DEPARTMENT OF COMPUTER SCIENCE

COURSE CSC3002F- COMPUTER SCIENCE 3002

TASK: NETWORKS ASSIGNMENT 1

DATE: 29 FEBRUARY 2024

1. We know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. We have not allowed and will not allow anyone to copy our work, intending to pass it off as their own work.
3. We acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is our own work.
4. We hereby release our publication (excepting the contributions from others as indicated under point 2 above), under the Creative Commons Attribution Noncommercial Share Alike 2.5 South Africa License.

SIGNED DATE: 29 February 2024

Surname, Name	**Student Number**	**Section(s) authored**
Tshem, Mziwokholo	TSHMZI006	CLIENT 1, SERVER IMPLEMENTATION
Modise, Omolemo	MDSOMO001	CLIENT 2, SERVER IMPLEMENTATION
Fall, Baye-Saliou	FLLBAY001	CLIENT 3,, SERVER IMPLEMENTATION

TABLE OF CONTENTS

INTRODUCTION

In this report we will be taking you through our chat application that involves creating a peer-to-peer chat application that uses UDP for real-time media streams and TCP for signaling and initiating peer communication. A central server facilitates initial client connections, and clients can query the server for available peers and communication parameters. The client, designed for TCP interaction with the server and the server maintains an active list of peer clients for efficient coordination. The goal is to implement a streamlined application, emphasising the integration of UDP and TCP sockets for effective peer communication.

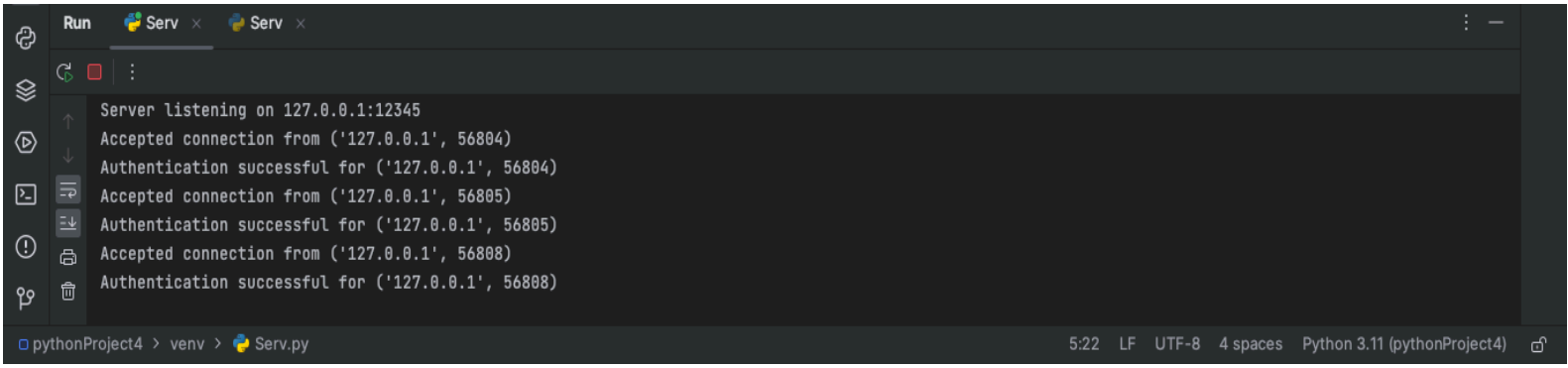
SERVER IMPLEMENTATION

Implementation

Binding and Initializing Sockets:

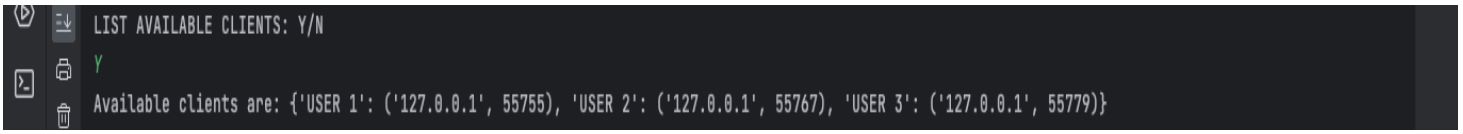
Using socket.socket(), the server creates a TCP socket, and binds it to the given host and port ().
It uses listen() to check for incoming connections.

