Message Delivery Approach

The approach used for the message delivery part in the client-server communication, specifically focusing on the protocol chosen, the programming language used, and how the program handles multiple client scenarios.

1. Protocol Used: TCP (Transmission Control Protocol)

The message delivery part of the client-server communication in the provided code uses TCP (Transmission Control Protocol) as the underlying transport protocol. TCP is a reliable, connection-oriented, and stream-oriented protocol. It ensures that data sent by the client is received in the same order by the server, and it guarantees that data is delivered without loss or corruption. TCP establishes a reliable and persistent connection between the client and the server before data exchange begins.

TCP is well-suited for scenarios where data integrity and reliability are crucial, such as file transfers or applications that require the guaranteed delivery of messages.

2. Programming Language: Python

The provided code is written in Python, a high-level, versatile programming language with a strong focus on simplicity and readability. Python's ease of use and extensive libraries make it a popular choice for network programming and developing client-server applications.

3. Handling Multiple Clients Scenario: Single-Threaded Design

The current implementation of the provided code follows a single-threaded design, where the client operates in a linear, single-threaded manner. It connects to the server, sends a message or a file, waits for a response, and then repeats the process. This design is suitable for scenarios where the client interacts with the server sequentially.

However, the single-threaded design has limitations when it comes to handling multiple clients simultaneously. In this design, the client can handle only one connection at a time. If multiple clients attempt to connect concurrently, they will have to wait until the server finishes processing the requests from previous clients. This approach could lead to increased latency and reduced scalability.

4. Potential Improvements for Multiple Clients Handling: Multi-Threading or Asynchronous Design

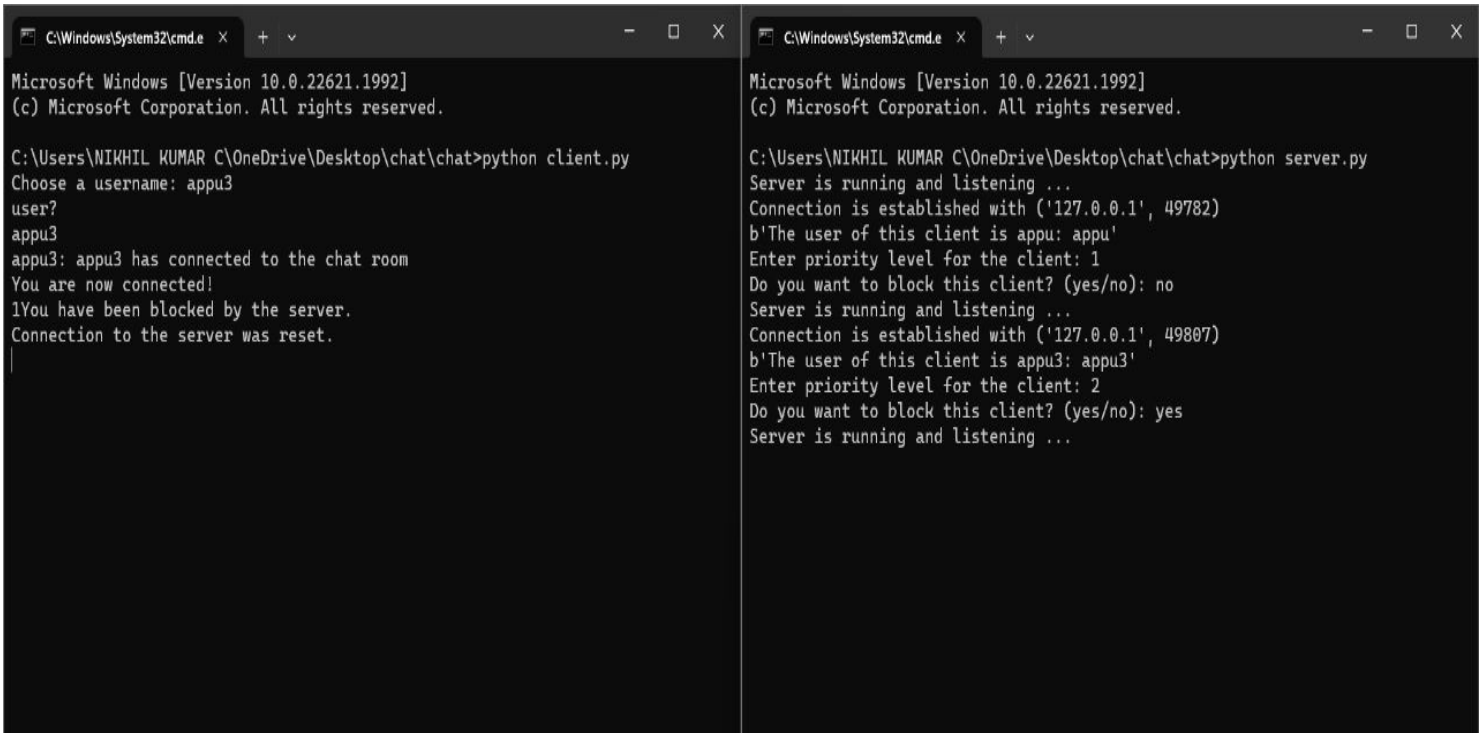
To improve the handling of multiple clients concurrently, the program could be enhanced using one of the following approaches:

