TESSENGER: A Simple Chat Application

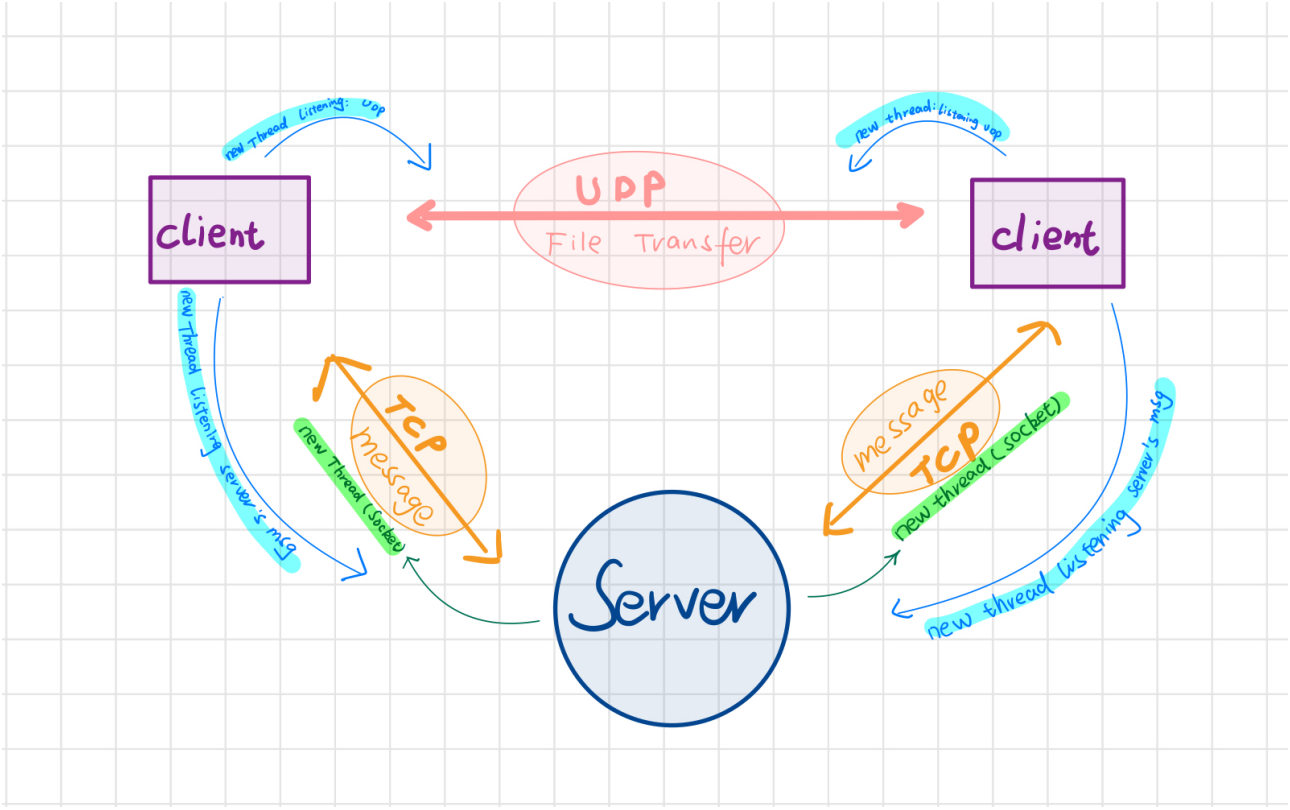
Introduction

TESSENGER is a simple chat application that allows users to communicate with each other using a client-server architecture. The application supports features such as user authentication, one-to-one messaging, group messaging, file sharing. The client-server communication is handled over TCP for command exchange and UDP for file and video transmission.

Environment

- Python 3.11

Components



1. Server

The server component of TESSENGER is responsible for handling incoming client connections, authenticating users, and managing communication between clients. It listens for incoming connections on a specified TCP port and facilitates communication between clients.

