Computer Networks Project Report

## Samuel Armstrong (sma180007), Risvee Ayub (rna180002), Vishak Bandi (vxb180017)

### Protocol Design:

1. Message format for sending and receiving math calculations:
   1. The format of this is fairly straightforward. The Server sends a message to the client asking it to input a math problem to be solved and the math problem must be in the following format: Only +, -, /, and \* are the allowed operators, and parentheses are allowed, all numbers, parentheses, and operators must be separated by spaces, and the input must be valid mathematical expression. The server will then dispense the answer to the problem back to the client.
2. Message format for joining and terminating connection:
   1. Joining: The Server is opened, opening a socket at a port that both client and server know. The client then runs, using the prior opened socket to connect to the server, where the client must provide their name where prompted, being in the format of a string. Then the server asks for a math problem to solve and we go to (1) above.
   2. Terminating: The client will terminate its connection with the server by typing the message stop instead of a math problem, and the client will stop running.
3. Format for keeping logs of clients’ activities at server side:
   1. The log of client activities on the server side is kept in a text file generated when the server is started, “log.txt”. If the server is started, runs without error, and is stopped properly, the log will begin with the exact time and the message “Server Started.”, and will end with the exact time and the message “Server Stopped.”. The exact time is obtained with Java’s LocalDateTime.now() method, which gives the exact date and time in the format YYYY-MM-DDTHH:MM:SS.sss (The T is an actual T, not a measure of time).The Server logs the time, name of the user, and the message “connected” every time a new connection with a client is established. Every time a client sends a problem to the server to be solved, it is logged, with the exact time, the name of the client, and the contents of the problem all being put into the log. Once this problem is solved and the answer sent to the client, the Server logs the exact time, the name of the client, and the answer to their problem. If the server experiences an error for some reason, the log will simply display everything logged up until that point.

### Programming Environment:

We all did separate parts of the coding on our own, so we all used different environments and compiled it into a Github repository. I used Eclipse, Risvee used Visual Studio Code, and Vishak used Visual Studio Code, all on Windows machines to write the code initially, then tested a bit using the UTD Linux servers. Most of the work at the end editing code and fixing bugs was done by Risvee using Visual Studio Code on a Windows system. We then tested everything on a Linux machine with the terminal.

### Compilation and Execution:

We used a Linux terminal to run our code, so without makefile you use the following lines, in a Linux terminal:  
javac Server.java  
javac Client.java  
java Server  
java Client  
When using a makefile, use the following lines:  
make  
java Server  
java Client  
After this you simply give your name to the server via its connection with the client and you can use the server to solve your math problems.

### Parameters Needed:

The parameters need before launching the program are the port number and address of the server, which is set in our code but can be replaced if you want to use something else for either of these. The client need only give their name and the problem they want solved, in the format described in the protocols above, and the server will do this for them. The only required parameters from the user are the user’s name and whatever problem they want to be solved.

### Issues and Bugs:

Our client and server connect well, but we ran into issues with the sockets and getting the server and client to communicate successfully. While it does work sometimes, most of the time any attempt to send in a problem to be solved will result in an error, unfortunately our code does not work as intended.

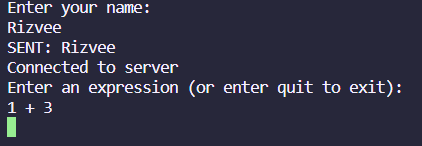
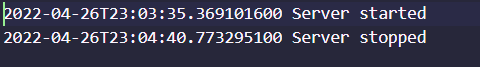
### Challenges Faced:

Our first challenge was a lack of communication, as none of us established communication until about 2 weeks ago, and then we had issues with no one responding to the group trying to meet, so we ended up beginning work on it about 3 days before it was due. When we finally divided up the labor, we still had issues with trying to get information from each other, but we managed to make it work. To get into specifics, we ran into a big issue getting the server and client to connect, as they were both written separately, but after some debugging we fixed it. We also ran into a big issue, where our code does not like to communicate well, so it doesn’t quite work.

### What we learned:

The biggest thing learned during this project was that we needed to start sooner, as our waiting until the last minute made this project much more difficult than it should’ve been had we started a week before we did. We also learned a bit about Socket Programming, as for me personally, I didn’t quite get it in my operating systems class, but I do now.

### Output Screenshots:

Client Output:  
  
Server Output:  
  
Server Log: Attached as log.txt

### Work Division:

Our final work division was Risvee writing the code for the server, Vishak writing the code from the client, I write the report, and I helped attempt to edit both teammates code when we put it all together to try to make it fit.

Addendum:  
We will continue working on the code and submit another late version if we get it to work tomorrow.