Programming Assignment 2 - TCPChat

Implementation details | Information sources | Program execution | Examination criteria

Purpose with the assignment

We want you to continue to have a look at the fundamentals of distributed systems.

- Use basic communication methods to better understand.
- Practice the use of terminology concerning failure model.
- Understand the benefits and drawbacks with streaming.
- Understand the benefits and drawbacks with fixed interfaces.
- Understand JSON

This assignment touches on topics from lectures: Introduction to distributed systems, characterization of distributed systems, system models, interfaces, invocation models.

Description of the assignment

In this assignment, you will improve your simple chat server from assignment 1. The clients and server communicate TCP streams. In other words, seen from the perspective of the OSI session and transport layers the architecture is connectionless. The chat server should be able to handle an arbitrary number of clients.

Requirements on the implementation

- A list of participants should be available.
- Crashed clients should be marked "disconnected" and be able to re-join.
- It should be possible to join and leave the chat server.
- Everyone in the chat room should be alerted when someone enters/leaves the chat room.
- You must use the designated ChatMessage Interface.
- You must implement the '/help' command.
- You must define, describe and implement one extra command.
- You should provide a desktop client.
- You must use JSON to communicate between the client(s) and the server.
- You need to use a threaded server.

Requirements on the report:

- A section should discuss which failure model you have chosen and why. Are there faults that you cannot handle? Why?
- You should discuss the difference between UDP and TCP communication. What is better using TCP? What is better using UDP? Why?

- You also need to describe how your solution can be started (which commands, which ports are needed)?

Implementation details

This section contains an outline of the theoretical background necessary to complete the exercise. For further details, refer to the course literature and the numerous sources on the Internet.

TCP Stream communication

This assignment requires you to implement communication using TCP streams. You must only transmit messages of the type ChatMessages, which is provided to you.

Server requests

A client should be able to send 7 different types of requests to the server, help, join, leave, list, tell, broadcast, and YOUR_UNIQUE_COMMAND. The requests are listed below:

Help message

A help message should return a string with the available commands on the server.

```
/help
```

On your screen, this should be displayed as

```
help join leave broadcast tell list qotd
```

Using the help command along with a parameter should return a description on how to use that particular command.

```
/help qotd
```

On your screen, this should be displayed as

```
/qotd - returns the quote of the day
```

Connection request

A connection request should be sent to the server during client initialization. It effectively acts as a “subscription request”, announcing the client’s presence to the server so that the server can send chat messages to the client as needed. A connection request should contain the name of the user making the request. The server either accepts or denies the request (e.g., a request could be denied if the user name was already taken, since the communication protocol will likely depends on unique user names) and sends an appropriate response message to the client.

Broadcast message

A broadcast message should be distributed to all clients, preceded with the name of the user sending the message. If user Alfred broadcasts message “Hello world!”, all other clients should see the following line (or similar) in their chat window:

```
Alfred: Hello world!
```

Private message

A private message should be sent to a single user. The request must therefore contain the name of the user to whom the message should be sent. If Alfred wants to send a private message to Bruce, he types

```
/tell Bruce, DotA tonight?
```

On Bruce’s screen, this should be displayed as

```
<Alfred tells you> : DotA tonight?
```

Disconnect

There should be a way to leave the chat in a controlled way.

```
/leave Bye folks!
```

List

There should be a way to list participants in the chat.

```
/list
```

Marshalling and unmarshalling

You need to use the serializable class ChatMessage to communicate.

Handling multiple clients

The server must keep track of all connected clients; their names, IP addresses and port numbers. The code framework (see next paragraph) contains code for maintaining a collection of users. The `ClientConnection` class is used to store user details, and the `Server` object has a `Vector m_connectedClients` of `ClientConnection` objects that holds all connected clients. The Server member methods `addClient()`, `sendPrivateMessage()` and `broadcast()` have been completed in the code framework and can be used to maintain the connected users and send messages to them (although you need to supply the actual message sending code in the

`sendMessage()` method in the `ClientConnection` class). The framework does not have any code for disconnecting clients; you have to implement this by yourselves.

Existing code framework

The JsonTest project is the only new code you will get. You are encouraged to continue using your UDPChat-code.

Information sources

You are encouraged to search for information by yourselves. However, the sources listed below are good starting points.

Program execution

To run the programs use Eclipse "run configuration..." or, open two command shells. In the first shell, enter the directory with your implementation files and type:

Examination criteria

Implementation

The implementation should correctly handle all three types of server requests. It should be possible to connect a reasonable number of clients

Documentation

Each student turns in a very short (max 1-2 pages) report (in PDF format) describing their solution. The report should contain

- A short overview of the solution, detailing parts that you think are important including the theoretical parts on failure models and invocation solutions.
- A reflection of what you have learned in the exercise and feedback on the exercise itself.

You may use either Swedish or English for your report and code. Grammar, good document layout and spell checking are required – badly written reports will be discarded without comments. The same goes for source code with sloppy indentation or commenting.

Releasing (submitting) your code and documentation

Hand in using SCIO / Assignments.

1. Attach your PDF-report named YOUR_LOGIN_tcp_report.pdf
2. Attach your server-JAR named YOUR_LOGIN_tcp_server.jar
3. Attach your desktop-client-JAR named YOUR_LOGIN_tcp_client.jar
4. Attach Your source files named YOUR_LOGIN_tcp_source.zip

Happy coding!

This assignment was created by Sanny Syberfelt and further developed by Marcus Brohede, Petter Henriksson and András Márki.