Experiment No 1

Network commands

Aim: Familiarization with network commands in python

Theory:

Python provide 2 level of access to network programming
i) low-level access: At the low-level you can access the basic socket support of the operate. System.
ii) high level access: at the high level allow to implement proto-col like HTTP, FTP

Socket as end point in a bidirectional communication channel Socket use different protocols for determine the connection type for pass to post communication bw Client and Server
Socket programming is a way of connecting two node on a network to communicate with each other. One Socket

listen on a particular portal at to, which the other socket reach out to the other to form a communication

To create a socket we use the socket.
socket() . method
socket . socket (socket_family, socket_type - protocol=0)

Socket . server meth
S. bind(): Bind address to the socket. the address contains the pair of host name and the port number
S. listen(): State the TCP listener
S. accept(): passively accept the TCP client connection and block . the connection occurs

Socket client method
S. connect(): Actively start the TCP server connection

Socket general method:
S. send(): Send the TCP message
S. sendto(): send the UDP message
S. recv(): Receive the TCP.
S. recvfrom(): Receive the UDP message
S. close(): close the socket

socket.gethostname() returns the hostname

TCP → Transmission control protocol is a connection-oriented protocol. means it requires connection to establish b/w the client and server before data can be sent

→ UDP (or Datagram Protocol) is a connectionless protocol. It doesn't establish a connection before sending data. It sends packets without guaranteeing their delivery order or error-free. It is faster than TCP but less reliable than TCP

Result:

The python network programming commands are familiarized

Experiment No 2
Client Server using TCP

Name: Basil Anil
Class: S7DS
Roll No 14

Aim: To implement client_server connection using TCP protocol

Code:

Server:

```
import socket

def start_server(host='127.0.0.1', port=65432):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
server_socket:
        server_socket.bind((host, port))
        server_socket.listen()
        print(f"Server listening on {host}:{port}")

        conn, addr = server_socket.accept()
        with conn:
            print(f"Connected by {addr}")
            while True:
                data = conn.recv(1024)
                if not data:
                    print("Connection closed by client.")
                    break
                print(f"Received: {data.decode()}")
                response = input("Enter response: ")
                conn.sendall(response.encode())

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

Client:

```
import socket
```

```
def start_client(host='localhost', port=64532):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
client_socket:
            client_socket.connect((host,port))
            message='Hello'
            client_socket.sendall(message.encode())
            print(f"Send :{message}")

            data=client_socket.recv(1032)
            print(f"Recived : {data.decode()}")

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

26/7/24

Experiment no 2

Client - Server Chat

Aim: To write a program to implement client server connection with server programming using TCP protocol

Algorithm

Server

Step 1: Create a TCP/IP socket
Step 2: Bind the socket to a specific host and port
Step 3: Listen for incoming connection
Step 4: Accepts a connection and enter a loop to receive data
Step 5: Echoes the received data back to the client

Client

Step 1: Create a TCP/IP socket
Step 2: Connect to the server
Step 3: Enter a loop to send message to the server

Step 4: Recv the echoed message from the server

Output

Server

```
server listening on ('127.0.0.1', 65432)
connected by ('127.0.0.1', 65263)
Received : hello
Respon : hey
connection closed by client
```

Client

```
connect to server at 127.0.0.1 : 65432
Enter message to send ('exit' to quit) : hello
Received from server : hey
Enter message to send (type 'exit' to quit) :
exit
closing connection
```

26/7/24

Experiment No-3

Client - Server Chat using UDP

Aim: Write a python program to implement client - server communication using socket programming with UDP as transport layer protocol

