Megan Tran
Gabriel Goldstein
CS 4390.502
Term Project

Project Report

a) **Names and NetIDs for the group members:**
Megan Tran: MJT170002
Gabriel Goldstein: GJG180000

b) **Protocol design**
Message format for sending and receiving math calculation:
The client accepts the following format from System.in for a math problem (typed by the user):

- **<problem>** = <operand> <operator> <operand> |
  <operand> <operator> <problem>

Where:

- **<operator>** = + | - | * | /
- **<operand>** = any number

Explanation: The format for a math message is at least 1 instance of a number (positive or negative) followed by a simple operator (plus, minus, times, or divide) followed by a second number. There will be one space in between each operand and operator.

Ex:

- 1 + 2
- 3 * 4 + 5

The client will send the math problem as a string in the following format to the server:

- <**problem**> + “\n”

Explanation: The message the client sends to the server contains the problem entered by the user followed by a new line character. The server receives the message in that format and can then operate on the math problem.

The server will send the answer back to the client in the following string format:

- <**answer**> + “\n”

Explanation: The message the server sends to the server contains a number, which is the answer to the problem the client sent, plus a new line character.

Message format for joining and terminating connection:

- Joining connection:
  - Client sends IP address and port number of local machine.
    - IP address = “127.0.0.1”
    - Port number = 6789
    - Note: Client and server must be run on same machine

  - Server forms new socket connection and acknowledges connection to client (done implicitly through built in java Socket library)
- Upon successful connection:
  - client sends name as string as follows:
    - <name> + "\n"
      - where <name> is a string and is followed by a new line character
      - The name will have been read upon starting the client program through command line arguments
- Upon receiving name:
  - Server stores name for client
  - server acknowledges name with the following message:
    - "Name received by server"
  - Client waits until it receives "Name received by server"
    - Once message received, client proceeds to starts taking user input
- Terminating connection:
  - User types "stop" in System.in (instead of a math problem)
  - Client reads "stop" from input buffer and sends message to notify server
    - Message format: "stop \n"
  - Server receives stop acknowledges by sending "stopped"
  - Server closes all connections, including socket connection and input and output buffers
  - Client receives "stopped" message
  - Client closes input and output buffers

<u>Format for keeping logs of clients' activities at server side:</u>

Format of message to log file:
- [<Time>]: <clientName> <action> <value>

Where:
- **<Time>** = current time. Time has the following format
  - [HH:mm:ss timezone MMMM dd, yyyy]:
- **<clientName>** = name of client thread
- **<action>** = "connected" | "requested" | "received" | "disconnected"
- **<value>** =
  - if <action> = "connected"
    - <value> = null
    - Purpose: To note that the client has successfully connected. No value needed in this case
  - if <action> = "requested"
    - <value> = math expression | "stop"
    - Purpose: To note that server has received the math expression from the user or to note that the client has sent a stop command.

- if <action> = “received”
  - <value> = numeric answer to math expression
  - Purpose: To note that server has solved and sent the answer to the math expression to the client
- if <action> = “disconnected”
  - <value> = “(connected for <duration> seconds)”
    - <duration> = time client was connected to server
  - Purpose: To note that server has disconnected from the client and closed all connections

Explanation: Each activity log entry begins with the time, timezone, and date of the request, respectively. This is followed by the client’s name, an action, and a value (which may be an empty string).
Possible actions are: connected, requested, received, and disconnected.
Possible values are: an empty string (for the initial connection there is no value), a math problem, an answer to a sent math problem, the command “stop” (to end the connection), or the total time connected.

Ex:
```
[11:29:46 CST November 12, 2020]: client5 connected
[03:01:17 CST November 18, 2020]: client4 requested 3+1+4
[03:20:50 CST November 14, 2020]: client7 received 125
[08:30:57 CST November 15, 2020]: client2 requested stop
[03:21:41 CST November 11, 2020]: client3 disconnected (connected for 48 seconds)
```

c) The programming environment you used
Visual Studio Code and IntelliJ IDEA Community Edition

d) How to compile and execute our programs
To compile and run server:
```
make Server
```

To compile and start a client (open in new terminal, repeat for each client)
```
make Client name="<name>"
```
(Example: make Client name="client1")

To solve math problems (in client terminal):
```
<math problem>
```
(Example Format: 2 + 2 * 4)

To stop: (execute commands in respective terminals)
To stop client:
```
stop
```

To stop server:

```
stop
    (make sure no clients are connected)
```

e) **Parameters needed during execution (i.e., IP, port, may be name)**
   The only parameter needed during execution is the name you wish to assign the client when connecting (a command line argument).

   Ex:

   ```
   make Client name="<name>"
   ```

   Where:

   <name> is the input (to be placed inside the quotation marks)

f) **Good use of comments throughout your files and code**
   We made thorough, detailed comments in our code as we were working on our project to make sure all of our code was clear, readable, and understandable. Each section of code is accompanied by a descriptive comment to show exactly what its purpose is. Functions are also preceded with an explanation of their purpose.

g) **If your application is not complete, specify what works and what doesn't**
   Our application is fully complete and functions as specified in the term project instructions with the assumptions listed below.

h) **Challenges faced**
   We struggled a bit with managing the different connected clients and dealing with their threads. We also had trouble stopping the program smoothly without errors, but we were able to figure it out. We also had trouble figuring out how to implement the makefile, and we ultimately decided to have it accept arguments from the user.

i) **What you have learned doing project**
   We learned a lot about thread management during the course of this project. We learned how to run multiple threads at once. We also learned how to send messages back and forth between a client and a server. We did this project completely through peer programming, and learned that it helps a lot to be able to have someone else to look at your code and catch any mistakes you make. We were also able to see tcp in action, and how connections remain persistent until explicitly closed by client or server.

Assumptions:

- No parentheses will be included in the mathematical equation
- There will be one space in between each operand and operator
- The client and server must be run on the same machine

Video:

https://youtu.be/1GErm_oZs_4

Megan Tran
Gabriel Goldstein
CS 4390.502
Term Project

Design Document (overview of program)

I. Socket Connections
- Client and Server utilize TCP connections for sockets
- Server has a welcoming thread to await each incoming client
- Upon receiving new client, server makes new socket connection and starts a new thread for that client
- Upon closing socket connection (describe below), server ends client thread

II. Sending and receiving Math Expressions
- Client awaits input from System.in
- Client sends math problem to server through output buffer
- Server reads math problem from input buffer
- Server solves math expression (described below)
- Server sends answer to client
- Client awaits response from server
- Client prints answer to console when received

III. Solving Math Expression (Two ways)

1.
- Server invokes Script Engine Manager
- Server checks if external script was successful in parsing and solving expression
- Server parses and solves expression locally if external script unsuccessful

2. On some machines, script engines do not work, so we have coded a backup solution in case:
- Client uses modified implementation of Dijstra's shunting yard problem, where parentheses are not considered (only basic operators and operands)

IV. Closing Connections
- Client awaits "stop" input from System.in
- Client sends stop message to server through output buffer
- Server acknowledges stop message by sending "stopped" message to client
- Server closes input and output buffers
- Server closes socket connection
- Client waits from socket connection to close, then closes input and output buffers