Experiment 2 – RPC/RMI

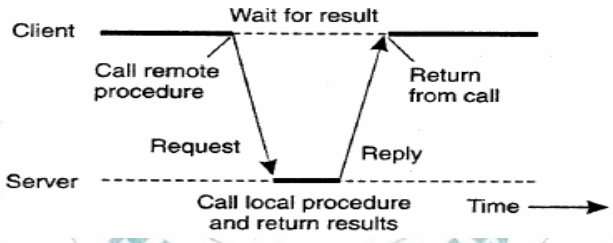
Learning Objective: Student should be able to Built a Program for Client/server using RPC/RMI

Tools : Java

Theory:

Remote Procedure call

A remote procedure call (RPC) is an inter-process communication that allows a computer program to cause a procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.
It further aims at hiding most of the intricacies of message RPC allows programs to call procedures located on other machines. But the procedures 'send' and 'receive' do not conceal the communication which leads to achieving access transparency in distributed systems.
Example: when process A calls a procedure on B, the calling process on A is suspended and the execution of the called procedure takes place. (PS: function, method, procedure difference, stub, 5 state process model definition)Information can be transported in the form of parameters and can come back in procedure result. No message passing is visible to the programmer. As calling and called procedures exist on different machines, they execute in different address spaces, the parameters and result should be identical and if machines crash during communication, it causes problems.



Client Stub: Used when read is a remote procedure. Client stub is put into a library and is called using a calling sequence. It calls for the local operating system. It does not ask for the local operating system to give data, it asks the server and then blocks itself till the reply comes.
Server Stub: when a message arrives, it directly goes to the server stub. Server stub has the same functions as the client stub. The stub here unpacks the parameters from the message and then calls the server procedure in the usual way.
Summary of the process:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's as sends the message to the remote as.
4. The remote as gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local as.
8. The server's as sends the message to the client's as.
9. The client's as gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Implementation of RPC

The implementation of an RPC mechanism is based on the concept of stubs, which provide a perfectly normal (local) procedure call abstraction by concealing from programs the interface to the underlying RPC system. We saw that an RPC involves a client process and a server process. Therefore, to conceal the interface of the underlying RPC system from both the client and server processes, a separate stub procedure is associated with each of the two processes. Moreover, to hide the existence and functional details of the underlying network, an RPC communication package (known as RPCRuntime) is used on both the client and server sides. Thus, implementation of an RPC mechanism usually involves the following five elements of program

1. The client
2. The client stub
3. The RPCRuntime
4. The server stub
5. The server

The interaction between them is shown in Figure 4.2. The client, the client stub, and one instance of RPCRuntime execute on the client machine, while the server, the server stub, and another instance of RPCRuntime execute on the server machine. The job of each of these elements is described below.

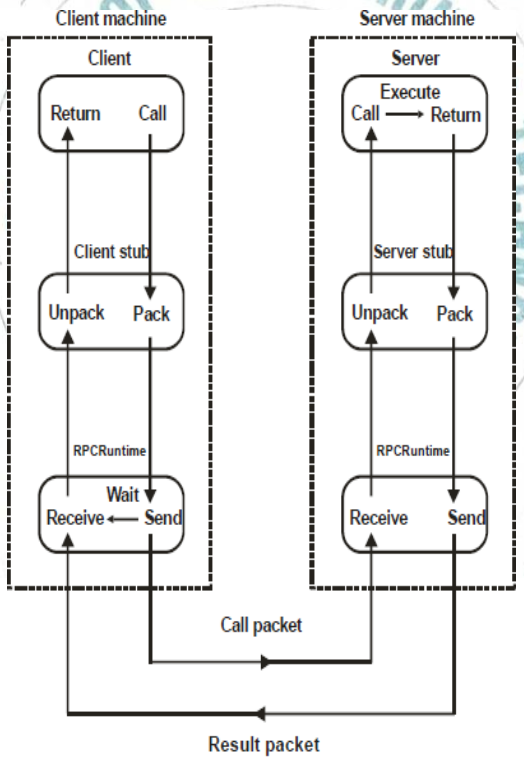
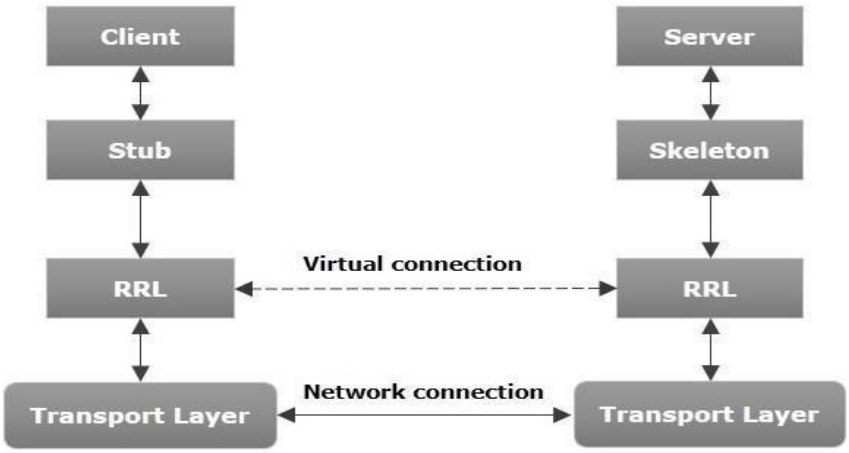


