EC3022D
COMPUTER NETWORKS

ASSIGNMENT-2

Submitted by,
Salvin Saji
B200930EC
EC-04

Aim :
Create a chat application using socket programming.

Theory :
Socket programming is a way of communicating between processes running on different devices over a network. In Python, the socket module provides a way to work with sockets, allowing us to create networked applications.
To create a chat application using socket programming in Python, we can follow these steps:

1. Create a server: The server is responsible for accepting incoming connections from clients and handling those connections. We can create a server using the socket module in Python.
2. Bind the server to a port: The server needs to be bound to a port number on the host machine. This allows clients to connect to the server at that port number. We can use the bind() method of the socket object to bind the server to a port.
3. Listen for incoming connections: The server needs to be set up to listen for incoming connections from clients. We can use the listen() method of the socket object to make the server listen for incoming connections.
4. Accept incoming connections: Once a client connects to the server, the server needs to accept that connection and create a separate socket object to communicate with that client. We can use the accept() method of the socket object to accept incoming connections from clients.
5. Create a client: Clients are responsible for connecting to the server and sending and receiving messages to and from the

server. We can create a client using the socket module in Python.

6. Connect the client to the server: The client needs to be connected to the server at a specific IP address and port number. We can use the connect() method of the socket object to connect the client to the server.
7. Send and receive messages: Once the client is connected to the server, the client can send and receive messages to and from the server using the send() and recv() methods of the socket object.

To create a chat application using socket programming, we can implement these steps by setting up a server that accepts incoming connections from clients and creates a separate thread for each connected client. Each thread listens for messages from its respective client and broadcasts those messages to all other connected clients. Clients can send messages to the server, which broadcasts those messages to all other connected clients. In summary, socket programming in Python provides a simple and powerful way to create networked applications like chat applications. By following the steps outlined above, we can create a chat application that allows clients to connect to a server and send and receive messages in real-time.

Server code :

```
import socket
import threading

PORT = 9095
IP = socket.gethostbyname(socket.gethostname())
```

```
HEADER = 64
FORMAT = 'utf-8'
DISC_MSG = '!DISCONNECT'

server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server.bind((IP,PORT))

client_list = []

def send(msg,client):
    msglen = len(msg)
    len_enc = str(msglen).encode(FORMAT)
    len_enc += b' '*(HEADER - len(len_enc))
    client.send(len_enc)
    client.send(msg.encode(FORMAT))

def broadcast(msg,conn):
    for client in client_list:
        if client!=conn:
            send(msg,client)

def handle_client(conn,addr):
    print(f'[SERVER]{addr} connected')
    client_list.append(conn)
    connected = True

    while connected:
        msglen = conn.recv(HEADER).decode(FORMAT)
        if msglen:
            mlen = int(msglen)
```

```
            msg = conn.recv(mlen).decode(FORMAT)
            print(f"{msg}")
            broadcast(msg,conn)
            if(msg.split(' ')[-1] == DISC_MSG):
                connected = False
                client_list.remove(conn)
    conn.close()
    print(f"[SERVER] active connections = {threading.activeCount()
- 2}")

def start():
    server.listen()
    print(f"[SERVER] server started at {IP}:{PORT}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args =
(conn,addr))
        thread.start()
        print(f"[SERVER] active connections =
{threading.activeCount() - 1}")

print('[SERVER] Starting ...')
start()
```

Code Explanation :

This Python script defines a server socket that listens for incoming connections from client sockets. The socket library is used to create a server socket that binds to a specified IP address and port number.

The script then defines several constants, including the header size for messages, the encoding format, and the disconnect message.
The script defines three functions: send(), broadcast(), and handle_client(). The send() function takes a message and a client socket as input, encodes the message, and sends it to the client using the socket. The broadcast() function takes a message and a connection as input and sends the message to all clients except for the one specified by the connection.
The handle_client() function is the main function that handles each client connection. It first adds the client to a list of active connections and prints a message indicating that the client has connected. It then enters a loop that receives messages from the client, prints them to the console, and broadcasts them to all other clients using the broadcast() function. If the message contains the disconnect message, the function removes the client from the list of active connections and exits the loop. The function then closes the connection and prints the number of active connections to the console.
The start() function starts the server socket and enters a loop that listens for incoming connections. For each new connection, the function creates a new thread that runs the handle_client() function and starts the thread.
Finally, the main program prints a message indicating that the server is starting and calls the start() function.

Client Code :

```
import socket
```

```
import threading

PORT = 9095
IP = socket.gethostbyname(socket.gethostname())
HEADER = 64
FORMAT = 'utf-8'
DISC_MSG = '!DISCONNECT'

client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
client.connect((IP,PORT))


def send(msg):
    msglen = len(msg)
    len_enc = str(msglen).encode(FORMAT)
    len_enc += b' '*(HEADER - len(len_enc))
    client.send(len_enc)
    client.send(msg.encode(FORMAT))

def receive():
    global connected
    while connected:
        msglen = client.recv(HEADER).decode(FORMAT)
        if msglen:
            mlen = int(msglen)
            msg = client.recv(mlen).decode(FORMAT)
            print(msg)

def send_msg():
    global connected
```

```
    while connected:
        m = input()
        if m == '!DISCONNECT':
            connected = False
        m = f'[{name}] : {m}'
        send(m)

print("Welcome to chatroom")
name = input("Enter your name : ")
connected = True

s_thread = threading.Thread(target = send_msg)
s_thread.start()

r_thread = threading.Thread(target = receive)
r_thread.start()
```

Code explanation :

This Python script defines a simple client-server chat application using socket programming. The socket library is used to create a client socket that connects to a server socket at a specified IP address and port number.

