kết nối thành công thì Server sẽ gửi danh sách Files cho phép Client download và Client sẽ hiển thị danh sách đó lên màn hình.

Ví dụ hiển thị trên màn hình:

```

Nhập IP server: 192.168.7.55
Nhập cổng server: 6265
Error connecting to server: [WinError 10061] No connection could be made because the target machine actively refused it
Nhập IP server: 192.168.7.55
Nhập cổng server: 6264
Trying to reconnect in 5 seconds...

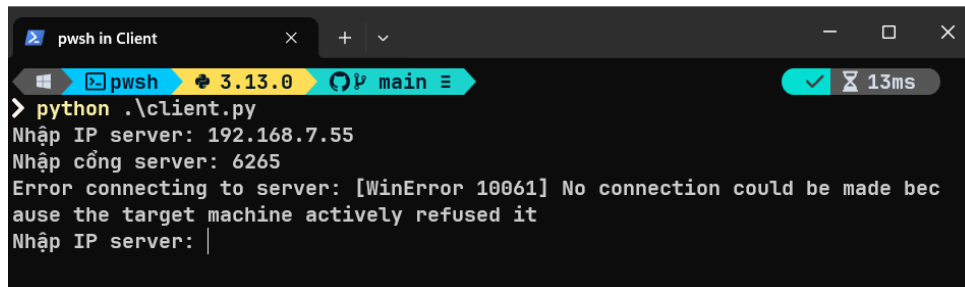
Connected to server.
-----
Available files on the server:
BTC_T-100.JPG: 4.12MB
File 2.zip: 10.00MB
File1.zip: 5.00MB
File16B.bin: 1.00GB
-----

Monitoring input.txt for download requests.
-----
Files not found on the server:

No new files to download!
-----

```

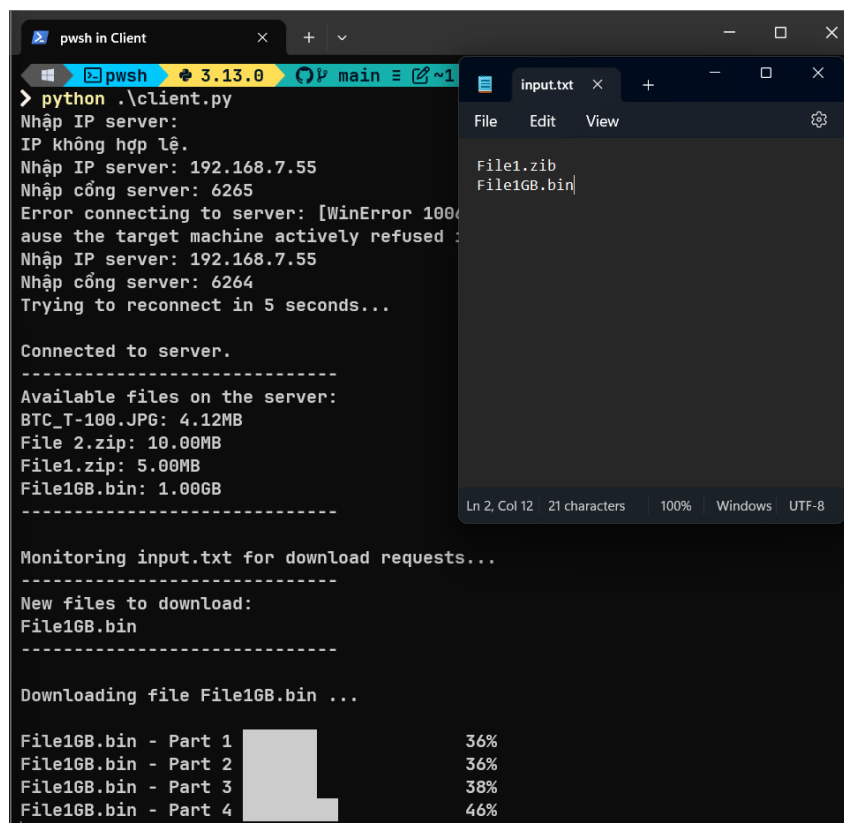
Hình 8: Client nhập IP của Server và kết nối thành công



```
pwsh in Client
> python .\client.py
Nhập IP server: 192.168.7.55
Nhập cổng server: 6265
Error connecting to server: [WinError 10061] No connection could be made bec
ause the target machine actively refused it
Nhập IP server: |
```