Features:

- User Authentication: Users are required to provide a valid username and password for authentication.

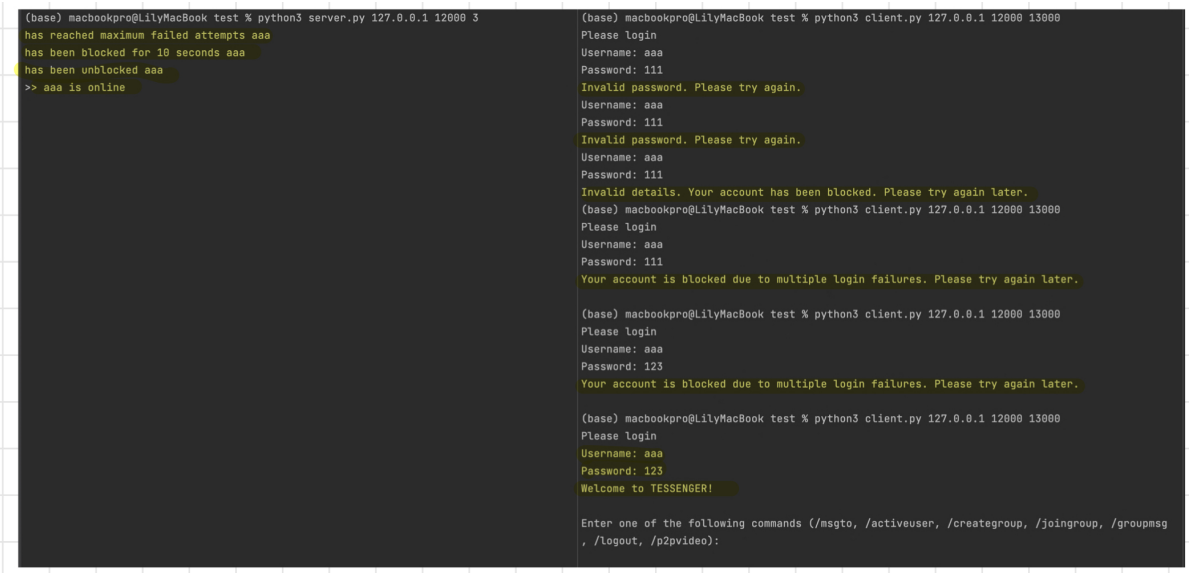
- Blocklist: A blocklist is implemented to temporarily block users who exceed the maximum number of login attempts.
- Group Messaging: Users can create and join group chats to communicate with multiple users simultaneously.
- File Sharing: Clients can send files to each other using a combination of TCP and UDP protocols.
- Logging: Server logs user activity, messages, and group chat interactions.

2. Client

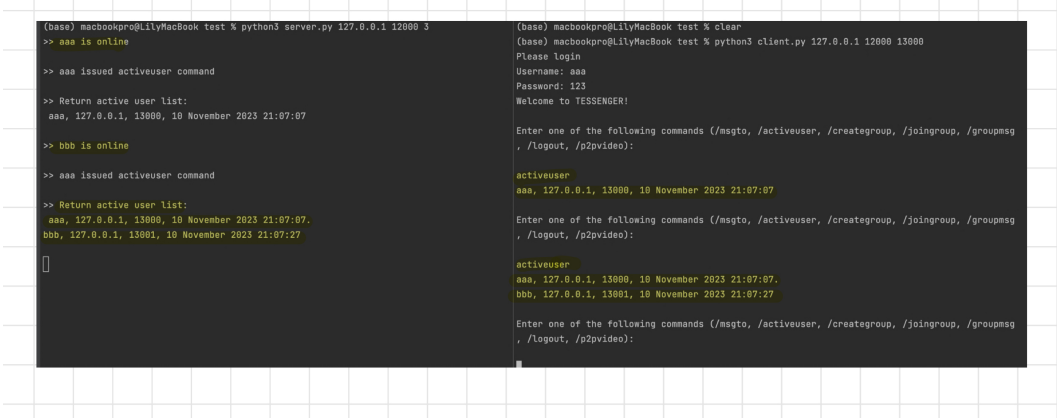
The client component of TESSENGER provides a command-line interface for users to interact with the server. Users can log in, send messages, create or join group chats, and share files. The client establishes a TCP connection with the server for command exchange and utilizes UDP for file and video transmission.

