

BÁO CÁO PROJECT SOCKET

Người thực hiện: Dũ Quốc Huy

Thành phố Hồ Chí Minh, tháng 04 năm 2024

Mục Lục

1	Mô tả dự án	2
2	Công nghệ được sử dụng	2
2.1	Môi trường phát triển	2
2.2	Boost	2
2.3	OpenSSL	2
2.4	UTF8 <=> UTF16	2
2.5	MultiThread	3
3	Các chức năng chính của đồ án	3
3.1	Tạo kết nối	3
3.1.1	Mô tả	3
3.1.2	Minh họa	3
3.2	Gửi text	4
3.2.1	Mô tả	4
3.2.2	Quy trình xử lý	4
3.2.3	Minh họa	5
3.3	Gửi file	5
3.3.1	Mô tả	5
3.3.2	Quá trình xử lý	5
3.3.3	Minh họa	6
3.4	Kết nối nhiều Client	7
3.4.1	Mô tả	7
3.4.2	Minh họa	7
4	Tự đánh giá	7
4.1	Ưu điểm	7
4.2	Nhược điểm	7
	Tài liệu tham khảo	8

1. Mô tả dự án

- Xây dựng chương trình tương tác giữa máy gửi (Client) và máy nhận để xử lý các yêu cầu (Server) sử dụng giao thức TCP. Giao diện được thực hiện thông qua giao diện dòng lệnh (Command line tool)
- Đảm bảo thực hiện được các yêu cầu
 - Thiết lập kết nối (senddata, receivedata)
 - Gửi text (sendtext)
 - Gửi file (sendfile)
- Một số tính năng nâng cao:
 - Khả năng gửi text kèm Unicode và Emotion
 - Thực hiện thao tác khác trong quá trình gửi file
 - Server cho phép nhiều kết nối/ yêu cầu, khả năng xử lý dữ liệu đảm bảo tính bảo mật, an toàn và toàn vẹn dữ liệu.
- Người sử dụng thực hiện gửi các request thông qua giao diện terminal của Client với các lệnh được quy định trước. (Xem thông tin từng lệnh bên dưới)

2. Công nghệ được sử dụng

2.1. Môi trường phát triển

- Môi trường: Windows
- Ngôn ngữ C++ (C++17 Standard)
- Công cụ: Visual studio

2.2. Boost

Qua quá trình tìm hiểu, dự án này sử dụng Boost.asio để thực hiện xây dựng kết nối TCP đồng thời xử lý kết nối và gửi nhận dữ liệu bất đồng bộ.

2.3. OpenSSL

Để đảm bảo tính toàn vẹn và bảo mật dữ liệu, dự án có áp dụng mã hóa SHA256 (từ OpenSSL) cho việc mã hóa text và file trong quá trình gửi và nhận.

2.4. UTF8 <=> UTF16

Việc nhận và hiển thị thông tin, xử lý dữ liệu gửi và nhận từ Client đều sử dụng các phương thức chuyển đổi linh hoạt giữa string <=> wstring. Giao diện được thiết lập ở hiển thị UTF16. Đảm bảo nhập và hiển thị được UNICODE.

2.5. MultiThread

Để đảm bảo các yêu cầu: trong quá trình gửi file có thể nhận các request khác ở Client; Server có khả năng tiếp nhận nhiều kết nối, xử lý các yêu cầu cùng 1 lúc. Dự án áp dụng tính đa luồng và xử lý bất đồng bộ có sẵn từ Boost.

