PD LAB

ASSIGNMENT - 3

Name: Raunak Thanawala

Registration Number: 231070051

Branch: Computer Engineering

Batch: 3

**Aim:-**

To create a messaging application in python using socket programming and tkinter.

**Theory:-**

Socket programming is a way to enable communication between two terminals over a network.

Sockets provide a way for software applications to send and receive data, allowing for network-based communication.

It is a fundamental technology for creating networked applications like web servers, chat clients, and multiplayer games.

A socket is an endpoint for sending or receiving data across a computer network.

Types of Sockets:

Stream Sockets (TCP):

Provide a reliable, two-way, connection-based byte stream.

TCP (Transmission Control Protocol) ensures that data is delivered accurately and in order.

Datagram Sockets (UDP):

Provide a connectionless, unreliable messaging service.

UDP (User Datagram Protocol) is faster and more efficient for applications that do not require guaranteed delivery of data.

Client-Server Model: Socket programming often follows a client-server architecture, where the server waits for incoming connections, and the client initiates a connection to the server.

Steps for Socket Programming

1. Create a Socket:

A socket is created using the socket() function, specifying the address family (such as IPv4 or IPv6) and the socket type (such as TCP or UDP).

1. Bind the Socket (For Servers):

Bind the socket to an IP address and port number using the bind() method, which specifies the address and port to listen for incoming connections.

1. Listen for Connections (For Client): For a server, the socket needs to listen for incoming connections using the listen() method, which allows the server to accept incoming requests.
2. Accept Connections (For Server): The server uses the accept() method to accept a connection from a client. This method returns a new socket object representing the connection and the address of the client.
3. Connect to the Server (For Client): For clients, connect to the server using the connect() method, which establishes a connection to the server's socket.
4. Send and Receive Data : Data is transmitted between the client and server using the send() and recv() methods for TCP sockets, or sendto() and recvfrom() for UDP sockets.
5. Close the Socket (For Server): Once the communication is finished, close the socket using the close() method to free up resources.

**Code and Output:**

1. Server Code:

import socket

import threading

import tkinter as tk

from tkinter import scrolledtext

HOST = '127.0.0.1'

PORT = 12340

class ChatServer:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Chat Server")

self.dark\_mode = False

*# Create GUI components*

self.chat\_area = scrolledtext.ScrolledText(root, state='disabled')

self.chat\_area.pack(padx=10, pady=10)

self.message\_frame = tk.Frame(root)

self.message\_frame.pack(padx=10, pady=10, fill=tk.X)

self.message\_entry = tk.Entry(self.message\_frame)

self.message\_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)

self.message\_entry.bind("<Return>", self.send\_message)

self.toggle\_button = tk.Button(self.message\_frame, text="🌙", command=self.toggle\_mode)

self.toggle\_button.pack(side=tk.RIGHT)

*# Initialize server socket and client list*

self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.clients = {} *# Dictionary to hold clients and their usernames*

*# Start the server*

try:

self.server\_socket.bind((HOST, PORT))

self.server\_socket.listen()

self.update\_chat\_area("Server started, waiting for connections...")

except socket.error as e:

self.update\_chat\_area(f"Socket error: {e}")

self.root.quit()

return

*# Start thread to accept connections*

threading.Thread(target=self.accept\_connections, daemon=True).start()

def accept\_connections(self):

"""Accept new client connections."""

while True:

try:

client\_socket, client\_address = self.server\_socket.accept()

threading.Thread(target=self.handle\_client, args=(client\_socket,), daemon=True).start()

except socket.error as e:

self.update\_chat\_area(f"Socket error while accepting connections: {e}")

def handle\_client(self, client\_socket):

"""Handle incoming messages from a client."""

try:

username = client\_socket.recv(1024).decode('utf-8')

self.clients[client\_socket] = username

self.update\_chat\_area(f"Connection from {username} ({client\_socket.getpeername()})")

except socket.error as e:

self.update\_chat\_area(f"Socket error while receiving username: {e}")

client\_socket.close()

return

while True:

try:

message = client\_socket.recv(1024).decode('utf-8')

if not message: *# Client has disconnected*

break

self.broadcast(f"{username}: {message}", client\_socket)

except socket.error as e:

self.update\_chat\_area(f"Socket error while handling client {username}: {e}")

break

*# Cleanup client connection*

self.remove\_client(client\_socket)

def broadcast(self, message, source\_socket):