Algorithm

Step 1: Start
Step 2: Import the 'socket' module
Step 3: Define 'start_server' function with default parameters 'host = 127.0.0.1' and port = '65432'
Step 4: Within 'start_server' function.
  Step 4.1: Create a UDP socket using 'socket.socket(socket.AF_INET, socket.SOCK_DGRAM)'.
  Step 4.2: Bind the socket to the provided 'host' and 'port'
  Step 4.3. Print the message indicating the server is listening on the specified host and port
  Step 4.4: Enter an infinite loop to continuously

listen for incoming messages
Step 4.4: Receive a message and the sender address
Step 4.42: Decode and print message and the sender address
Step 4.43: If the message is 'exit' print a closing confirmation message, break the loop, and close the connection
Step 4.44: Otherwise, prompt the user to enter a response message
Step 4.45: Send the response message back to the sender using 'server_socket.sendto(response.encode(), addr)'
Step 4.5: Print server shut down message
Step 4.6: Check if the script is executed as the main program
Step 4.6.1: If true, call the 'start_server' function
Step 4.7: Stop

Client code

Step 1: Start
Step 2: Import socket module
Step 3: Define 'start_client' function with default

params = '127.0.0.1' and port = 65432'

Step 4: Within 'start_client' function:

Step 4.1: Create a UDP socket

Step 4.2: Print a message indicating the client is connected to the server at the specified host and port.

Step 4.3: Enter a infinite loop to continuously send and receive messages.

Step 4.3.1 Prompt the user to enter a message to send to the server.

Step 4.3.2: If the message is empty send the empty message to the server, print a closing connection message and break the loop

Step 4.3.3: Encode the message and send it to the server using

Step 4.3.4: Receive a response from the server.

Step 4.3.5: Print the received response from the server

Step 5: Check if the script is executed as the main program

Step 6: Stop

Experiment No 3
Client Server using UDP

Name: Basil Anil
Class: S7DS
Roll No 14

29/8/24

Aim: To implement client_server connection using UDP protocol

Code:

Client:

```
import socket

def start_udp_client(host='localhost', port=65433):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as
client_socket:
        message = 'Hello, Server!'
        client_socket.sendto(message.encode(), (host, port))
        print(f"Sent message: {message}")

        data, _ = client_socket.recvfrom(1024)
        print(f"Received echo: {data.decode()}")

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

Server:

```
import socket

def start_udp_server(host='localhost', port=65433):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as
server_socket:
        server_socket.bind((host, port))
        print(f"Server listening on {host}:{port}")

        while True:
```

```
        data, addr = server_socket.recvfrom(1024)
        if data:
            print(f"Received data: {data.decode()} from {addr}")
            response = f"Echo: {data.decode()}"
            server_socket.sendto(response.encode(), addr)

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

Output

Server

Server listening on 127.0.0.1: 65432
Received message from ('127.0.0.1', 62988): hello
Enter response: hey
Received message from ('127.0.0.1', 62988): hello.
Enter response: hey
Received message from ('127.0.0.1', 62988): exit
Received exit command from ('127.0.0.1', 62988).
closing connection
Server has shut down

Client

Connected to server at 127.0.0.1: 65432
Enter message to send (type 'exit' to quit
hello.
Received from server: hey
Enter message to send (type 'exit' to quit):
exit
closing connection

29/7/24

12

Experiment - 4

Multi-user chat using TCP

Aim:

write a python program to implement multi user chat service with TCP as transport layer protocol using GUI

Algorithm

client.py

step 1: Initialize the Tkinter Root Window
create the root window. Set the window title to 'client'.

step 2: Create GUI components
add a text area for displaying messages
add an entry field for inputting messages
Bind the entry field to send message on pressing enter

step 3: Set up the client socket
create a TCP socket
connect to the server

step 4: Start listening for messages
start a new thread to listen for incom

my message
Step 5: Run the Tkinter main loop
Step 6: Continously receive message from the server
Step 7: Display each message in the text area
Step 8: Get the message from the entry field and send it
Step 9: Close the client