Features:

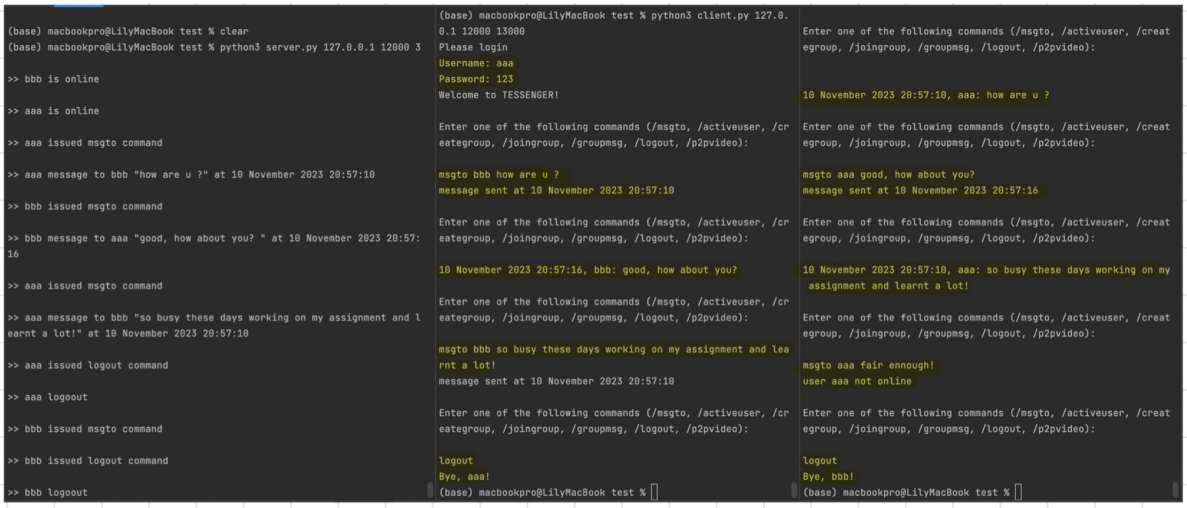
- User Authentication: Users must log in with a valid username and password.



- Command Line Interface: Users interact with the application using command-line commands (/msgto, /activeuser, /creategroup, /joingroup, /groupmsg, /logout, /p2pvideo).
  - activeuser

