JOBSHEET 2

Interoperabilitas

“PEMROGRAMAN REMOTE PROCEDURE CALL (RPC)”

Disusun oleh:

Program Studi Teknologi Rekayasa Komputer
Jurusan Teknik Elektro

POLITEKNIK NEGERI SEMARANG
2023

I. Tujuan Instruksional Khusus

Setelah menyelesaikan praktikkum ini, mahasiswa diharapkan :

1. Memahami konsep RPC
2. Menjelaskan pengunaan RPC
3. Mengimplementasikan RPC dengan Bahasa pemrograman Java

II. Dasar Teori

RPC adalah sebuah metode yang memungkinkan kita untuk mengakses sebuah prosedur yang berada di komputer lain. Untuk dapat melakukan ini sebuah *server* harus menyediakan layanan *remote procedure.* Pendekatan yang dilakukan adalah sebuah *server* membuka *socket,* lalu menunggu *client* yang meminta prosedur yang disediakan *server.* RPC masih menggunakan cara primtif dalam pemrograman yaitu menggunakan paradigma *procedural programming*.

Tujuan

RPC digunakan untuk administrasi sistem sehingga seorang administrator jaringan dapat mengakses sistemnya dan mengelola sistemnya darimna saja, selama sistemnya terhubung ke jaringan.

Socket

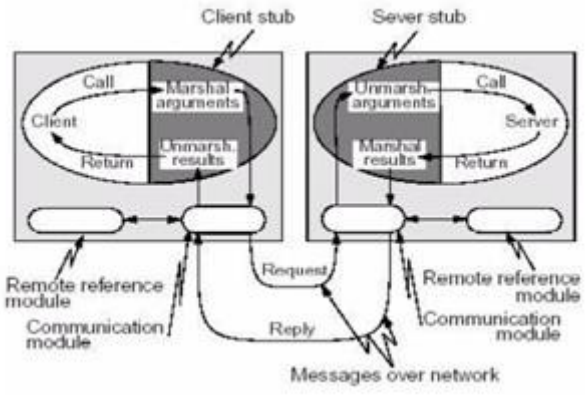
RPC menggunakan socket untuk berkomunikasi dengan proses lainnya.

Cara Kerja RPC

Tiap prosedur yang dipanggil dalam RPC, maka proses ini harus berkoneksi dengan server remote dengan mengirimkan semua parameter yang dibutuhkan, menunggu balasan dari server dan melakukan proses kemudian selesai. Proses di atas disebut juga dengan stub pada sisi klien. Sedangkan Stub pada sisi server adalah proses menunggu tiap message yang berisi permintaan mengenai prosedur tertentu.

Implementasi RPC

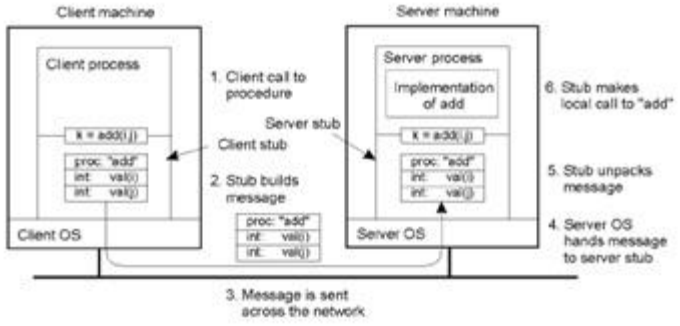
Untuk proses nya kurang lebih sama dengan RMI. Kalau RMI kita mengenal proxy dan skeleton, pada RPC dikenal dengan Stub(Client stub dan Server stub).



Gambar Ilustrasi Implementasi RPC

Remote Reference Modul dan Communication Modul berada pada tatanan sistem operasi. Contoh implementasi adalah Sun Microsystems Open Network Computing (ONC) : RPC specification, XDR (eXternal Data Representation) standard, UDP atau TCP transport protocol. Xerox Courier : RPC model, Data representation standard, XNS (Xerox Network Systems) SPP (Sequenced Packet Protocol) sbg transport protocol, Apollo s Network Computing Architecture (NCA), RPC protocol, NDR (Network Data Representation).

Langkah-langkah dalam RPC



Prosedur client memanggil client stub

Client stub membuat pesan dan memanggil OS client

OS client mengirim pesan ke OS server

OS server memberikan pesan ke server stub

Server stub meng-unpack parameter-parameter untuk memanggil server

Server mengerjakan operasi, dan mengembalikan hasilnya ke server stub

Server stub meng-pack hasil tersebut dan memanggil OS server

OS server mengirim pesan (hasil) ke OS client

OS client memberikan pesan tersebut ke client stub

Client stub meng-unpack hasil dan mengembalikan hasil tersebut ke client

III. ALAT DAN BAHAN

Dalam penggunaan modul praktikum ini, anda harus menyediakan beberapa perangkat dan keperluan pendukung seperti:

1. Laptop atau Komputer Desktop
2. Aplikasi Java Netbeans, Eclipse
3. Akses Internet

IV. PRAKTIKUM

Pada praktikum kali ini, membuat dua aplikasi yang terdiri dari aplikasi server dan aplikasi client. Dua program tersebut yakni program client (RPCClient.java) dan server (RPCServer.java). Di bawah ini memperlihatkan bagaimana RPC (*Remote procedure call*) disimplementasikan di dalam Bahasa Pemrogaraman Java menggunakan pustaka bawaan Socket, SocketServer dan yang terkait.

