Design Document

Design

As far as designing aspect goes, we kept it simple. We followed the same design as the one in the demo except for a few tweaks here and there. For the database we went with a 2D array. To make it easier on we, made each user take an indexing of 3. That way we would be able to fit the name, Ip address and port. We also used if statements inside the for loop for the client and server. We tried going with python at first and looked at the book's code. Once we figured out that we wouldn't have a lot of trouble handling the memory, we switched back to C and modified her code!

Description of message

For active users, only 3 would be allowed online at once. Anymore and there would be a FAILURE message. Once the users registered, they would be able to interact with each other if they were in the same contact list. Users also had the freedom to pass the following commands to the server:

1. **register <contact-name> <IP-address> <port> :** This is a command a user could pass to the server to register their info like IP address and port. If the total number of users is less than 3, the return message would output SUCCESS. If there are already 3 users, the return message would output FAILURE.
2. **create <contact-list-name> :** The user can pass this command to the server to create a list that others would be able to join. A SUCCESS message will appear if they are able to create a list and FAILURE if they are unable .
3. **query-lists:** This is a command the user can input to check what lists are already created. It will display the name and the users to the user. Should the contact fail to be removed, a return code of FAILURE will appear.
4. **join <contact-list-name> <contact-name>:** When this command is passed, the server will look up the name as well as port and IP address. If it doesn't find anything, then a SUCCESS message will appear. If not than a FAILURE message will appear.
5. **leave <contact-list-name> <contact-name> :** This command will remove the named contact list and its associated information from the named contact list.
6. **exit <contact-name> :** This command lets the user know that they want to exit the IM and a success message. The user contact name will be removed as a well as from the contact list. If everything goes smooth, a return message of SUCCESS will be outputted if not then FAILURE would be outputted.
7. **save <file-name>:** This command will save all the info to a text file called *file-name.txt*
8. **im-start <contact-list-name> <contact-name>:** This command in short is to let the server know that you would like to start a message with everyone in the list. The only

way this command gets a FAILURE is if the contact name isn't in the list. Otherwise, it is SUCCESSFUL

9. **im-complete <contact-list-name> <contact-name>:** This command tells the server that the messaging has been completed. This command should pair with a corresponding im-start by the same named contact. If that is the case, then SUCCESS is returned. Else it is FAILURE that is returned.

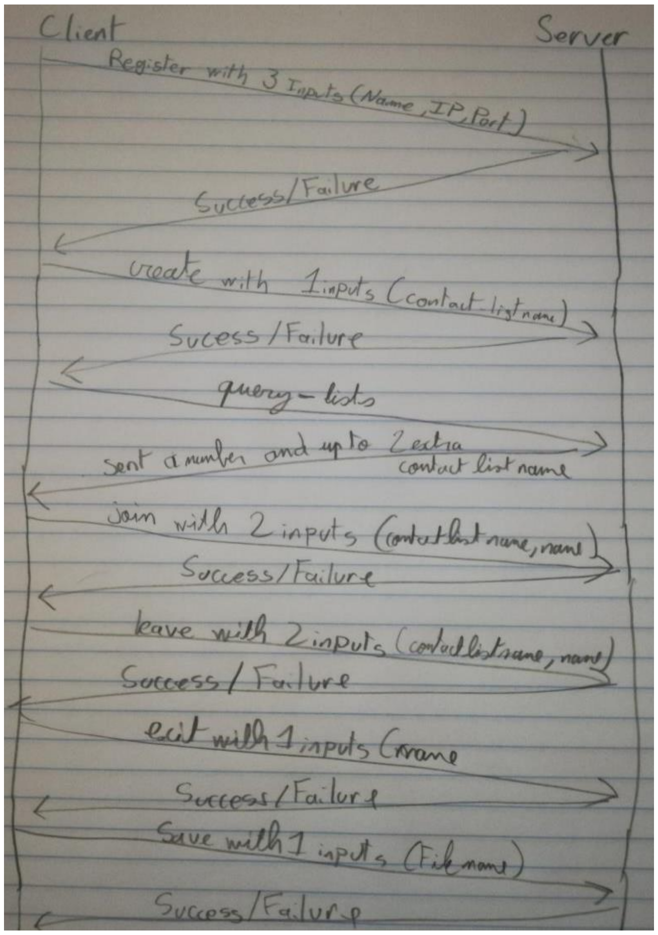
Milestone Testing

We tested out commands with hard code. Basically, we found a case where it would have to return FAILURE and another case where it would return SUCCESS. We then applied it to every command and ensured that it was working correctly!

