Submission: UDP Chat Program Supporting Multiple Clients

Overview:

This project implements a network chat program that supports multiple clients using the UDP protocol. The server tracks multiple clients, broadcasts messages to them asynchronously, allows setting nicknames, and handles clients exiting the chat room.

Step 1: Server Supporting Multiple Clients

Server Code:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Collections.Generic;

class UdpChatServer
{
    private static int serverPort = 8080;
    private static UdpClient udpServer;
    private static List<IPEndPoint> connectedClients = new List<IPEndPoint>(); // List to
        store multiple clients

    public static async Task Main(string[] args)
    {
        udpServer = new UdpClient(serverPort);
        Console.WriteLine($"Server is listening on port {serverPort}...");

        while (true) {
            try {
                // Receiving messages from clients
                UdpReceiveResult receivedResult = await udpServer.ReceiveAsync();
                IPEndPoint clientEndPoint = receivedResult.RemoteEndPoint;
                string receivedMessage = Encoding.UTF8.GetString(receivedResult.Buffer);

                // Add client if it's a new connection
                if (!connectedClients.Contains(clientEndPoint)) {
                    connectedClients.Add(clientEndPoint);
                    Console.WriteLine($"New client connected: {clientEndPoint.Address}:
        {clientEndPoint.Port}");
                    BroadcastMessage($"Client {clientEndPoint.Address}:
        {clientEndPoint.Port} has joined the chat.", clientEndPoint);
                }

                // Output the received message
                Console.WriteLine($"Client {clientEndPoint.Address}:{clientEndPoint.Port}
        says: {receivedMessage}");

                // Broadcast the message to all other clients
                BroadcastMessage($"Client {clientEndPoint.Address}:{clientEndPoint.Port}
        says: {receivedMessage}", clientEndPoint);
            } catch (Exception ex) {
                Console.WriteLine("Error receiving message: " + ex.Message);
            }
        }
    }
```

```
    // Broadcast a message to all connected clients except the sender
    private static async void BroadcastMessage(string message, IPEndPoint sender)
    {
        byte[] messageBytes = Encoding.UTF8.GetBytes(message);
        foreach (var client in connectedClients)
        {
            if (!client.Equals(sender)) // Don't send the message back to the sender
            {
                await udpServer.SendAsync(messageBytes, messageBytes.Length, client);
            }
        }
    }
}
```

Step 2: Adding Nicknames

Client Code:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;

class UdpChatClient
{
    private static UdpClient udpClient;
    private static string serverIp = "127.0.0.1";  // Server IP
    private static int serverPort = 8080;          // Server port#
    private static int localPort = 0;              // Auto-allocation
    private static string nickname;                // User's nickname

    public static async Task Main(string[] args)
    {
        udpClient = new UdpClient(localPort);
        Console.WriteLine("Enter your nickname:");
        nickname = Console.ReadLine();

        if (string.IsNullOrEmpty(nickname)) {
            nickname = "Anonymous";
        }

        Console.WriteLine($"UDP Chat Client started as {nickname}... Type messages to send
        to the server...");

        // Send the nickname as the first message to the server
        await SendNicknameAsync();

        // Receiving messages from the server asynchronously
        Task receiveTask = ReceiveMessagesAsync();

        // Message sending loop
        await SendMessagesAsync();
    }

    private static async Task SendNicknameAsync()
    {
        IPEndPoint serverEndPoint = new IPEndPoint(IPAddress.Parse(serverIp), serverPort);
        byte[] nicknameBytes = Encoding.UTF8.GetBytes(nickname);
        await udpClient.SendAsync(nicknameBytes, nicknameBytes.Length, serverEndPoint);
    }

    private static async Task SendMessagesAsync()
```

```
    {
        IPEndPoint serverEndPoint = new IPEndPoint(IPAddress.Parse(serverIp), serverPort);

        while (true) {
            try {
                string message = Console.ReadLine();
                if (!string.IsNullOrEmpty(message)) {
                    string fullMessage = $"{nickname}: {message}";
                    byte[] messageBytes = Encoding.UTF8.GetBytes(fullMessage);
                    await udpClient.SendAsync(messageBytes, messageBytes.Length,
    serverEndPoint);
                    Console.WriteLine($"Sent: {fullMessage}");
                }
            } catch (Exception ex) {
                Console.WriteLine("Error sending message: " + ex.Message);
            }
        }
    }

    private static async Task ReceiveMessagesAsync()
    {
        IPEndPoint serverEndPoint = new IPEndPoint(IPAddress.Any, 0);

        while (true) {
            try {
                UdpReceiveResult receivedResult = await udpClient.ReceiveAsync();
                string receivedMessage = Encoding.UTF8.GetString(receivedResult.Buffer);
                Console.WriteLine($"Server: {receivedMessage}");
            } catch (Exception ex) {
                Console.WriteLine("Error receiving message: " + ex.Message);
            }
        }
    }
}
```

Step 3: Handling Client Exit

Client Exit Code:

```
private static async Task SendMessagesAsync()
{
    IPEndPoint serverEndPoint = new IPEndPoint(IPAddress.Parse(serverIp), serverPort);

    while (true) {
        try {
            string message = Console.ReadLine();

            if (message == "/exit") {
                await SendExitMessageAsync(serverEndPoint);
                udpClient.Close();
                Console.WriteLine("You have left the chat.");
                break; // Exit the loop and terminate the client
            }

            if (!string.IsNullOrEmpty(message)) {
                string fullMessage = $"{nickname}: {message}";
                byte[] messageBytes = Encoding.UTF8.GetBytes(fullMessage);
                await udpClient.SendAsync(messageBytes, messageBytes.Length,
    serverEndPoint);
                Console.WriteLine($"Sent: {fullMessage}");
            }
        } catch (Exception ex) {
            Console.WriteLine("Error sending message: " + ex.Message);
        }
```

```
    }
}

private static async Task SendExitMessageAsync(IPEndPoint serverEndPoint)
{
    string exitMessage = $"{nickname} has left the chat.";
    byte[] exitMessageBytes = Encoding.UTF8.GetBytes(exitMessage);
    await udpClient.SendAsync(exitMessageBytes, exitMessageBytes.Length, serverEndPoint);
}
```

Server Cleanup Code:

```
private static async void BroadcastMessage(string message, IPEndPoint sender)
{
    byte[] messageBytes = Encoding.UTF8.GetBytes(message);
    List<IPEndPoint> clientsToRemove = new List<IPEndPoint>();

    foreach (var client in connectedClients) {
        if (!client.Equals(sender)) {
            try {
                await udpServer.SendAsync(messageBytes, messageBytes.Length, client);
            } catch (Exception) {
                clientsToRemove.Add(client); // If sending fails, mark the client for
     removal
            }
        }
    }

    // Remove clients that failed to receive the message
    foreach (var client in clientsToRemove) {
        connectedClients.Remove(client);
        Console.WriteLine($"Client {client.Address}:{client.Port} removed from the
     list.");
    }
}
```

Result:

- A UDP server is implemented that supports multiple clients.
- Clients can join, set nicknames, chat, and exit the chat room.
- Messages are broadcasted from the server to all clients, and the server handles both client connections and disconnections properly.

Screenshots:

https://ibb.co/jgygbTT