NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

PROJECT REPORT

COURSE TITLE

DATA STRUCTURES AND ALGORITHMS

COURSE CODE

CS-216

COURSE TEACHER

MS ALISHBA MASOOD

GROUP MEMBER S

OMER AFTAB TC-23074

MUHAMMAD OWAIS TC-23045

DEPARTMENT

TELECOMMUNICATION ENGINEERING SPRING -2025

CONTENT

1) INTRODUCTION: -

This project is a ***TCP-based Chat Room Server Using Python Language*** that allows multiple clients to connect and communicate with each other in real-time. The server broadcasts messages from one client to all other connected clients, creating a group chat environment. The server uses a linked list to manage connected clients, ensuring efficient addition and removal of clients as they join or leave the chat.

1.1) PROBLEM STATEMENT:

In real-time communication systems such as chat servers, managing a fluctuating number of client connections efficiently poses a significant challenge. Traditional static data structures like arrays lack the flexibility needed to handle dynamic user activity, where clients can join or disconnect at any moment. Moreover, ensuring smooth concurrent message handling, maintaining session integrity, and logging conversations add to the complexity. To address these issues, this project proposes a solution that employs a singly linked list for dynamic client management, threading for simultaneous client handling, and file logging to preserve the chat history all integrated into a Python-based server application designed for real-time text-based communication.

1.2) USED OF DATA STRUCTURE :

2. **Dynamic Memory**: No need for fixed size, adjusts as clients join/leave.
3. **Efficient Client Management**: Easy to add/remove clients with pointers.
4. **Simple Traversal**: Can easily iterate through clients for message broadcasting.
5. **Flexible Data Handling**: Efficient removal of disconnected clients.
6. **Scalable**: Handles growing numbers of clients without memory issues.

2) FUNCTIONALITY:

The TCP-based chat room server with a linked list allows for dynamic client management, where each client's details (socket, address) are stored in a linked list. The server can efficiently broadcast messages to all clients by traversing the list. It handles client connections and disconnections in real-time, adding and removing clients as they join or leave. The linked list structure ensures flexible memory management, allowing for scalability as the number of clients grows. Additionally, threading enables the server to handle multiple clients simultaneously, ensuring smooth communication.

3) CODE EXPLANATION:-

3.1) NODE STRUCTURE:

```
class node(object):
    def __init__(self, client, nick, ip):
        self.client = client
        self.ipv4 = ip
        self.nickname = nick
        self.nextnode = None
```

This node contain the information of a single client like its IP address, nickname and also the pointer of the next client node.

3.2) CLASS DESIGN:

```
class clients(object):
    def __init__(self):
        self.head = None

    def __iter__(self):
        current = self.head
        while current:
            yield current
            current = current.getnextnode()
```

The Client class help to maintain all the client connected to the server while using ("_iter_") function so we can access a class in the form of loop.

3.3) BROADCAST FUNCTION:

```
def broadcast(message, sender):
    for i in clients_list:
        if i.getclient() != sender:
            try:
                i.getclient().send(message)
            except:
                continue
```

It broadcast the message of a client to all other connected client while check that the message is not forward to the origin client.

3.4) CHAT HISTORY:

```
def logmsg(dmsg):
    try:
        with open("chat.txt", "a", encoding="utf-8") as file:
            file.write(dmsg + "\n")
    except Exception as e:
        print(f"Failed to write to log: {e}")
```

This function record each and every message conversation between different clients for security purpose.

4) OPERATIONS OF LINKED LIST:

There are many operations which can be performed using linked list data structure but the primary operations we used in our project are:-

4.1) INSERTION OPERATION:

```
    def addclient(self, client, nick, ip):
        new_client = node(client, nick, ip)
        new_client.setnextnode(self.head)
        self.head = new_client
```

It creates a new Client Node for the connected client and also it can Inserts the new node at the head of the linked list. It updates the client_list_head to point to the new node.

4.2) DELELTION OPERATION:

```
def removeclient(self, client):
    current_node = self.head
    prev_node = None
    while current_node and current_node.getclient() != client:
        prev_node = current_node
        current_node = current_node.getnextnode()
    if not current_node:
        return
    if prev_node is None:
        self.head = current_node.getnextnode()
    else:
        prev_node.setnextnode(current_node.getnextnode())
```

Traverses the linked list to find the client node matching the given connection and also it can removes the node from the linked list by adjusting the next pointers of adjacent nodes.

4.3) SERACHING OPERATION:

```
def allclients(self):
    current_client = self.head
    count = 0
    print("All Clients....")
    while current_client:
        print(current_client)
        count += 1
        current_client = current_client.getnextnode()
    print("Total number of clients:", count)
```

Iterates through the linked list to find the IP addresses of all connected clients and print the list of number of clients connected to it.