"""Broadcast a message to all clients except the source socket."""

self.update\_chat\_area(message)

*# Broadcast the message to all clients*

for client in list(self.clients):

if client != source\_socket:

try:

client.send(message.encode('utf-8'))

except socket.error:

self.remove\_client(client)

def send\_message(self, event=None):

"""Send a server message to all clients."""

message = self.message\_entry.get()

if message:

self.broadcast(f"Server: {message}", None)

self.message\_entry.delete(0, tk.END)

def update\_chat\_area(self, message):

"""Update the chat area with a new message."""

self.chat\_area.configure(state='normal')

self.chat\_area.insert(tk.END, message + '\n')

self.chat\_area.configure(state='disabled')

self.chat\_area.yview(tk.END)

def toggle\_mode(self):

"""Toggle between light and dark mode."""

if self.dark\_mode:

self.root.configure(bg="white")

self.chat\_area.configure(bg="white", fg="black")

self.message\_entry.configure(bg="white", fg="black")

self.toggle\_button.configure(bg="lightgrey", fg="black")

self.dark\_mode = False

else:

self.root.configure(bg="black")

self.chat\_area.configure(bg="black", fg="white")

self.message\_entry.configure(bg="black", fg="white")

self.toggle\_button.configure(bg="darkgrey", fg="white")

self.dark\_mode = True

def remove\_client(self, client\_socket):

"""Remove a client from the list and close its connection."""

if client\_socket in self.clients:

username = self.clients.pop(client\_socket)

client\_socket.close()

self.update\_chat\_area(f"{username} ({client\_socket.getpeername()}) disconnected.")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

server = ChatServer(root)

root.mainloop()

1. Client Code:

import tkinter as tk

from tkinter import simpledialog, scrolledtext

import socket

import threading

HOST = '127.0.0.1'

PORT = 12340

class ChatClient:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Chat Client")

self.dark\_mode = False

*# Prompt for username*

self.username = simpledialog.askstring("Username", "Enter your username:", parent=root)

if not self.username:

self.root.quit()

return

*# Create GUI components*

self.chat\_area = scrolledtext.ScrolledText(root, state='disabled')

self.chat\_area.pack(padx=10, pady=10)

self.message\_frame = tk.Frame(root)

self.message\_frame.pack(padx=10, pady=10, fill=tk.X)

self.message\_entry = tk.Entry(self.message\_frame)

self.message\_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)

self.message\_entry.bind("<Return>", self.send\_message)

self.toggle\_button = tk.Button(self.message\_frame, text="🌙", command=self.toggle\_mode)

self.toggle\_button.pack(side=tk.RIGHT)

*# Initialize and connect socket*

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

self.socket.connect((HOST, PORT))

self.socket.send(self.username.encode('utf-8')) *# Send username to server*

except socket.error as e:

self.update\_chat\_area(f"Connection error: {e}")

self.root.quit()

return

*# Start the receive thread*

threading.Thread(target=self.receive\_messages, daemon=True).start()

def send\_message(self, event=None):

"""Send a message to the server."""

message = self.message\_entry.get()

if message:

try:

self.socket.send(message.encode('utf-8'))

self.message\_entry.delete(0, tk.END)

self.update\_chat\_area(f"You: {message}")

except socket.error:

self.update\_chat\_area("Failed to send message. Disconnected from server.")

def receive\_messages(self):

"""Receive messages from the server."""

while True:

try:

message = self.socket.recv(1024).decode('utf-8')

if message:

self.update\_chat\_area(message)

except socket.error:

self.update\_chat\_area("Connection closed by the server.")

break

def update\_chat\_area(self, message):

"""Update the chat area with a new message."""

self.chat\_area.configure(state='normal')

self.chat\_area.insert(tk.END, message + '\n')

self.chat\_area.configure(state='disabled')

self.chat\_area.yview(tk.END)

def toggle\_mode(self):

