CSE 489/589
Programming Assignment 1 Stage 1 Report
Text Chat Application

Notes: (IMPORTANT)

- ➔ Your submission will **NOT** be graded without submitting this report. It is required to use this report template. Do not add sections for Stage 2, as they will not be graded.
- ➔ One of your group members select <File> - <Make a copy> to make a copy of this report for your group, and share that Google Doc copy with your teammates so that they can also edit it.
- ➔ Report your work in each section (optional). **Describe** the method you used, the obstacles you met, how you solved them, and the results. You can take screenshots at key points. There are NO hard requirements for your description.
- ➔ For a certain command/event, if you successfully implemented it, **attach the screenshot of the result from the grader (required).** You will get full points if it can pass the corresponding test case of the automated grader.
- ➔ For a certain command/event, if you tried but failed to implement it, properly describe your work. If you can get some points (not full) from the grader, also **attach the screenshot of the result from the grader (required).** We will partially grade it based on the work you did.
- ➔ Do **NOT** claim anything you didn't implement. If you didn't try on a certain command or event, leave that section blank. **We will run your code**, and if it does not match the work you claimed largely, you and your group won't get any partial grade score for this WHOLE assignment.
- ➔ This report has 5 bonus points. Bonus points grading will be based on your **description** in each section, including the sections where you got full points from the grader.
- ➔ Maximum score for Stage 1: 42 + 5 = 47

1. Group and Contributions

- Name of member 1: Brecken McGeough
  - UBITName: breckenm
  - Contributions: All
-

2. SHELL Functionality

[5.0] Application Startup (startup)

Grader screenshot:

Description:
How to start- in my pa1 directory in stones.cse.buffalo.edu, run make clean then make, and then run ./assignment1 s 4322. In any other .cse.buffalo.edu, run ./assignment1 c <port>. My server works like this, it creates a socket, binds that socket to the port, and starts to listen to incoming connections. When a successful connection is made, it inserts the new connection information (their listening port, external ip, and external hostname, along with lots of other info) into a linked list of all currently connected clients. Can interface directly with the server shell and depending on the commands will display different things to the server console, or send information to a client (who requests it). The client has its own interface where it can print out different things depending on the command inputted. When 'LOGIN <server_external_ip> <server_listening_port>' is inputted into the client console (assuming its running), will create a connection with the server and immediately send info about itself to identify. Client and server can converse with eachother by sending and receiving messages on their listening ports. Accepts shell commands by comparing to see if inputted command is either AUTHOR\n, IP\n, PORT\n, etc.. and does what each command is supposed to do. If input doesnt match any command or command isnt spelled properly, uppercase, etc, then prints error.

3. Test results

[0.0] AUTHOR (author)

Your submission will not be graded if the AUTHOR command fails to work.
Grader screenshot:

Description:
If the command 'AUTHOR' is typed into the console (server or client), it returns the statement "I, breckenm, have read and understood the course academic integrity policy". It just prints out to the console.

[5.0] IP (ip)

Grader screenshot:

Description:
If the command 'IP' is types into the console (server or client), it returns the external ip address of the machine (server or client) you ran it on. It gets the external ip address by creating a dummy udp socket to Google's public DNS server and then getting the ip address with the inet_ntop() function. Then it prints out the ip.

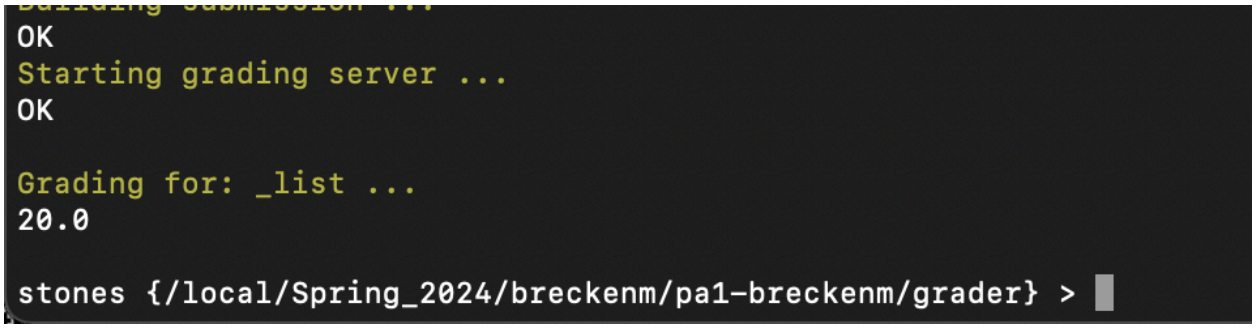
[2.5] PORT (port)

