

Multi Threaded Chat Server

Written by:

Mr. Shalev Atsis

B.Sc. Computer Science (Expected 2026)

HIT – Holon Institute of Technology



[Shalev Atsis](#)



+972-58-5060699



Shalevatsis@gmail.com

Project Overview

This project implements a distributed chat system using Python and TCP sockets. The server can handle multiple clients simultaneously and facilitates both public and private messaging. The system demonstrates key principles of network communication and distributed systems.

Files included

- **server.py:** Python script for running the server.
- **server.exe:** Executable version of the server.
- **client.py:** Python script for running the client.
- **client.exe:** Executable version of the client.
- **Multi Threaded Chat Server.pdf:** Current file.
- **chat_documentation.txt: ONLY AFTER RUNNING THE PROGRAM –** message logging of latest session.

Program explanation and code description

This project consists of two main components: a server and a client. Both implemented using Python's socket and threading libraries, with the server using the TCP protocol for communication.

The system implements bi-directional communication between the clients and the server, ensuring both sending and receiving messages in real time.

The server handles multiple client connections simultaneously. Communication between clients occurs in the form of private messages (direct messages between clients) and broadcast messages (messages sent to all connected clients). Additionally, all chat activity is logged into a file, and the server can be gracefully shut down.

The server listens to incoming client requests and acts as an intermediary between clients, ensuring that messages are delivered appropriately. Once a client connects to the server, the server establishes communication, which includes opening a chat between the client and another target client, whose name is specified during the request. Every client is uniquely identified by a name, which is used for handling private messages.

The client connects to the server, where they can send messages (both private and broadcast), and receive messages from the server and other clients in real time. Clients can also gracefully disconnect from the server by sending a specific exit command.

Server features

1. **Multithreaded client handling:** each client connection is handled in a separate thread to ensure the server can handle multiple clients simultaneously.
2. **Private and broadcast messaging:** Clients can send messages to all users (broadcast) or specific clients (private).
3. **Graceful shutdown:** The server can be shut down gracefully by typing 'Terminate server' in the console, ensuring all client connections are properly closed.
4. **Logging:** The server logs chat interactions into a file named 'chat_documentation.txt'.

Server Code breakdown:

a. Global Variables:

- *host* and *port*: Define the server's hostname and the port number to listen on.
- *clients*: A dictionary storing the client name as the key and the client connection as the value.
- *shutdown_event*: A threading event that signals when the server should shut down.
- *clients_lock*: A lock used to ensure thread safety when accessing the clients dictionary.

b. Server Socket Setup (*start_server*):

- A socket object is created and bound to the specified host and port.
- The server starts listening for incoming connections and delegates connection acceptance to a separate thread (*accept_connection*).

c. Client Connection Handling:

- *accept_connection*: Waits for incoming client connections. When a connection is accepted, it calls *register_client* to handle the new client.
- *register_client*: Receives the client's name, stores their connection in the clients dictionary, and sends a welcome message. A separate thread is started to handle communication with the client (*client_handler*).

d. Client Communication:

- *client_handler*: Listens for messages from the client. If a message is received, it is processed:

- If it's an *exit* message, the client is notified, and their connection is closed.
- If it's a *private message*, it is sent to the target client using *send_to_client*.
- If it's a broadcast message, it is sent to all other clients via *broadcast_message*.
- *send_to_client*: Sends a private message to a specific client, ensuring the target exists.
- *broadcast_message*: Sends a message to all connected clients, excluding the sender.

e. Server Shutdown (*shutdown_server*):

- When the server shuts down, it informs all connected clients that the server is going down, closes their connections, and clears the *clients* dictionary.

f. Logging:

- All messages exchanged between clients are logged in *chat_documentation.txt*. This includes when clients join, leave, and the content of their messages.

g. Graceful Shutdown:

- The server continuously waits for the user to type '*Terminate server*' in the console to trigger a shutdown sequence, including notifying and disconnecting clients.

