Report: Socket Programming Assignment

# 1. Introduction

Our project involves the creation of a simple client-server communication application using TCP sockets in Python. Our objective is to implement a chat-like system where multiple clients can connect to a single server. The server can accept messages from clients, echo responses, and maintain a session-based in-memory cache of client connections.

The solution is divided into two Python scripts: Server.py and Client.py. The server handles multiple clients using threads, ensures each client receives a unique name, and responds to special requests like showing connection status. The following report describes our code, challenges encountered, testing procedures, and potential improvements.

# 2. Code Overview

## 2.1 Server.py

The server is responsible for accepting incoming client connections and managing them using threads. Key components of the server include:

* **Socket Creation and Binding:** The server creates a socket using IPv4 addressing (AF\_INET) and TCP (SOCK\_STREAM) protocols, binding it to localhost on port 12345.
* **Handling Multiple Connections with Threads:**A new thread is spawned for every client that connects to the server. Each client is assigned a name like Client01, Client02, and so on.
* **In-Memory Cache for Client Sessions:**The server stores client details such as the connection start time and disconnection time in an in-memory dictionary.
* **Response Logic:**
  + **Echo:** The server echoes messages received from the client, appending ACK to confirm receipt.
  + **Status Check:** If the client sends the keyword status, the server returns the current cache with details of all active and disconnected sessions.
  + **Connection Closure:** When a client sends the message exit, the server logs the disconnection time and closes the connection.

## 2.2 Client.py

The client program allows users to interact with the server. Key features of the client program include:

* **Establishing a Connection:**The client connects to the server on the same port (12345). Upon successful connection, the server sends an introductory message with the client’s assigned name and the connection time.
* **User Commands:**
  + **Sending Messages:** Clients can input any message, which is sent to the server. The server echoes the message in uppercase with ACK.
  + **Status Request:** If the client sends status, the server responds with the current session cache.
  + **Exit Command:** If the client sends exit, the server closes the connection and sends a disconnection message to the client.

# 3. Challenges Faced and Solutions

1. **Handling Multiple Client Connections:**Implementing multi-threading for handling multiple clients simultaneously was initially challenging. The solution was to use the threading module to spawn a new thread for each connected client.
2. **Connection Management and Resource Handling:**Properly closing connections was crucial to prevent resource leaks. Using shutdown(SHUT\_WR) before closing sockets ensured all data was sent before terminating connections.
3. **Maintaining Client Cache Consistency:**Ensuring the correct logging of connection and disconnection times required precise management within each thread. This was handled by updating the in-memory cache immediately upon connection and disconnection.

# 4. Test Results

The application was tested with the following scenarios:

## 4.1 Single Client Test:

* **Scenario:** One client connects, sends a few messages, requests the server status, and exits.
* **Result:** The server correctly echoed messages, responded with the cache status, and logged the disconnection time.

## 4.2 Multiple Client Test:

* **Scenario:** Three clients connected simultaneously, sent messages, requested status, and exited.
* **Result:** Each client was correctly assigned a unique name, and the server handled all interactions without errors.

## 4.3 Handling Exceeding Client Limit:

* **Scenario:** A fourth client attempted to connect when the server limit was set to 3.
* **Result:** The server denied the additional connection, ensuring the limit was enforced.

4.4 Screenshots:The server console showing multiple client connections.

* A client sending a message and receiving an echoed response.
* The server responding to the status request.
* A client exiting and the server logging the disconnection time.
* Pictures 1 and 2 are from the terminal, where Client.py is run and the third picture is the python output from the Server.py file.

A screenshot of a computer

Description automatically generatedA black screen with white text

Description automatically generatedA screenshot of a computer

Description automatically generated

# 5. Possible Improvements

Given more time, the following improvements could be made:

1. **GUI for Client Application:**Developing a graphical user interface for clients would enhance usability and make interactions more intuitive.
2. **Encryption for Secure Communication:**Encrypting messages between the server and clients would ensure data privacy and security.
3. **Persistence of Cache Data:**Storing the session cache in a file or database would allow for historical analysis of connections.
4. **Dynamic Client Limits:**Allowing the server to adjust the maximum client limit dynamically based on system load could make it more robust.

# 6. Conclusion

Our project provided hands-on experience with socket programming and multi-threading in Python. It demonstrated to us how to build a reliable client-server communication system and handle multiple connections. Despite minor challenges, the final implementation met all the requirements, including managing client connections, maintaining session caches, and handling various commands.

This assignment deepened our understanding of networking concepts like TCP communication, connection management, and server-client interactions. With additional time, further enhancements could be implemented to improve usability, security, and scalability.

# 7. References

* Python Documentation:<https://docs.python.org/3/library/socket.html>
* Threading in Python:<https://docs.python.org/3/library/threading.html>

Our report covers the complete design and testing process for the client-server communication application created in Python. It demonstrates a thorough understanding of socket programming and provides ideas for future improvements.