Grader screenshot:

Description:
Prints out the port the machine is listening on to the console(server or client). 4322 for server or whatever port the user typed in when running the client script. Opted not to bind input port to client socket for this part and just print it out, but will be sure to bind client port to socket for stage2.

[20.0] LIST (_list)

Grader screenshot:



Description:
If the server types LIST command in, it will iterate through the linked list and put all of the currently online clients (clients have int in their struct: 0 for offline, 1 for online) ports into an int

array. Then, the array of online client ports is sorted from least to greatest using the built in qsort function. Then, the sorted ports array is iterated through and for each iteration it iterates through the linked list checking to see if the port matches a clients port. If a match is found, the format string specified in the handout containing the id (add 1 every time a client is matched to a port), ip address, hostname, and port all stored in that clients struct is concatenated to a string stored outside both loops and ended with a newline character, so when string is printed out, it will do newline for each client to abide by format. It does this until it has collected all the info from every client with matching ports in the sorted ports list. What's left is a string of all properly formatted online client information separated by newline characters and in order by their port number least to greatest. Then this is printed to the console. If the client types in LIST, displays the global string 'list' to the clients console. When a client logs into the server, it immediately calls LIST on the server side and does everything described above except instead of printing it to the servers console, the server sends the string to the client where the client copies the string into the global list string, so LIST is a snapshot of the clients logged into the server at the time the client connected (until they call REFRESH)

As soon as client types LOGIN into the client console, it will create a socket and try to connect to the given server ip and using the specified server port. If there is a successful connection, it will send a string to the server containing both the clients external ip address and external hostname. When the server receives this information, it will break up the string into the ip and hostname and will create a new client structure with the information (client external ip, hostname, port, file descriptor, etc…) and insert into the linked list that stores all the connected clients, initializing and filling in all other fields with initial values (new clients inserted on top of linked list like stack to reduce time complexity). Then it will send back a message letting the client know it was successfully added. If client with matching ip and hostname and port and file descriptor already exists in the clients linked list, it will not create a new node and just set their online value to 1 in their client struct to show that they are online, if they are new then it will create their client struct with their info and insert them into the linked list as described above. Then, after successful connection client sends 'LIST' to server, and when server gets 'LIST' command it will send string of all clients including all necessary info and formatted properly abiding by the format for LIST (refer to LIST above for more details). Client receives this string and stores it in global list string which will be printed out when the client calls LIST. LOGIN does not print the online client list, it just gets the list and stores it for future LIST calls.

[2.0] LOGIN Exception Handling (exception_login)

Grader screenshot:

Description:
I couldn't get full credit for login exception handling.

[5.0] REFRESH (refresh)

Grader screenshot:

Description:
Sends 'LIST' to server and updates global list variable with string of clients sent back from server in same way specified above in LOGIN, then displays it. For info on how LIST command sent to server gets list of current online clients, see LIST description.

[2.5] EXIT (exit)

Grader screenshot:

Description:
The client will send the server its external ip, hostname, port and a message saying its exiting. The server will then iterate through the linked list storing the clients and when it finds the node with the ip and hostname and port matching that of the recently exited client sent it will set their online status to 0 so they will no longer show up in LIST and it will remove their file descriptor from the watchlist. Will set their online status to 0 and remove fd from watchlist if they quit the terminal without exiting as well. Then, client will exit with status 0.