Hình 9: Client nhập sai IP của Server, thực hiện nhập lại

Bước 2: Client sẽ thực hiện nhập tên file muốn download vào dòng mới của file input.txt. Sau đó Client sẽ quét file text này để bắt đầu thực hiện download từng file. Đối với các yêu cầu mà sai tên file thì Client sẽ thực hiện báo warning lên màn hình hay nhập các file đã download về rồi thì Client sẽ bỏ qua.



```
pwsh in Client
> python .\client.py
Nhập IP server:
IP không hợp lệ.
Nhập IP server: 192.168.7.55
Nhập cổng server: 6265
Error connecting to server: [WinError 10061] No connection could be made bec
ause the target machine actively refused it
Nhập IP server: 192.168.7.55
Nhập cổng server: 6264
Trying to reconnect in 5 seconds...

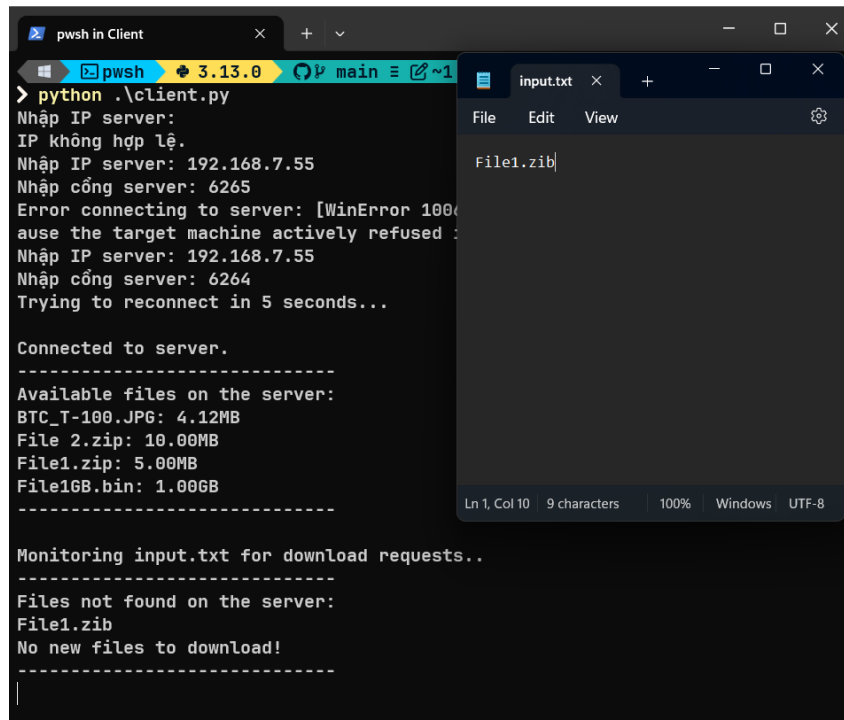
Connected to server.
-----
Available files on the server:
BTC_T-100.JPG: 4.12MB
File 2.zip: 10.00MB
File1.zip: 5.00MB
File16B.bin: 1.00GB
-----

Monitoring input.txt for download requests...
-----
New files to download:
File16B.bin
-----

Downloading file File16B.bin ...

File16B.bin - Part 1 36%
File16B.bin - Part 2 36%
File16B.bin - Part 3 38%
File16B.bin - Part 4 46%
```

