**Kuncham Padma Priyanka_M01633574**

**Data Communication and Network**

# CSCI 4/5300 Project: Part 2

## Project Description:

Simple Python Server with Multi-Threaded Client Handling
Developed in a simplified Instant Message (IM) system that consists of two or more user programs, i.e., user1, user2 and user3

## Abstract:

This document describes a Python script that implements a basic server using sockets and multi-threading (can use visual studio source code editor).

The server listens for incoming connections on a specified IP address and port, establishes a connection with each client, and spawns a new thread to handle communication with each connected client independently.

The communication involves receiving and sending messages in a simple text-based format. The server allows multiple clients to connect simultaneously and communicate with the server independently.

## Code Performance:

- User1 starts first, listening at a given port number and waiting for outside connections.
- User2 and user3 connects to user1.
- User2 and user3 talks to user1 from the standard input (keyboard) until user2 or user3 enters "#".

- User1 prints everything received from user2 or user3 on the screen until it receives "#". Then, user1 replies to user2 and user3 simultaneously one by one at a time from the standard input (keyboard) until user1 enters "#".
- User2 and User3 prints everything received from user1 on the screen until it receives "#".
- User1 or User2 or User3 sends an “Exit” to the other side if it is done.
- Steps 3-5 is repeated until both users send an “Exit” to the other side.
- After both users receive the “Exit”, the connection is closed.

## Socket Connection Explanation:

socket.socket(socket.AF_INET, socket.SOCK_STREAM): Creates a new socket using the socket module. The parameters socket.AF_INET specify the address family (IPv4 in this case), and socket.SOCK_STREAM specify the socket type (TCP).

server_socket.bind((host, port)): Binds the socket (server_socket) to a specific host and port. Binding essentially associates the socket with a specific network address (in this case, a combination of a host and a port number). This step is necessary for the socket to listen for incoming connections on a specific network interface and port.

Socket: A socket is an endpoint for sending or receiving data across a computer network. It provides a programming interface for network communication.

Address Family (AF_INET): Specifies the address family used by the socket. In this case, it's AF_INET, which indicates IPv4. IPv4 addresses are commonly used in networking.

Socket Type (SOCK_STREAM): Specifies the socket type. SOCK_STREAM indicates a socket that provides a reliable, stream-oriented connection (TCP).

Binding: Binding a socket involves associating it with a specific network address. In the case of a server socket, it's binding to a specific combination of a host (in this case, '0.0.0.0', which means it will listen on all available interfaces) and a port number (in this case, 12345). Binding allows the server to listen for incoming connections on that specific address and port.

**TCP connections work in a simple server-client model:**

Server: The server creates a socket and binds it to a specific address and port. It then listens for incoming connections using server_socket.listen(). When a client connects, the server accepts the connection using server_socket.accept(), creating a new socket (conn) dedicated to that specific client.

Client: The client creates a socket and connects to the server using the server's address and port. Once the connection is established, the client can send and receive data through the socket.

In summary, the bind operation is crucial for specifying the network address to which the server socket is bound, allowing it to listen for incoming connections. The combination of socket.AF_INET, socket.SOCK_STREAM, and the bind operation sets up a TCP server to accept connections on a specific address and port.

## Source Code:

```
#user1:
import socket   #Imports the Python socket

import threading #Imports the threading module

def handle_client(conn, addr):  #function with a socket connection and the
address of client as parameters.
    print(f"Connection established with {addr}")

    while True:
        data = conn.recv(1024).decode('utf-8')
        print(f"Received from {addr}: {data}")

        if data == "#" or data == "Exit":
            break

        reply = input(f"Reply to {addr}: ")
        conn.send(reply.encode('utf-8'))

        if reply == "#" or reply.lower() == "Exit":
            break
```

```
    conn.close()
    print(f"Connection with {addr} closed")

#setup of server
def main():
    host = '0.0.0.0'
    port = 12345

#Creates a socket using IPv4 (AF_INET) and TCP (SOCK_STREAM).
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#Binds the socket to the specified host (0.0.0.0) and port (12345).
    server_socket.bind((host, port))
#Listens for incoming connections with a maximum queue size of 5.
    server_socket.listen(5)  # Listen for up to 5 connections

    print(f"Waiting for connections on port {port}")

    while True:
        conn, addr = server_socket.accept()

        # Start a new thread to handle the communication with the connected
client
        client_thread = threading.Thread(target=handle_client, args=(conn,
addr))
        client_thread.start()

if __name__ == "__main__":
    main()



#Source Code:
#User2
import socket


def main():
    host = '192.168.12.132'  # Replace with user1's IP address
    port = 12345

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


    client_socket.connect((host, port))

    while True:
        message = input("Talk to user1: ")
        client_socket.send(message.encode('utf-8'))
```

```
        if message == "#" or message == "Exit":
            break

        data = client_socket.recv(1024).decode('utf-8')
        print(f"Received from user1: {data}")

        if data == "#" or data == "Exit":
            break

    client_socket.close()
    print("Connection closed")


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

```
#user3:
import socket


def main():
    host = '192.168.12.132'  # Replace with user1's IP address

    port = 12345

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))

    while True:
        message = input("Talk to user1: ")
        client_socket.send(message.encode('utf-8'))

        if message == "#" or message== "Exit" :
            break

        data = client_socket.recv(1024).decode('utf-8')
        print(f"Received from user1: {data}")

        if data == "#" or data == "Exit":
            break

    client_socket.close()
    print("Connection closed")

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