

TP : Implémentation d'une Calculatrice avec RPC en Java (Sockets)

Nom : DIAWARA NANA
Filière : CyberSecurity
Cours : Distributed Applications

Introduction

Le but de ce TP est de simuler une communication de type RPC (*Remote Procedure Call*) entre un client et un serveur en Java. Le serveur expose des opérations arithmétiques (addition, soustraction, multiplication, division) comme des procédures distantes. Le client envoie une requête spécifiant l'opération à effectuer et les deux opérandes, puis reçoit le résultat.

Cette implémentation repose sur les sockets TCP pour établir une communication fiable entre client et serveur, avec un format de message simple basé sur les chaînes de caractères.

Objectifs pédagogiques

- Implémenter un système RPC simplifié en Java.
- Utiliser les **sockets TCP** pour échanger des requêtes.
- Formater et interpréter des messages de type "operation:val1:val2".
- Appliquer les concepts d'entrée/sortie réseau.

Structure du projet

```
RPCProject/  
    RPCServer.java  
    RPCClient.java
```

1. Code du Serveur : RPCServer.java

```
import java.io.*;  
import java.net.*;  
  
public class RPCServer {  
    public static void main(String[] args) {  
        try (ServerSocket serverSocket = new ServerSocket(12345)) {  
            System.out.println("RPC Server started on port 12345...");  
        }  
    }  
}
```

```

        while (true) {
            Socket socket = serverSocket.accept();
            new Thread(() -> handleClient(socket)).start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void handleClient(Socket socket) {
    try (
        BufferedReader in = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(),
            true)
    ) {
        String request = in.readLine();
        String[] parts = request.split(":");
        String function = parts[0];
        double a = Double.parseDouble(parts[1]);
        double b = Double.parseDouble(parts[2]);

        double result = switch (function) {
            case "add" -> a + b;
            case "sub" -> a - b;
            case "mul" -> a * b;
            case "div" -> b != 0 ? a / b : Double.NaN;
            default -> Double.NaN;
        };

        out.println(result);

    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

2. Code du Client : RPCClient.java

```

import java.io.*;
import java.net.*;

public class RPCClient {
    public static void main(String[] args) {
        try (
            Socket socket = new Socket("localhost", 12345);
            BufferedReader userIn = new BufferedReader(new
                InputStreamReader(System.in));
            PrintWriter out = new PrintWriter(socket.getOutputStream(),
                true);
            BufferedReader in = new BufferedReader(new InputStreamReader(
                socket.getInputStream()))
        ) {
            System.out.println("Connected to RPC Server");
            System.out.print("Enter operation (add/sub/mul/div): ");

```

```

        String op = userIn.readLine();
        System.out.print("Enter operand a: ");
        String a = userIn.readLine();
        System.out.print("Enter operand b: ");
        String b = userIn.readLine();

        String request = op + ":" + a + ":" + b;
        out.println(request);

        String response = in.readLine();
        System.out.println("Result: " + response);

    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

3. Instructions d'exécution

1. Compiler les deux fichiers :

```
javac RPCServer.java RPCClient.java
```

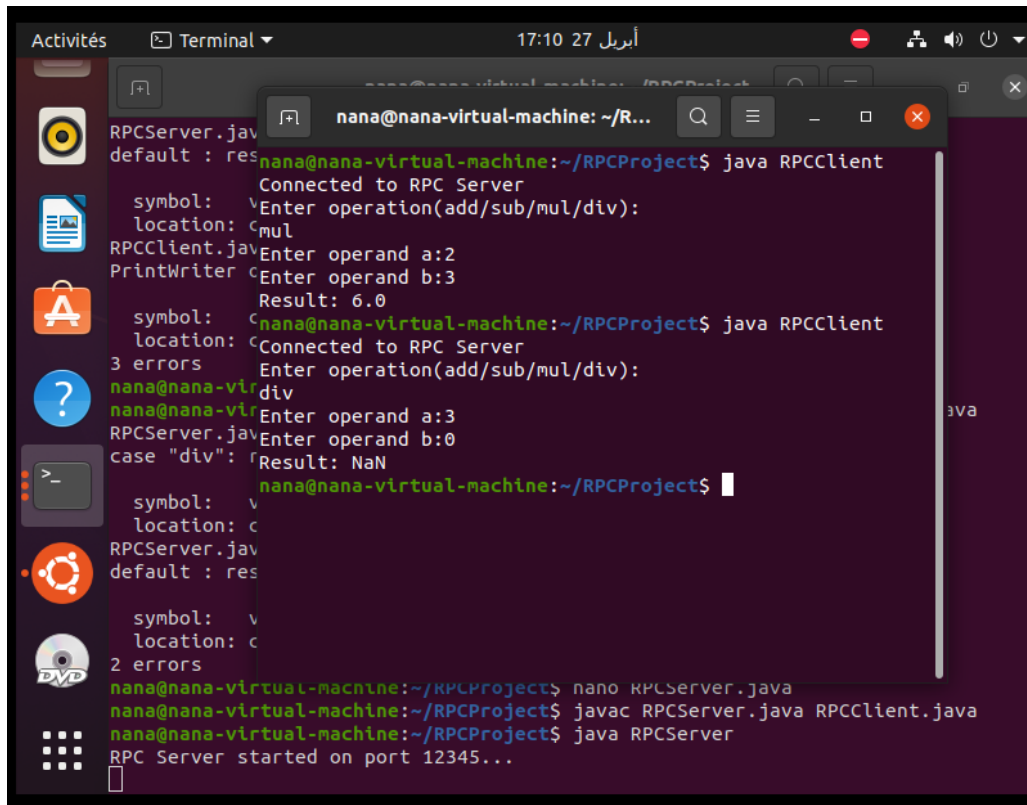
2. Lancer le serveur dans un terminal :

```
java RPCServer
```

3. Lancer le client dans un autre terminal :

```
java RPCClient
```

4. Capture d'exécution (exemple)



```
nana@nana-virtual-machine: ~/RPCProject$ java RPCClient
Connected to RPC Server
Enter operation(add/sub/mul/div):
mul
Enter operand a:2
Enter operand b:3
Result: 6.0
nana@nana-virtual-machine:~/RPCProject$ java RPCClient
Connected to RPC Server
Enter operation(add/sub/mul/div):
div
Enter operand a:3
Enter operand b:0
Result: NaN
nana@nana-virtual-machine:~/RPCProject$ nano RPCServer.java
nana@nana-virtual-machine:~/RPCProject$ javac RPCServer.java RPCClient.java
nana@nana-virtual-machine:~/RPCProject$ java RPCServer
RPC Server started on port 12345...
```

FIGURE 1 – Exécution : saisie des opérandes et affichage du résultat

Conclusion

Ce TP a permis de simuler un système de calcul distribué basé sur une architecture client-serveur. En utilisant les sockets TCP, nous avons mis en place une communication de type RPC simplifiée, dans laquelle le client spécifie une opération et reçoit un résultat calculé par le serveur.

L'approche adoptée illustre la base des systèmes distribués et prépare à des technologies plus avancées telles que Java RMI, gRPC ou les microservices via HTTP/REST. Le format de message simple utilisé ici pourrait facilement évoluer vers une sérialisation plus formelle (JSON, XML, Protobuf) pour des systèmes plus robustes.