Ghulam Ishaq Khan Institute of Engineering Sciences and Technology Topi

CE4I3: Computer Networks

Project: Instant Messaging

Submitted To:

Muhammad Salman Saeed

Submitted By:

Abdullah Khan (2020026)

Aiman Gohar (2020059)

Project Title: Instant Messaging System

Introduction

The Instant Messaging System is a chat system made in Python. It lets users talk to each other instantly over the internet. It has three parts: a server, a client, and a graphical interface. The server manages connections between clients and helps send messages. The client allows users to compose and read messages, enabling communication with others in various locations. The GUI, developed with **tkinter**, furnishes a user-friendly interface, simplifying interaction with the chat system.

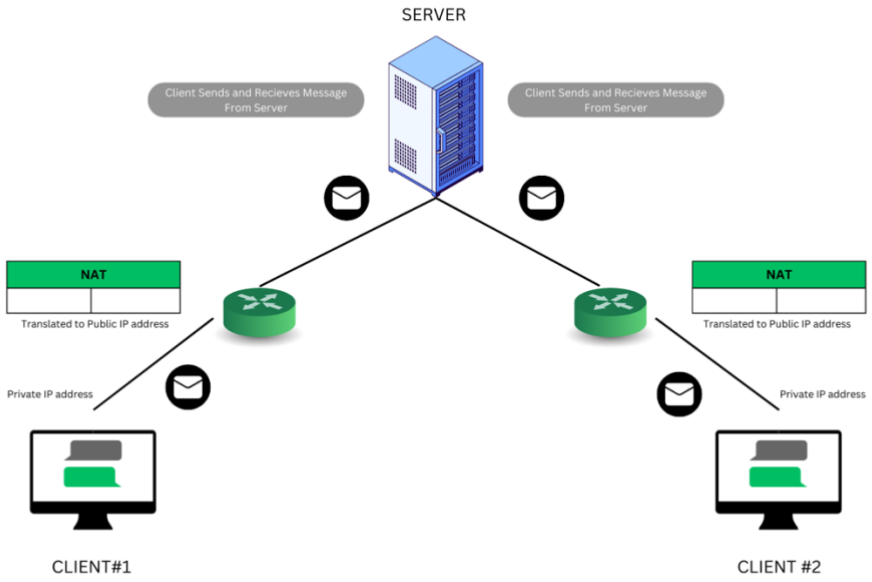
Implementation Overview

Design

The Instant Messaging System is developed using Python, employing libraries such as **socket** for network communication between the server and clients, and **threads** to enable a client to communicate with multiple users simultaneously. The **tkinter** library is utilized for developing a simple GUI, providing a user-friendly interface, and simplifying interaction with the chat system.

System Architecture

The system follows a client-server architecture. The server component listens for incoming connections on a specified IP address and port, while the client component connects to the server to send and receive messages. The GUI provides an intuitive interface for users to interact with the application.



Components

The Instant Messaging System architecture consists of the following main components;

Server

- Handles incoming client connections: Utilizes Python's socket library to create a TCP/IP socket and listens for incoming connections from clients.

- Manages message relay between clients: Receives messages from clients and broadcasts them to all other connected clients.
- Utilizes multithreading for concurrent client handling: Creates a new thread for each client connection, allowing multiple clients to connect and communicate simultaneously.

Client

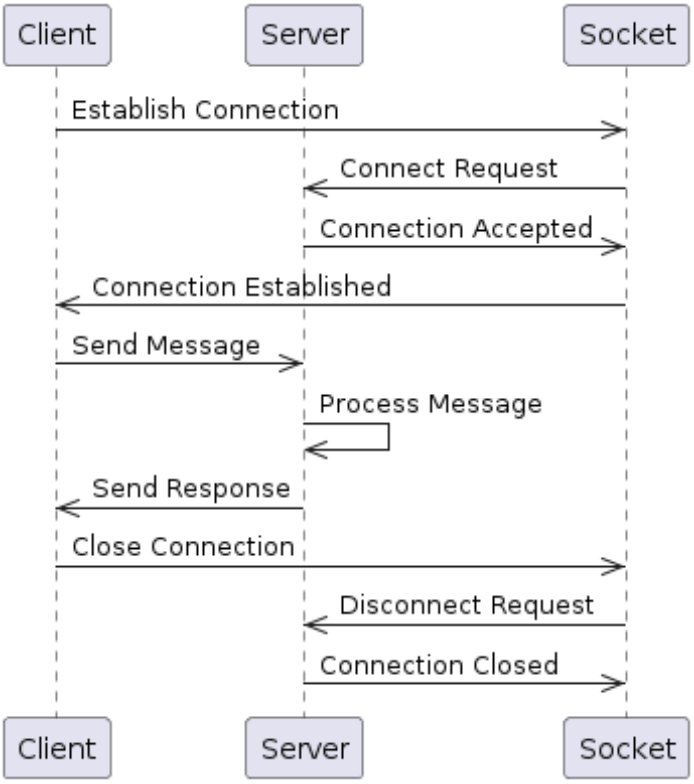
- Connects to the server: Establishes a connection to the server using TCP sockets, enabling communication between the client and server.
- Sending messages: Allows the user to input and send messages to the server for broadcasting.
- Receives messages: Listens for messages from the server and displays them to the user.

