Project Report

CS 4390 Final Project

Authors:
Rosie Wang (rdw170000)
Sanjana Sankaran (sxs179530)
Agatha Lam (ajl170230)

1. Protocol Design

A Message class was used to store all information needed about a message by the protocol.

Message.java

Field	Type	Purpose
clientID	String	To store client ID
data	String	To store math request
answer	String	To store math reply
syn	boolean	To request for a connection
ack	boolean	To acknowledge syn or fin messages received
fin	boolean	To request for disconnection
err	boolean	To indicate error handling math request

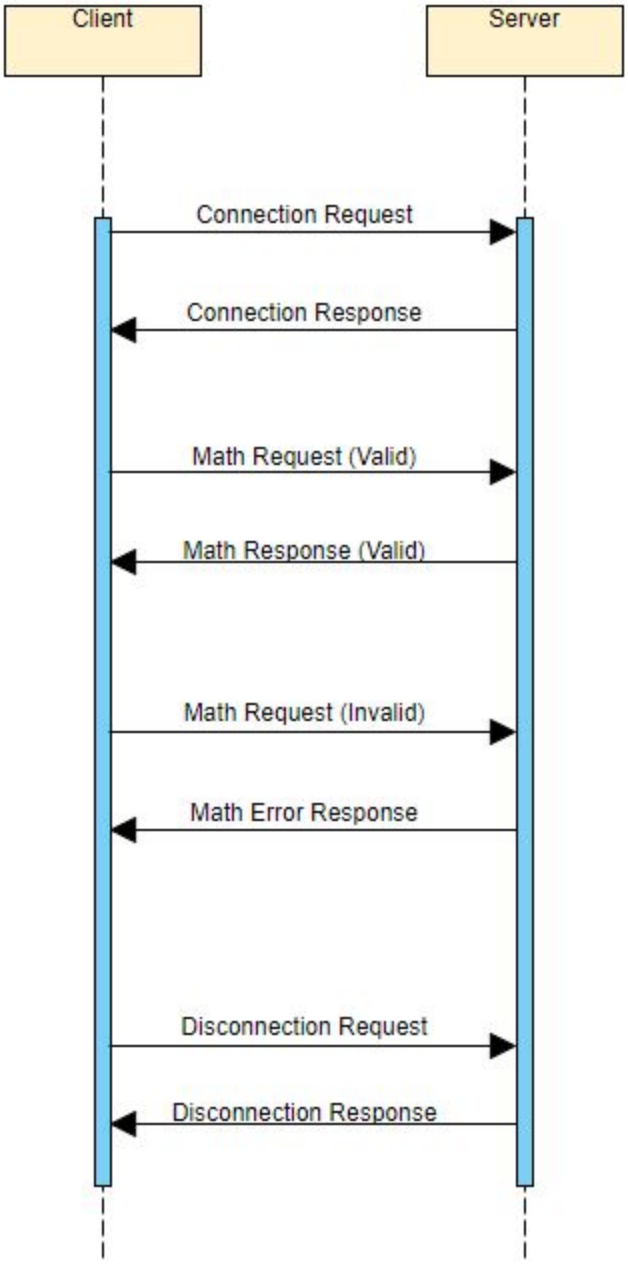
The Message class allowed for us to have 3 different types of client request messages:

	Connection Request	Math Request	Disconnection Request
clientID	*[clientID]*	*[clientID]*	*[clientID]*
data		*[equation]*	
answer			
syn	*true*	*false*	*false*
ack	*false*	*false*	*false*
fin	*false*	*false*	*true*
err			

And 4 different types of server response messages:

	Connection Response	Math Response	Math Error Response	Disconnection Response
clientID	*[clientID]*	*[clientID]*	*[clientID]*	*[clientID]*
data		*[equation]*	*[equation]*	
answer		*[answer]*		
syn	*true*	*false*		*false*
ack	*true*	*false*		*true*
fin	*false*	*false*		*true*
err			*true*	

Network Sequence Diagram



1.1 Server Design

TCPServer simulates a running math server which can make math calculations for math requests sent by its clients. TCPServer is able to handle multiple client connections and handle client requests in order. TCPServer generates separate **TCPServerThreads** threads to handle requests for each connection it has, including connection requests, math requests and disconnection requests.

TCPServerThreads is created when the TCPServer recognizes a connection by a new client on the serverSocket. Upon starting a TCPServerThread, it will wait for a proper connection request from the client and send back a connection response, recognizing an established connection. The client's id will be obtained and it will begin to log the client's events in the TCPServer log including:

- Client name
- Client connection/disconnection
- Client math requests
- Client total connection time

If a proper connection request is not received, the TCPServerThread will terminate the connection. After proper connection, it can receive math requests. The math expression is evaluated and the message containing the answer is logged.

