CSL2090: Principles of Computer Systems
Project Report: Chatroom Application

Team Members:
Anuj Rajan Lalla B22AI061
Abhinav Swami B22AI003

May 3, 2024

Abstract

This report discusses the design and implementation of a chatroom application developed as a project for CSL2090: Principles of Computer Systems. The application utilizes key networking and operating system concepts such as socket programming, message and file transfer, multithreading, and process synchronization.

Contents

1 Introduction

The chatroom application facilitates real-time messaging and file sharing among users connected over a network. The application is built using Python, emphasizing networking via sockets and concurrent handling of user connections through multithreading.

2 Networking Concepts Used

2.1 Socket Programming

The application employs TCP/IP sockets to establish a continuous stream between the client and the server, allowing for two-way communication. The sockets are initialized using the `socket.socket` function and connected through `socket.connect` for clients and `socket.accept` for the server.

2.2 Message Transfer

Text messages are transmitted across the network using the established sockets. Each message from the client is prefixed with the user's name and then encoded and sent over the network. The server receives these messages and broadcasts them to all other connected clients.

2.3 File Transfer

The application supports file transfer, where a user can send a file to other users. The file is read in chunks, sent through the socket, and reconstructed at the receiving end. The server plays a middleman role, receiving files from one client and forwarding them to others.

3 Operating System Concepts Used

3.1 Multithreading

Both the client and server utilize threading to handle multiple tasks simultaneously. The client uses separate threads for sending and receiving messages, while the server uses threads to manage connections and handle incoming data from different clients without blocking.

3.2 Process Synchronization

Synchronization is crucial for managing access to shared resources like connection lists and UI components in a multithreaded environment. Python's threading model and certain thread-safe operations (like appending to a list) are inherently used to prevent data corruption.

4 Application Layer

This chatroom operates at the application layer of the OSI model. It handles the specifics of the application protocol, including the format and transmission of messages and files, defining how data is packaged and received at both ends.

5 Code Analysis and Further Insights

5.1 Message Transfer

5.1.1 Client-Side Implementation

On the client side, message transfer is handled by the `Send` class, which inherits from `threading.Thread`. The user's message is captured via the `input()` function and sent using the socket's `sendall` method, which ensures complete data transmission. This is seen in the code snippet:

```
message = input('{}: '.format(self.name))
self.sock.sendall(f'{self.name}: {message}'.encode('utf-8'))
```

This method sends the message encoded in UTF-8 to ensure compatibility across different systems and networks.

5.1.2 Server-Side Implementation

On the server side, the `ServerSocket` class handles incoming messages. It listens for data from each client, decodes it, and then broadcasts the message to all other connected clients. The broadcasting is implemented in the `sendAll` function of the server:

```
def sendAll(self, message, source):
    for client_socket in list(self.curr_connections):
        if client_socket != source:
            client_socket.sendall(message.encode('utf-8'))
```

This function iterates over all connected clients and forwards the received message, ensuring that the sender does not receive their own message.

5.2 File Transfer

5.2.1 Client-Side Implementation

File transfer from the client side is initiated by selecting a file through the GUI and sending a file header followed by the file data in chunks. This is managed by the `send_file` method in the `Client` class:

```
self.sock.sendall(f'FILE:{filename}:{filesize}'.encode('utf-8'))
with open(file_path, 'rb') as f:
    bytes_read = f.read(1024)
    while bytes_read:
        self.sock.sendall(bytes_read)
        bytes_read = f.read(1024)
```

Here, the client first informs the server about the file by sending a descriptive header, then sends the file data in 1024-byte chunks.

5.2.2 Server-Side Implementation

The server receives the file data, stores it temporarily, and then forwards it to all other clients. This is done in the `handle_file_transfer` method within the `ServerSocket` class:

```
def handle_file_transfer(self, initial_msg):
    _, filename, filesize = initial_msg.split(':')
    file_data = b''
    while filesize > 0:
        data = self.server_socket.recv(min(1024, filesize))
        file_data += data
        filesize -= len(data)
    self.server.broadcast_file(self.server_socket, filename, file_data)
```

After receiving the entire file, the server broadcasts it to other clients using the `broadcast_file` method.

5.3 Usage of Tkinter in the Client Interface

The client interface is built using Tkinter, providing a user-friendly environment for interaction. The `setup_gui` method within the `Client` class sets up the graphical elements:

```
window = ctk.CTk()
window.title('Chatroom')
```

Widgets such as text inputs, buttons, and message displays are defined here, enabling users to type messages, send files, and view chat history. The use of custom Tkinter widgets (from the `customtkinter` module) enhances the visual appeal and user experience.

5.4 Conclusion of Analysis

This detailed examination of both message and file transfer processes, along with the client-server interaction and GUI setup, provides a comprehensive view of how different computer system principles are applied in real-world software development. These implementations highlight the practical use of multithreading, networking protocols, and user interface design, crucial for modern software applications.

6 Screenshots

Screenshots added for reference

7 Conclusion

The chatroom application effectively demonstrates the application of core networking and operating system principles in a real-world software project. The successful implementation of socket programming, multithreading, and data transfer protocols highlights the practical application of course concepts.