Client features

1. **User input:** The client sends messages to the server, and handles user input for both normal and exit scenarios.
2. **Receiving messages:** A separate thread continuously receives and prints messages from the server.
3. **Graceful exit:** The client disconnects from the server gracefully when the user types 'exit' or when the server disconnects.

Server Code breakdown:

a. Connection Setup:

- The client creates a socket object and attempts to connect to the server using the specified host and port.
- Upon connection, the client sends its name to the server for identification.

b. Message Receiving:

- *receive_messages*: This function runs in a separate thread. It listens for messages from the server and prints them. If the server closes the connection, the thread stops, and the client is notified.

c. Message Sending:

- The main thread is responsible for taking user input and sending it to the server. If the input is 'exit', the client sends a message indicating the user has left the chat and then disconnects.

d. Graceful Exit:

- The client uses a *disconnect_event* to signal the disconnection process. This event is set when the user types 'exit' or if the server unexpectedly disconnects.

e. Error Handling:

- The client handles socket errors during message receiving and sending. If an error occurs, it signals disconnection and ensures the client exits properly.

Installation and setup

1. **Clone the repository:** Download all files mentioned earlier and make sure they are located at the same folder and ready to use.
2. **Important remark:** We have implemented two methods to run the program.

There are different instructions to run each method.

- a. First method is running the server and client through exe files – this allows to run as much clients as desired without limitation (regardless memory or hardware limitations in your computer). This option was tested and works perfectly. There might be a chance that the antivirus or firewall at your computer will not allow running unknown exe files without giving trust. If there is any issue, you are kindly requested to use the other method.
- b. The second method is running the server and clients through IDE.

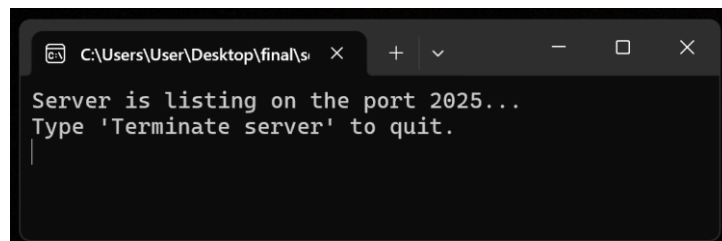
3. **First Method – running the program with exe files:**

a. **Run the server:**

- i. Double click 'server.exe'.
- ii. Server terminal will appear.

The server provides some basic instructions.

If server termination is desired, it is needed to type 'Terminate server' in server terminal.



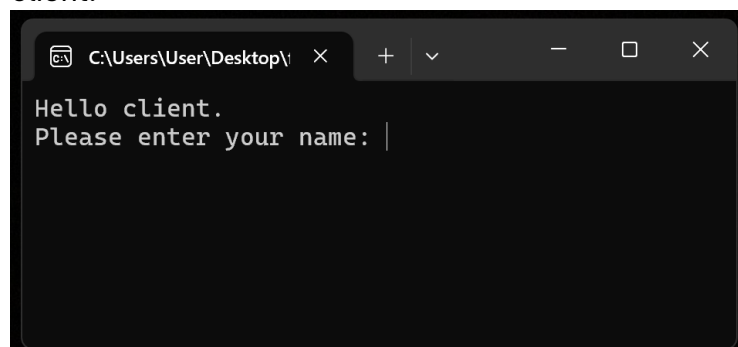
```
C:\Users\User\Desktop\final\server.exe
Server is listing on the port 2025...
Type 'Terminate server' to quit.
|
```

- iii. Server is now available.

b. **Run clients:**

- i. Double click 'client.exe'.
- ii. Client terminal will appear.

The client first demands entering the name of current client.



```
C:\Users\User\Desktop\client.exe
Hello client.
Please enter your name: |
```

- iii. After entering the name, client connects to the server and provides basic instructions for usage.