a. Multi-Threading: In a multi-threaded design, each client connection is managed by a separate thread. This allows the server to handle multiple clients simultaneously, as each thread can independently process client requests. Multi-threading can significantly improve the server's responsiveness and throughput, making it better suited for scenarios with a high number of concurrent client connections.

b. Asynchronous Design: An asynchronous approach, often implemented using Python's Asuncion library, allows the server to handle multiple clients concurrently without the need for separate threads. Asynchronous programming utilizes non-blocking I/O operations, enabling the server to efficiently switch between different client connections as needed, without blocking the entire program. This approach can be more memory-efficient compared to multithreading, making it suitable for handling a large number of simultaneous connections with relatively low resource usage.

Bonus Points Implementation

1. Blocking Suspicious user (1st it has been accepted and then blocked)



2. Text file transfer

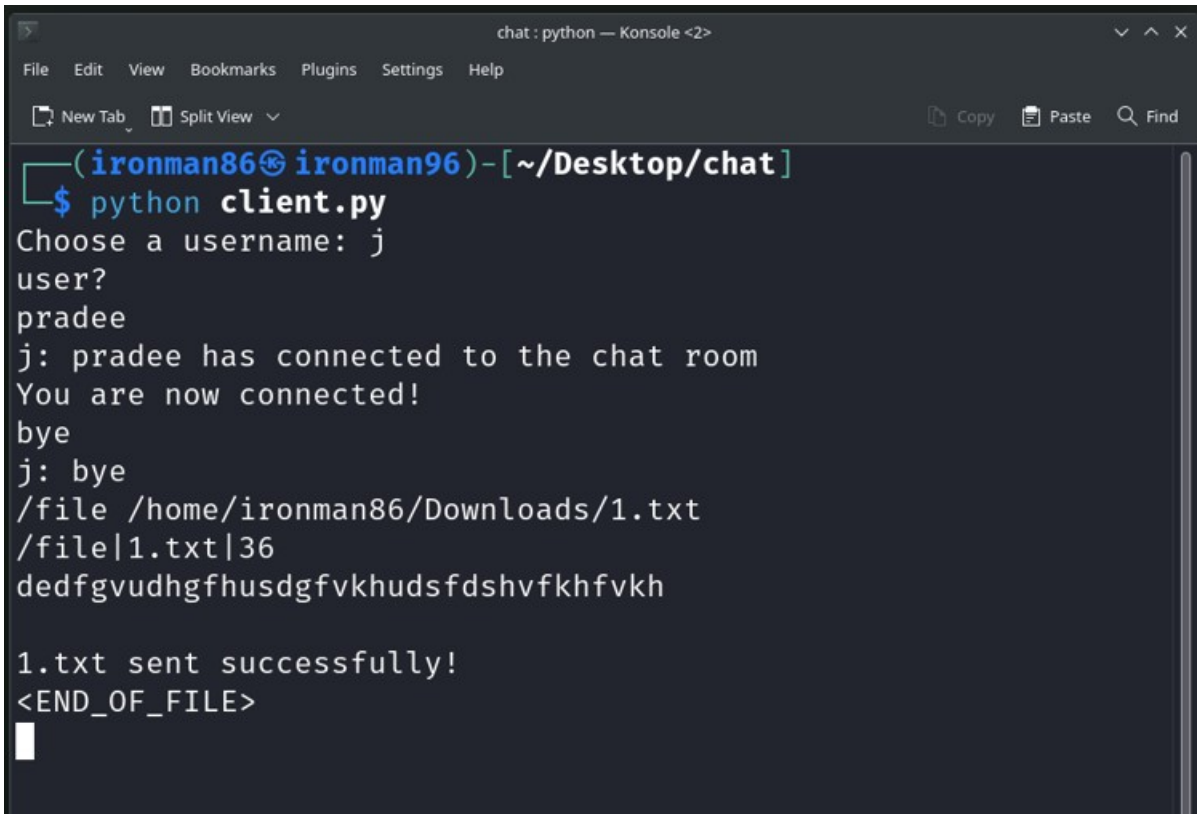
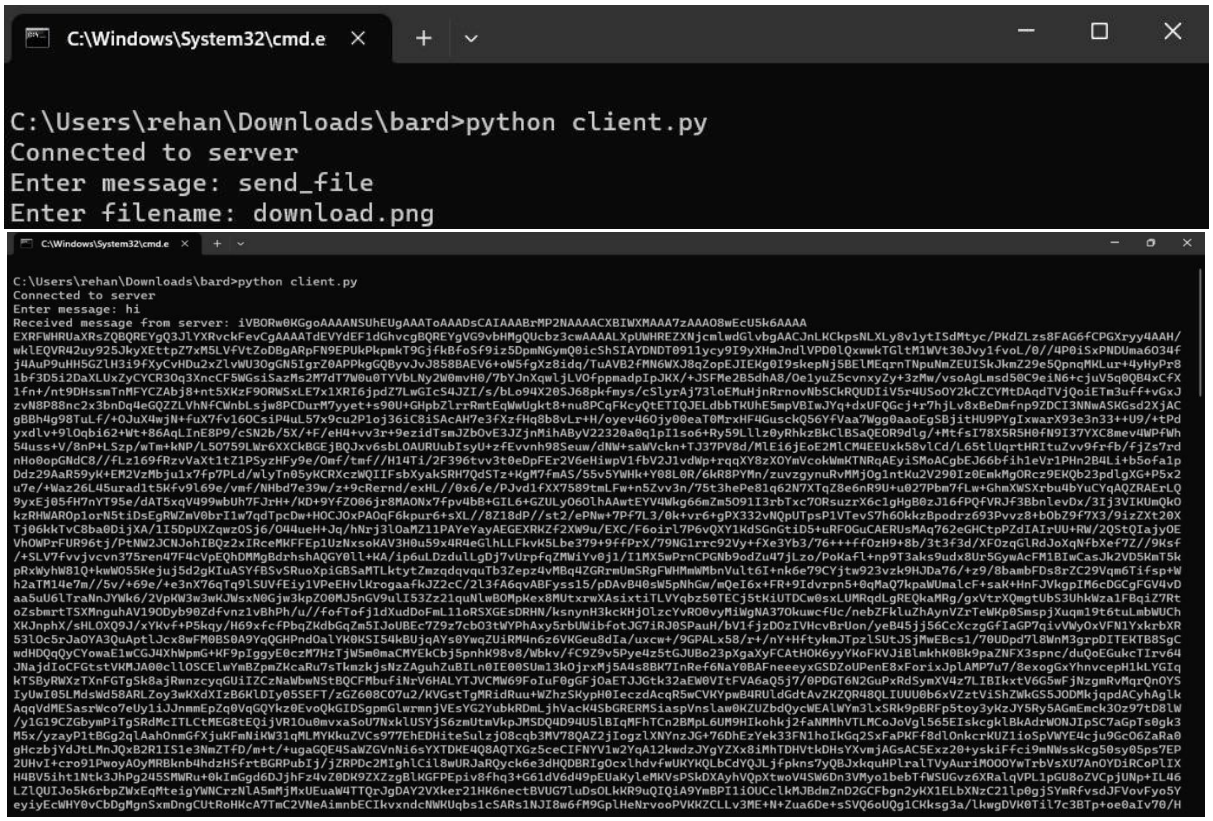
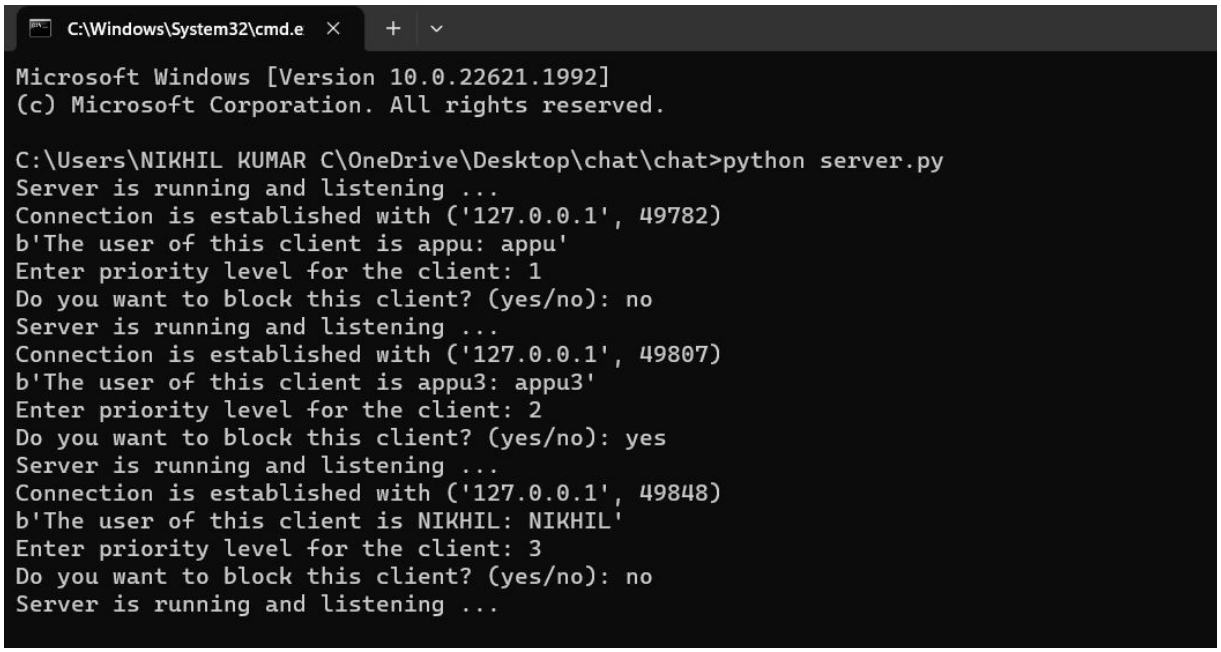