Server.Py

Algorithm

Step 1: Create the root window
Step 2: Set the window title to 'Server'
Step 3: Add a text area and entry fields
Step 4: Create a TCP socket and bind it to the port
Step 5: Start a new thread to accept incoming client connections
Step 6: Run the main loop
Step 7: Continuously accept new client connections and add each new client to the client list

step 8: Continuously receive message from the client and broadcast each message

step 9: Close all client sockets and the server socket as well

Experiment No 4

Multiuser Chatroom using TCP

Name: Basil Anil
Class: S7DS
Roll No 14

```
import socket
import threading

class ChatServer:
    def __init__(self, host='127.0.0.1', port=12342):
        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.server_socket.bind((host, port))
        self.server_socket.listen()
        self.clients = []

    def broadcast(self, message, client_socket):
        for client in self.clients:
            if client != client_socket:
                try:
                    client.send(message)
                except:
                    client.close()
                    self.clients.remove(client)

    def handle_client(self, client_socket):
        while True:
            try:
                message = client_socket.recv(1024)
                if message:
                    self.broadcast(message, client_socket)
                else:
                    client_socket.close()
                    self.clients.remove(client_socket)
                    break
            except:
                client_socket.close()
```

```
            self.clients.remove(client_socket)
            break

    def start(self):
        print("Server started...")
        while True:
            client_socket, addr = self.server_socket.accept()
            print(f"New connection from {addr}")
            self.clients.append(client_socket)
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

if __name__ == "__main__":
    server = ChatServer()
    server.start()
```

Aim: To implement multi user chatbox using TCP protocol

Code:

Server:

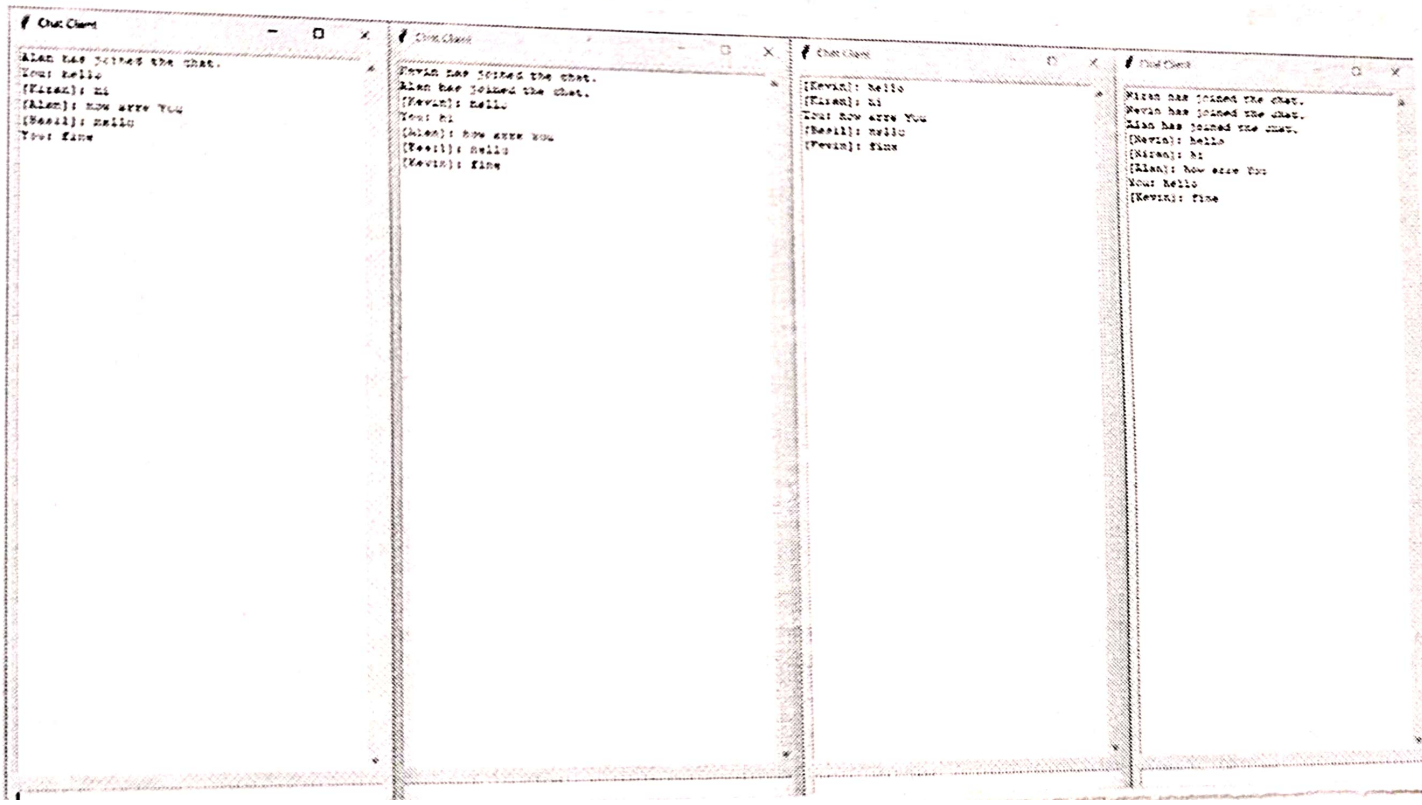
```
import socket
import threading

class ChatServer:
    def __init__(self, host='127.0.0.1', port=12342):
        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.server_socket.bind((host, port))
        self.server_socket.listen()
        self.clients = []

    def broadcast(self, message, client_socket):
        for client in self.clients:
            if client != client_socket:
                try:
                    client.send(message)
                except:
                    client.close()
                    self.clients.remove(client)
```

```
    def handle_client(self, client_socket):
        while True:
            try:
                message = client_socket.recv(1024)
                if message:
                    self.broadcast(message, client_socket)
                else:
                    client_socket.close()
                    self.clients.remove(client_socket)
                    break
            except:
                client_socket.close()
                self.clients.remove(client_socket)
                break

    def start(self):
        print("Server started...")
        while True:
            client_socket, addr = self.server_socket.accept()
            print(f"New connection from {addr}")
            self.clients.append(client_socket)
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

if __name__ == "__main__":
    server = ChatServer()
    server.start()

Client:
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext
from tkinter import simpledialog

class TCPChatClient:
    def __init__(self, host='127.0.0.1', port=12342):
```

