Distributed Systems

Lab 2

Horia Radu - k55592hr

Task 2.4: Protocol Design

COMMANDS LIST: (replace what is in the parentheses with your input)

REGISTER (screenName)
DISCONNECT
MESSAGE (message)
PRIVATE (screenName of person you want to msg) (message)
LIST

*when using the client, you need to enter a screenName when launching it
*if it is taken, you will be told and will need to REGISTER another one

1. - The registration of the user will be done with a command “REGISTER (screenName)” followed by the screenName the user will want to use.
   - To be noted is that, if the user did not register, thus not having a screenName, they will not be allowed to use any commands aside from “REGISTER” and just receiving public messages.
   - In case the user will enter an already registered screenName, they will be greeted by an error, telling them that the screenName is already in use.
   - For the register command,to be sure no unregistered user can use commands, I will check for the register command first, and if the user is not registered, he won't be allowed to do anything.
   - Pseudocode:

```
if command is “REGISTER”:
        iterate through all the current connections
                check if the screenName is already in use
        if it is not in use:
                register it for the current user
else if command is “DISCONNECT”:
        disconnect user  // Make sure the user can disconnect
                                // even if he is not registered
else if current screenName is empty:
        throw ERROR that user doesn’t have a screenName
else if command is ….. // Continue with the other commands
```

2. User requests will be implemented as written commands, they will be put right after the REGISTER and DISCONNECT commands, and the check to see if the user is registered
   a. For messaging everyone use the command:
      “MESSAGE (your message here)”

```
      else if command is “MESSAGE”:
              iterate through all the connections:
```

send the message

b. For a private message use:
“PRIVATE (user you want to send a message to) (message)”
In case the user does not exist, the client will be given a prompt acknowledging him about that.
else if command is “PRIVATE”:
iterate through all the connections:
Check to see if the receiver screenName exists
If receiver screenName found:
send message to receiver
send confirmation to sender
If not receiver was found:
throw ERROR that user was not found

c. For seeing a list of connected users use:
“LIST”
else if command is “LIST”:
iterate through all the connections:
add all the connection names to a list
send the list of the active screenNames to the sender
+ a number of all current client connections

3. If the command entered is not valid or the user does not exist, a reply from the server will be sent, which lets the client know the problem.
- If the command is invalid or written wrongly, the reply will be:
“ERROR UNKNOWN_COMMAND: (your command + message)”
With all the commands being included in if’s, this will be added at the end, if no commands were recognised.
if command:
do command
else if command:
do command
etc...
else:
give UNKNOWN_COMMAND ERROR

- If the screen name does not exist when trying to private message, the reply will be:
“ERROR USER_NOT_FOUND: (screen name)”
if no client found with the screen name in all connections:
throw USER_NOT_FOUND ERROR

- If the screenName you are trying to use already exists, the reply will be:
"ERROR: Already registered screenName (screen name) "
if screenName (user is trying to enter) is in connections.screenNames:
throw Already registered screenName ERROR

- If you are not registered when trying to use commands:
    "ERROR: Not registered screenName"
    if current screenName is empty:
        throw Not registered screenName ERROR

4. For the close request, there will be a command: “DISCONNECT”
   This will disconnect the user and remove his screen name from the server.
   if command is “DISCONNECT”:
       remove client from the current clients
       remove client connection from the server and his name
       send acknowledgement to the client that he successfully disconnected
       send acknowledgement to the server that the client disconnected
       close user’s connection

Task 2.7: Critique

1. Is the protocol you created for this exercise stateless? Explain why or why not, or explain why this concept may not be applicable in the context of this application.
    a. I consider the protocol for this exercise to be stateful.
       I find it similar to a Simple Mail Transport Protocol (SMTP), which is a stateful protocol.
       This protocol I’ve just made is connection based, with users being able to send multiple commands/requests, and disconnect at any point.
       In the context of this application, one of the main points that make this a stateful protocol is that users can be in two different states; either registered, thus being allowed to use any command as they please, or unregistered, which blocks a number of commands in the application.
       Also, the sockets make this protocol stateful, as they hold various states for the server and the server response to the users.

2. Describe how the current implementation could be extended/improved to allow subgroups of users to be created and messages to be sent only to members of a subgroup.
    a. This feature could surely be added to the current implementation without changing its structure.
       For this we need to first add a command to create a group, and include the creator in it (“CREATE_GROUP *group name*”), which will have as attribute a group name, here we can also add a command to delete a group, but this would take more work and require adding different permissions.
       We will need to be able to store all the groups and its participants in the server. For this we could use a dictionary.
       The group will be added to an empty dictionary in the server, which will hold all the group names with their specific participants.
       After that we can create a command to list all groups (“LIST_GROUPS *group name*”) with their names, just like the list for the current users.
       Then another command for joining a group (“JOIN_GROUP *group

name*”) with an attribute for the group name, which will add the registered user to the corresponding group dictionary.
(here we will need to check that the user isn't already in that group)
Here it would be good to also add a command for exiting the group (“LEAVE_GROUP *group name*”).
When a user disconnects, we can either kick him from the group, or keep them in the group, and next time he connects to the client, he will need to use the same screenName.
Finally, the group messaging would work just like the private message command, but in a loop, so that we send the message to the whole group. It would go in the group dictionary and send the messages, iterating through the whole dictionary (“MESSAGE_GROUP *group name* *message*”).