3. Các chức năng chính của đồ án

3.1. Tạo kết nối

3.1.1. Mô tả

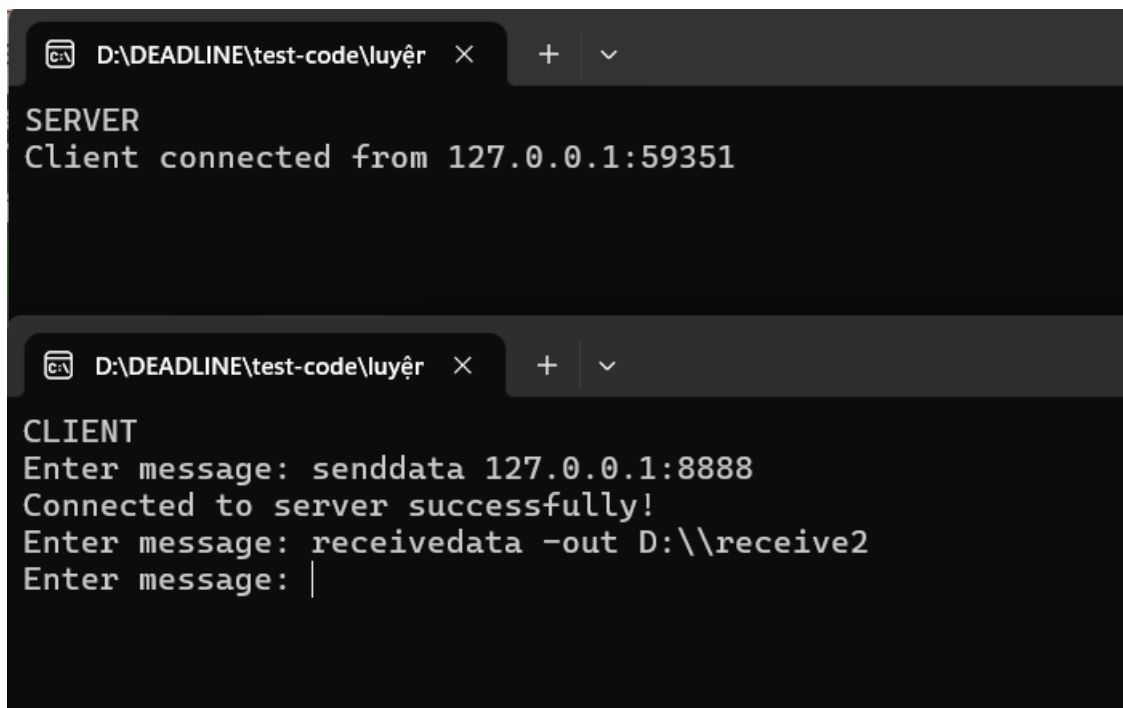
Để khởi tạo kết nối từ Client đến Server cần 2 yêu cầu. Đảm bảo Server đã được khởi động sẵn chờ Client kết nối. Lưu ý: Server được mặc định chạy ở IPV4, port 8888.

- Tạo kết nối: Với giao diện của Client, người dùng nhập lệnh: **senddata <ip>:<port>**. Nếu thông tin kết nối đúng, Client có thể kết nối được đến Server.
- Yêu cầu thư mục lưu file của Client: người dùng nhập lệnh: **receivedata -out <path>**.

Phía Server khi nhận được yêu cầu sẽ thực hiện như sau:

- Luôn giữ trạng thái chờ Client kết nối
- Khi nhận được kết nối từ Client, xuất ra màn hình thông tin Client kết nối và tạo ra luồng để xử lý kết nối từ Client đó.
- Khi nhận được yêu cầu **receivedata -out <path>**. Server thực hiện gán path để lưu các file sắp tới được nhận từ Client.

3.1.2. Minh họa



```
D:\DEADLINE\test-code\luyện tập > SERVER
Client connected from 127.0.0.1:59351

D:\DEADLINE\test-code\luyện tập > CLIENT
Enter message: senddata 127.0.0.1:8888
Connected to server successfully!
Enter message: receivedata -out D:\\receive2
Enter message: |
```

Hình 1: Tạo kết nối ban đầu

3.2. Gửi text

3.2.1. Mô tả

Nội dung của văn bản gửi đi có thể là ASCII, không giới hạn số lượng nhập vào. Ngoài ra có thể gửi thêm Unicode, Emotion.

Để xác thực dữ liệu từ Client được toàn vẹn có thể sử dụng việc mã hóa file trước và gửi hash qua Server để Server sau khi nhận dữ liệu text thì so sánh hash với nhau.

Để thực hiện yêu cầu gửi đoạn văn, người dùng nhập lệnh **sendtext Dũ Quốc Huy**. Có thể nhập trực tiếp unicode trên màn hình terminal, còn với emotion, ở windows có thể dùng Windows + . để mở hộp thoại và chọn Emotion muốn gửi