"""Toggle between light and dark mode."""

if self.dark\_mode:

self.root.configure(bg="white")

self.chat\_area.configure(bg="white", fg="black")

self.message\_entry.configure(bg="white", fg="black")

self.toggle\_button.configure(bg="lightgrey", fg="black")

self.dark\_mode = False

else:

self.root.configure(bg="black")

self.chat\_area.configure(bg="black", fg="white")

self.message\_entry.configure(bg="black", fg="white")

self.toggle\_button.configure(bg="darkgrey", fg="white")

self.dark\_mode = True

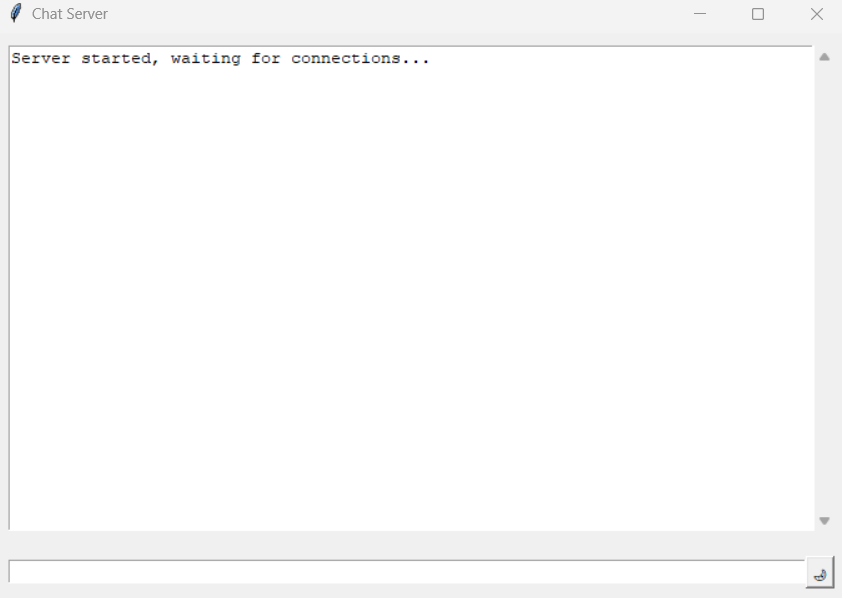
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

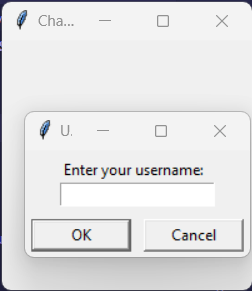
client = ChatClient(root)

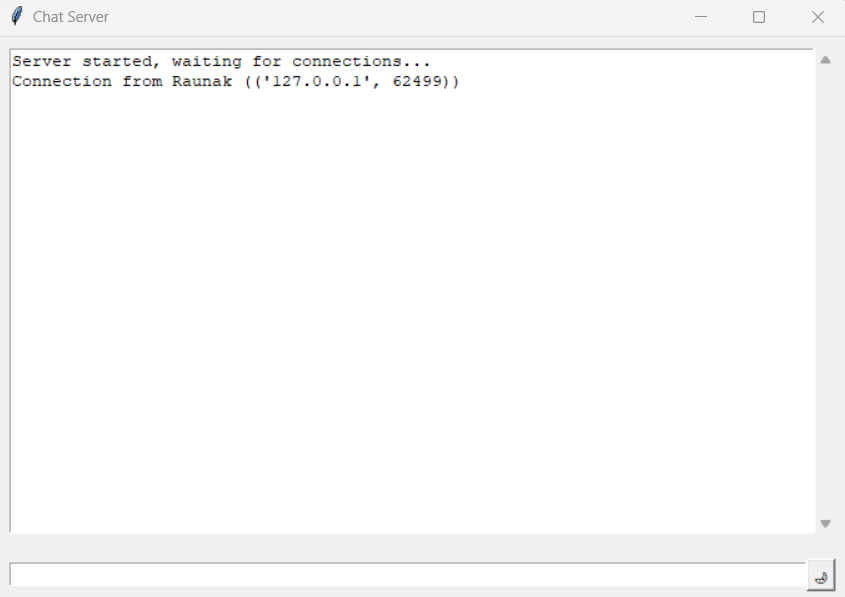
root.mainloop()