Hình 10: Client nhập tên file muốn tải vào input.txt



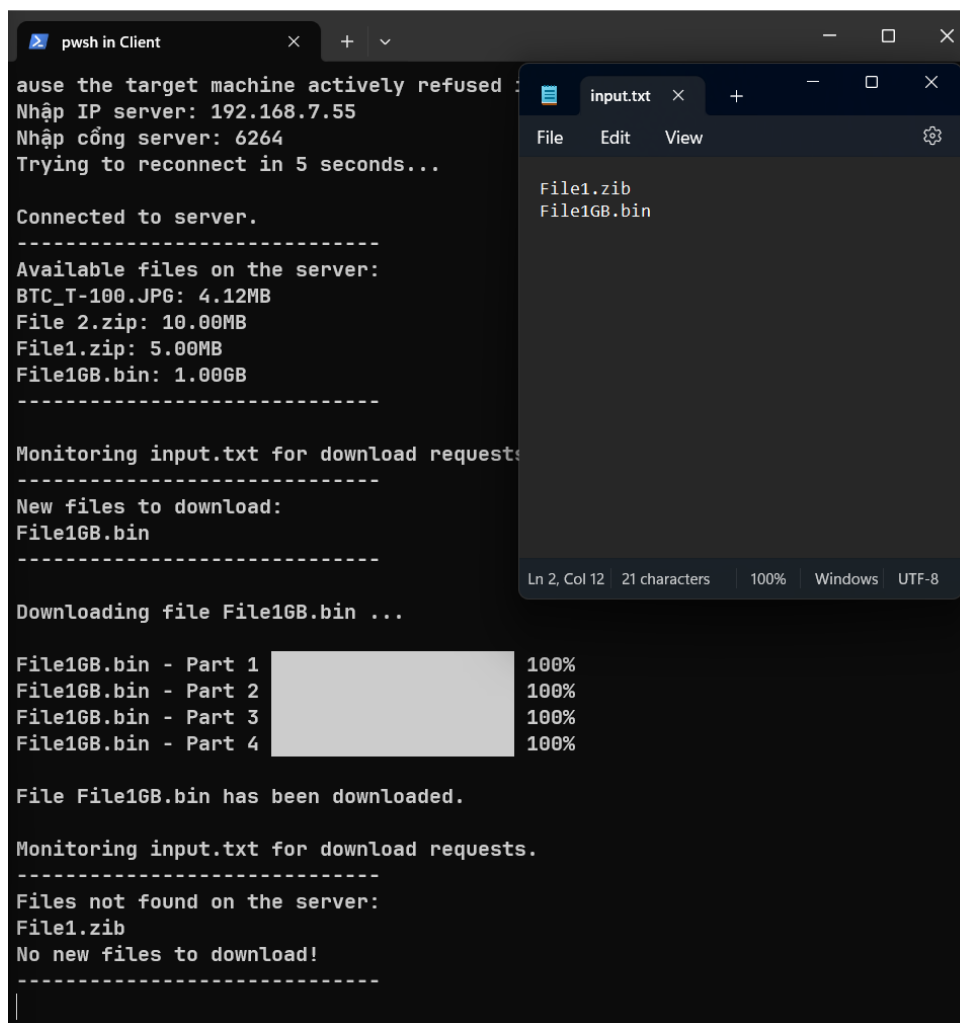
```
pwsh in Client
> python .\client.py
Nhập IP server:
IP không hợp lệ.
Nhập IP server: 192.168.7.55
Nhập cổng server: 6265
Error connecting to server: [WinError 10061]
ause the target machine actively refused :
Nhập IP server: 192.168.7.55
Nhập cổng server: 6264
Trying to reconnect in 5 seconds...

Connected to server.
-----
Available files on the server:
BTC_T-100.JPG: 4.12MB
File 2.zip: 10.00MB
File1.zip: 5.00MB
File1GB.bin: 1.00GB
-----

Monitoring input.txt for download requests..
-----
Files not found on the server:
File1.zip
No new files to download!
-----
|
```