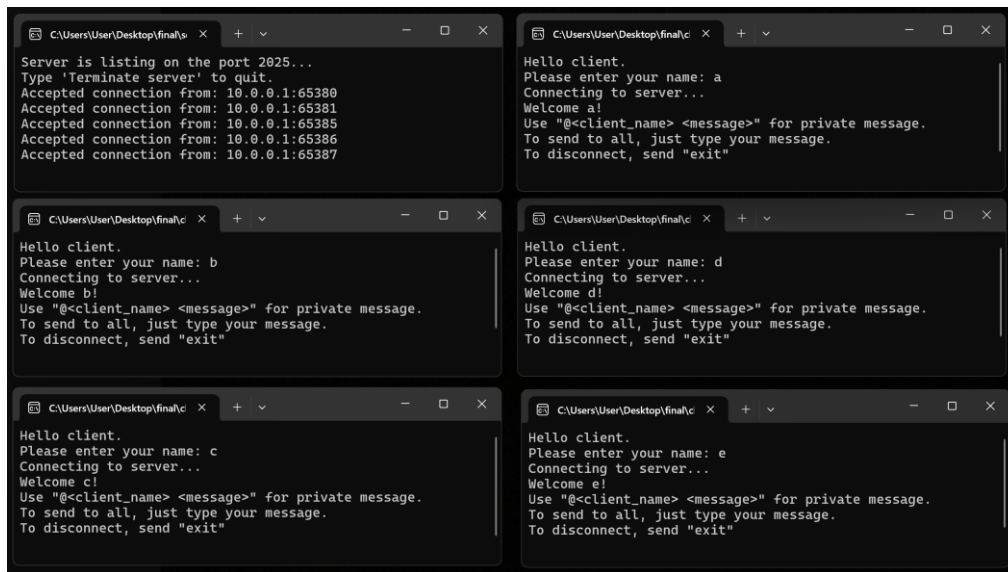
- iv. The instructions are:
 - Use the form '@client_name message' in order to write a private message to another client named 'client_name'.
 - In order to send a message to every client connected to the server at the moment, you can just type your message and press enter.
 - In order to disconnect, type 'exit' and enter.
- v. It is possible to repeat the process in order to join as many clients desired (No clients limit tested – not requested in task. Server supports at least five for sure).

4. Second method – running the program through IDE:

- a. Open the folder in which you saved the files.
- b. Duplicate 'client.py' in and rename in form 'client_i.py' (i equals the index of each client 1,2,3, etc.).
- c. Open 'server.py' and all 'client_i.py' through your IDE.
- d. Run each file.
- e. Follow steps b.ii-vii in the first method.

Input/Output examples

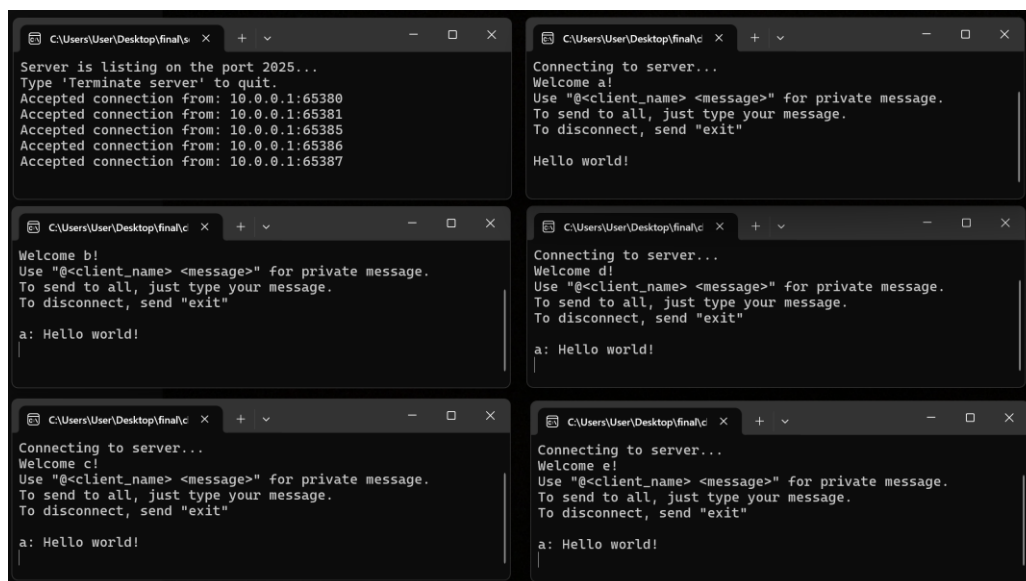
1. **Five clients are connecting:**
 - a. Server is running.
 - b. Client names is 'a' to 'e'.