Fig. 3.2 : Implementation of RPC mechanism

Remote Method Invocation (RMI)

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.
RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.rmi.

The following diagram shows the architecture of an RMI application.



Working of an RMI Application

The following points summarize how an RMI application works −

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

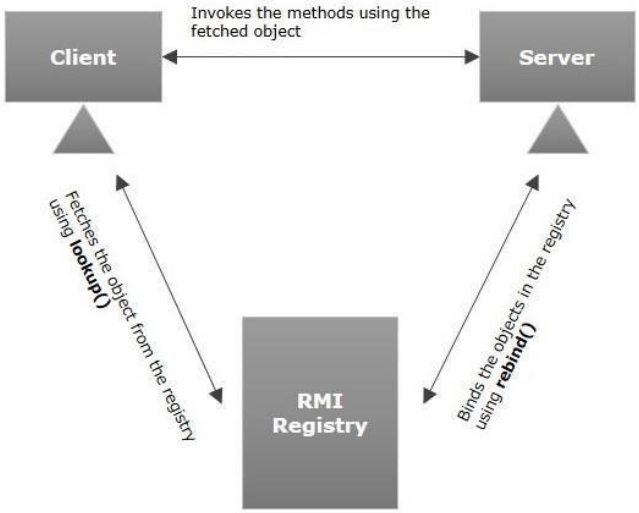
Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as marshalling.
At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as unmarshalling.

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using bind() or reBind() methods). These are registered using a unique name known as bind name.
To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using lookup() method).

The following illustration explains the entire process −



To write an RMI Java application, you would have to follow the steps given below −

- Define the remote interface
- Develop the implementation class (remote object)
- Develop the server program
- Develop the client program
- Compile the application
- Execute the application

Code:

RMI_Interface:

```
package pkg_RMI;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMI_Interface extends Remote {
   void displayMessage() throws RemoteException;
   int factorial(int n) throws RemoteException;
}
```

```
package pkg_RMI;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.Scanner;
import java.rmi.NotBoundException;
import java.rmi.Naming;
public class RMI_Client {
 public static void main(String[] args) throws
MalformedURLException, RemoteException,
NotBoundException{
        try {
```

RMI_Client:

```
                RMI_Interface helloAPI = (RMI_Interface) Naming.lookup("rmi://localhost:1880/hello");
//                      helloAPI.displayMessage();
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter a number to find the factorial: ");
                int n = sc.nextInt();
                int ans = helloAPI.factorial(5);
                System.out.println("Factorial of "+n+" is "+ans);
        }
        catch(Exception e)
        {
                System.out.println("The RMI APP is not running...");
                e.printStackTrace();
        }
```

```
  }
}
```

RMI_Server:

