GOVERNMENT COLLEGE OF ENGINEERING, JALGAON
(An Autonomous Institute of Government of Maharashtra)
Department of Computer Engineering

Name: Vaibhav Bharat Sontakke	PRN No: 1942207
Class: T. Y. B. Tech.	Batch: L4
Date of Performance: _____	Date of Completion: _____
Subject: DSL	Sign. of Teacher with Date: _____

Aim: Program for Distributed Application using RPC

Title: Design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

Software Requirements: Ubuntu OS, Eclipse IDE 4.11

Theory :

Remote Procedure Call

Objective:

Remote Procedure Call (**RPC**) is a protocol that one program can **use** to request a service from a program located in another computer on a network without having to understand the network's details. A procedure call is also sometimes known as a function call or a subroutine call. **RPC uses** the client-server model

A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the

required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

In distributed computing, a **remote procedure call** (**RPC**) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote.[1] This is a form of client–server interaction (caller is client, executor is server), typically implemented via a request–response message-passing system. In the object-oriented programming paradigm, RPC calls are represented by remote method invocation (RMI).

The RPC model implies a level of location transparency, namely that calling procedures is largely the same whether it is local or remote, but usually they are not identical, so local calls can be distinguished from remote calls. Remote calls are usually orders of magnitude slower and less reliable than local calls, so distinguishing them is important.

RPCs are a form of inter-process communication (IPC), in that different processes have different address spaces: if on the same host machine, they have distinct virtual address spaces, even though the physical address space is the same; while if they are on different hosts, the physical address space is different. Many different (often incompatible) technologies have been used to implement the concept.

RPC Implementation Mechanism:

- RPC mechanism uses the concepts of stubs to achieve the goal of semantic transparency.
- Stubs provide a local procedure call abstraction by concealing the underlying RPC mechanism.
- A separate stub procedure is associated with both the client and server processes.
- RPC communication package known as RPC Runtime is used on both the sides to hide existence and functionalities of a network.

- implementation of RPC involves the five elements of program:
    1. Client
    2. Client Stub
    3. RPC Runtime
    4. Server stub
    5. Server

The sequence of events in a remote procedure call are given as follows:

1. Client

- A Client is a user process which initiates a RPC
- The client makes a normal call that will invoke a corresponding procedure in the client stub.

2. Client Stub

Client stub is responsible for the following two tasks:

i. On receipt of a call request from the client, it packs specifications of the target procedureand arguments into a message and asks the local RPCRuntime to send it to the server stub.

ii. On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

3. RPCRuntime

- Transmission of messages between Client and the server machine across the network is handled by RPCRuntime.
- It performs Retransmission, Acknowledgement, Routing and Encryption.
- RPCRuntime on Client machine receives messages containing result of procedure execution from server and sends it client stub as well as the RPCRuntime on server machine receives the same message from server stub and passes it to client machine.
- It also receives call request messages from client machine and sends it to server stub.

4. Server Stub

Server stub is similar to client stub and is responsible for the following two tasks:

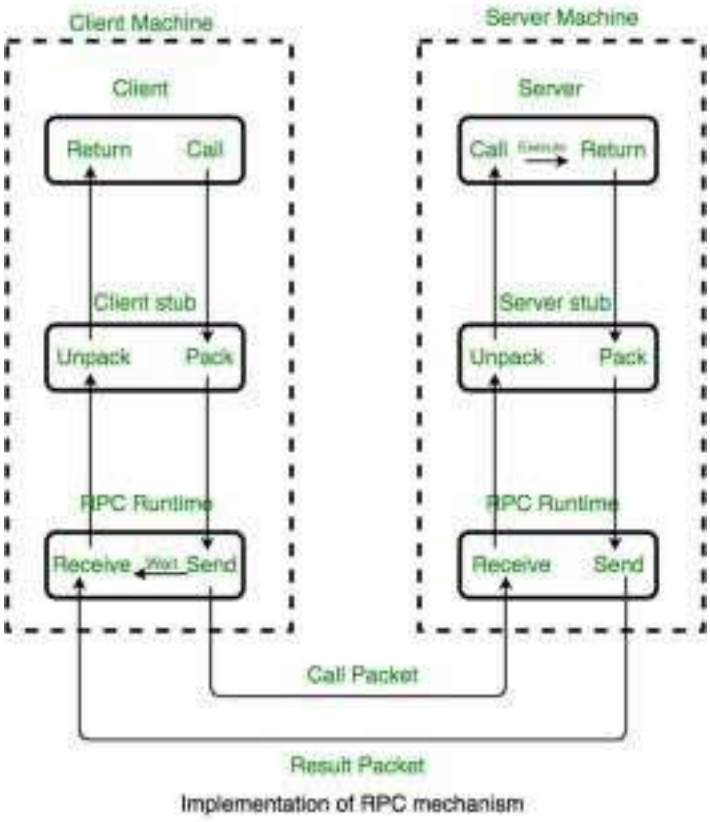
i. On receipt of a call request message from the local RPCRuntime, it unpacks and makes anormal call to invoke the required procedure in the server.

ii. ii. On receipt of the result of procedure execution from the server, it unpacks the result intoa message and then asks the local RPCRuntime to send it to the client stub.

5. Server

When a call request is received from the server stub, the server executes the required procedure and returns the result to the server stub. Thus we have successfully implemented RPC mechanism for a file transfer across a network

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message. The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.



Implementation of RPC mechanism

- **ALGORITHM:**

  Steps:

  1. Create the IDL**,** Open terminal
    - sudo apt-get adv –keyserver pgp.mit.edu –recv-keys A4A9406876FCBD3C456770C88C718D3B5072E1F5
    - sudo apt-get updates
    - sudo apt-get install rpcbind
    - mkdir exp2
    - cd exp2
    - gedit fact.x

  2. add following code in itstruct intpair { int a;
  };
  program FACT_PROG {
  version FACT_VERS {
  int FACT(intpair) = 1;
  } = 1;
  } = 0x23451111;

  3. save and exit the file
    - rpcgen -a -C fact.x
    - gedit Makefile.fact

  4. find the following line in
  the fileCFLAGS += -g
  and change it to:
  CFLAGS += -g -DRPC_SVC_FG

5. find the following line in the same fileRPCGENFLAGS = and change it to:
   RPCGENFLAGS = -C
6. save and exit the filegedit fact_client.c
7. save and exit the filegedit fact_server.c
8. save and exit the file
9. compilemake -f Makefile.fact
10. In one terminal, run:
    sudo ./fact_server
11. In another terminal, run:
    cd exp2
    $exp2/./fact_client localhost

Conclusion: In this Practical, we learnt about development of distributed application using RPC

……

Mrs. Archana Chitte

Name & Sign of Course Teacher