The figure shows six terminal windows. The top-left window is the server, displaying: "Server is listing on the port 2025...", "Type 'Terminate server' to quit.", and five "Accepted connection from: 10.0.0.1:65380" through "10.0.0.1:65387". The other five windows are clients, each showing: "Hello client.", "Please enter your name: [name]", "Connecting to server...", "Welcome [name]!", and the usage instructions: "Use '@<client_name> <message>' for private message. To send to all, just type your message. To disconnect, send 'exit'" where [name] is 'a' through 'e'.

Figure 1 - Five clients are connecting

2. **Client sent a message:**
 - a. Server is running.
 - b. Five clients are connected.
 - c. a sent a message.

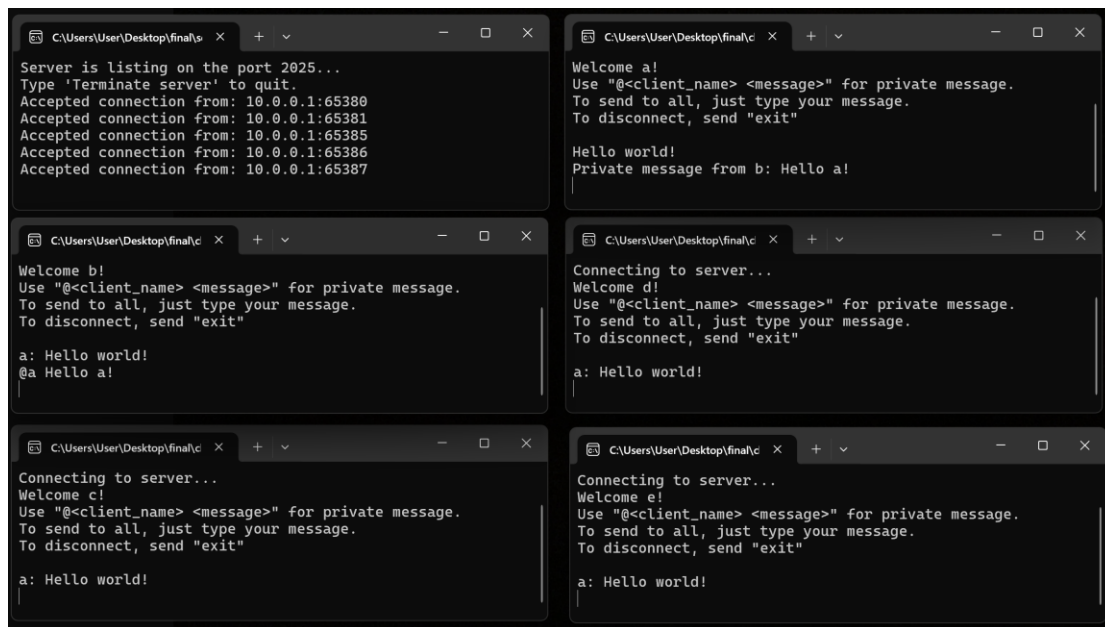


The figure shows the same six terminal windows as Figure 1. In the server window, the five accepted connections remain. In the client window for 'a', the message "a: Hello world!" has been sent, and this message is now visible in the input/output area of all five client windows (clients 'a' through 'e').

