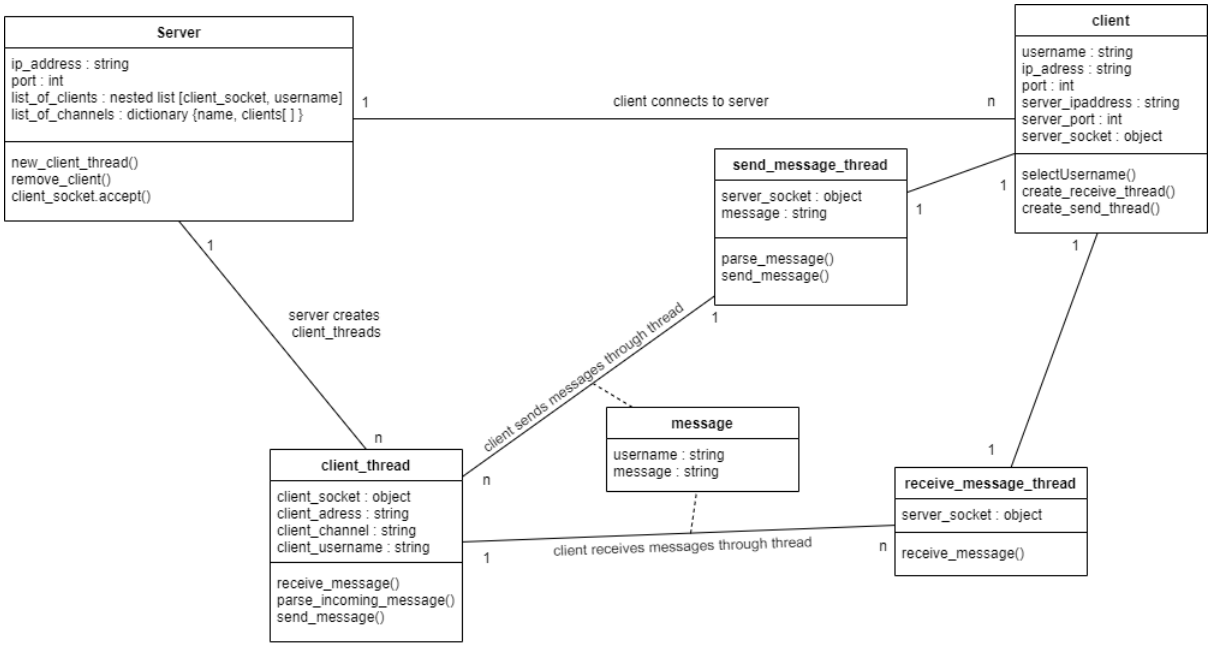
Jesse Pasanen 0545937

Distributed Systems

Assignment 1



Class Diagram

The basic concept of the program is to host a server that manages messages sent from multiple clients. The server is threaded and thus handles each client on it's own thread after establishing a connection with them. The biggest upside to this is that the server can now handle multiple users at once without them having to wait in a queue. This also is completely transparent to the users. The threads share the memory and update the list of clients on their own accord for every other thread as well. The client also has two threads, one for sending messages and one for receiving messages. After some research, I noticed that the 'select' Python library is often used in this kind of situations, but sadly it doesn't work on windows how I wanted it to; And that is the reason of threading the client as well.

For transferring data, TCP was used because it establishes a connection between the client and the server. TCP was also used because it's way more reliable as it has all kinds of features that make sure that the message was received and without errors.

The server stores clients in a nested list with the client socket and the username. Chat channels are stored in a dictionary with their names and clients that are connected to them as a list. The program doesn't have the feature to add new channel through the CLI as it wasn't a requirement. Channels can be added manually through the code. Normally this information would be in a database, but the program does not feature a database and resets every time it is restarted. Reason for this is it not being a requirement.

Each time a new client connects, the server accepts the connection and then creates a new thread for the client. After that the thread handles all incoming and outgoing traffic between the server and the client. When first connecting to the server, the client is asked to give a username. Parsing is done on the client side and the comparison whether the username exists is done on the server side. After selecting an unused username, the client is given instructions on how to continue.

The user can join a group chat (channel) by typing ‘/join [channel name]’. By typing joining, the user can chat with other people that have also joined the channel. The available channels are listed below. The user can also send a private message by typing ‘/msg [username] [message]’. The user can send a private message even if they have not joined a channel. Third option is exiting by typing ‘/exit’. All these commands have error messages that help the user navigate the system. The parsing for these command is done server-side, but in hindsight it definitely should’ve been done client side.

After successfully inserting a username

'Testuser' perspective of sending messages

'Newuser' perspective of sending messages

```
Server started...
Running...
New user connected from 127.0.0.1:61098.
New user connected from 127.0.0.1:61101.
User 'Testuser' (127.0.0.1:61098) connected to channel 'channel1'
User 'Newuser' (127.0.0.1:61101) connected to channel 'channel1'
[channel1][Newuser]: Hello!
[channel1][Testuser]: Hi there!
User 'Newuser' sent a private message to user 'Testuser'
User 'Newuser' (127.0.0.1:61101) disconnected from channel 'channel1'
User 'Newuser' (127.0.0.1:61101) connected to channel 'channel2'
User 'Testuser' (127.0.0.1:61098) disconnected from channel 'channel1'
User 'Testuser' (127.0.0.1:61098) connected to channel 'channel2'
[channel2][Testuser]: Now I'm in channel2!
[channel2][Newuser]: Me too.
User Newuser (127.0.0.1:61101) disconnected.
(127.0.0.1:61101) Connection lost.
User Testuser (127.0.0.1:61098) disconnected.
(127.0.0.1:61098) Connection lost.
```

Server perspective of sending messages

```
[Server]: You're currently not in a chatroom. Use '/join [Channel name]' to join.
/
Invalid command. Available commands:
'/join [Channel name]'
'/msg [username] [message]'
'/exit'

/join
Usage: '/join [Channel name]'
/join notexistchannel
[Server]: Channel not found.
[Server]: You're currently not in a chatroom. Use '/join [Channel name]' to join.
/msg
Usage: '/msg [username] [message]'
/msg Newuser
Usage: '/msg [username] [message]'
/msg Newuser Hello, This is a private message
[Server]: Username 'Newuser' not found.
[Server]: You're currently not in a chatroom. Use '/join [Channel name]' to join.
/msg Testuser Hello me!
[Server]: You can't sent a private message to yourself.
[Server]: You're currently not in a chatroom. Use '/join [Channel name]' to join.
/exit 1
Command '/exit' doesn't take any arguments.
/exit
Disconnected.
```

Error messages to guide the user

All requirements are met. Given requirements below.

Client Requirments

- set nickname
- connect to the server by IP address
- send text messages to other connected clients
- Chat should support several channels
- Chat should support private messages
- Client should show messages from other clients
- Client should be able to disconnect from the server

Server Requirments

- Handle several clients connections
- Transmit messages between clients

How to use:

1. Start server with ‘python server.py’
2. Open one or more terminals to start clients
3. Start client with ‘python client.py [server ip address] [server port]’

The server IP address in ‘127.0.0.1’ and the port is ‘1234’.
The program should have clear instructions to move onward from there.

Source code available:

https://github.com/Jezerig/Distributed-Systems-Assignment-1-TCP-Chat