3.2.2. Quy trình xử lý

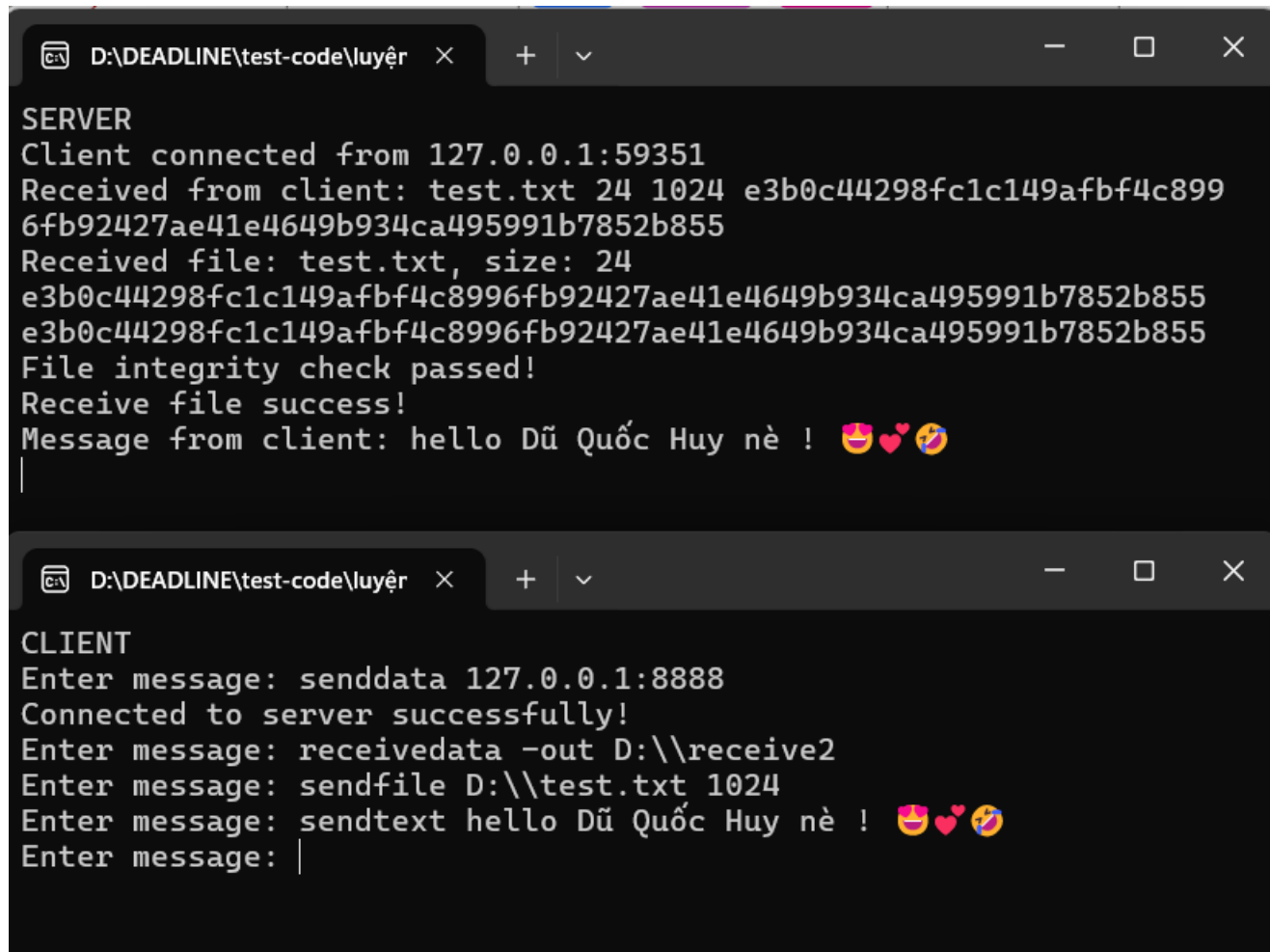
Client khi nhận được lệnh sendtext từ người dùng

1. Đọc dữ liệu text và tạo một hashed_text cho text ban đầu
2. Gửi message lần 1: sendtext <text_size> <hashed_text>
3. Gửi message lần 2: dữ liệu chỉ chứa text mà người dùng muốn gửi

Server khi nhận được yêu cầu sendtext:

1. Đọc dữ liệu từ message lần 1: text_size để tạo buffer cho việc nhận text, hashed_text để so sánh với dữ liệu lát sau nhận được
2. Đọc dữ liệu từ message lần 2: đọc và hash sau đó so sánh với hashed_text từ Client để đảm bảo dữ liệu nhận được đầy đủ.

