Chat in python using parallel processing and socket programming.

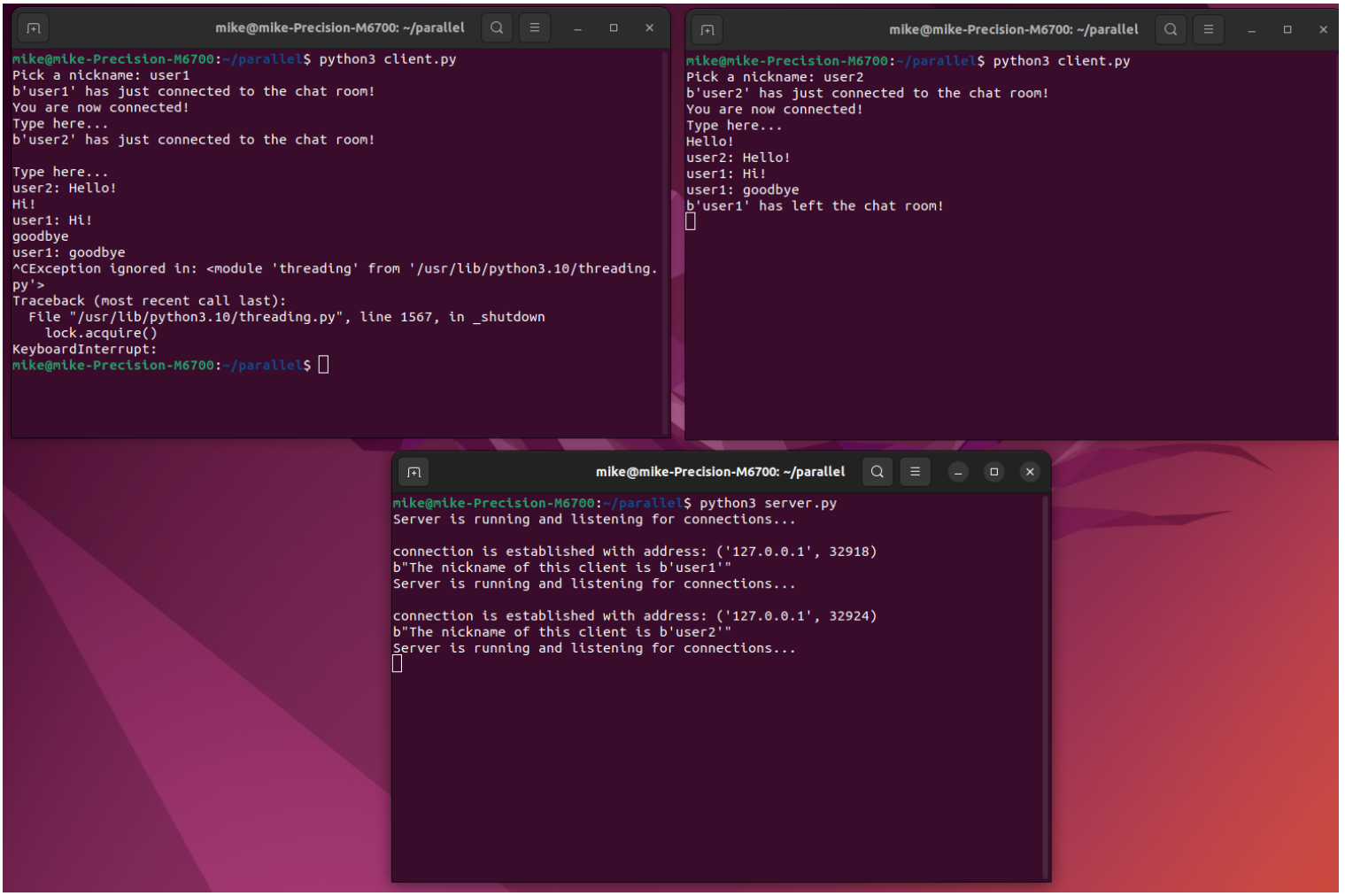
Zervas Michail
May 2023

Overview

The Python server-client chat application performs real-time communication between multiple clients via a central server using Transmission Control Protocol (TCP). The server program manages incoming connections from various clients, distributing messages from one user to all currently connected clients. On the other hand, the client program allows users to join the server, send and receive messages from the chat room in real-time. This program employs both socket programming and multithreading techniques to handle multiple client connections concurrently.

