Computer Networks Project Report

NetworkCalc—A Java socket-programming calculator application

Student: Shoaib Huq, sxh200053
Student: Agastya Bose, axb200123

We present NetworkCalc, a client-server networking application using Java sockets that allows multiple clients to simultaneously send randomly-generated arithmetic expressions to the server, which evaluates and returns the computed result.

1 Protocol Design

The messaging protocol employed by our application uses the following types of messages to communicate between the client(s) and the server. Each message consists of the message type followed by the client's name (except in the case of error messages from the server to clients), and optionally some further information, depending on the message. Each component of a message is separated by the colon symbol (`:`). The messages are broadly classified into the following:

1.1 Client-to-server Messages

1.1.1 Connection request (`JOIN`)

These are the messages that a client sends to the server when it first connects to it. It establishes their connection. The client awaits an acknowledgment of this request from the server before proceeding on to issue other requests.

- Format: `JOIN:<ClientName>`
- Example: `JOIN:Alice`

1.1.2 Calculation request (`CALC`)

These are the messages that a client sends to the server to request it to perform a calculation of the provided arithmetic instruction.

- Format: `CALC:<ClientName>:<ArithmeticExpression>`
- Example: `CALC:Alice:12+7*3`

1.1.3 Disconnection request (`LEAVE`)

These are the messages that a client sends to the server after it has received the response to its last calculation request and is about to disconnect and terminate its session.

- Format: `LEAVE:<ClientName>`
- Example: `LEAVE:Alice`

1.2 Server-to-client Messages

1.2.1 Acknowledgment (ACK)

This is the response from the server that each client shall await after sending out a JOIN request. The server sends out this message to newly connected clients when it is ready to handle their requests.

- Format: ACK:<ClientName>:Welcome
- Example: ACK:Alice:Welcome

1.2.2 Calculation response (RES)

The server responds to calculation requests with calculation responses. This is how the server communicates how it processed the input data (from the arithmetic expression) back to each client.

- Format: RES:<ClientName>:<Result>
- Example: RES:Alice:33

1.2.3 Error message (ERR)

This is the default message sent by the server when it can't recognize or parse a request from a client. The server tries to be as descriptive as it can about the error encountered in processing the request.

- Format: ERR:<ErrorDescription>
- Example: ERR:Invalid Expression Format

1.3 Server Logging Format

1.3.1 Connection log (CONNECT)

This entry is recorded whenever a new client successfully joins the server. It includes the client's name and the IP address from which the connection was established.

- Format: [YYYY-MM-DD hh:mm:ss] CONNECT - <ClientName>: Connected from <IP:Port>
- Example: [2025-04-27 03:39:33] CONNECT - Alice: Connected from /127.0.0.1:34278

1.3.2 Calculation request log (CALC_REQUEST)

This entry is logged whenever the server receives a valid calculation request from a client. It contains the name of the client and the full expression that was submitted.

- Format: [YYYY-MM-DD hh:mm:ss] CALC_REQUEST - <ClientName>: Expression received: <Expression>
- Example: [2025-04-27 03:39:41] CALC_REQUEST - Alice: Expression received: 5+3*2

1.3.3 Calculation result log (CALC_RESPONSE)

After evaluating a valid expression, the server logs the result it computed for that request. This confirms that the server successfully processed and responded to the client's message.

- Format: [YYYY-MM-DD hh:mm:ss] CALC_RESPONSE - <ClientName>: <Expression> = <Result>
- Example: [2025-04-27 03:39:41] CALC_RESPONSE - Alice: 5+3*2 = 11

1.3.4 Disconnection log (DISCONNECT)

When a client disconnects from the server (either gracefully or unexpectedly), this entry records the event and the total duration of the client's session.

- Format: `[YYYY-MM-DD hh:mm:ss] DISCONNECT - <ClientName>:  Client disconnected after <Duration> seconds`
- Example: `[2025-04-27 03:39:45] DISCONNECT - Alice:  Client disconnected after 12 seconds`

1.3.5 Error log (ERR)

Any time the server encounters a malformed message or fails to process a request, an error entry is written to the log. This helps with debugging and transparency.

