Report: Networks Assignment

Urja Arora: z5374579

Program Design

Language Used

The assignment can be run using Python3.

File Structure

```
├── client.py
└── server.py
```

Implementation

The Client contains both TCP and UDP sockets to manage different commands. Most syntactical error handling, such as the format of commands, is done at the client. It comprises three threads: one for handling TCP communication with the server, one for receiving UDP messages, and another for handling user input.

The Server, on the other hand, contains only a TCP socket to manage communication with clients. It contains a single thread and can deal with multiple clients concurrently. It manages all commands (except /p2pvideo) that the client can use and all error handling except command syntax, which is managed at the client's end. It makes use of signals to ensure a graceful server shutdown.

Application layer message format

- Commands initiated by the client are of the following syntax:
  `<keyword_command> [command inputs].`
- For example, the following client command is interpreted by the server to handle messages between users.

```
/msgto {sender_username} {receiver_username} {message}
```

How the system works

Running the Application

Server

```
python3 server.py SERVER_PORT MAX_INVALID_ATTEMPTS
```

Client

```
python3 client.py SERVER_IP SERVER_PORT CLIENT_UDP_PORT
```

Closing the Application

Server is closed using Ctrl-C and client using the ‘/logout’ command.

Note: All clients should be logged out before shutting down the server

Design tradeoffs

1. To replicate the scenarios where the 'Enter command' prompt was used, I added a short timer instead of an event listener, as it better replicated the sample interactions.
2. I chose to keep all my code in a single file for both the client and server, aiming to reduce complexity rather than increasing modularity.
3. I opted for multithreading in both the server and the client, trading off simplicity for more advanced communication.
4. I opted not to use JSON for client-server interaction for the sake of simplification.
5. I chose not to use a logging module, instead manually logging files to prioritize simplicity over complexity.

Assumptions

1. The localhost is hardcoded as the server host in server.py.
2. Timers were used to wait for server messages.
3. Timers were used for peer-to-peer (p2p) file transfer.
4. Assumed reliability for p2p.
5. Assumed the username would not contain any whitespace.

Possible improvements and extensions

1. The code could be much more modular as several functionalities were repeated at the server.
2. All error handling could be moved to the server-side.
3. The code could be broken down into multiple files.
4. Timers should not have been used; instead, event handlers should have been implemented to guarantee the correct output.
5. Reliability could have been implemented for p2p.

Broken features

As far as I am aware, there are no broken features within the specified requirements.

Borrowed code

The following code was borrowed from online resources/webcms and changed according to the requirements of the assignment.

1. The server shutdown code

```
def shutdown_server(sig, frame):
    print("\n===== Server is shutting down gracefully =====")
    if os.path.isfile('userlog.txt'):
        os.remove('userlog.txt')
    if os.path.isfile('messagelog.txt'):
        os.remove('messagelog.txt')
    for group in groups:
        if os.path.isfile(f'{group}_messagelog.txt'):
            os.remove(f'{group}_messagelog.txt')
    print("===== All log files deleted =====")
    server_tcp_socket.close()
    sys.exit(0)

signal.signal(signal.SIGINT, shutdown_server)
```

2. File transfer in receive_udp_messages

```
def receive_udp_messages(client_udp_socket, client_username, server_host):
    while True:
        try:
            data, sender_address = client_udp_socket.recvfrom(1024)
            receivedMessage = data.decode()

            if receivedMessage.split(" ")[0] == '/logout':
                break

            chunks = receivedMessage.split()
            audience_username = chunks[1]
            original_filename = chunks[2]

            server_filename = f"{audience_username}_{original_filename}"
            # From the internet
            with open(server_filename, 'ab') as file:
                   while True:
                    data, address = client_udp_socket.recvfrom(1024)
                    if data == b"EOF":
                        break
                    file.write(data)

            print(f"File ({original_filename}) received from {audience_username}\n")
            print("Enter one of the following commands (/msgto, /activeuser, /creategroup, /joingroup, /groupmsg, /p2pvideo, /logout):")
        except:
```

3. File transfer in /p2pvideo

```
with open(filename, 'rb') as file:
        message = message
        client_udp_socket.sendto(message.encode(), server_address)
        data = file.read(1024)
        while data:
            client_udp_socket.sendto(data, server_address)
            data = file.read(1024)
            time.sleep(0.00001)
        end_marker = b"EOF"
        client_udp_socket.sendto(end_marker, server_address)
```

4. Initial code was taken from assignment resources on webcms