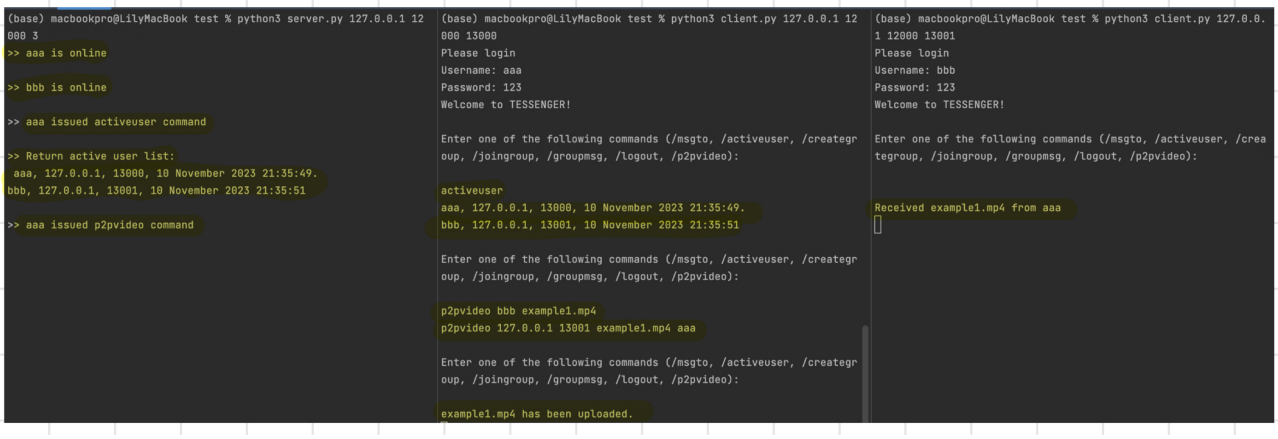


- msgto/logout



- Group Messaging: Users can create and join group chats to communicate with multiple users simultaneously.
  - creategroup/joingroup/groupmsg

- File Sharing: Clients can send and receive files from other users.



Design

- **Server Implementation:**

In the server script, threading is employed to handle multiple client connections concurrently. Each incoming client connection is processed in a separate thread using the socket_target function.

```
def main(server_ip, tcp_port, max_attemps):
    socket_list = {}
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(
        socket.SOL_SOCKET,
        socket.SO_REUSEPORT,
        1)
    s.bind((server_ip, tcp_port))
    s.listen()
    setupLogs()
    lock = threading.Lock()
    while True:
        conn, addr = s.accept()
        threading.Thread(target=socket_target, args=(conn, socket_list,
max_attemps, lock)).start()
```

The socket_target function is responsible for handling the communication with an individual client. This function is executed in a separate thread for each client connection.

- **Client Implementation:**

In the client script, threading is used to create two separate threads: one for receiving messages from the server (read_server function) and another for executing commands entered by the user (execute_command function).

```
def main(server_ip_, tcp_port_, udp_port_):
    global udp_port, server_ip
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    udp_port = udp_port_
    server_ip = server_ip_
```

```
    try:
        s.connect((server_ip, tcp_port_))
        print('Please login')
        threading.Thread(target=UDP_recv, args=()).start()
        threading.Thread(target=read_server, args=(s,)).start()

        execute_command(s)
    except:
        print('Connection refused!')
```

In the main function, two threads are started concurrently. One thread executes the UDP_recv function, which handles UDP message reception, and the other thread executes the read_server function, which reads messages from the server.

```
def read_server(s):
    global running, auth
    while running:
        try:
            content = s.recv(2048).decode('utf-8')
            print(content)
            if not content or 'Bye' in content or 'blocked' in content or
'error' in content:
                disconnect(s)
                running = False
                break
            if 'Invalid password' not in content:
                print(prompt)
            if 'Welcome' in content:
                auth = True
            elif 'p2pvideo' in content:
                _, addr, receive_port, file_name, sender =
re.split(r'\s', content)
                UDP_send((addr, int(receive_port)), file_name, sender)
```

```
        except OSError as e:
            print(f"Error reading from server: {e}")
            running = False
            break
```

- **P2Pvideo Implementation:**

UDP is utilized for file transmission between clients. The UDP file transfer is implemented through two functions: UDP_send in the client script and UDP_recv in the server script. These functions work together to send and receive files over UDP.

Usage

To run the TESSENGER application, execute the following commands:

Server:

python server.py <server_ip> <tcp_port> <max_attempts>

- <server_ip>: IP address where the server is hosted.
- <tcp_port>: Port for TCP communication.
- <max_attempts>: Maximum login attempts allowed before temporary blocking.

Client:

python client.py <server_ip> <tcp_port> <udp_port>

- <server_ip>: IP address where the server is hosted.
- <tcp_port>: Port for TCP communication.
- <udp_port>: Port for UDP communication.

Conclusion

TESSENGER provides a simple and functional chat application that enables users to communicate securely and efficiently. The combination of TCP for command exchange and UDP for file and video transmission ensures a seamless user experience. The application's features, including user authentication, group messaging, file sharing make it a practical solution for online communication.