Figure 2 - Client sent a message

3. **Client sent a private message:**
 - a. Server is running.
 - b. Five clients are connected.

c. b replies a private message to a.



The figure shows six terminal windows arranged in a 3x2 grid. The top-left window shows the server's startup log, including the port (2025) and five accepted connections from 10.0.0.1. The top-right window shows client 'a' being welcomed and receiving a private message from client 'b'. The middle-left window shows client 'b' being welcomed and sending a private message to client 'a'. The middle-right window shows client 'd' being welcomed. The bottom-left window shows client 'c' being welcomed. The bottom-right window shows client 'e' being welcomed.

Figure 3 - Client b sent a private message to a

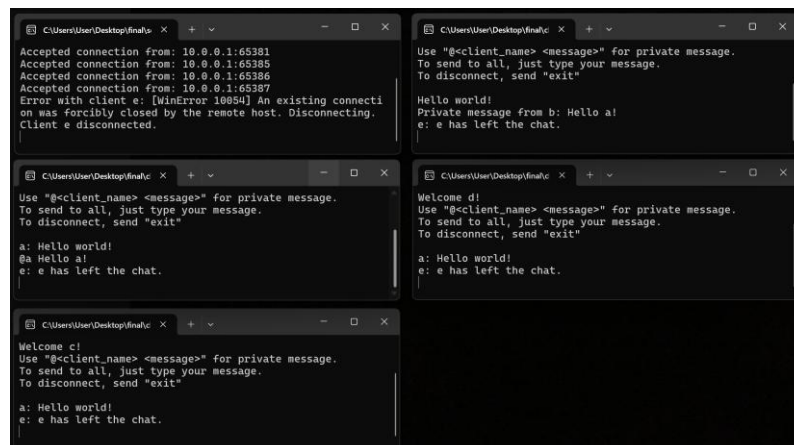
4. Client left the chat:

- a. Server is running.
- b. Five clients were connected.
- c. Client e left the chat.



The figure shows a single terminal window for client 'e'. It displays the welcome message and instructions. The user has typed 'a: Hello world!' and 'exit' on the next line, but has not yet pressed the enter key.

Figure 4 - Client e terminal before pressing enter

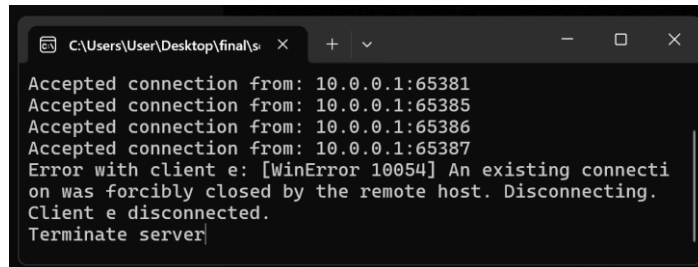


The figure shows six terminal windows in a 3x2 grid. The top-left window shows the server log with an error message: 'Error with client e: [WinError: 10054] An existing connection was forcibly closed by the remote host. Disconnecting. Client e disconnected.' The top-right window shows client 'a' receiving a message from client 'b' and a status update: 'e: e has left the chat.' The middle-left window shows client 'b' sending a private message to client 'a'. The middle-right window shows client 'd' being welcomed. The bottom-left window shows client 'c' being welcomed. The bottom-right window shows client 'e' being welcomed, with the status update 'e: e has left the chat.' visible.