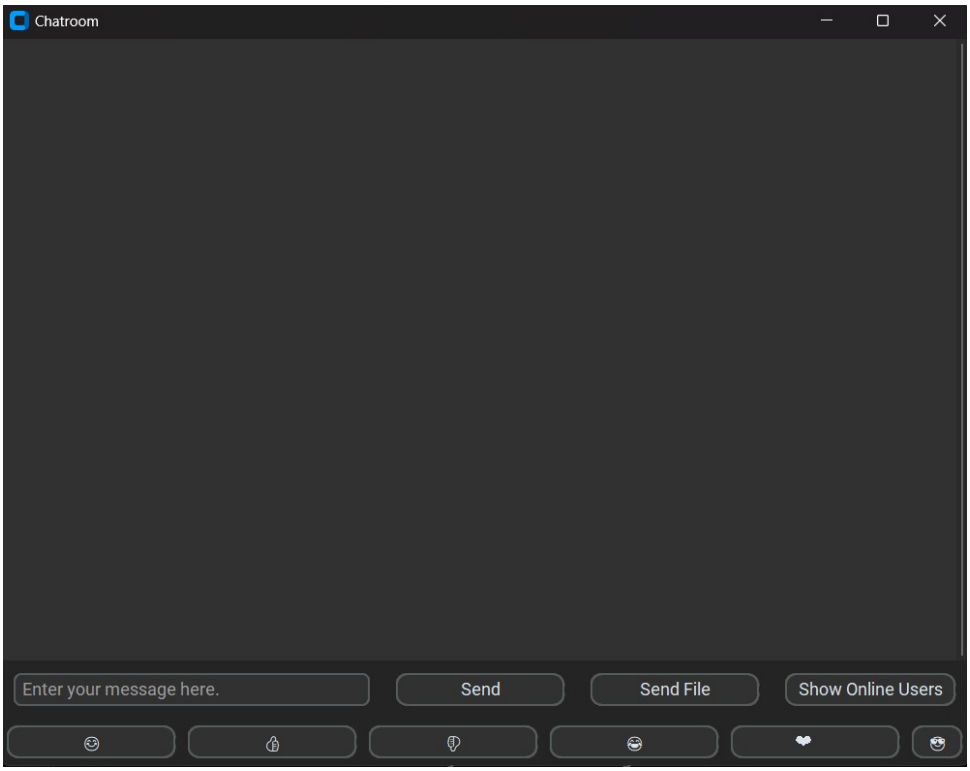

Figure 1: Chatroom client interface

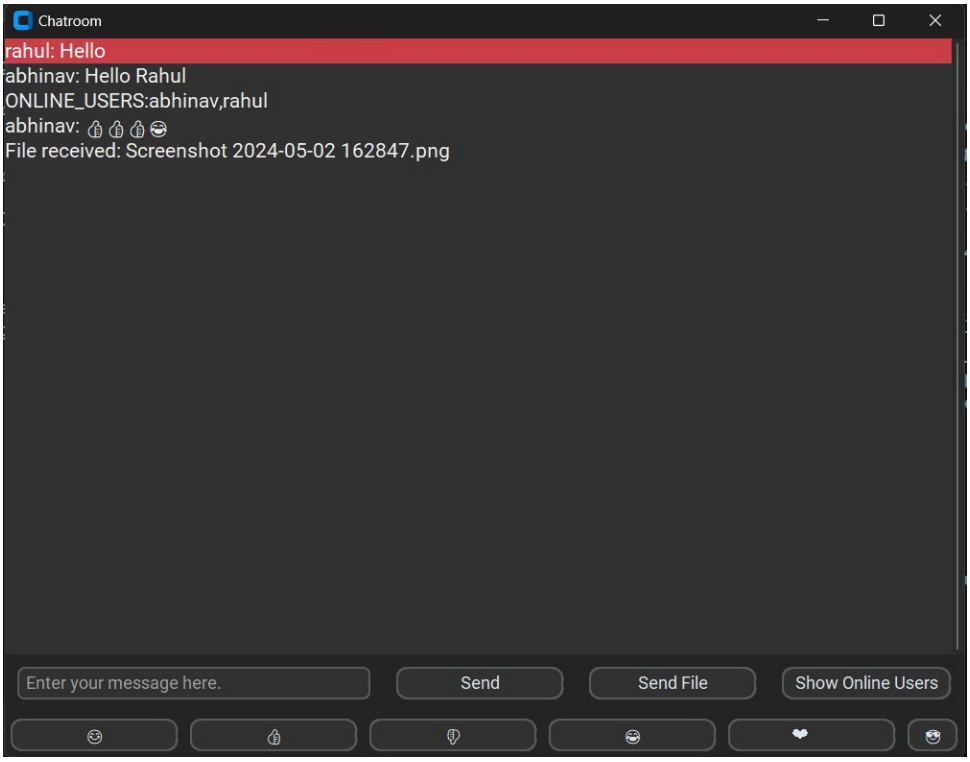

Figure 2: Some functionalities