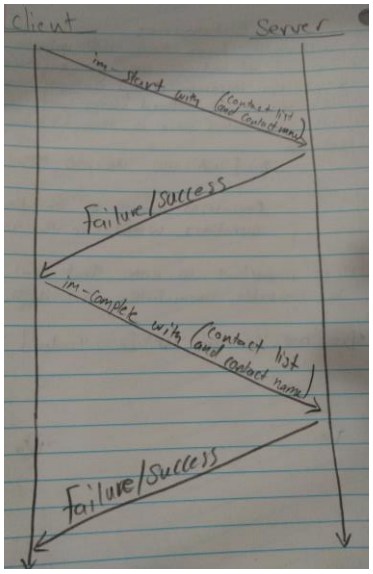
Full Project Testing:

There was no need to retest the Milestone progress since it was already doing what it was supposed to do. But to test the full thing, we created a state (a state variable) to lock the different servers & clients when the im-start command is initiated. This is would run in a loop until the im-complete command with the appropriate arguments where passed. The server would return SUCCESS if no errors happened and everything ran smoothly. FAILURE would be returned if either an incorrect command and arguments were passed or the loop stopped without the im-complete command. After both halves were complete(Milestone and Full Project Testing) and tested separately, we combined them into a one testing-for-all testing method and ran it on our messaging app

Time space Diagram


Client
Server
Register with 3 Inputs (Name, IP, Port)
Success/Failure
create with 1 inputs (contact list name)
Sucess/Failure
query-lists
sent a number and up to 2 extra contact list name
join with 2 inputs (contact list name, name)
Success/Failure
leave with 2 inputs (contact list name, name)
Success/Failure
exit with 1 inputs (name
Success/Failure
Save with 1 inputs (File name)
Success/Failure

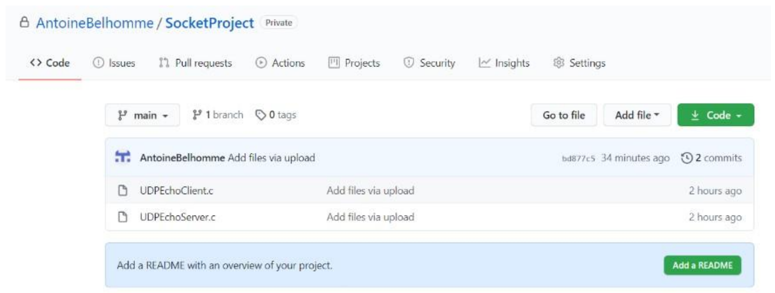
Full Project Time Space Diagram


client
Server
im-start with (contact list and contact name)
Failure/success
im-complete with (contact list and contact name)
Failure/Success

Links to GitHub (It's private per request of professor)

Part 1:

https://github.com/AntoineBelhomme/SocketProject



AntoineBelhomme / SocketProject Private

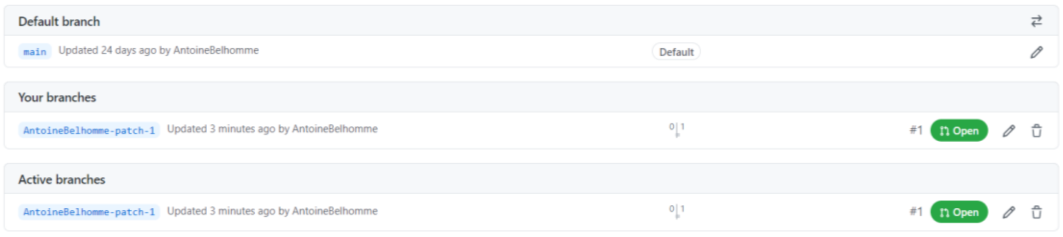
AntoineBelhomme Add files via upload bd877c5 34 minutes ago 2 commits

UDPEchoClient.c	Add files via upload	2 hours ago
UDPEchoServer.c	Add files via upload	2 hours ago

Add a README with an overview of your project.

Part 2:

https://github.com/AntoineBelhomme/SocketProject



Default branch

main Updated 24 days ago by AntoineBelhomme Default

Your branches

AntoineBelhomme-patch-1 Updated 3 minutes ago by AntoineBelhomme 0 1 #1 Open

Active branches

AntoineBelhomme-patch-1 Updated 3 minutes ago by AntoineBelhomme 0 1 #1 Open

Links to demo

First demo:

https://www.youtube.com/watch?v=OA5Wl_Eo_7M

Second demo:

https://youtu.be/_RlOdSOQ4A4