Authentication of the client:



The server waits for a client's default password to be submitted in order to authenticate the connection.
The server authenticates the client by comparing the password that was received with the expected password ("CSC3@A1").
The server notifies the client of its successful authentication by sending a "OK" message along with the client's IP address and port for UDP communication.

Managing and Communicating with Clients:



The handle_client() function manages each client connection in a different thread.
The server waits for incoming messages from the client within the handling function.

If the client sends the command "LIST," the server responds by sending a list of available client addresses.

Design

Multithreading for Concurrency:

The server manages several client connections at once by using multithreading. The server may serve several customers at once without stalling since each client connection is handled in its own thread.

Mechanism of Authentication:

A basic authentication method based on a default password is implemented by the server. In order to be authorised and communicate with the server, clients need to enter the correct password.

Client Management:

To keep track of client labels and addresses, the server keeps a dictionary called client addresses. As a result, the server can effectively track and manage connected clients.

Functionality

Client Authentication:

In order to authenticate with the server, clients need to supply the correct default password, which is "CSC3@A1". The process of authentication guarantees that the server can only be accessed by authorized clients.

Client Communication:

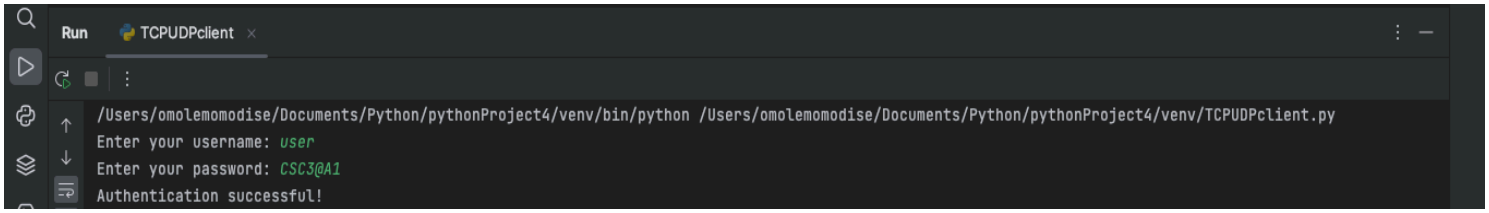
Clients with authentication are able to message the server. Upon request, the server offers a list of clients that are available.

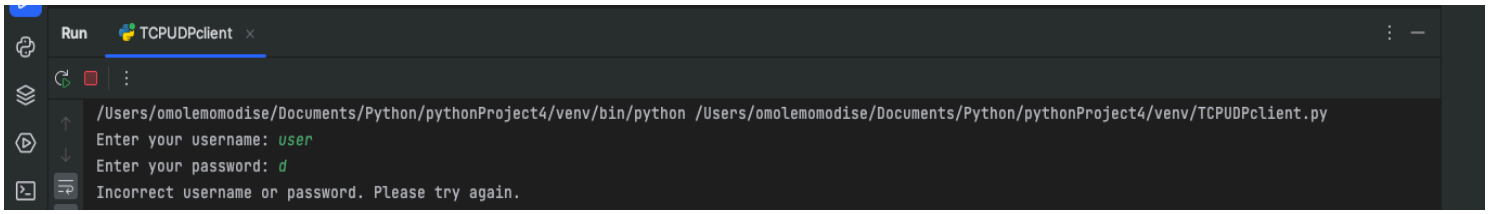
Scalability and Concurrency:

Because of its multithreaded architecture, the server is scalable and able to serve many clients at once. It can manage numerous client connections at once.