3.2.3. Minh họa



The image shows two terminal windows. The top window, titled 'D:\DEADLINE\test-code\luyện', displays the server's output. It shows a client connection from 127.0.0.1:59351, receipt of a file named 'test.txt' (24 bytes), a successful integrity check, and a message from the client: 'hello Dĩ Quốc Huy nè !' followed by three emojis (🥰, ❤️, 🌈). The bottom window, also titled 'D:\DEADLINE\test-code\luyện', displays the client's input. It shows the user entering the server address '127.0.0.1:8888', connecting successfully, and then entering the commands 'receivedata -out D:\\receive2', 'sendfile D:\\test.txt 1024', and 'sendtext hello Dĩ Quốc Huy nè !' followed by the same three emojis. The prompt 'Enter message:' is visible at the end of the input line.

```
SERVER
Client connected from 127.0.0.1:59351
Received from client: test.txt 24 1024 e3b0c44298fc1c149afb4c899
6fb92427ae41e4649b934ca495991b7852b855
Received file: test.txt, size: 24
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
File integrity check passed!
Receive file success!
Message from client: hello Dĩ Quốc Huy nè ! 🥰❤️🌈

CLIENT
Enter message: senddata 127.0.0.1:8888
Connected to server successfully!
Enter message: receivedata -out D:\\receive2
Enter message: sendfile D:\\test.txt 1024
Enter message: sendtext hello Dĩ Quốc Huy nè ! 🥰❤️🌈
Enter message: |
```

Hình 2: Gửi dữ liệu text

3.3. Gửi file

3.3.1. Mô tả

Người dùng có thể thực hiện gửi file đến server (quy định path lưu file ở server từ lệnh connect ở trên). File gửi có khả năng không giới hạn kích thước, lên đến Gigabyte, trong quá trình gửi file có khả năng thực hiện thêm các request khác, dữ liệu từ file đảm bảo tính toàn vẹn của tập tin gửi.

Để thực hiện gửi file, người dùng có thể dùng lệnh **sendfile <path> <buffer_size>**. Trong đó, path là đường dẫn đến tập tin của Client, buffer_size là kích thước gói tin mà người dùng muốn chia.

3.3.2. Quá trình xử lý

Khi Client nhận được yêu cầu như trên, sẽ thực hiện theo quá trình sau:

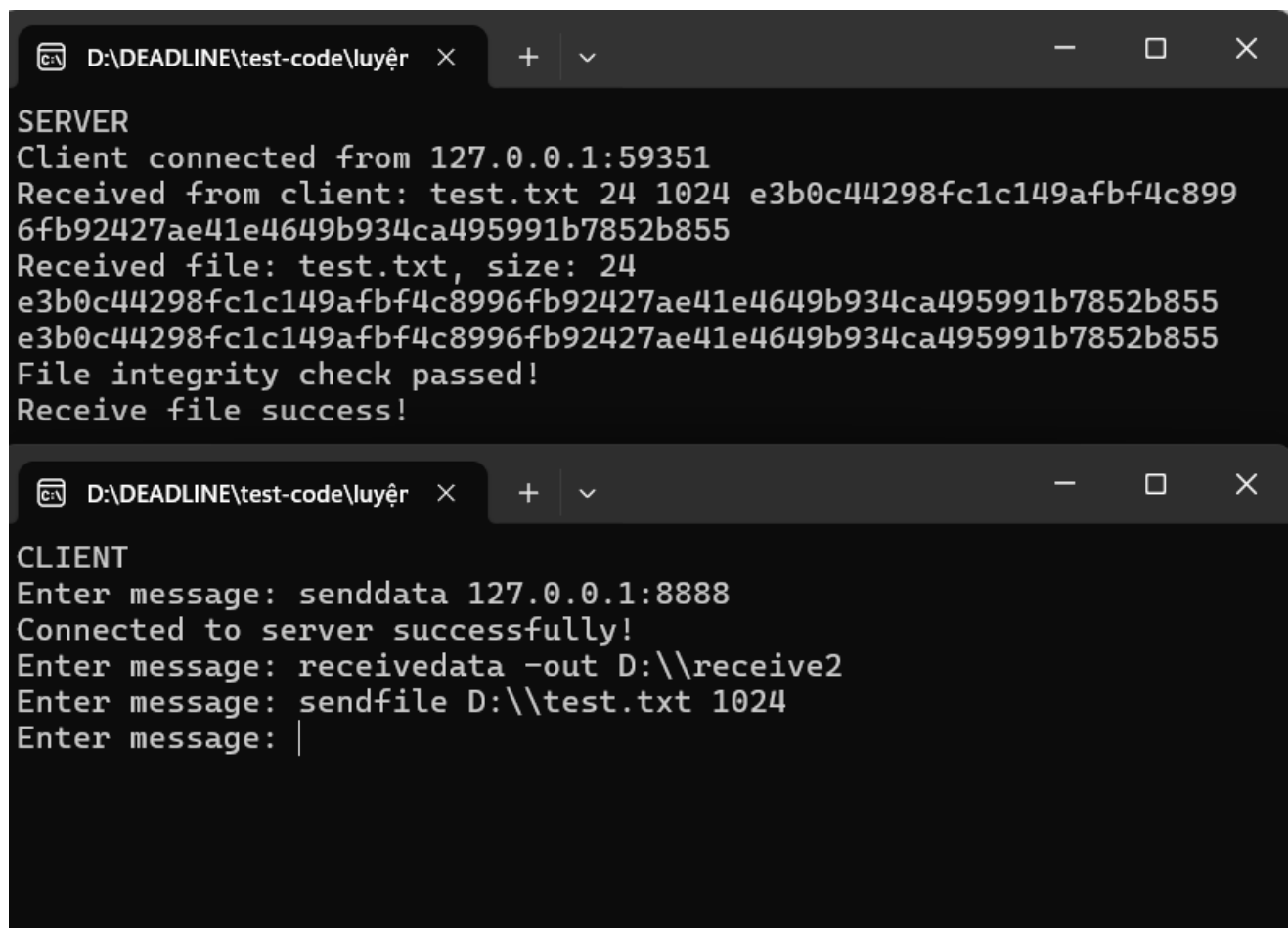
1. Thực hiện lấy từng thông tin đầu vào
2. Kiểm tra đường dẫn file

