| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Practical Number:** 2 | |
| **Aim of Practical:**  Implement a Server calculator containing ADD(),MUL(),SUB() etc. | | | |
| **DOP:** 31.8.23 | | **DOS:** 10.10.23 | |
| **CO Mapped:**  CO1 | **PO Mapped:**  - | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 2

**Aim:** Implement a Server calculator containing ADD(),MUL(),SUB() etc.

**Theory:**

Remote Procedure Call (RPC) using Datagram is a communication mechanism in distributed computing where programs on different computers can invoke procedures or methods on remote systems as if they were local, using the User Datagram Protocol (UDP) for communication. Here's a brief explanation:

RPC Concept: RPC is a method of interprocess communication that allows a program to cause a procedure (subroutine) to execute in another address space, typically on a remote server. The calling program can pass arguments to the remote procedure, and the procedure can return results.

Datagram: Datagram refers to the use of UDP, a connectionless transport protocol, for communication between the client and server. Unlike TCP, UDP doesn't provide reliability and ordering guarantees, making it suitable for scenarios where low latency is more critical than guaranteed delivery.

Simplicity and Efficiency: RPC over Datagram is often chosen for its simplicity and efficiency. It's well-suited for situations where a small amount of data needs to be transmitted quickly, and occasional packet loss or out-of-order delivery can be tolerated.

Stateless: UDP is stateless, meaning each packet is independent of others. This makes it a good choice for stateless services or operations that can be retried if a packet is lost.

Error Handling: RPC over Datagram typically involves less overhead for error handling and connection setup compared to RPC over TCP. However, it may require additional mechanisms for handling lost or duplicate packets if reliability is crucial.

Example Use Cases: RPC over Datagram is commonly used in scenarios such as online gaming, real-time multimedia streaming, and situations where the client and server exchange small, time-sensitive messages.

Challenges: The use of UDP introduces challenges such as packet loss, packet duplication, and dealing with out-of-order packets. To ensure reliability, additional mechanisms like acknowledgment and retransmission may be implemented at the application level.

Security: RPC over Datagram may require additional security mechanisms to protect against unauthorized access and data tampering since UDP doesn't inherently provide encryption or authentication.

**Code:**

File: ServerCalcultor.java

package vesitajayk57.practical2;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class ServerCalculator {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(12345)) {

System.out.println("Server is running and listening on port 12345...");

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress());

while (true) {

// Create input and output streams for communication with the client.

BufferedReader input = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);

// Read the math expression from the client.

String expression = input.readLine();

if (expression == null || expression.isEmpty()) {

clientSocket.close();

System.out.println("Client disconnected.");

break;

}

System.out.println("Received Expression: " + expression);

// Evaluate the expression and send the result back to the client.

try {

double result = evaluateExpression(expression);

output.println("Result: " + result);

} catch (Exception e) {

output.println("Error: " + e.getMessage());

}

// Close the client socket.

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static double evaluateExpression(String expression) throws Exception {

// Implement your expression evaluation logic here.

// You can use libraries like javax.script.ScriptEngine for this purpose.

// For simplicity, we'll assume that the expression is in a valid format.

// Example evaluation using ScriptEngine (JavaScript evaluator):

// ScriptEngineManager manager = new ScriptEngineManager();

// ScriptEngine engine = manager.getEngineByName("js");

// return (double) engine.eval(expression);

// For demonstration purposes, we'll use a simple parser here:

String[] tokens = expression.split(" ");

if (tokens.length != 3) {

throw new Exception("Invalid expression format");

}

double operand1 = Double.parseDouble(tokens[0]);

double operand2 = Double.parseDouble(tokens[2]);

String operator = tokens[1];

switch (operator) {

case "+":

return operand1 + operand2;

case "-":

return operand1 - operand2;

case "\*":

return operand1 \* operand2;

case "/":

if (operand2 == 0) {

throw new Exception("Division by zero");

}

return operand1 / operand2;

default:

throw new Exception("Invalid operator");

}

}

}

File: ClientCalculator.java

package vesitajayk57.practical2;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.Scanner;

public class ClientCalculator {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 12345);

// Create input and output streams for communication with the server.

PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

// Create a scanner to read user input.

Scanner scanner = new Scanner(System.in);

// Loop to keep sending math expressions until the user provides no input.

while (true) {

// Prompt the user for a math expression.

System.out.print("Enter a math expression (e.g., '5 + 3') or press Enter to exit: ");

String expression = scanner.nextLine();

// If the user provides no input (presses Enter), exit the loop.

if (expression.isEmpty()) {

break;

}

// Send the user's input expression to the server.

output.println(expression);

// Receive and print the result.

String result = input.readLine();

System.out.println("Server Response: " + result);

}

// Close the socket and scanner.

socket.close();

scanner.close();

} catch (IOException e) {

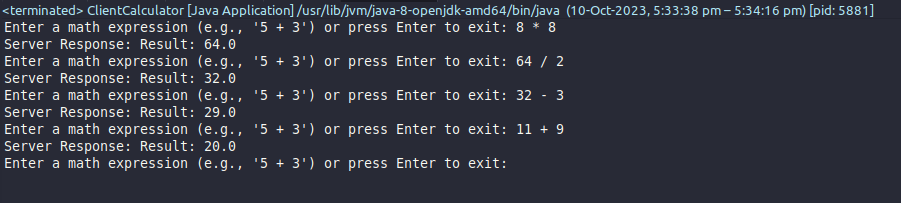
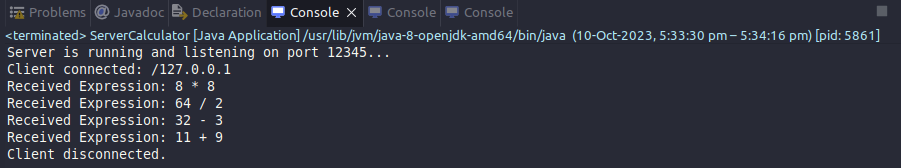
e.printStackTrace();

}

}

}

**Output:**



**Conclusion:**

I learnt how to implement a Server calculator containing ADD(),MUL(),SUB().