RPCServer.java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import
java.io.PrintStream;
import
java.net.InetAddress;
import
java.net.ServerSocket
; import
java.net.Socket;


public class RPCServer {

    private final ServerSocket serverSocket;

    @SuppressWarnings("CallToThreadStartDuringObjectConstruction")
    public RPCServer(int port) throws IOException {
        serverSocket = new ServerSocket(port);

        String localIP = InetAddress.getLocalHost().getHostAddress();
        System.out.println("Server is running on " + localIP + ":" + port); while (true)
        {
            Socket rpcClient = serverSocket.accept();
            String address =
            rpcClient.getRemoteSocketAddress().toString();
            System.out.println("New client connected : " +
            address);
            new Thread(() -> {
                try {
                    addHook(rpcClient);
                } catch (IOException ex) {
                    System.err.println("Client disconnected " + address);
                }
```

```
            }).
            sta
            rt(
             );
        }
    }

    private void addHook(Socket rpcClient) throws IOException {

              BufferedReader reader = new
                   BufferedReader(new
InputStreamReader(rpcClient.getInputStr
        eam())); String line;
        while ((line = reader.readLine()) != null) {
            System.out.println("Client request : " +
            line);
             String[] commands = line.split(":",
                             3);
            int result;
                         int operand1 =
                  Integer.parseInt(commands[1]);
                         int operand2 =
                  Integer.parseInt(commands[2]);

            String message =
                   "";

                  switch
              (commands[0]) {

                  case
                 "add":
                    result = (operand1 + operand2);
                    message = operand1 + " + " + operand2 + " = " + result;
                    break;

                  case
                 "sub":
                    result = (operand1 - operand2);
                    message = operand1 + " - " + operand2 + " = " + result;
                    break;

                  case
                 "mul":
                    result = (operand1 * operand2);
                    message = operand1 + " * " + operand2 + " = " + result;
                    break;

                  case
                 "div":
                    result = (operand1 / operand2);
                    message = operand1 + " / " + operand2 + " = " + result;
                    break;

                  case
                 "mod":
                    result = (operand1 % operand2);
                    message = operand1 + " % " + operand2 + " = " + result;
                    break;

            }

            PrintStream printStream = new PrintStream(rpcClient.getOutputStream(),
            true);
            printStream.println(message);
        }
```

```
    }

    public static void main(String[] args) throws
        Exception { RPCServer server = new
        RPCServer(3000);
    }
}
```

RPCClient.java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import
java.io.PrintStream;
import java.net.Socket;
import
java.util.Scanner;


public class RPCClient {

    private final PrintStream printStream;

    @SuppressWarnings("CallToThreadStartDuringObjectConstruction")
    public RPCClient(String ipAddress, int port) throws IOException { Socket rpcClient =
    new Socket(ipAddress, port);
         new Thread(()
              -> {
             try {
                 BufferedReader reader = new BufferedReader(new
InputStreamReader(rpcClient.getInputStream(
                 ))); String line;

                 while ((line = reader.readLine()) != null) {
                     System.out.println("Server response : " +
                     line);
     System.out.print("\nCommands [add, sub, mul, div, mod, exit] :
                                  ");
                 }
             } catch (IOException ex) {
                 System.err.println("\nDisconnected!
                 !"); System.exit(0);
             }
         }).start();

         printStream = new PrintStream(rpcClient.getOutputStream(), true);

    }

    public void sendMessage(String operation)
        { Scanner scan = new
        Scanner(System.in);
        System.out.print("\nEnter 1st number :
        "); int f1 = scan.nextInt();

        System.out.print("Enter 2nd number : ");
        int s1 = scan.nextInt();

        printStream.println(operation + ":" + f1 + ":" + s1);

    }

    public static void main(String[] args) {
```

```
        try {
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter server ip address :
    "); String ipAddress = scan.nextLine();

    System.out.print("Enter connection port : ");
    int port = scan.nextInt();

             RPCClient client = new RPCClient(ipAddress,
             port); System.out.println("\nConnected to
             server\n");

    System.out.print("Commands [add, sub, mul, div, mod, exit] : ");

               while
              (true) {

     scan = new Scanner(System.in);

                 String command = scan.nextLine();

                 if
                     (command.equals("exit
                     ")) { System.exit(0);
                 }

                 client.sendMessage(comma
     nd); System.out.print("\n");
             }
         } catch (IOException ex) {
             System.err.println("\nUnable to
             connected!");
         }

     }

}
```

Server Output:

Server is running on 172.20.52.46:3000
New client connected : /172.20.52.46:52843
Client request : add:10:20
Client request : sub:50:20
Client request : mul:10:2
Client request : div:100:5
Client request : mod:1234:10
Client disconnected /172.20.52.46:52843

Client Output:

Enter server ip address : 172.20.52.42
Enter connection port : 3000

Connected to server
Commands [add, sub, mul, div, mod, exit] :
add
Enter 1st number : 10

```
Enter 2nd number : 20
Server response : 10 + 20 = 30

sub
Enter 1st number : 50
Enter 2nd number : 20
Server response : 50 - 20 = 30

mul
Enter 1st number : 10
Enter 2nd number : 2
Server response : 10 * 2 = 20

div
Enter 1st number : 100
Enter 2nd number : 5
Server response : 100 / 5 = 20

mod
Enter 1st number : 1234
Enter 2nd number : 10
Server response : 1234 % 10 = 4

exit
```

V. TUGAS

Buat kelompok dengan 2 anggota didalamnya kemudian aplikasi RPC Client Server diatas diubauh menjadi Graphic User Interface (GUI). Kemudian ujicoba secara real dengan anggota kelompok anda yang bertindak sebagai server dan sebagai client.

VI. KESIMPULAN