Socket programming is an essential part of the server-client communication process. Socket programming refers to the method by which two nodes (here clients) interact with one another on a network via a listening server. The server program creates a listening socket that constantly listens for incoming client connections, while each connected client has its own socket to send and receive messages to and from the server.

Multithreading in this chat program enables multiple clients to connect to the server and chat with each other concurrently. When a client connects to the server, the server creates a new thread to handle that client's connection. This allows the server to handle multiple client connections simultaneously, without having to wait for one connection to complete before accepting another.



How To Use

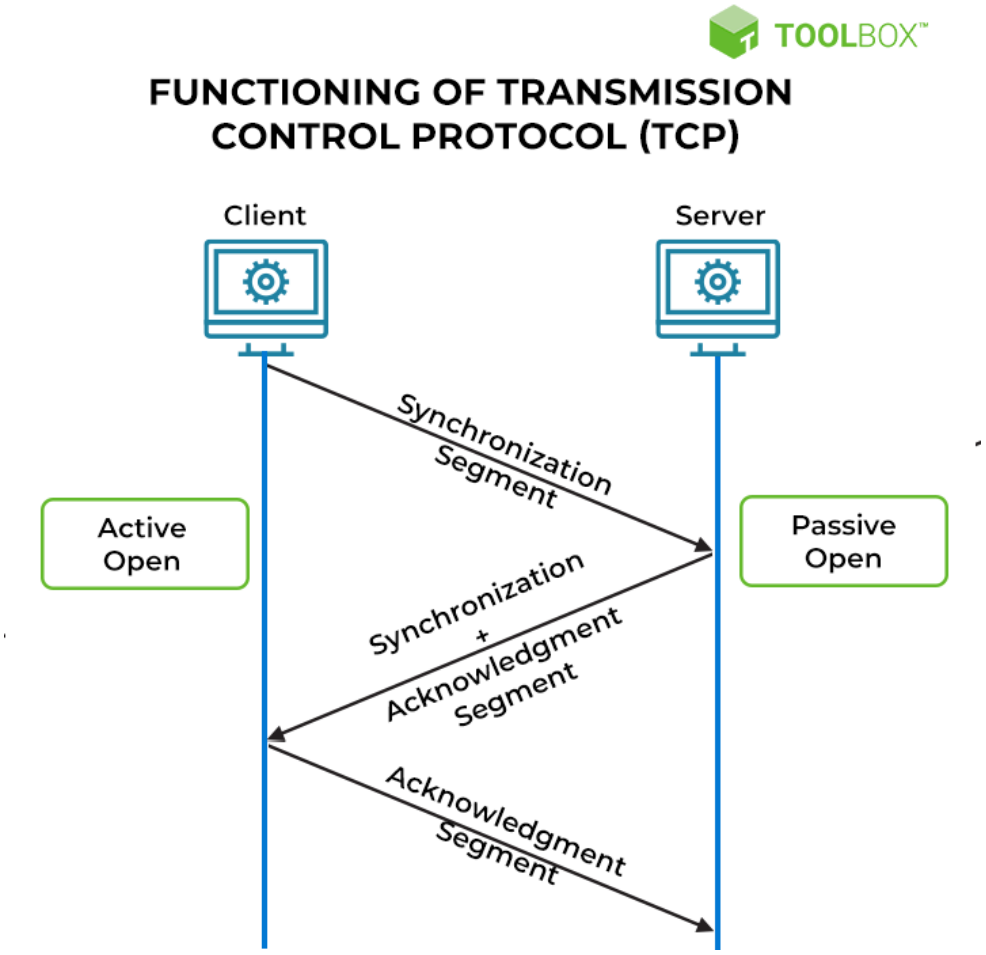
Running the server and client:

1. Start the server from terminal with the command: *$python server.py*
2. Start a client/user from a new terminal with the command: *$python client.py*
3. Enter the username of the user and write any message.
4. To log out a user, simply press Ctrl+C from the client's terminal.