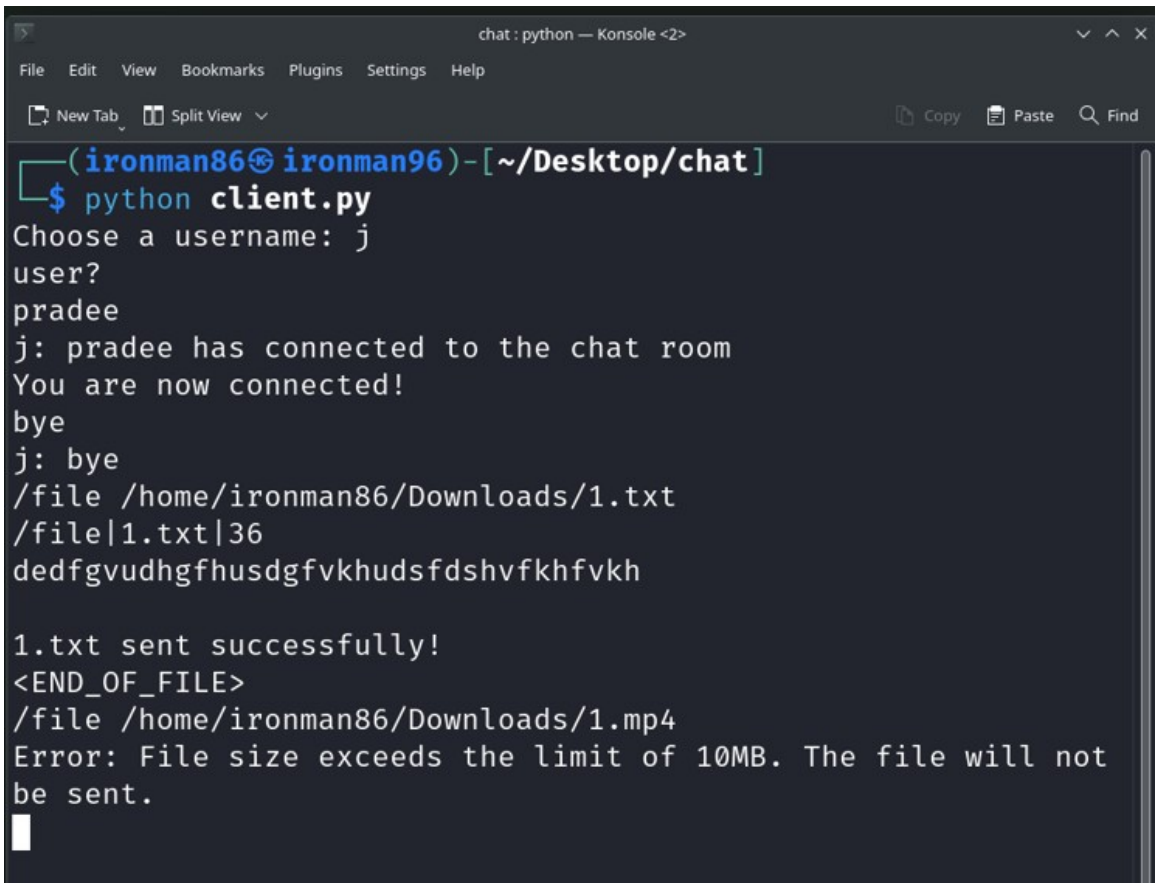
Image Transfer



3. Priority based messaging



4. Buffer management of files



S.NO.	NAME	CONTRIBUTION	GRADING
1.	PRADEEP YN	Share documents or audio files or image files	5
2.	PRADEEP KUMAR	Priority based messaging	5
3.	NIKHIL KUMAR C	Blocking Suspicious user	5
4.	HARSHITH REDDY C	Buffer management of files	5

GITHUB LINK:- https://github.com/Harshith-reddy-c/Socket-Programming-