Lab 2 - Report

Iva Jorgusheska, UID: 11114620

March 13, 2024

1 Description of the protocol

In this report, I give an explanation of the design of my protocol for the messaging system for lab 2, and the key ideas taken into consideration.

1.1 Starting the server - OnStart method

I will use the onStart method which is called when the server starts to initialize global variables that will later be accessed and updated by all the users that connect to the server. The variable 'nrUsersConnected' will be initialized to 0 when the server starts. Later, on connect and on disconnect for each user it will be increased, or decreased by 1 accordingly.
In the onStart method I will initialize a list of all the sockets of the users connected. When a user connects, its socket will be added. Also, equally important is to remove the socket from the list when a user disconnects. In this way, we will always have an up-to-date list of connected users and their sockets which can be used to easily send messages to all the users. When a user connects or disconnects, as requested, an informing message will be sent to all the users connected using their sockets in the socket list. Moreover, I will initialize a dictionary in which the key will be the socket, and values will be the screen name of the user. It will keep track of all the names of the users. It will also be updated regularly when a user connects or disconnects.

1.2 onConnect method

I will use the onConnect method to set the variables needed for each user connecting to the server. I set the first message flag to true when the user is connected, which indicates the user needs to register. After the first message flag is set to false, the user can proceed with the other commands. Moreover, in the onConnect method I increase the number of users connected to the server which is a variable initialized to 0 when the server starts running. As requested I will also send a message to all the users connected using the sockets list that a new user has joined the server using the sockets list.
To make the server more user-friendly, I will send a message to the user connecting what their first message should look like, i.e. how to register. I can send a message to the user by using the socket which is provided as a parameter to the function.

1.3 onMesage method

In the onMessage method, we need to make sure the first thing the user does is register to the server. Hence, I will have a flag that is set to firstMessage = True and only after correct first message is processed will be changed to False. I will check that the first command is $register. If it is not, an appropriate message will be displayed that the user must register first. When the user registers, he provides his desired screen name as a parameter, "$register name". If the name entered by the user is already present as a value in the dictionary of users, he will get warned that the name is occupied, and he needs to register with another name. To be sure that the user is given the chance to register until he provides a valid name, the flag that we are not on the first message anymore will be removed only when the user enters a valid name and successfully logs in. The key in the mentioned 'users' dictionary will be the socket the user is connected to and the value will be the valid screen name he used to register. I believe this design idea will help me get the socket of the recipient of the message, or message all users easily, and do all kinds of operations by just knowing the name of the user, or

their socket.
After the first message flag is removed, I will check the messages are commands if they begin with $, since I defined commands to begin with $ in the previous part of the exercise where we had to distinguish between ordinary messages and commands. I will have the following commands which will be enough to provide a full functionality for the server: $quit, $messageUser, $messageAll, $listAllUsers, and $listAllCommands. If the message starts with $, but is none of the commands available then a message that it is an invalid command will be sent to the user. Otherwise, I check and match with the commands and their requirements. $messageUser must be followed by a name of a user connected to the server, you can message yourself. Then followed by a message as long as the user wants. If the 'messageUser' function is not followed by enough parameters, an appropriate message explaining how the command should be used will be sent to the user. To send the message to the specific user, I will find its socket from the key-value dictionary. I will use the name provided to find the key-value pair and then get the key - the socket for the provided value - name. The recipient will get the name of the user sending the message as well as text that the message was only sent to him. The name of the sender is easy to get, and can be saved in a variable at the beginning when the user registers.
The $messageAll function only takes the message as a parameter. When I send a message to all the users, I will send it to all the sockets connected saved in the sockets list. I just check it is not the user sending the message and skip that one. The users will receive the name of the sender and the information that he sent the message to all the users. For both sending an individual message or to all users, the sender will get a confirmation message.
The $listAllUsers command does not require any parameters so we will check that the command contains only one word. Then using the 'users' dictionary, I can easily get the names of the users which will be the values in the dictionary. I will check if the dictionary is empty just in case, but it cannot be since at least the user who is issuing the command will be connected to the server.
The $listAllCommands method will send all the available commands and their explanations as a message to the user who issued the command.
Regardless of being the first message or not, the user can write the command $quit to disconnect from the server. In that case, I will issue the command "socket.close()" which will automatically call the onDisconnect method. Additionally, a confirming message that the user has disconnected will be provided.

1.4 onDisconnect method

In the onDisconnect method, I will remove the socket of the user that is disconnecting from the list of sockets, and send a message to all the users to inform them about the new number of connected users. I also remove the key-value pair from the 'users' dictionary. In this way when we search for a name to check if we can send a message to it, we will always have an updated list and will not be able to send messages to disconnected users.

The protocol is efficient, provides all the desired functionality, and yet follows a simple logic where the user dictionary and the sockets list provide the crucial information needed.

2 Pseudocode of the protocol

2.1 Starting the server - OnStart method

function OnStart
    $usersConnected \leftarrow 0$
    Initialize *socketsList* ▷ List of sockets of connected users
    Initialize *usersDictionary* ▷ Dictionary to map sockets to screen names
end function

2.2 onConnect method

```
function onConnect(userSocket)
    Set firstMessage ← True
    usersConnected ← usersConnected + 1
    Add userSocket to socketsList
    Broadcast "New user connected" to all users
    Send "Please register your screen name using $register" to userSocket
end function
```

2.3 onMessage method

```
function onMessage(userSocket, message)
    if firstMessage then
        if first command is $register then
            Extract username from message
            if username is not already taken then
                Set firstMessage to False
                Add user to users dictionary with userSocket and username
                Send "You have registered successfully." to userSocket
            else
                Send "The name is already occupied." to userSocket
            end if
        else
            Send "Please register first" to userSocket
        end if
    else
        if Message starts with $ then
            ProcessCommand(userSocket, message)                ▷ Process command
        else
            Send "Invalid command" to userSocket
        end if
    end if
end function
function ProcessCommand(userSocket, message)
    Extract command from message
    if command is $quit then
        Send a disconnect message to the user
        Close the userSocket
        Return False indicating disconnection
    else if command is $messageAll then
        Extract message content
        Send the message to all users except the sender
        Send a confirmation message to the sender
    else if command is $messageUser then
        Extract recipient username and message content
        if recipient exists then
            Send the message to the recipient
            Send a confirmation message to the sender
        else
            Send "Recipient is not connected" to the sender
        end if
    else if command is $listAllUsers then
        Construct a string containing all usernames
        if there are no users then
            Send "No users are connected" to the sender
        else
```

Send the list of users to the sender
end if
else if command is $listAllCommands **then**
Construct a message containing all available commands
Send the message to the sender
else
Send "Invalid command" to the sender
end if
end function

2.4 onDisconnect method

function onDisconnect(userSocket)
Remove *userSocket* from *socketsList*
Broadcast "User disconnected" to all users
Remove entry for *userSocket* from *usersDictionary*
end function