Hình 11: Nếu như file Client nhập không có trên server sẽ thông báo

Bước 3: Client gửi yêu cầu cho Server và bắt đầu thực hiện download file theo như mô tả ở phần 2.1 đối với giao thức TCP hoặc 3.1 đối với UDP.



```
pwsh in Client
ause the target machine actively refused :
Nhập IP server: 192.168.7.55
Nhập cổng server: 6264
Trying to reconnect in 5 seconds...

Connected to server.
-----
Available files on the server:
BTC_T-100.JPG: 4.12MB
File 2.zip: 10.00MB
File1.zip: 5.00MB
File1GB.bin: 1.00GB
-----

Monitoring input.txt for download requests
-----
New files to download:
File1GB.bin
-----

Downloading file File1GB.bin ...

File1GB.bin - Part 1 ██████████ 100%
File1GB.bin - Part 2 ██████████ 100%
File1GB.bin - Part 3 ██████████ 100%
File1GB.bin - Part 4 ██████████ 100%

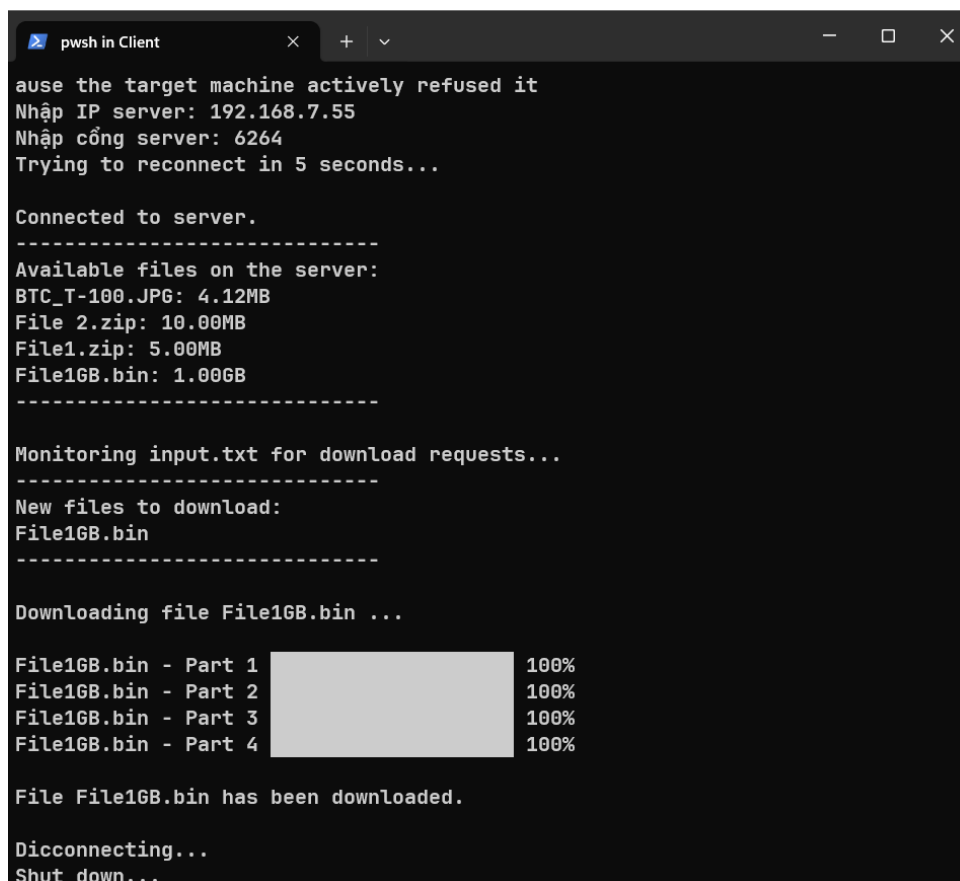
File File1GB.bin has been downloaded.

Monitoring input.txt for download requests.
-----
Files not found on the server:
File1.zib
No new files to download!
-----
```

Hình 12: Client download file thành công

Nếu như Client nhập tên file muốn download vào dòng mới của input.txt thì cứ mỗi 5s chương trình sẽ tự quét lại input.txt và thực hiện download.

