**ASSIGNMENT TITLE:** Design a distributed application using RPC

**AIM:** To demonstrate the use of Remote Procedure Call (RPC) using client server architecture to calculate the square of given number.

**OBJECTIVES:** To study and implement RPC and client server architecture.

**THEORY:**

Remote Procedure Call:

Basic characteristic of RPC is transparency. Transparency is of two types:

1. Syntactic Transparency: In this the RPC & LPC syntax is identical.
2. Semantic Transparency: In this RPC & LPC have same semantics.

Elements of RPC:

The basic elements of RPC code are:

1. Client
2. Client Stub
3. RPC Runtime
4. Server Stub
5. Server

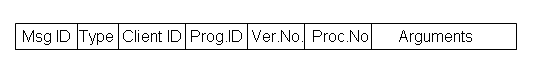
In this type of communication:

1. Client calls a remote procedure by sending packets containing parameters.
2. It is packed and sent to the client.
3. The server receives the packet, unpacks it, computes the result and sends he result back to the client

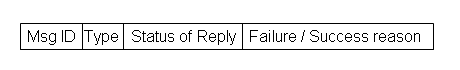
Stub Generation: Generation of strubs for client and server is of two types:

1. Manual: Provide a set of transmission functions from which user constructs his own stubs.
2. Automatic: It uses Interface Definition Language (IDL) i.e. List of procedures selected.

* RPC message format:



* RPC Reply format:



-Marshalling arguments and results:

a) Taking arguments of client and result.

b) Encoding a message.

c) Decoding the message.

**INPUT:** Number whose square is to be found.

**OUTPUT:** Square of number.

**PLATFORM:** Linux.

**CONCLUSION:** Thus, RPC has been studied and implemented on Linux platform.

**FAQs :**

1. What is the difference between RPC and LRPC?

Answer

Difference between RPC and LRPC:

RPC (Remote Procedure Call): A communication protocol that allows a program to execute procedures or functions on a remote server as if they were local. RPC typically involves marshalling parameters, sending requests over a network, executing the procedure on the server, and unmarshalling the results.

LRPC (Local Remote Procedure Call): A variant of RPC used for communication between processes on the same machine. LRPC avoids the overhead of network communication by directly invoking procedures in another process's address space.

1. What does rpcgen do?

Answer

Rpcgen:

Rpcgen is a tool used to generate code for RPC (Remote Procedure Call) implementations. It takes an interface description file as input, typically written in the RPC Language (XDR), and generates client and server stub code in C or other programming languages. rpcgen automates the process of marshalling and unmarshalling parameters, handling network communication, and defining data structures for RPC-based applications.

1. What is meant by packing and unpacking of RPC messages

Answer

Packing and unpacking of RPC Messages:

Packing: Refers to the process of converting data from its native representation in memory to a standardized format suitable for transmission over a network. This involves serializing data structures, such as converting integers to network byte order and encoding complex data types.

Unpacking: The reverse process of packing, which involves decoding received data from the network and reconstructing data structures in memory. Unpacking converts data from the standardized network format back to its native representation. Proper packing and unpacking are essential for ensuring interoperability and data integrity in RPC communication.

CODE:

IDL.X

struct values{

float num1;

float num2;

char operation;

};

/\*Programme, version and procedure definition\*/

program COMPUTE{

version COMPUTE\_VERS{

float ADD(values) =1;

float SUB(values)=2;

float MUL(values)=3;

float DIV(values)=4;

float POW(values)=5;

} =7;

} = 456123789;

IDL\_client.c

#include "IDL.h"

#include <stdio.h>

*float* compute\_7(*char* \**host*, *float* *a*, *float* *b*, *char* *op*)

{

    CLIENT \*clnt;

*float* \*result\_1;

    values add\_7\_arg;

*float* \*result\_2;

    values sub\_7\_arg;

*float* \*result\_3;

    values mul\_7\_arg;

*float* \*result\_4;

    values div\_7\_arg;

*float* \*result\_5;

    values pow\_7\_arg;

    if (*op* == '+')

    {

        add\_7\_arg.num1 = *a*;

        add\_7\_arg.num2 = *b*;

        add\_7\_arg.operation = *op*;

        clnt = clnt\_create(*host*, COMPUTE, COMPUTE\_VERS, "udp");

        if (clnt == NULL)

        {

            clnt\_pcreateerror(*host*);

            exit(1);

        }

        result\_1 = add\_7(&add\_7\_arg, clnt);

        if (result\_1 == (*float* \*)NULL)

        {

            clnt\_perror(clnt, "call failed");