TCPServer.java

```
create server welcome socket

while true
        wait for socket connection
        create server thread to handle connected socket
        start server thread
```

TCPServerThreads.java

```
endConnection = false
synReceived = false

while !endConnection
        RECEIVE(message)
        if !synReceived
                if message.syn                          // handle connection request
```

```
                    response = syn-ack message
                    synReceived = true
                    LOG(clientID connection)
                else
                    endConnection = true
                    LOG(invalid connection attempt)
                    close socket
            else
                if message.fin                          // handle disconnection request
                    response = fin-ack message
                    endConnection = true
                    LOG(clientID disconnection)
                    LOG(clientID total connection time)
                    close socket
                else                                    // handle math request
                    response = SOLVE(message)
                    LOG(clientID math request)
            SEND(response)
```

TCPServer Log Format

The log format is based on the common Logger class format.

Date Time Class Function
Log Setting: message

For example:

```
Nov 19, 2020 12:00:00 AM TCPServer log
INFO: client151dsj343-12h3-234j-wer1-2kfgobnskd45 has
connected
```

These are the formats for logging activities:

```
[CLIENT] has connected
[CLIENT] request: [math expression]
[CLIENT] has disconnected
[CLIENT] has connected for [time connected] ms
```

1.2 Client Design

TCPClient simulates a math client which sends 3-6 randomly generated math expressions to a server for math calculations at random times. In order to establish a connection, it will first send a message with the syn field marked to request for a connection. It will wait to receive a syn-ack response to confirm that the server has established and acknowledged the connection request. It can then send math requests. To disconnect, it will send a message with the fin field marked to request for disconnection. Once it has received a fin-ack response to confirm the server has disconnected and acknowledged the disconnection request, the client can properly close the socket.

This math client can only connect to math servers at its local IP address. TCPClient and TCPServer do not need to be configured. Both will communicate on the same port 6789 by default. Therefore, any other programs or clients outside of this simulation using the same port will create errors with the server.

The generation of a random math equation/expression string (not shown in pseudocode below) is done recursively by calling the method generateRandomEquation(int numOps, boolean topLevel). The resulting equation has the following properties:

- 1 - 4 operators (aka 2 - 5 numbers)
- operators are any of `+`, `-`, `*`, and `/`
- numbers are integers between 0 - 30
- divide by zero is handled by Script Engine as `Infinity`
- parentheses are optionally added at each level of recursion except the topmost

TCPClient.java

```
create client connection socket
generate random client ID

connectionAttempts = 0
do                                                  // connection request
        SEND(syn message)
        RECEIVE(reply)
        connectionAttempts++
while !reply.syn and !reply.ack and connectionAttempts <= 3

for i = random int 3-6 to 0                         // math request
        mathRequest = GENERATE_RANDOM_EQUATION(random int 1-4, true)
        SEND(mathRequest)
        RECEIVE(reply)
        WAIT(random int 0-3000)     // time in ms
```

do // disconnection request
SEND(fin message)
RECEIVE(*reply*)
while !*reply.fin*

close socket

2. Programming Environment Used

Our team used a variety of programming environments including Visual Studio Code, Eclipse, IntelliJ and Vim as code editors. Code was commonly maintained via GitHub.

3. Compilation and Execution

There are no parameters needed for execution.

In cs1.utdallas.edu server, use the following commands:

```
PATH:/usr/local/bin:/usr/bin:/bin:/usr/ccs/bin:/sbin:/
usr/sbin:/usr/local/openwin/bin:/usr/openwin/bin

make classes

make server
   - or "java TCPServer"

make client
   - or "java TCPClient"
```

To simulate multiple client connections running on the server, run make client on multiple terminals. Open log.txt to view server log file.

4. Challenges Faced

Editing the server to handle more than one client proved to be a challenge because it involves OS concepts with threading and understanding how to split the responsibilities between the main TCPServer thread and TCPServerThreads threads. Additionally, we faced a problem where the threads were not starting correctly.

We also struggled with creating log files which did not lock readers and writers. Simple log files required writers to close the file before changes could be available to readers. In order to keep actively updating the log files of a math server which should continue to stay up and running, we would need to close and initialize the writer constantly, wasting resources and lowering performance. We wanted the log files to be accessible and readable when the server thread was running so we learned about the Logger class in Java.

5. Lessons Learned

We learned about Java Socket class and ServerSocket class:

- How to communicate using ports and IP addresses
- How to use the DataOutputStream to send messages through the socket
- How to use InputStream to receive messages from the socket

We learned about how threads can be used in networks and to handle multiple connections. Thus, we learned about the Java Thread class:

- The difference between Thread.start() and Thread.run()
- How start() and run() both use the run function in threads
- How to start a thread
- How to delay a thread using Thread.sleep()

We learned about the Java Logger class:

- How to set the logging.properties file to change the logging output
- How to stop sending output to the command line
- How to send output to a file using the FileHandler class

We learned about the Java Random class:

- How to generate a random integer in a certain range using Random.nextInt()
- How to generate a random boolean value using Random.nextBoolean()
- How to generate a random math expression

We learned about the Java UUID class:

- How to generate a random UUID using UUID.*randomUUID*().toString()

Lastly, we learned about the Javax ScriptEngine interface and Javax ScriptEngineManager class:

- How to use the *eval* method to calculate or evaluate the math expression and output the given result