Computer Networks Project Design

NetworkCalc Implementation Summary

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

1 Design Goals and Assumptions

The primary objective of this project was to implement a lightweight, concurrent client-server application that enables multiple clients to send arithmetic expressions to a central server for evaluation. A key design goal was to maintain a simple and human-readable communication protocol that could be parsed easily using basic string operations. The server was designed to handle multiple clients simultaneously using Java's thread pool executor, while ensuring that all incoming calculation requests are processed in the order they were received through a centralized FIFO queue. Additionally, the server was required to evaluate each arithmetic expression using the Shunting Yard algorithm to follow operator precedence, and to maintain a detailed log of all client activity including connections, disconnections, requests, and responses.

In building this application, several assumptions were made. It was assumed that each client provides a unique identifier (name) when connecting to the server. The user running the client application on each new run will provide this name after being prompted for it. It was also assumed that clients would submit only syntactically valid expressions composed solely of non-negative integer operands not exceeding 100 and operators from the set $\{+, -, *, /, \%\}$, and that all messages exchanged between the client and server would conform to the predefined protocol format. Each expression generated would have at least 3 and at most 8 operands. No arithmetic expression generated by the client will contain parentheses. However, the server does, in fact, support evaluating parenthesized expressions, if needed. Finally, it was assumed that the server would be used in a controlled environment where the number of concurrently connected clients does not exceed five, allowing the use of a fixed-size thread pool to manage client connections efficiently.

2 High-level Overview

On a very high level, this is how the client(s) and the server in our network application interact:

1. Client connects to the server, sends `JOIN:<name>`.
2. Server responds with `ACK:<name>:Welcome`.
3. Client sends requests of the format `CALC:<name>:<expression>` at random times.
4. Server enqueues each incoming `CALC` request, and processes it in order. It evaluates each expression using the Shunting Yard Algorithm, and returns `RES:<name>:<result>` to the correct client.
5. Each client, after it has sent a predetermined number of requests, will eventually send the `LEAVE:<name>` message and disconnect.
6. Server acknowledges the disconnection and logs it along with the total connection duration.

3 Server Implementation

Algorithm 1 `MathServer` main execution flow

```
1: function MathServer
2:     Setup log directory and fresh server.log
3:     Launch RequestProcessor thread
4:     Start background thread to watch for "quit" input
5:     Create TCP ServerSocket on port 12345
6:     while True do
7:         Accept new client connection ← clientSocket
8:         Launch new ClientHandler thread for clientSocket
9:     end while
10: end function
11: function RequestProcessor
12:     while True do
13:         req ← dequeue from requestQueue
14:         Log "CALC_REQUEST - req.clientName: req.expression"
15:         if req.expression is valid then
16:             result ← Evaluate req.expression using Shunting Yard Algorithm
17:             Send "RES:clientName:result" to client
18:             Log "CALC_RESPONSE - clientName: expression = result"
19:         else
20:             Send "ERR:message" to client
21:             Log error
22:         end if
23:     end while
24: end function
25: function ClientHandler(clientSocket)
26:     while input from client ≠ null do
27:         Parse message as cmd, payload
28:         if cmd == JOIN then
29:             Save client name and connection timestamp
30:             Log "CONNECT - name: from IP:port"
31:             Send "ACK:name:Welcome"
32:         else if cmd == CALC then
33:             Enqueue CalcRequest to requestQueue
34:         else if cmd == LEAVE then
35:             Send "ACK:name:Goodbye"
36:             break
37:         else
38:             Send "ERR:Invalid Expression Format"
39:         end if
40:     end while
41:     Compute connection duration
42:     Log "DISCONNECT - name: Client disconnected after X seconds"
43:     Close clientSocket
44: end function
```

4 Client Implementation

Algorithm 2 `MathClient` main execution flow

function MathClient
  Connect to server at `HOST:PORT`
  Prompt user for client name
  Send `JOIN:<clientName>`
  while received line from server **do**
    if line starts with `ACK:<clientName>:` **then**
      break
    end if
  end while
  Launch background thread to run ReadLoop
  Randomly choose number of expressions $n \in [3, 8]$
  for $i \leftarrow 1$ to n **do**
    Wait random delay $d_i \in [1, 10]$ seconds
    Generate expression e_i with $3 - 10$ operands and random operators
    Send `CALC:<clientName>:<expression>`
  end for
  Wait random time $t \in [10, 20]$ seconds
  Send `LEAVE:<clientName>`
  Shutdown client
end function
function ReadLoop
  while client is running **do**
    Read line from server
    if line is not null **then**
      Display line
    else
      Exit loop
    end if
  end while
end function