GUI

- Provides a graphical interface: Creates a window with widgets (e.g., text box, button) for user interaction.
- Displays message history: Shows the history of messages exchanged between clients.
- Sending messages: Provides a text input field for the user to type messages and a button to send them to the server.

Workflow

1. **Server Setup:** Run the server.py file to establish a connection with the server. The server listens for incoming client connections.
2. **Client Connection:** Clients establish connections to the server upon running the client.py file.
3. **Messaging:** Clients can send and receive messages in real time using a user-friendly interface after running the gui.py file. Messages are transmitted between clients via the server**.**
4. **Termination:** Clients can gracefully disconnect from the server when they finish their session.



Server.py

```
import socket
import threading

# Server configuration
HOST = '127.0.0.1'
PORT = 5555

# List to store connected clients
clients = []

# Function to handle client connections
def handle_client(client_socket, username):
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            if message:
                broadcast(username + ': ' + message)
        except:
            # Client has disconnected
            print(username + ' has disconnected.')
            client_socket.close()
            clients.remove(client_socket)
            broadcast(username + ' has left the chat.')
            break

# Function to broadcast messages to all connected clients
def broadcast(message):
    for client in clients:
        client.send(message.encode('utf-8'))

# Main function to start the server
def main():
    # Create a socket object
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Bind the socket to the host and port
    server_socket.bind((HOST, PORT))

    # Start listening for incoming connections
    server_socket.listen()

    print('Server is listening on {}:{}'.format(HOST, PORT))

    # Accept incoming connections and handle them in separate threads
    while True:
        client_socket, client_address = server_socket.accept()
        print('Connection from', client_address)
```

```
        # Prompt client for username
        client_socket.send('Enter your username: '.encode('utf-8'))
        username = client_socket.recv(1024).decode('utf-8')
        clients.append(client_socket)

        # Welcome message
        client_socket.send('Welcome to the chat!'.encode('utf-8'))

        # Start a new thread to handle the client
        client_thread = threading.Thread(target=handle_client, args=(client_socket,
username))
        client_thread.start()

if __name__ == "__main__":
    main()
```

Client.py

```
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext, simpledialog

# Server configuration
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 5555

# Function to receive messages from the server
def receive_messages(client_socket, chat_window):
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            chat_window.insert(tk.END, message + '\n')
            chat_window.see(tk.END)  # Scroll to the bottom of the chat window
        except:
            # Server has disconnected
            chat_window.insert(tk.END, 'Disconnected from the server.\n')
            client_socket.close()
            break

# Function to send messages to the server
def send_message(client_socket, message_entry):
    message = message_entry.get()
    if message:
        client_socket.send(message.encode('utf-8'))
        message_entry.delete(0, tk.END)

# Main function to start the client
def main():
    # Create a socket object
```

```
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the server
    client_socket.connect((SERVER_HOST, SERVER_PORT))

    # Create the GUI
    root = tk.Tk()
    root.title('IM Client')

    # Chat display
    chat_window = scrolledtext.ScrolledText(root, width=50, height=20)
    chat_window.grid(row=0, column=0, columnspan=2)

    # Message entry
    message_entry = tk.Entry(root, width=40)
    message_entry.grid(row=1, column=0)

    # Send button
    send_button = tk.Button(root, text='Send', command=lambda:
send_message(client_socket, message_entry))
    send_button.grid(row=1, column=1)

    # Receive messages from the server
    receive_thread = threading.Thread(target=receive_messages, args=(client_socket,
chat_window))
    receive_thread.start()

    root.mainloop()

if __name__ == "__main__":
    main()
```

Deployment Instructions

- If not, download and install Python 3 from the official Python website (https://www.python.org/) according to your operating system.
- Download the project files (**server.py**, **client.py**, **gui.py**) and place them in a designated directory on your system. Ensure that all files are located in the same directory for proper functionality.
- Run the server by executing server.py.
- Run the client by executing client.py.
- The GUI will open, allowing you to send and receive messages.

Conclusion

The Instant Messaging Application provides a simple yet effective solution for real-time communication between users. By following the client-server architecture and utilizing Python's socket library, the application allows users to exchange messages seamlessly. The graphical user interface enhances user experience, making it accessible to users of all technical levels.

Dependencies

- Python 3
- socket library