Figure 5 - Client left the chat

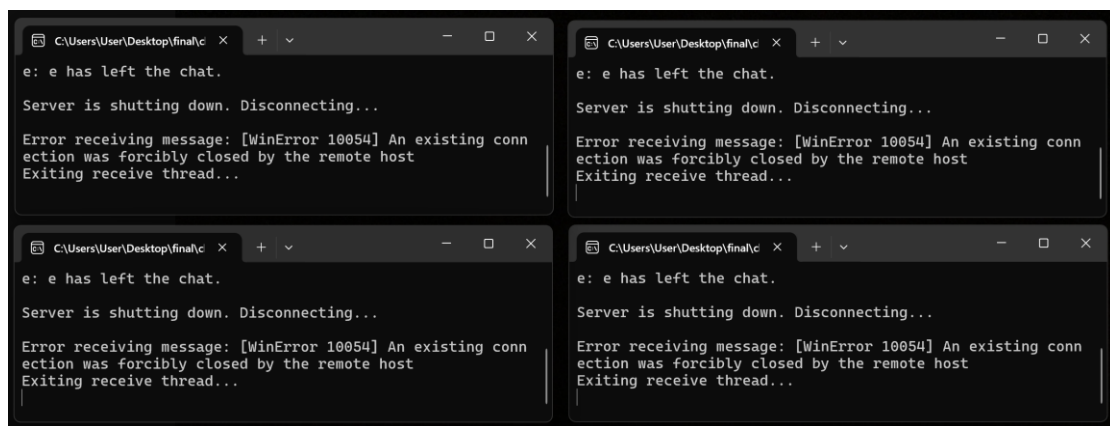
5. Server termination while clients connected

- Server is running.
- Four clients are connected.
- Server is performing graceful shutdown.



```
C:\Users\User\Desktop\final\server>
Accepted connection from: 10.0.0.1:65381
Accepted connection from: 10.0.0.1:65385
Accepted connection from: 10.0.0.1:65386
Accepted connection from: 10.0.0.1:65387
Error with client e: [WinError 10054] An existing connecti
on was forcibly closed by the remote host. Disconnecting.
Client e disconnected.
Terminate server
```

Figure 6 - Server termination - server terminal before pressing enter



```
C:\Users\User\Desktop\final\client>
e: e has left the chat.
Server is shutting down. Disconnecting...
Error receiving message: [WinError 10054] An existing conn
ection was forcibly closed by the remote host
Exiting receive thread...

C:\Users\User\Desktop\final\client>
e: e has left the chat.
Server is shutting down. Disconnecting...
Error receiving message: [WinError 10054] An existing conn
ection was forcibly closed by the remote host
Exiting receive thread...

C:\Users\User\Desktop\final\client>
e: e has left the chat.
Server is shutting down. Disconnecting...
Error receiving message: [WinError 10054] An existing conn
ection was forcibly closed by the remote host
Exiting receive thread...

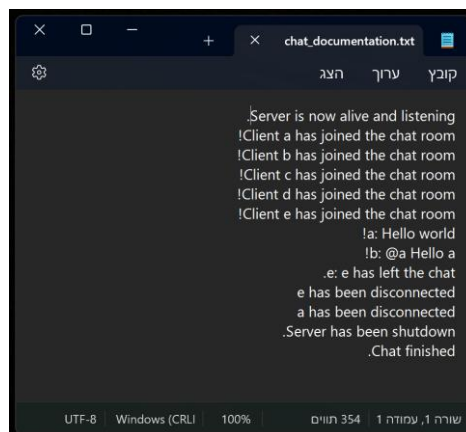
C:\Users\User\Desktop\final\client>
e: e has left the chat.
Server is shutting down. Disconnecting...
Error receiving message: [WinError 10054] An existing conn
ection was forcibly closed by the remote host
Exiting receive thread...
```

Figure 7 - Server termination - clients terminals

- Client's terminals will be automatically closed when pressing enter in the terminal.

6. Chat documentation

- Chat documentation of the session is available after finishing the session in the directory where 'server.exe' is located.



```
chat_documentation.txt
Server is now alive and listening
!Client a has joined the chat room
!Client b has joined the chat room
!Client c has joined the chat room
!Client d has joined the chat room
!Client e has joined the chat room
!a: Hello world
!b: @a Hello a
.e: e has left the chat
e has been disconnected
a has been disconnected
.Server has been shutdown
.Chat finished
```

Figure 8 - Chat documentation