SHARED CLIENT IMPLEMENTATION: TSHMZI006

- **Authentication and TCP Connection:**





The TCP host and port for the server connection are defined at the beginning of the code.
Using tcp_socket.connect((tcp_host, tcp_port)), it generates a TCP socket (tcp_socket) and connects to the server.
A loop is used to require the user to input their password and username in order to authenticate. The server receives the entered credentials and authenticates them. When the server says "OK," the loop ends and the user is verified. If not, the user is encouraged to attempt again and receives an error notice.

- **Establishing a UDP Connection:**

  The server provides the client with its IP address and port number after authentication. The IP address and port are extracted after splitting the data, which is supplied as a string (client_ip_port).
  The IP address and port of the client are used to build and bind a UDP socket (udp_socket).
  To continuously receive UDP communications, a second thread called receive_thread is established. The receive_messages function is executed on this thread and prints the received messages along with the IP address or username of the sender.

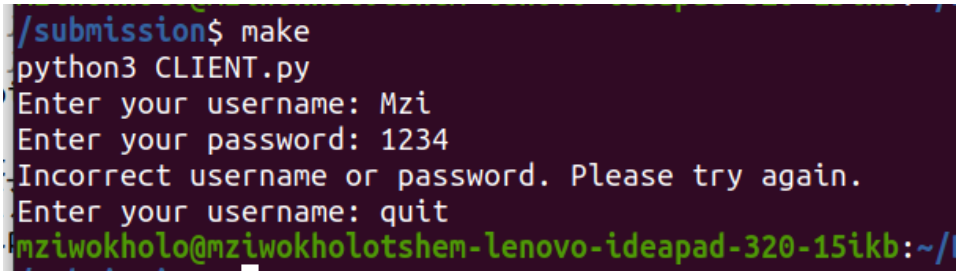
- **Sending and Receiving Messages:**

  The main loop allows the user to interact with the client. It prompts the user to input either "LIST" to request a list of available clients or to enter the username of the recipient and a message to send.
  If the user inputs "LIST," the client sends a request to the server (tcp_socket.send(b"LIST")) and receives the list of available clients from the server.
  The available clients are displayed to the user.
  The user can then input the recipient's username and a message. The client looks up the recipient's IP address and port from the available clients and sends the message using the UDP socket.

CLIENT 1 ADDITIONAL IMPLEMENTATION: TSHMZI006

- **Exiting the Code:**

  The user can exit the code at any time by entering "quit" during the username input, password input, list request, recipient username input, or message input prompts.
  If the user enters "quit," the sys.exit(0) statement is used to exit the code.



Significance of this feature

The inclusion of the 'quit' mechanism in the code is an essential component that greatly improves the client application's overall resilience and usefulness. When using a chat system, where users may need to quickly disconnect or browse away from the programme, users frequently value having a simple and intuitive option to end the session. This feature guarantees a controlled and smooth exit procedure in addition to giving users an easy way to end the application at numerous input prompts. The solution eliminates abrupt terminations, ensures correct resource cleanup before the application shuts, and improves user experience by enabling users to exit by a designated command, like inputting 'quit'. This focus on user control

CLIENT 2 ADDITIONAL FEATURE: MDSOMO001

Timestamps

Implementation:

The timestamp is generated using Python's datetime module, specifically the datetime.now() method.
The timestamp is formatted using the %Y-%m-%d %H:%M:%S format specifier, which represents the year, month, day, hour, minute, and second of the current time.

Benefits:

Clarity: Timestamps provide context for messages, making it easier for recipients to understand when they were sent or received.
Chronological Order: Timestamps make it simple for users to follow the messages' chronological order, particularly in crowded chat rooms.

Synchronization: In situations involving real-time communication, timestamps let users efficiently coordinate their responses and synchronize their talks.

User Experience:

The speed at which users can determine how recent a message is and adjust their response accordingly improves communication effectiveness.

CLIENT 3 ADDITIONAL FEATURE: FLLBAY001

1 to 2

The additional feature to the base client implementation is a feature where a single client can send a message to two other clients at the same time. This feature makes it easy for the client to communicate with other clients for cases where the message is long and retyping it may take time. Thus the feature advances ease of use for the client.

Figure 1. A client sending a message to two other clients