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

        # Get username
        self.username = simpledialog.askstring("Username", "Enter your
username:", parent=tk.Tk())
        if not self.username:
            self.username = "Anonymous"

        self.client_socket.send(self.username.encode('utf-8'))

        self.root = tk.Tk()
        self.root.title("TCP Chat Client")
        self.create_widgets()
        self.receive_thread = threading.Thread(target=self.receive_messages)
        self.receive_thread.start()

    def create_widgets(self):
        self.text_area = scrolledtext.ScrolledText(self.root, state='disabled')
        self.text_area.pack(padx=10, pady=10)

        self.entry = tk.Entry(self.root)
        self.entry.pack(fill='x',
padx=10, pady=10)
        self.entry.bind("<Return>", self.send_message)

        self.text_area.tag_configure('red', foreground='red')

        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

    def send_message(self, event=None):
        message = self.entry.get()
        if message:
            self.text_area.config(state='normal')
            self.text_area.insert(tk.END,"You: "+ message + '\n', 'red')
            self.text_area.config(state='disabled')
            self.text_area.yview(tk.END)
            self.client_socket.send(message.encode('utf-8'))
```

```
        self.entry.delete(0, tk.END)

    def receive_messages(self):
        while True:
            try:
                message = self.client_socket.recv(1024).decode('utf-8')
                if message:
                    self.text_area.config(state='normal')
                    self.text_area.insert(tk.END, message + '\n', 'black')
                    self.text_area.config(state='disabled')
                    self.text_area.yview(tk.END)
                else:
                    break
            except:
                break

    def on_closing(self):
        self.client_socket.close()
        self.root.destroy()

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    client = TCPChatClient()
    client.run()
```

Output


Chat Client
Alan has joined the chat.
You: hello
[Kiran]: hi
[Alan]: how are You
[Sunil]: hello
You: fine
Chat Client
Kevan has joined the chat.
Alan has joined the chat.
[Kevan]: hello
You: hi
[Alan]: how are You
[Sunil]: hello
[Kevin]: fine
Chat Client
[Kevan]: hello
[Kiran]: hi
You: how are You
[Sunil]: hello
[Kevin]: fine
Chat Client
Kiran has joined the chat.
Kevin has joined the chat.
Alan has joined the chat.
[Kevin]: hello
[Kiran]: hi
[Alan]: how are You
You: hello
[Kevin]: fine