Bước 4: Client muốn kết thúc chương trình sẽ bấm Ctrl+C khi này chương trình sẽ thực hiện dọn sạch socket và kết thúc chương trình. Tương tự với Server nếu như muốn kết thúc chương trình.



```
ause the target machine actively refused it
Nhập IP server: 192.168.7.55
Nhập cổng server: 6264
Trying to reconnect in 5 seconds...

Connected to server.
-----
Available files on the server:
BTC_T-100.JPG: 4.12MB
File 2.zip: 10.00MB
File1.zip: 5.00MB
File1GB.bin: 1.00GB
-----

Monitoring input.txt for download requests...
-----
New files to download:
File1GB.bin
-----

Downloading file File1GB.bin ...

File1GB.bin - Part 1 ██████████ 100%
File1GB.bin - Part 2 ██████████ 100%
File1GB.bin - Part 3 ██████████ 100%
File1GB.bin - Part 4 ██████████ 100%

File File1GB.bin has been downloaded.

Dicconnecting...
Shut down...
```

Hình 13: Nhập Ctr+C để kết thúc chương trình

Về cơ bản thì chỉ cần thực hiện các bước đơn giản là: Biên dịch, chạy chương trình bên Server trước và các Client sau; Client Nhập IP và PORT của Server muốn kết nối; Cập nhật tên file muốn tải vào input.txt; Kết thúc chương trình bằng Ctrl+C.

Tài liệu

- [1] Carmine Scarpitta. *Why can TCP packets pass through firewalls but UDP can't?* <https://www.quora.com/Why-can-TCP-packets-pass-through-firewalls-but-UDP-cant>. Tham khảo cách fix TCP không thể kết nối giữa hai máy.
- [2] saiganesh4. *Reliable-Data-Transfer-Applcation*. <https://github.com/saiganesh4/Reliable-Data-Transfer-Applcation/blob/main/server.py>. Tham khảo cài đặt cơ chế RDТ. Truy cập 09-12-2024.
- [3] Ths. Huỳnh Thuy Bảo Trân. *Transport Layer*. Bài giảng tiếng việt học kỳ 1 năm học 2024-2025. Tham khảo Kịch bản giao tiếp, cơ chế RDТ,... 2024.
- [4] Keith Ross Pearson Jim Kurose. *Chapter_v8.0 – TransportLayer.pptx*. Bài giảng tiếng anh học kỳ 1 năm học 2024-2025. Tham khảo Kịch bản giao tiếp, cơ chế RDТ,... 2024.
- [5] David M. Beazley. *Reliable-Data-Transfer-Applcation*. <http://www.dabeaz.com>. Học lập trình socket cho python. Truy cập 05-12-2024.
- [6] Python Website. *socket — Low-level networking interface*. <https://docs.python.org/3/library/socket.html#socket.socket.recv>. Tham khảo tất cả các hàm trong socket. Truy cập 06-12-2024.
- [7] Idiot Developer. *Multithreaded Client Server in Python || Socket Programming in Python*. https://www.youtube.com/watch?v=xceTFWy_eag. Học cài đặt thread cho python.
- [8] John Humphreys. *UDP cannot connect to anything other than 127.0.0.1*. <https://stackoverflow.com/questions/8126519/udp-cannot-connect-to-anything-other-than-127-0-0-1>. Tham khảo cách fix bug không thể kết nối các địa chỉ khác ngoại trừ địa chỉ loopback.
- [9] knittl. *How can I find out where a UDP connection fails?* <https://serverfault.com/questions/366270/how-can-i-find-out-where-a-udp-connection-fails>. Tham khảo cách fix không thể kết nối giữa các socket trong giao thức UDP.
- [10] wikipedia. *ANSI escape code*. https://en.wikipedia.org/wiki/ANSI_escape_code. Tham khảo dịch chuyển con trỏ để in ra màn hình.