- Format: `[YYYY-MM-DD hh:mm:ss] ERR - <ClientName>:  <ErrorDescription>`
- Example: `[2025-04-27 03:39:41] ERR - Alice:  Invalid Expression Format`

2 Programming Environment

The project was primarily developed in a Unix environment. Shoaib utilized the MacOS environment whereas Agastya used GNU/Linux. Both utilized the Visual Studio Code editor extensively. The programming language used to implement the project is Java. It was chosen because both teammates are familiar with it and it has extensive first-party networking and concurrency support in the form of standard libraries. Java also has a lot of documentation and guides available online to learn these topics. Additionally, because of Java's use of the Java Virtual Machine, this project can run on any device with a JDK installed. We have included both a Makefile as well as a Windows batch script to automate the process of compilation and execution on a wide variety of OS architectures. Further information on this may be found in the next section.

3 How to Compile and Execute

3.1 Prerequisites

Before compiling and running the NetworkCalc application, ensure you have:

- Java Development Kit (JDK) 21 or later installed
- Java added to your system PATH
- Make utility (for Unix systems) or access to Command Prompt (for Windows)

We have included both a Makefile (for Unix devices) as well as a build.bat batch file (for Windows devices) to automate the compilation and execution process necessary to run the application. Make sure you run the following commands from the terminal/command prompt while in the project root directory.

3.2 Compilation

Unix:

make

Windows:

```
.\build.bat compile
```

This will compile and place the required Java class files in the project's `bin/` sub-directory.

3.3 Starting the Server

Unix:

```
make run-server
```

Windows:

```
.\build.bat run-server
```

The server will start and listen on port 12345. You can shut down the server by typing `quit` in the terminal.

3.4 Starting a Client

Unix:

```
make run-client
```

Windows:

```
.\build.bat run-client
```

When prompted, enter a name for each client started. The client will then send a connection request to the server, which, upon being acknowledged, will lead it to randomly generate arithmetic expressions and send them to the server.

4 Execution Parameters

4.1 Server Configuration

The server has several configurable parameters defined as constants within `MathServer.java`:

- **PORT:** The port number on which the server listens (default: 12345)
- **Thread Pool Size:** Maximum number of concurrent client connections (default: 5)

4.2 Client Configuration

The client application also provides several configurable parameters within `MathClient.java`:

- **HOST:** The server hostname to connect to (default: localhost)
- **PORT:** The server port to connect to (default: 12345)
- **MIN_EXPRESSION_COUNT:** Minimum number of expressions to send (default: 3)
- **MAX_DELAY_SECONDS:** Maximum random delay between expressions (default: 10)

The client is configured to generate between 3 and 8 random expressions. Within each expression, there will be 3 to 10 operands. The expression will pull this set of random operators: {+, -, *, /, %}.

5 Learning & Challenges

5.1 Technical Skills Acquired

Working on developing this application provided opportunities to learn and apply socket programming for client-server communication, multi-threaded programming using Java's concurrency utilities, scheduled task execution for asynchronous operations, and most importantly for this class, protocol design principles for network applications.

5.2 Challenges Faced

Several technical challenges were encountered and overcome:

- **Concurrent Client Handling:** Managing multiple client connections simultaneously required careful thread management and synchronization.
- **Expression Evaluation:** Implementing a robust expression parser that correctly handles operator precedence and various edge cases required implementing the Shunting Yard algorithm.
- **Protocol Design:** Creating a clean, efficient protocol that handles all necessary interactions while remaining simple was a balancing act.
- **Error Handling:** Addressing various failure modes, such as server disconnection, malformed expressions, and client timeouts required comprehensive error handling.
- **Resource Management:** Ensuring proper cleanup of resources (sockets, threads, file handles) under all termination scenarios was challenging.