5) OUTPUT:-

5.1.) Server Part:

```
[SERVER RUNNING] Listening on 0.0.0.0:5554...

connected with ('127.0.0.1', 51633)
All Clients....

ip: ('127.0.0.1', 51633),
NickName = owais
Total number of clients: 1
connected with ('127.0.0.1', 51634)
All Clients....

ip: ('127.0.0.1', 51634),
NickName = ali

ip: ('127.0.0.1', 51633),
NickName = owais
Total number of clients: 2
('127.0.0.1', 51634) Disconnected
```

5.2.) Client Part:

```
Choose nickname: owais
Connected With Server.

ali joined the server

ali left the chat
```

6) LIMITATION:-

i) No room for private chat between two clients.

ii) GUI is not used in server interfacing and connection.

Iii) Server cannot remove the client by itself until or unless client leave the server connection.

7) OVERALL CODE: -

7.1.)(SERVER SIDE)

```
import socket
import threading
class node(object):
    def __init__(self, client, nick, ip):
        self.client = client
        self.ipv4 = ip
        self.nickname = nick
        self.nextnode = None
    def __str__(self):
        return f'\nip: {self.ipv4},\nNickName = {self.nickname}'
    def setnextnode(self, value):
        self.nextnode = value
    def getnextnode(self):
        return self.nextnode
    def getclient(self):
        return self.client
    def getnick(self):
        return self.nickname
    def getip(self):
        return self.ipv4
class clients(object):
    def __init__(self):
        self.head = None
    def __iter__(self):
        current = self.head
        while current:
```

```
        yield current
        current = current.getnextnode()
    def addclient(self, client, nick, ip):
        new_client = node(client, nick, ip)
        new_client.setnextnode(self.head)
        self.head = new_client
    def removeclient(self, client):
        current_node = self.head
        prev_node = None
        while current_node and current_node.getclient() != client:
            prev_node = current_node
            current_node = current_node.getnextnode()
        if not current_node:
            return
        if prev_node is None:
            self.head = current_node.getnextnode()
        else:
            prev_node.setnextnode(current_node.getnextnode())
    def search_client(self, client):
        current_client = self.head
        while current_client:
            if current_client.getclient() == client:
                return current_client
    def allclients(self):
        current_client = self.head
        count = 0
        print("All Clients....")
        while current_client:
```

```
            print(current_client)
            count += 1
            current_client = current_client.getnextnode()
        print("Total number of clients:", count)
host = '0.0.0.0'
port = 5554
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
clients_list = clients()
def broadcast(message, sender):
    for i in clients_list:
        if i.getclient() != sender:
            try:
                i.getclient().send(message)
            except:
                continue
def logmsg(dmsg):
    try:
        with open("chat.txt", "a", encoding="utf-8") as file:
            file.write(dmsg + "\n")
    except Exception as e:
        print(f"Failed to write to log: {e}")
def handle(client):
    while True:
        try:
            message = client.recv(1024)
            if not message:
```

```
            break
        decoded_msg = message.decode('ascii').strip()
        if decoded_msg.lower() == 'exit':
            current_client = clients_list.search_client(client)
            if current_client:
                broadcast(f'{current_client.getnick()} left the chat'.encode('ascii'), client)
                print(f'{current_client.getip()} Disconnected')
                clients_list.removeclient(client)
            client.close()
            break
        broadcast(message, client)
        logmsg(decoded_msg)
    except Exception as e:
        current_client = clients_list.search_client(client)
        if current_client:
            broadcast(f'{current_client.getnick()} left the chat'.encode('ascii'), client)
            print(f'{current_client.getip()} Disconnected due to error: {e}')
            clients_list.removeclient(client)
        client.close()
        break
def Server():
  print(f"[SERVER RUNNING] Listening on {host}:{port}...\n")
  while True:
    client, address = server.accept()
    print(f'connected with {str(address)}')
    client.send("Choose Nickname".encode('ascii'))
    nick = client.recv(1024).decode('ascii')
    clients_list.addclient(client, nick, address)
```

```
        clients_list.allclients()
        broadcast(f'{nick} joined the server \n'.encode('ascii'), client)
        client.send("Connected With Server. \n".encode('ascii'))
        thread = threading.Thread(target=handle, args=(client,))
        thread.start()
Server()
```

7.2.) CLIENT SIDE:-

```
import socket
import threading
nick = input("Choose nickname: ")
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('localhost', 5554))
def recieve():
    while True:
        try:
            msg = client.recv(1024).decode('ascii')
            if msg == "Choose Nickname":
                client.send(nick.encode('ascii'))
            else:
                print(msg)
        except:
            print("An error occurred...")
            client.close()
            break
def write():
    while True:
        msg_input = input("")
        if msg_input.strip().lower() == "exit":
```

```
            client.send("exit".encode('ascii'))
            client.close()
            break
        msg = f'{nick}: {msg_input}'
        client.send(msg.encode('ascii'))
recieve_thread = threading.Thread(target=recieve)
recieve_thread.start()
write_thread = threading.Thread(target=write)
write_thread.start()
```

8) CONCLUSION: -

The TCP based Chat Room Server project is not a easy task to accomplish. While creating this project it help us to enhance our knowledge related to networking as well as implementation of real life communication through coding .Using python language as a programming language since it provide a platform to accomplish our goal through a built in libraries and also easy in syntax to understand the complexity our easily.