Distributed Systems

Lab Exercise 2
COMP28112

Pavel Ghazaryan

University of Manchester
March 23 2023

Created Using LaTeX

Instructions to run the code

Open a terminal(command prompt) window and run the server:

python3 ./myserver.py localhost 8090

Open another terminal and run the client:

python3 ./myclient.py localhost 8090

To send a message press enter or press send button. You can open as many clients as You wish. Make sure ex2utils.py file is in the same folder.

Task 2.1: Your own basic server. **Your server code must be able to respond when a client connects, when a message is received and when a client disconnects by displaying a suitable message on the screen.**

In myserver.py:

```
class MyServer(Server):

    def onStart(self):
        print("Server running...")

    def onStop(self):
        print("\nServer ended")

    def onConnect(self, socket):
        self.numOfUsers += 1
        print("User connected, Current number of users: ",
            {self.numOfUsers})

    def onMessage(self, socket, message):
        print("Message received: ", message)
        message = message.upper().encode()
        socket.send(message)
        return True

    def onDisconnect(self, socket):
        self.users.pop(list(self.users.keys())[list(self.users.values()).index(socket)])
        self.numOfUsers -= 1
        print("Disconnected a user")
```

Task 2.2: Counting clients.

Your server code must be able to remember how many clients are connected at any one time, displaying a message on screen with this value every time a client connects or disconnects.

In myserver.py:

```
class MyServer(Server):

    def __init__(self):
        super(MyServer, self).__init__()
        self.users = {}
        self.numOfUsers = 0

    def onConnect(self, socket):
        self.numOfUsers += 1
        print("User connected, Current number of users: ",
            {self.numOfUsers})

    def onDisconnect(self, socket):
        self.users.pop(list(self.users.keys())[list(self.users.values()).index(socket)])
        self.numOfUsers -= 1
        print("Disconnected a user")
        print("\nCurrent number of users: ", self.numOfUsers)
```

Task 2.3: Accepting and parsing commands.

Your server code should be able to accept messages sent from the client in some sensible format, enabling interpretation of messages as they are sent to the server and displaying the commands and their parameters on screen.

```
    def onMessage(self, socket, message):
        (comm, sep, parameter) = message.strip().partition(' ')
        print("Command: ", comm)
        print("Message: ", parameter)
```

Task 2.4: Designing the protocol.

You should provide a description of your protocol design. It should include a list of the commands, what they do and the rationale behind each design decision. You should also write a pseudo-code that describes how your server will handle the various commands, in what order, etc.

I have defined a set of specific instructions that users must follow in order to use the messaging system. To send messages, users need to first register by using the command $REGISTER < username >$. The username they choose must be unique and cannot contain spaces. If the chosen username already exists, the user will be notified and asked to try again with a different username. Non-registered users are limited to using only the $REGISTER, HELP$, and $CLOSE$ commands, as they cannot send messages to any other online users.

If an unregistered user tries to type something, they will receive a notification that they need to register in order to carry out any actions. The "HELP" command allows new users to view the available command terms and their functionalities. Once registered, users have access to all commands, including $MESSAGE$, $DM < user >< message >$, and $LIST$.

The $MESSAGE$ command allows the user to send a message to all other connected users, while the $LIST$ command shows all connected users who are registered. Non-registered users cannot receive or send messages, and thus have no username to show. The DM command allows the user to send a message to a specific user by entering the username as the first parameter and the message as the second. If the entered username does not exist, the user will be notified that they have entered an incorrect username and asked to try again.

If a user enters a command and its parameters incorrectly, they will receive an error message instructing them to try again.

The server code should be organized in the following manner:

```
DEF onMessage(self, socket, message):
    Split the sent message into variables: Command and Parameters.

    CASE Command{

        "HELP":
            Output the list of all commands and their required parameters
            BREAK

        "CLOSE":
            Close the connection between socket and server
            BREAK

        "LIST":
            Sends user a list of all connected users
            BREAK

        "REGISTER":
            Add username and socket to a dictionary if username is
            valid & not already in the dictionary
            BREAK

        "MESSAGE":
            IF username is set
                Send message if there is more than 1 user in dictionary
                and parameters is not an empty string (meaning there is
                someone to send messages to). Else, say appropriate error
                message
            ELSE
                Tell user to register
            BREAK
```

```
"DM":
    IF username is set
        Split Parameters in to sendUser and Message.
        Send message to person if sendUser exists, is not
        the socket, and message is not an empty string
    ELSE
        Tell user to register
    BREAK

"":
    BREAK

DEFAULT:
    Tell socket that the command is not valid
}
```

Task 2.5: Implement your protocol in the server

Please see *myserver.py* file for implementation.

Task 2.6: Writing the client

For making the program user friendly I have implemented a gui using python built-in tkinter module. Please see *myclient.py* for implementation.