OUTPUT:

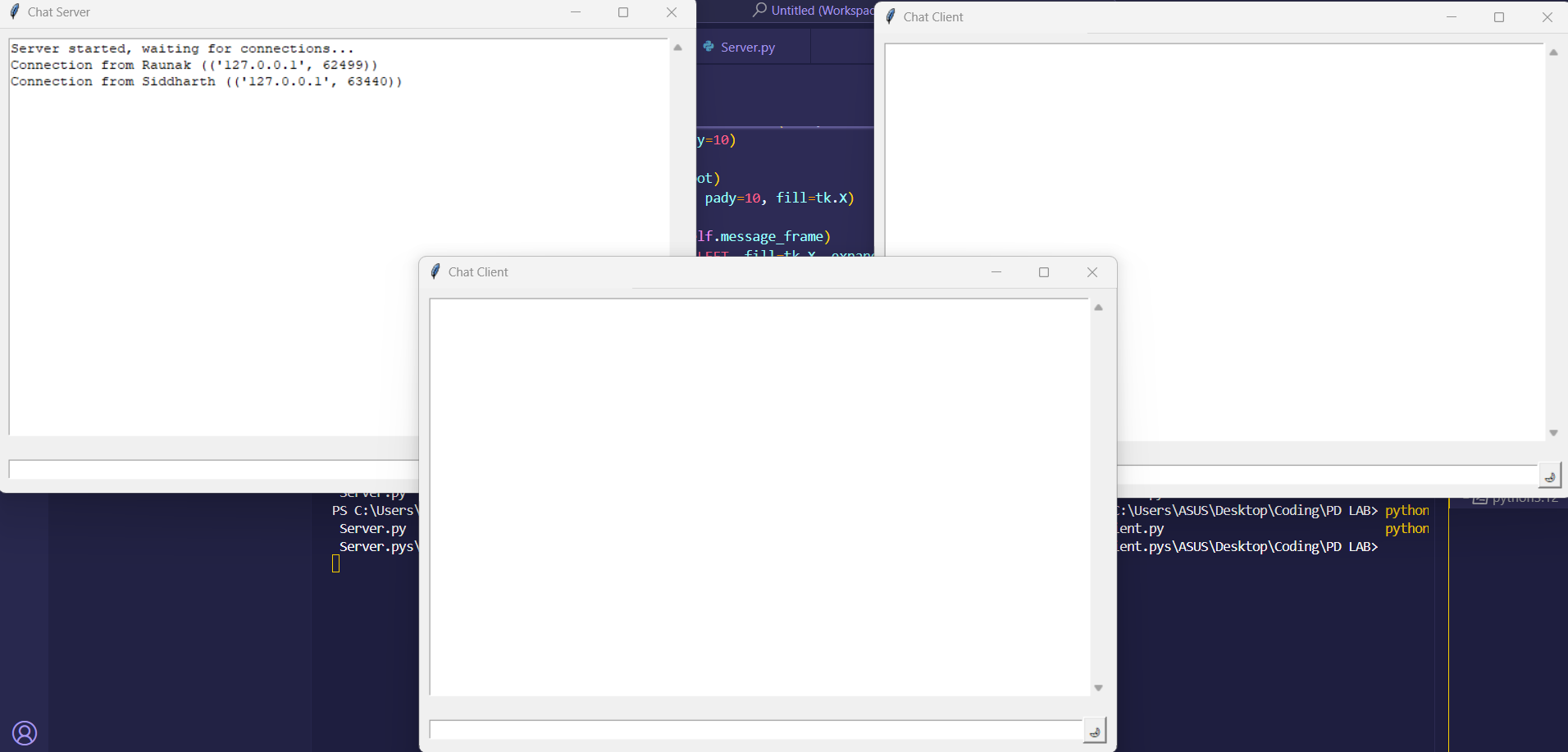


Initial Output for Server

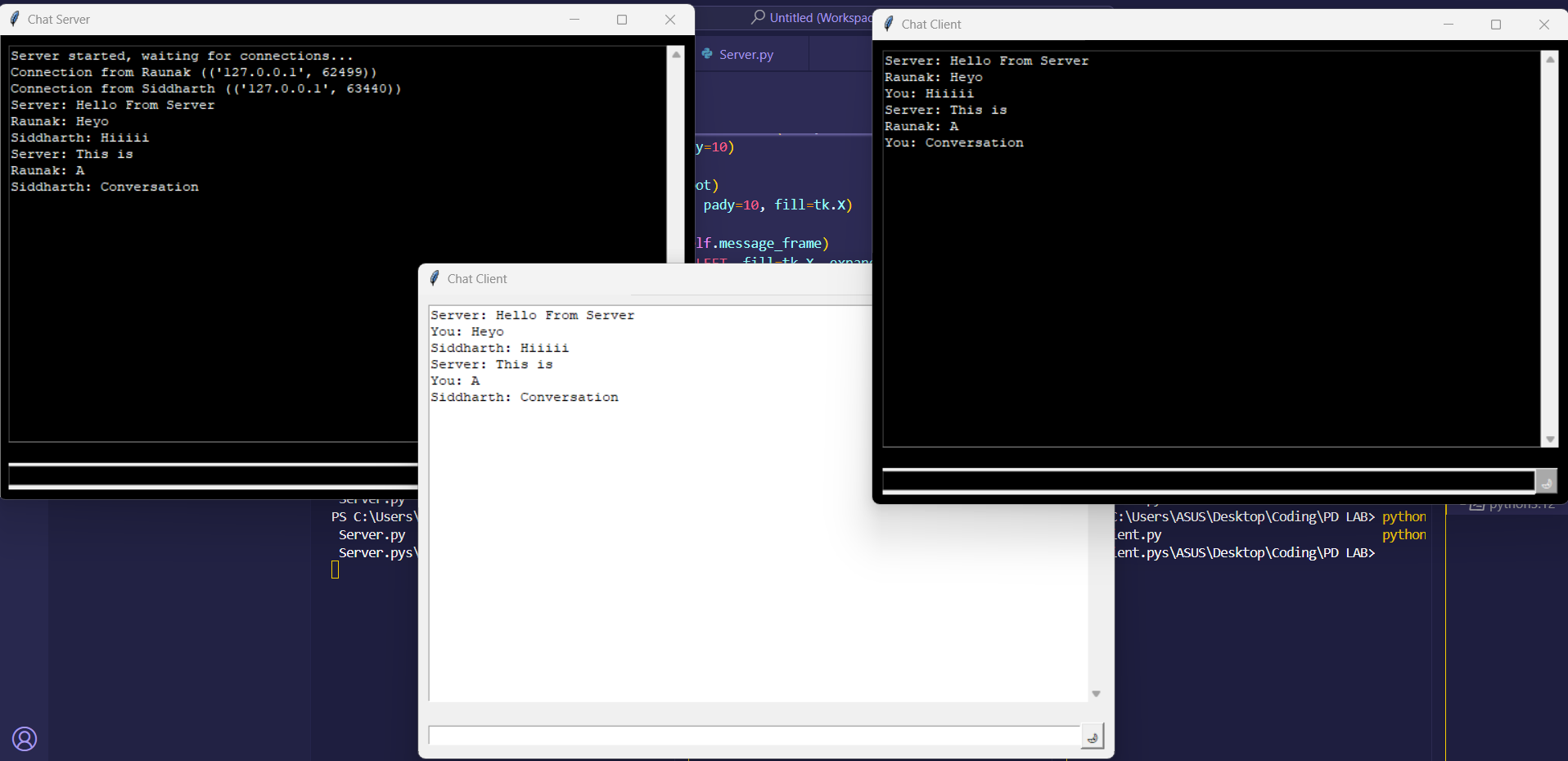
  
Initial Output for Client



After Entering your name and clicking OK



With Multiple Users



A Conversation between the Server and 2 Clients where the server and one of the clients is using dark mode

**Conclusion:**

Thus we have written a program to write a messaging application implementing socket programming and using tkinter to create an interface for the server and client to chat.