3. Lấy kích thước file
4. Lấy tên file (dùng để tạo file tương thích ở Server)
5. Hash dữ liệu file (cho việc đảm bảo toàn vẹn dữ liệu)
6. Send message lần 1 đến Server: `sendfile <file_name> <file_size> <buffer_size> <hashed_file>`
7. Send message nhiều lần đến Server: chia dữ liệu file thành từng gói với kích thước buffer size và gửi từng gói sang Server.

Khi Server nhận được yêu cầu có `command = "sendfile"`, sẽ thực hiện:

1. Phân tích dữ liệu message lần 1 của Client để lấy thông tin: `file_name`, `file_size`, `buffer_size`, `hashed_file`.
2. Dựa vào `buffer_size` để tạo buffer nhận từng gói tin, `file_size` để xác định khi nào nhận đủ dữ liệu, tạo `file_name` trong thư mục đã được quy định ở bước Connect.
3. Thực hiện đọc toàn bộ dữ liệu dựa vào các message sau. Với mỗi dữ liệu đọc được sẽ thực hiện ghi vào file.
4. Kiểm tra hash file vừa nhận được với hash nhận được từ Client để đảm bảo file đúng dữ liệu.

3.3.3. Minh họa



```
SERVER
Client connected from 127.0.0.1:59351
Received from client: test.txt 24 1024 e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Received file: test.txt, size: 24
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
File integrity check passed!
Receive file success!

CLIENT
Enter message: senddata 127.0.0.1:8888
Connected to server successfully!
Enter message: receivedata -out D:\\receive2
Enter message: sendfile D:\\test.txt 1024
Enter message: |
```