        }

        clnt\_destroy(clnt);

        return (\*result\_1);

    }

    else if (*op* == '-')

    {

        sub\_7\_arg.num1 = *a*;

        sub\_7\_arg.num2 = *b*;

        sub\_7\_arg.operation = *op*;

        clnt = clnt\_create(*host*, COMPUTE, COMPUTE\_VERS, "udp");

        if (clnt == NULL)

        {

            clnt\_pcreateerror(*host*);

            exit(1);

        }

        result\_2 = sub\_7(&sub\_7\_arg, clnt);

        if (result\_2 == (*float* \*)NULL)

        {

            clnt\_perror(clnt, "call failed");

        }

        clnt\_destroy(clnt);

        return (\*result\_2);

    }

    else if (*op* == '\*')

    {

        mul\_7\_arg.num1 = *a*;

        mul\_7\_arg.num2 = *b*;

        mul\_7\_arg.operation = *op*;

        clnt = clnt\_create(*host*, COMPUTE, COMPUTE\_VERS, "udp");

        if (clnt == NULL)

        {

            clnt\_pcreateerror(*host*);

            exit(1);

        }

        result\_3 = mul\_7(&mul\_7\_arg, clnt);

        if (result\_3 == (*float* \*)NULL)

        {

            clnt\_perror(clnt, "call failed");

        }

        clnt\_destroy(clnt);

        return (\*result\_3);

    }

    else if (*op* == '/')

    {

        if (*b* == 0)

        {

            printf("You are trying to divide by zero. Please insert a valid number.\n");

            exit(0);

        }

        else

        {

            div\_7\_arg.num1 = *a*;

            div\_7\_arg.num2 = *b*;

            div\_7\_arg.operation = *op*;

            clnt = clnt\_create(*host*, COMPUTE, COMPUTE\_VERS, "udp");

            if (clnt == NULL)

            {

                clnt\_pcreateerror(*host*);

                exit(1);

            }

            result\_4 = div\_7(&div\_7\_arg, clnt);

            if (result\_4 == (*float* \*)NULL)

            {

                clnt\_perror(clnt, "call failed");

            }

            clnt\_destroy(clnt);

            return (\*result\_4);

        }

    }

    else if (*op* == '^')

    {

        pow\_7\_arg.num1 = *a*;

        pow\_7\_arg.num2 = *b*;

        pow\_7\_arg.operation = *op*;

        clnt = clnt\_create(*host*, COMPUTE, COMPUTE\_VERS, "udp");

        if (clnt == NULL)

        {

            clnt\_pcreateerror(*host*);

            exit(1);

        }

        result\_5 = pow\_7(&pow\_7\_arg, clnt);

        if (result\_5 == (*float* \*)NULL)

        {

            clnt\_perror(clnt, "call failed");

        }

        clnt\_destroy(clnt);

        return (\*result\_5);

    }

    else

    {

        printf("Invalid operation. Please insert a valid operation.\n");

        exit(0);

    }

}

*int* main(*int* *argc*, *char* \**argv*[])

{

*char* \*host;

*float* number1, number2;

*char* oper;

    printf("Enter the 2 numbers followed by the operation to perform:\n");

    scanf("%f", &number1);

    scanf("%f", &number2);

    scanf("%s", &oper);

    host = *argv*[1];

    printf("Answer= %f\n", compute\_7(host, number1, number2, oper));

    exit(0);

}

IDL\_server.c

#include "IDL.h"

#include <stdio.h>

*float* \*

add\_7\_svc(values \**argp*, *struct* svc\_req \**rqstp*)

{

    static *float*  result;

    result = *argp*->num1 + *argp*->num2;

    return &result;

}

*float* \*

sub\_7\_svc(values \**argp*, *struct* svc\_req \**rqstp*)

{

    static *float*  result;

    result = *argp*->num1 - *argp*->num2;

    return &result;

}

*float* \*

mul\_7\_svc(values \**argp*, *struct* svc\_req \**rqstp*)

{

    static *float*  result;

    result = *argp*->num1 \* *argp*->num2;

    return &result;

}

*float* \*

div\_7\_svc(values \**argp*, *struct* svc\_req \**rqstp*)

{

    static *float*  result;

    result = *argp*->num1 / *argp*->num2;

    return &result;

}

*float* \*

pow\_7\_svc(values \**argp*, *struct* svc\_req \**rqstp*)

{

    static *float*  result;

    for (*int* i=0; i<*argp*->num2; i++)

    {

        result = result \* *argp*->num1;

    }

    return &result;

}

OUTPUT:

