Department of Computer Engineering

Academic Term : January - May 2024

Class: B.E. (Computer) Semester VIII Division: B

Subject Name: Distributed Systems CSC 801

Name	**Riya Rajesh Patil**
Roll No.	**9282**
Experiment No.	**2**
Experiment Title	**To implement Remote Procedure Call**

Aim - To implement Remote Procedure Call

cli.java

```
import java.io.*;
import java.net.*;
public class cli {

public static void main (String [] args) throws Exception {
Socket sock = new Socket("127.0.0.1", 3000);
BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
OutputStream ostream = sock.getOutputStream();
PrintWriter pwrite = new PrintWriter(ostream, true);
InputStream istream = sock.getInputStream();
BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
System.out.println("Client ready, type and press Enter key");
String receiveMessage, sendMessage, temp;
while (true) {
System.out.println("\nEnter operation to perform(add,sub,mul,div)....");
temp = keyRead.readLine();
```

```
sendMessage = temp.toLowerCase();
pwrite.println(sendMessage);
System.out.println("Enter first parameter :");
sendMessage = keyRead.readLine();
pwrite.println(sendMessage);
System.out.println("Enter second parameter : ");
sendMessage = keyRead.readLine();
pwrite.println(sendMessage);
System.out.flush();
if ((receiveMessage = receiveRead.readLine()) != null) {
System.out.println(receiveMessage);
}

}
}
}
```

server.java

```
import java.io.*;
import java.net.*;
class server {
public static void main(String[] args) throws Exception {
ServerSocket sersock = new ServerSocket(3000);
System.out.println("Server ready");
Socket sock = sersock.accept();
BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
OutputStream ostream = sock.getOutputStream();
PrintWriter pwrite = new PrintWriter(ostream, true);
InputStream istream = sock.getInputStream();
```

```
BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
String receiveMessage, sendMessage, fun;
int a, b, c;
while (true) {
fun = receiveRead.readLine();
if (fun != null) {
System.out.println("Operation : " + fun);
}
a = Integer.parseInt(receiveRead.readLine());

System.out.println("Parameter 1 : " + a);
b = Integer.parseInt(receiveRead.readLine());
if (fun.compareTo("add") == 0) {
c = a + b;
System.out.println("Addition = " + c);
pwrite.println("Addition = " + c);
}
if (fun.compareTo("sub") == 0) {
c = a - b;
System.out.println("Substraction = " + c);
pwrite.println("Substraction = " + c);
}
if (fun.compareTo("mul") == 0) {
c = a * b;
System.out.println("Multiplication = " + c);
pwrite.println("Multiplication = " + c);
}
if (fun.compareTo("div") == 0) {
c = a / b;
```

```
System.out.println("Division = " + c);
pwrite.println("Division = " + c);
}
System.out.flush();
}

}
}
```

Output -

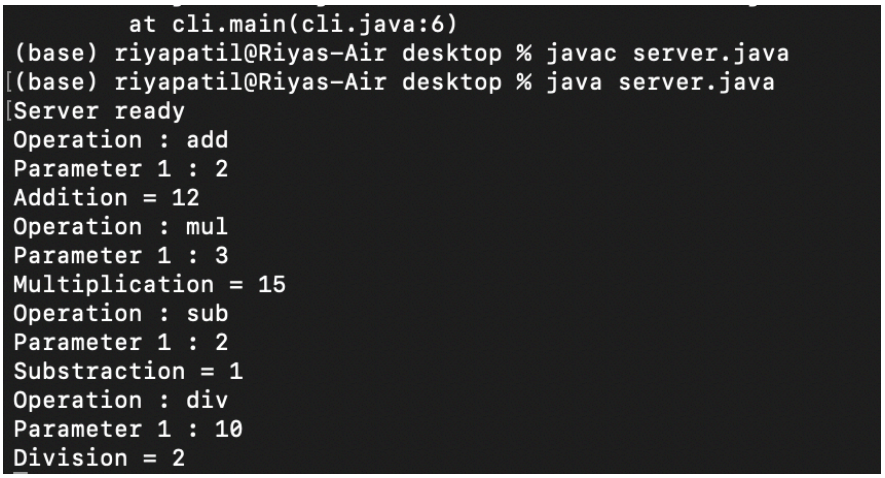
```
Last login: Tue Apr  9 17:44:27 on ttys000
(base) riyapatil@Riyas-Air ~ % cd desktop
(base) riyapatil@Riyas-Air desktop % javac cli.java
(base) riyapatil@Riyas-Air desktop % java cli.java
Client ready, type and press Enter key

Enter operation to perform(add,sub,mul,div)....
add
Enter first parameter :
2
Enter second parameter :
10
Addition = 12

Enter operation to perform(add,sub,mul,div)....
mul
Enter first parameter :
3
Enter second parameter :
5
Multiplication = 15

Enter operation to perform(add,sub,mul,div)....
sub
Enter first parameter :
2
Enter second parameter :
1
Substraction = 1

Enter operation to perform(add,sub,mul,div)....
div
Enter first parameter :
10
Enter second parameter :
5
Division = 2
```



Conclusion -

In conclusion the provided Java code demonstrates a form of Remote Procedure Call (RPC) between a client and a server. The provided code exemplifies a basic Remote Procedure Call (RPC) system. Clients initiate connections, send requests specifying operations and parameters to the server. The server executes these operations and sends back results. This facilitates remote execution of procedures, abstracting complexities of distributed computing through request-response communication over sockets.

Postlab -

1. **In which category of communication, RPC be included?**

   Remote Procedure Call (RPC) falls under the category of inter-process communication (IPC). IPC refers to mechanisms and techniques used by processes or threads to communicate and synchronize with each other, either within the same system or across different systems in a distributed environment. RPC specifically involves invoking procedures or functions in a remote address space, as if they were local, by sending messages between processes or systems over a network. Therefore, RPC is a form of communication that enables processes or systems to interact and collaborate across a network boundary.

2. **What are stubs? What are the different ways of stub generation?**

   Stubs are proxy components that act as placeholders or representatives of remote objects or services. They facilitate communication between client and server by providing a local interface that mirrors the remote interface. Stubs handle the marshalling (serialization) and unmarshalling (deserialization) of parameters and messages between the client and server, abstracting away the complexities of network communication.

There are different ways of generating stubs-
1. Manual Stub Generation: Developers can manually create stubs by writing code that mirrors the remote interface or API. This involves defining classes and methods that correspond to the remote object's interface, including logic for marshalling and unmarshalling parameters and messages. While this approach offers flexibility and control, it can be time-consuming and error-prone, especially for complex remote interfaces.

2. Compiler-based Stub Generation: Some programming languages and RPC frameworks provide built-in support for automatic stub generation during compilation. The compiler analyzes the remote interface definition and generates stub code based on it. This approach saves developers from manually writing stub code, ensuring consistency and reducing the chance of errors. Examples include Java's Remote Method Invocation (RMI) compiler and IDL compilers like CORBA's IDL-to-Java compiler.

3. **What is binding?**

Binding refers to the process of associating a network address or endpoint with a specified communication resource or service. Binding establishes a connection between the communication endpoint (such as a socket or port) and the underlying network infrastructure, allowing data to be sent and received over the network.

There are several types of binding:

1. Socket Binding: In socket programming, binding involves associating a socket with a specific network address (such as an IP address) and port number. This allows the socket to send and receive data over the network using the specified address and port.

2. Service Binding: In distributed systems, services are often bound to specific network addresses and ports so that clients can access them. For example, a web server may bind to port 80 (HTTP) or port 443 (HTTPS) to serve HTTP requests from clients.

3. Name Binding: Name binding involves associating a symbolic name or identifier with a network address or endpoint. This allows clients to refer to services or resources using human-readable names instead of numeric IP addresses.

4. Protocol Binding: Protocol binding refers to the association of a communication protocol with a network address or endpoint. For example, a server may bind to both IPv4 and IPv6 addresses to support communication over both IPv4 and IPv6 networks.

4. **Name the transparencies achieved through stubs**

Various transparencies achieved through stubs are -
1. Location Transparency: Stubs abstract the location of remote objects or services from the client. Clients interact with stubs as if they were invoking local procedures or methods, without needing to be aware of the physical location of the remote object. This transparency enables seamless communication with remote services regardless of their actual location in the network.

2. Access Transparency: Stubs provide access transparency by shielding clients from the details of how remote invocations are implemented and executed. Clients invoke methods on stubs in the same way they invoke local methods, without needing to know the underlying communication protocols or mechanisms used to transmit requests and receive responses.

3. Concurrency Transparency: Stubs can help achieve concurrency transparency by handling concurrent requests from multiple clients in a transparent manner. Clients can concurrently invoke remote methods on stubs without needing to manage synchronization or concurrency control mechanisms explicitly. The stubs handle concurrency issues internally, ensuring thread safety and proper synchronization.

4. Failure Transparency: Stubs abstract the handling of communication failures and errors from clients, providing failure transparency. Clients interact with stubs expecting successful method invocations, and the stubs handle communication failures, retries, and error recovery mechanisms internally. This transparency shields clients from the complexities of dealing with network failures and enhances system robustness and reliability.

5. Migration Transparency: In some RPC systems, stubs can provide migration transparency by allowing remote objects to be moved or migrated between different locations or servers without affecting clients. Clients continue to interact with stubs as usual, and the underlying system handles the migration of remote objects transparently, ensuring uninterrupted service availability.

.