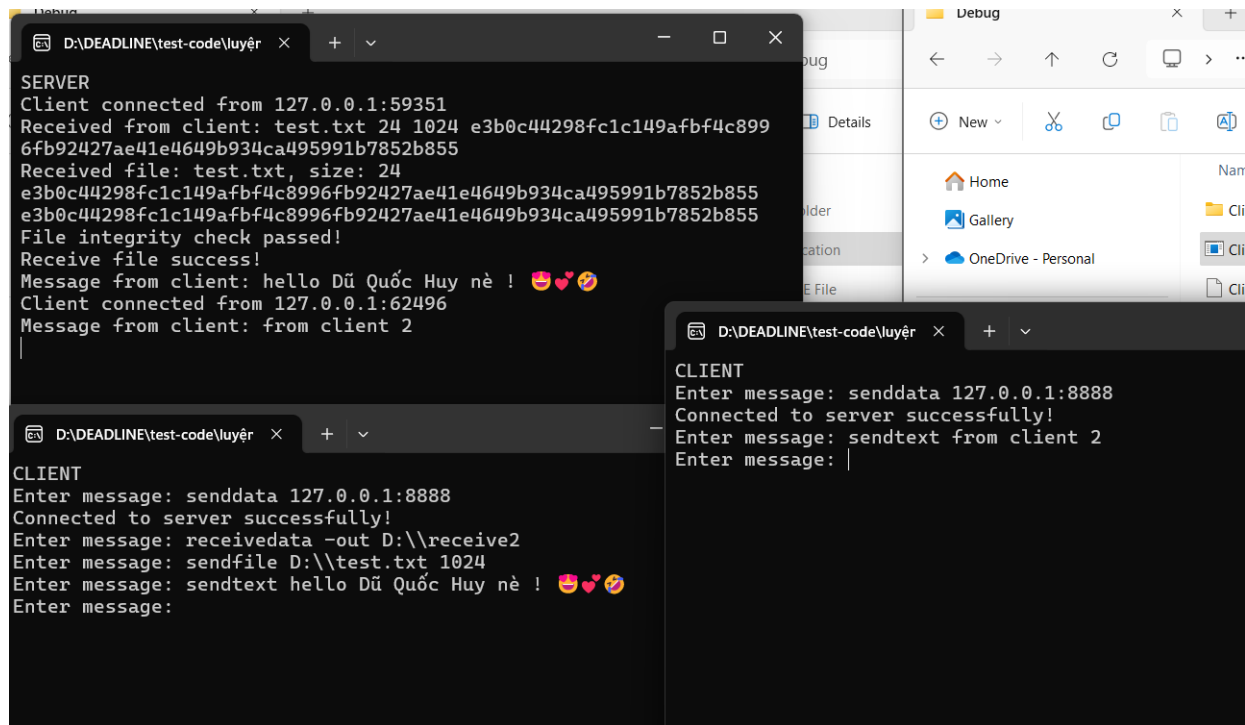
Hình 3: Gửi file đến Server

3.4. Kết nối nhiều Client

3.4.1. Mô tả

Khi Server đang được chạy có thể cho phép nhiều Client kết nối đến.

3.4.2. Minh họa



```
SERVER
Client connected from 127.0.0.1:59351
Received from client: test.txt 24 1024 e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Received file: test.txt, size: 24
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
File integrity check passed!
Receive file success!
Message from client: hello Dũ Quốc Huy nè ! 🍷💕🍷
Client connected from 127.0.0.1:62496
Message from client: from client 2

CLIENT
Enter message: senddata 127.0.0.1:8888
Connected to server successfully!
Enter message: receivedata -out D:\\receive2
Enter message: sendfile D:\\test.txt 1024
Enter message: sendtext hello Dũ Quốc Huy nè ! 🍷💕🍷
Enter message:

CLIENT
Enter message: senddata 127.0.0.1:8888
Connected to server successfully!
Enter message: sendtext from client 2
Enter message:
```

Hình 4: Nhiều Client kết nối đến Server

4. Tự đánh giá

Đây là đánh giá tạm thời của dự án tính tới ngày 31/03/2024:

4.1. Ưu điểm

Đã có thể thực hiện được các chức năng cơ bản được nêu ra ở trên như: tạo kết nối, gửi text, gửi file, xử lý lỗi và dữ liệu.

Ứng dụng được kiến thức về C++, xử lý bất đồng bộ, đa luồng, xử lý lỗi, trao đổi dữ liệu đảm bảo tính toàn vẹn và bảo mật (ở mức cơ bản), trao đổi thông qua giao thức TCP.

4.2. Nhược điểm

Việc sử dụng các chức năng được cung cấp bởi Boost vẫn còn hạn chế, chưa áp dụng được việc xử lý đa luồng hiệu quả trong việc xử lý nhiều tác vụ (sử dụng ThreadPool).

Dự án này sẽ tiếp tục được phát triển để vận dụng kiến thức trong quá trình tìm hiểu.

Tài liệu tham khảo

- [1] [Thư viện Boost.asio](#)
- [2] [Thư viện OpenSSL](#)
- [3] Các công cụ AI để hỗ trợ thông tin trong quá trình làm như Copilot và ChatGPT.