LAN Chat Project using Python

The LAN Chat project aims to create real-time communication between users within a Local Area Network (LAN) environment. By using Python's socket programming capabilities, we establish a client-server architecture where multiple clients can connect to a central server and exchange messages with each other.

Key Components:

1. **Server**: The central component of the project, responsible for managing connections from multiple clients, receiving and relaying messages between clients, and maintaining the overall functionality of the chat room.

2. **Clients**: Individual instances running on user devices, facilitating user interaction with the chat room. Clients connect to the server to send and receive messages, providing a seamless chatting experience.

Here are the steps to create the LAN Chat:

Step 1 - Set Up the Server

Process of the Implementation:

1. Import the modules which are the socket and Threading

```
import socket
import threading
```

   The **socket** module in Python provides a low level networking interface, which allows communication between processes over a network.

   The **threading** module in Python provides a high-level interface for creating and managing threads. Threads make it possible to handle multiple client connections simultaneously.

2. Implement the main Function which is sets the server's port and address.Then calls the ***initialize_server*** function with the specified address and port.

```
def main():
    my_port = 7000
    my_address = "192.168.1.4"
    initialize_server(my_address, my_port)
```

3. Implement the Initialize Server Function

```
def initialize_server(address, port):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((address, port))
    server_socket.listen()
    print(f"Server listening on {address}:{port}")
    clients_list = []
    accept_clients(server_socket, clients_list)
```

It establishes a TCP/IP socket for the server, binding it to a designated address and port for communication. It then listens for incoming client connections, printing a notification upon successfully initiating the server. Then invokes the ***accept_clients*** function to manage the incoming clients.

4. Accept clients to connect to the server

```
def accept_clients(server_socket, clients_list):
    while True:
        client, client_address = server_socket.accept()
        clients_list.append(client)
        client_thread(client, clients_list)
```

The ***accept_clients*** function accepts incoming client connections, continually monitoring for new connections. Upon connection, it adds the client's socket to a list named **clients_list**, effectively tracking all connected clients. Furthermore, it invokes the ***client_thread*** function to initiate communication management with the newly connected client, enabling the server to handle communication tasks concurrently for each client.

5. Create a new thread for each connected client

```
def client_thread(client, clients_list):
    client_threading = threading.Thread(target=thread_listening, args=(client, clients_list))
    client_threading.start()
```

Then Calls the thread_listening function to listen for messages from the client.

6. Listens for messages from the client

```
def thread_listening(client, clients_list):
    while True:
        message = client.recv(1024).decode()
        if message:
            print(f"Message from {client.getpeername()}: {message}")
            broadcast(message, clients_list)
        else:
            print(f"Client {client.getpeername()} disconnected")
            clients_list.remove(client)
            client.close()
```

The ***thread_listening*** function continuously listens for incoming messages from the client within a loop. Upon receiving a message, it decodes it and prints it alongside the client's peer address. Subsequently, the code calls the broadcast function to send the received message to all connected clients. If any client disconnected, it is removed from the **clients_list**, and its socket is closed to manage the disconnection.

7. Send the message to the all connected clients

```
def broadcast(message, clients_list):
    for client in clients_list:
        try:
            client.send(message.encode())
        except Exception as e:
            print(f"Error broadcasting message to {client.getpeername()}: {e}")
            clients_list.remove(client)
            client.close()
```

The function is responsible for Iterating through the **clients_list** and sends the message to each client and it handles exceptions if there is an error while broadcasting.

Step 2: Create the Client Script

Process of the Implementation

1. Imported the same modules as in the server which are Socket and Threading.
2. Asked users to enter their names and ensure there is a name not a blank space.
3. Created socket object which used to establish connections to other machines using TCP/IP over IPv4.
4. Since it is local chat between two different devices, the port was setted as the same as the port in server's script and the address was setted to the server's address which is **"192.168.1.4"**, then the socket object was connected to this port and address.
5. The function ***sending_msg*** was defined to handle sending messages to the server, it is started with a loop, so sending messages is repeated. It takes the message from the user and concatenates it with his name then sends it to the server.

```python
# Function to handle sending messages
def sending_msg():
    while True:
        # Get message from user
        sent_msg = input("Enter your message: ").strip()

        # Check if message is not empty
        if sent_msg:
            # Concatenate name with the message
            msg_name = name + " : " + sent_msg

            # Send message to server
            socket_obj.send(msg_name.encode())
```

6. The function ***recieveing_msg*** was defined to handle received messages with the maximum data could be received which is 1024 bytes, also it decodes the messages as we are dealing with sockets which encoded the message in the first place. Then it prints the message.

```python
# Function to handle receiving messages
def recieve_msg():
    while True:
        # Receive message from server
        recieved_msg = socket_obj.recv(1024).decode()

        # Print received message
        print(recieved_msg)
```

7. Created two threads for the two functions ***sending_msg*** and ***recieveing_msg*** to make sending and receiving messages between multiple clients and to happen simultaneously, ensuring that the program doesn't block while waiting for input or output from the user or the server.
8. In the end, starting the two threads to begin sending and receiving messages between clients.

```python
# Create threads for sending and receiving messages
thread_send = threading.Thread(target=sending_msg)
thread_receive = threading.Thread(target=recieve_msg)

# Start threads
thread_send.start()
thread_receive.start()
```

Step 3: Test the Server and Client Locally

In order to test our LAN chat we first run the server script on one machine within the LAN and run multiple instances of the client script on other machines within the same LAN.
First we opened the command prompt and wrote "***ipconfig***" command to check the ip address of the device which the server will run on it.

```
IPv4 Address. . . . . . . . . . . : 192.168.1.4
Subnet Mask . . . . . . . . . . . : 255.255.255.0
```

After that we checked that the port in the client's script is the same as the port in server's script which is **"7000"** and the address in the client's script is the ip address of the device that the server is running on which is **"192.168.1.4"**

The Server script is saved in **Server.py** file and the Client script is saved in **client.py.**

Start running the server

```
PS C:\Users\Aliaa\Desktop\Lan-chat> python .\Server.py
Server listening on 192.168.1.4:7000
```

We used two Clients the Clients
The First Client : Aliaa

```
PS C:\Users\Public> python .\Clients.py
Choose your name:
```

```
PS C:\Users\Public> python .\Clients.py
Choose your name: Aliaa
Enter your message:
```

The second client : Asmaa

```
Choose your name: Asmaa
Enter your message:
```

Aliaa wrote the message and sent it

```
PS C:\Users\Public\Lan-chat> python .\Clients.py
Choose your name: Aliaa
Enter your message:
What are u doing rn?
```

The message which Aliaa sent it appears to Asmaa

```
Choose your name: Asmaa
Enter your message:
Aliaa : What are u doing rn?
```

And then Asmaa responded

```
Choose your name: Asmaa
Enter your message:
Aliaa : What are u doing rn?
I am about to make cup of tea, do u want?
```

So the message appears to Aliaa

```
PS C:\Users\Public\Lan-chat> python .\Clients.py
Choose your name: Aliaa
Enter your message:
What are u doing rn?
Asmaa : I am about to make cup of tea, do u want?
```

And here is the running Server

```
PS C:\Users\Aliaa\Desktop\Lan-chat> python .\Server.py
Server listening on 192.168.1.4:7000
Message from ('192.168.1.4', 62726): Aliaa : What are u doing rn?
Message from ('192.168.1.4', 62728): Asmaa : I am about to make cup of tea, do u want?
```