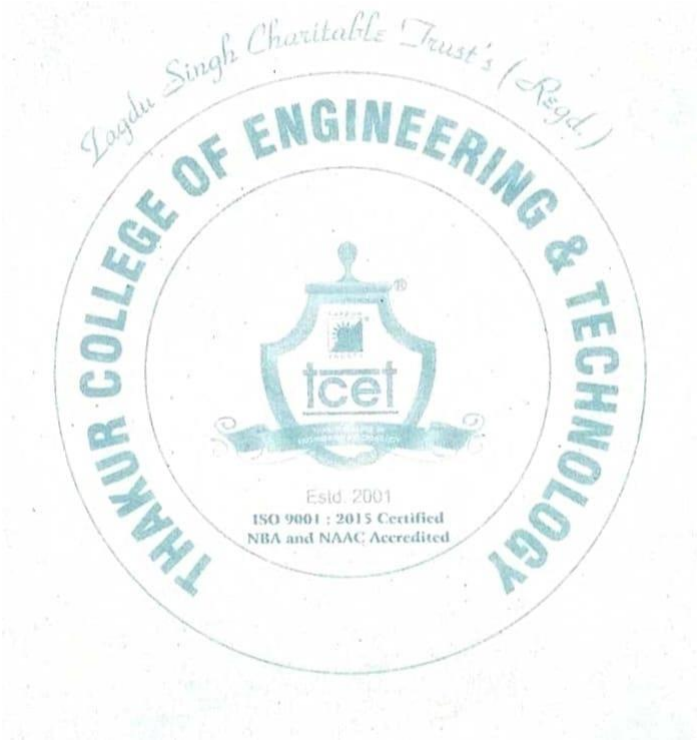
```
package pkg_RMI;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
public class RMI_Server extends UnicastRemoteObject implements RMI_Interface{
 public RMI_Server() throws RemoteException {
        super();
 }
 public static void main(String[] args)throws RemoteException, AlreadyBoundException {
        try {
                Registry registry = LocateRegistry.createRegistry(1880);
                registry.bind("hello", new RMI_Server());
                System.out.println("The RMI_Server is running and ready...");
        }
        catch (Exception e) {
           e.printStackTrace();
           System.out.println("The RMI_Server is not running...");
        }
 }
 @Override
 public void displayMessage()throws RemoteException{
        System.out.println("-----------------------------------------------");
        System.out.println("Hello Student!");
        System.out.println("-----------------------------------------------")
                                                                    }

 @Override
 public int factorial(int n)throws RemoteException{
        int fact = 1;
        for(int i=1;i<=n;i++) {
                fact *= i;
        }
        return fact;
 }
}
```

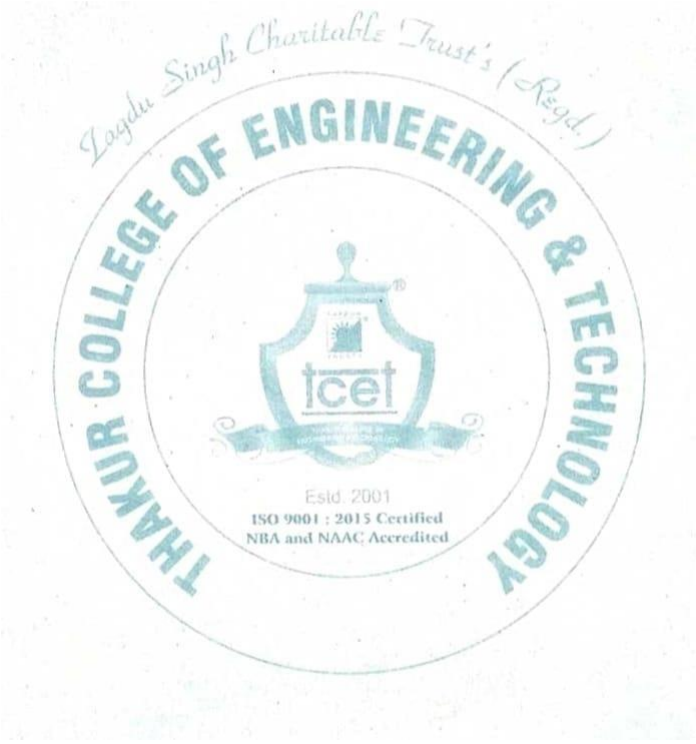
Output:

```
<terminated> RMI_Client [Java Application] C:\Program File
Enter a number to find the factorial:
5
Factorial of 5 is 120
```

For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				


Lagdu Singh Charitable Trust's (Regd.)
THAKUR COLLEGE OF ENGINEERING & TECHNOLOGY
tcet
Estd. 2001
ISO 9001 : 2015 Certified
NBA and NAAC Accredited


Yogdan Singh Charitable Trust's (Regd.)
THAKUR COLLEGE OF ENGINEERING & TECHNOLOGY
tcet
Estd. 2001
ISO 9001 : 2015 Certified
NBA and NAAC Accredited