The script then defines three functions: send(), receive(), and send_msg(). The send() function takes a message as input, encodes it, and sends it to the server. It first calculates the length of the message, encodes the length and the message in the specified format, and sends them to the server using the client socket.

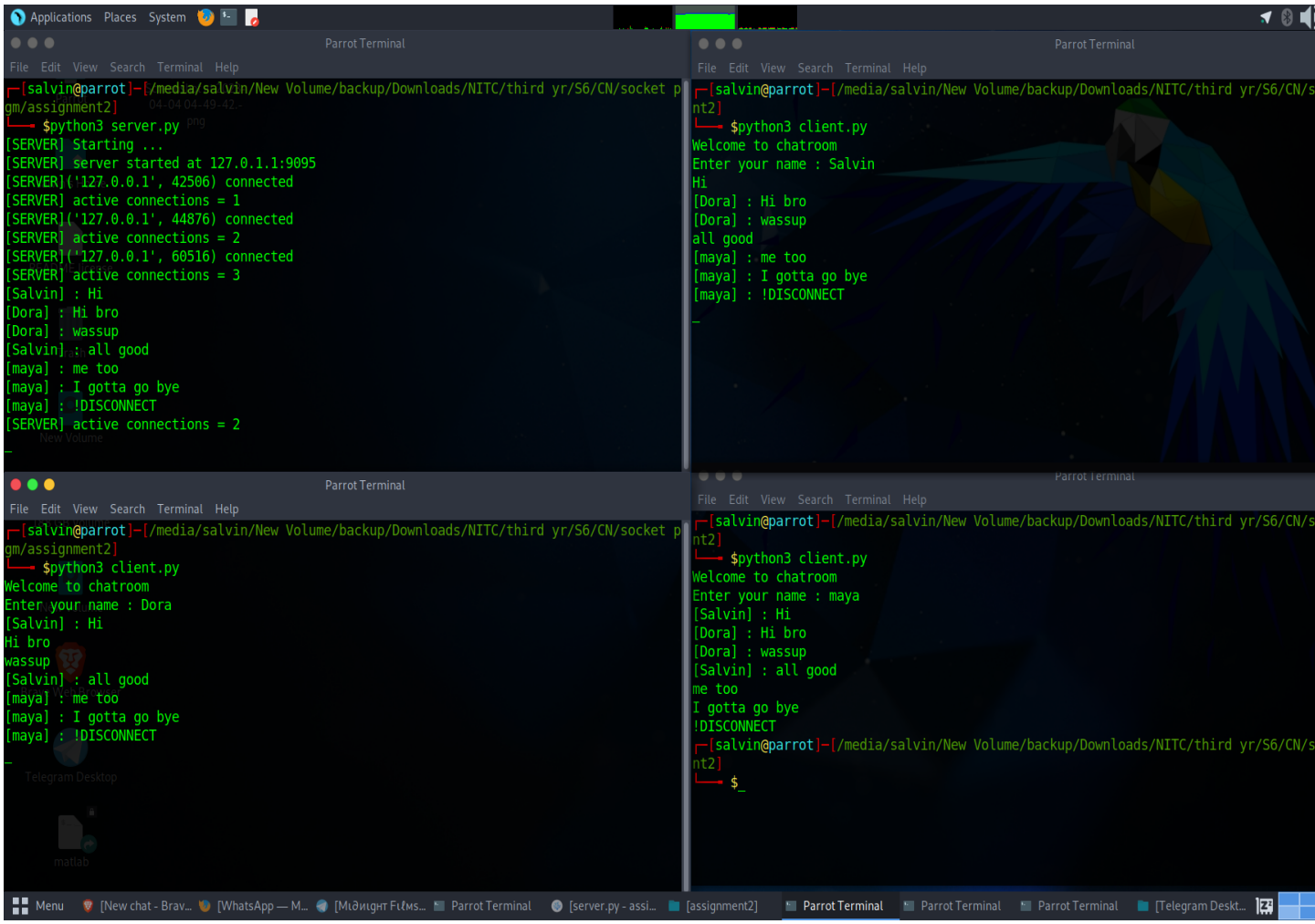
The receive() function continuously receives messages from the server and prints them to the console. It first receives the length of

the message from the server, converts it to an integer, and then receives the message itself. The message is then printed to the console.

The send_msg() function continuously takes user input from the console and sends it to the server using the send() function. If the user inputs the command !DISCONNECT, the connected variable is set to False, which causes the while loop to exit and the program to terminate.

Finally, the main program prompts the user to enter their name, starts the receive() and send_msg() functions in separate threads, and waits for the threads to complete.

Output :



Conclusion :

The report has discussed the basic concepts of socket programming, such as creating sockets, binding them to specific IP addresses and ports, and sending and receiving messages.

Furthermore, the report has demonstrated how to create a simple chat application that allows multiple clients to connect to a server and communicate with each other in real-time. The client-side

code and server-side code were provided, along with a brief explanation of their functionalities.