In order to terminate the app:

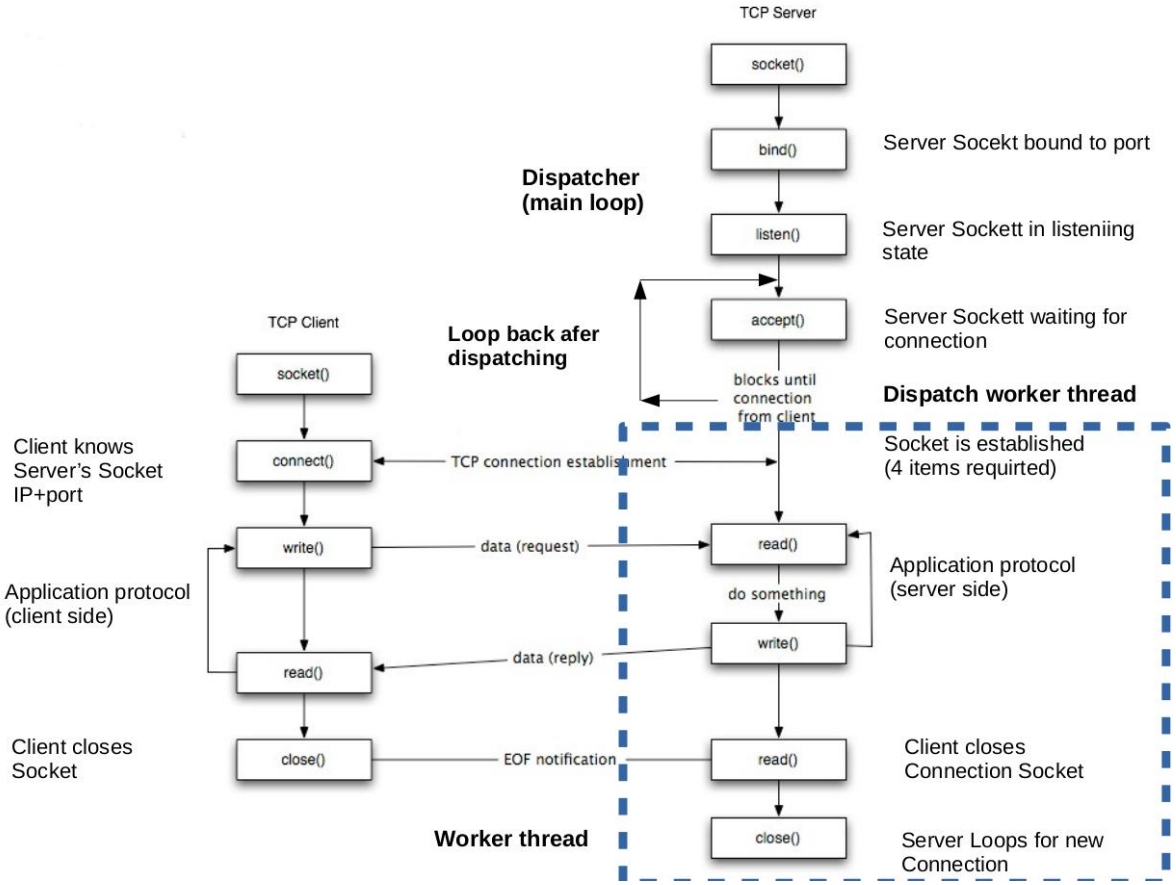
1. Press Ctrl+C on each client terminal to log out all users.
2. Wait until all clients have been disconnected from the server.
3. Press Ctrl+C on the server terminal to terminate the server.
4. Wait until the server has been completely shut down before exiting the terminal.

Protocol and Architecture



The reason that TCP was chosen over the UDP is because TCP guarantees that the data transferred remains intact and arrives in the same order in which it was sent, instead of UDP where there is no guarantee that the messages or packets sent would reach at all.

The general architecture that was used to implement this project was the server-client architecture and especially the TCP Multithreading server - client architecture as shown in the image below.



The server provides the ability to handle the connection and the communication of multiple clients with each other. The server runs continuously and listens for incoming client connections. Once a client connects, the server creates a new thread in order to handle that client's communication independently of other clients and also concurrently with the rest of the clients. Additionally, each client runs on its own thread, which allows the client to receive and send messages without blocking the main thread.

Sources

https://www.spiceworks.com/tech/networking/articles/tcp-vs-udp/
https://www.codingninjas.com/codestudio/library/socket-programming-with-multithreading-in-python
https://net-informations.com/python/net/thread.htm
https://ps5098252.medium.com/to-create-a-chat-application-using-the-concept-of-socket-programming-and-multi-threading-in-python-d792d4dd0e95
Margaritis K.G. (Parallel and Distributed Systems course, University of Macedonia) for the image of TCP server-client architecture.