5.3 Contributions

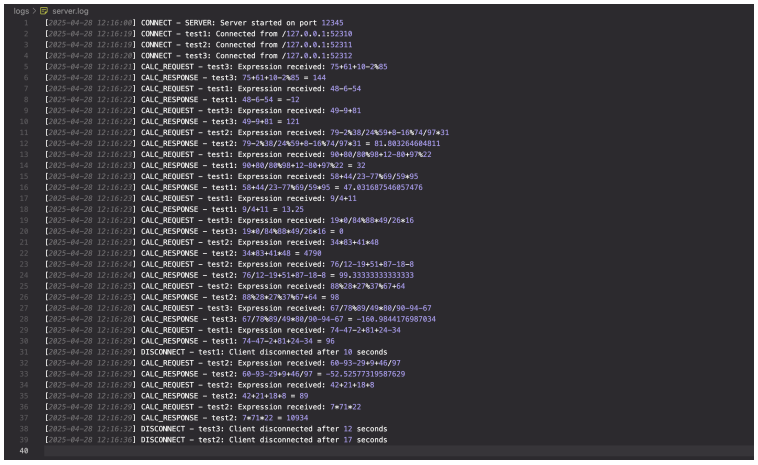
5.3.1 Shoaib

Shoaib took the lead on the client-side implementation and user interaction logic. He developed the client application to ensure intuitive communication with the server, including input validation and output formatting. Shoaib also oversaw the documentation of the project, writing this comprehensive report, usage instructions, and in-line code comments. He was additionally responsible for debugging and testing, ensuring that the overall system ran efficiently and as-expected under various conditions.

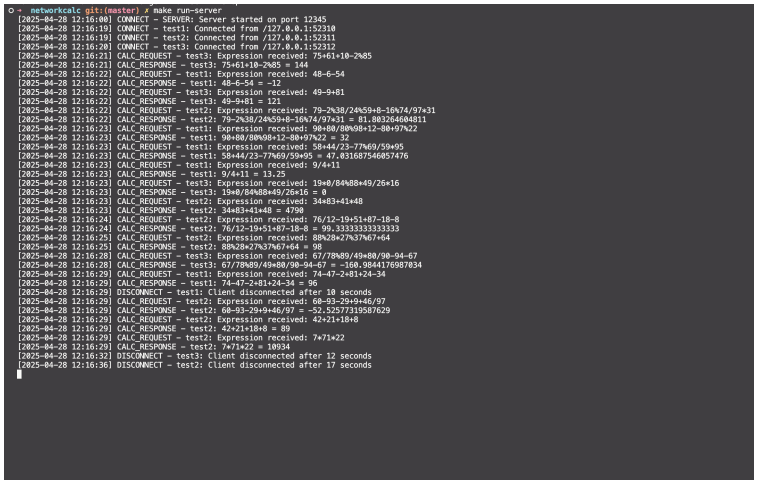
5.3.2 Agastya

Agastya was responsible for the backend development and system architecture of the project. He designed and implemented the server-side logic using Java, ensuring that all network communications followed the defined protocol. Agastya also handled the integration of logging, error handling, and concurrency management to make the system robust and reliable. In addition to backend development, he was in charge of testing for core functionalities along with Shoaib. He maintained the two Makefiles used to build and run the project. He also wrote the design document detailing the overall architecture of the server and client along with their interactions.

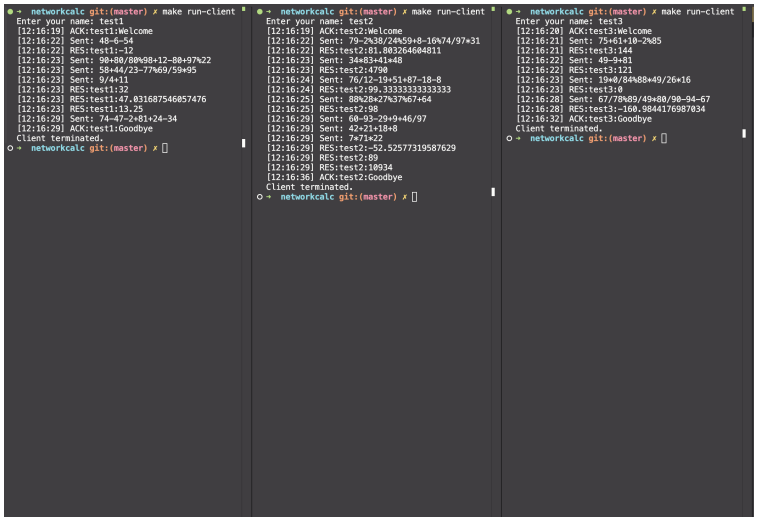
6 Output Screenshots



(a) Client output



(b) Server output



(c) Log output