**DISTRIBUTED COMPONENTS LAB**

**PRELAB – 2**

**1. RPC is not suitable to pass pointers to remote procedures. Justify**

* Values like integers, Booleans are easy since they can be copied into the message and sent without problems.
* When we need to pass pointers, we will have a confusion whether the pointer is to be passed or the value. What happens if the pointer is pointing to something in the middle of a complex structure with other pointers inside? Here a new problem arises whether we need to send the whole structure? Or the parties should interact with each other asking for values from each pointer the remote function uses?
* Since we have so many conflicts when we pass pointers to remote procedures, RPC is not suitable.

**2. Differentiate between SUN RPC and MS RPC:**

|  |  |
| --- | --- |
| **SUN RPC** | **MS RPC** |
| Originally developed by Sun, but now widely available on other platforms (including Digital Unix). Also known as Open Network Computing (ONC). | MSRPC was originally derived from open source software but has been developed further and copyrighted by Microsoft. |
| Sun RPC package has an RPC compiler (rpcgen) that automatically generates the client and server stubs. | The client stub code retrieves the required parameters from the client address space and delivers them to the client runtime library, which then translates the parameters into a standard Network Data Representation format to transmit to the server. |
| RPC package uses XDR (eXternal Data Representation) to represent data sent between client and server stubs. | The client stub then calls functions in the RPC client runtime library to send the request and parameters to the server.  If the server is located remotely, the runtime library specifies an appropriate transport protocol and engine and passes the RPC to the network stack for transport to the server. |

**3. Differentiate between Socket and RPC**

|  |  |
| --- | --- |
| **Socket** | **RPC** |
| Also commonly referred to as a "socket", used to describe an IP address and port, is a handle to a communication chain that can be used to communicate between different virtual machines or different computers. | RPC (Remote Procedure Call) is built on top of Socket. For an analogous desire, the main program running on one machine can call the subroutine prepared on another machine. |
| Each service opens a Socket and is bound to a port, with different ports corresponding to different services. | The basic principle of the RPC method is to ignore the specific details of the communication with the simplicity of the module call. |
| Sockets could be used to make the connection to the server | RPC is implemented on the basis of Socket. |

**4. Write down the functionalities of stub**

RMI (Remote Method Invocation) uses stub and skeleton object for communication with the remote object.

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

**5. How does the procedure-oriented middleware differ from Object oriented middleware? Give an example**

|  |  |
| --- | --- |
| **Procedure Oriented Middleware** | **Object Oriented Middleware** |
| Procedure-oriented middleware is characterized by a client converting the parameters of a procedure into a message, sending this message to the server (or host), who then converts the message back into the parameters. | Object-Oriented Middleware evolved more or less directly from the idea of remote procedure calls |
| Good examples of this type of middleware are Open Network Computing (ONC) from Sun Microsystems, Inc. and Distributed Computing Environment (DCE) from Open Software Foundation (OSF). | The first of these systems was OMG’s Common Object Request Broker Architecture (CORBA). |
| It supports exceptions by returning a message that a failure occurred. | The idea here is to make object-oriented principles available for the development of distributed systems. |

**6. List out all the issues in RPC and also mention how it is resolved**

**i) Single Threaded Servers:**

RPC “forces” the choice of a multi-threaded server instead of single threaded. This happens because the RPC model doesn’t allow the server to return without serving a response to the client. If the data requested by the client is not immediately available the server has to wait and can’t start serving new requests. The most obvious choice then is to make the server multi-threaded.

**Solution:**

Nowadays is hard, although not impossible, to imagine single-threaded servers being built.  Finagle is built around services which are asynchronous functions and gRPC offers both synchronous and asynchronous function calls.

**ii) The Two Army Problem**

This is also known as “Two Generals’ Problem” and it states that is impossible for two processes to agree on a decision over an unreliable network.

**Solution:**

This might be the main reason for timeouts when using RPC. Since the system can’t know that the other party received the message and it could wait indefinitely for a response it chooses to timeout and then act properly by canceling the request, as seen above, retrying after some time or some other approach suitable for the application in questions.

**iii) Exception Handling**

When a procedure is executed locally it either completes or fails entirely. Remote procedures introduce new errors regarding the communication over the network and also when one party fails.

**Solution:**

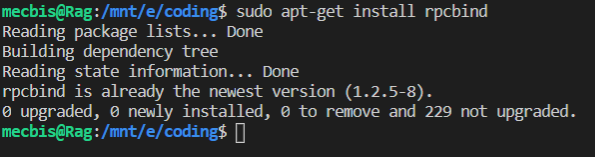
The communication should be stable so that the procedure calls will not introduce any new errors.

**7. What is marshalling / unmarshalling?**

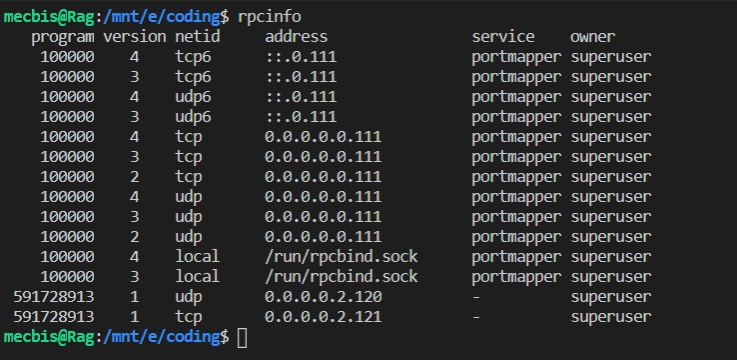
* **Marshalling** is the process of transforming the memory representation of an object to a data format suitable for the storage and transmission.
* **Unmarshalling** refers to the process of transforming a representation of an object that is used for storage or transmission to a representation of the object that is executable.

**8. Execute a simple calculator program using rpcgen**

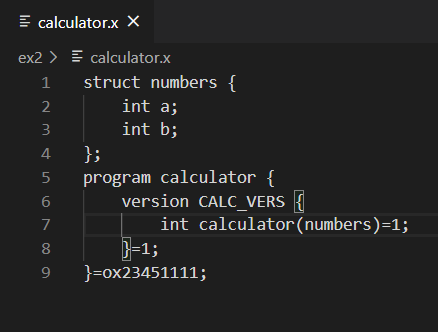
**Step 1:** The rpcbind package is installed in the first place.



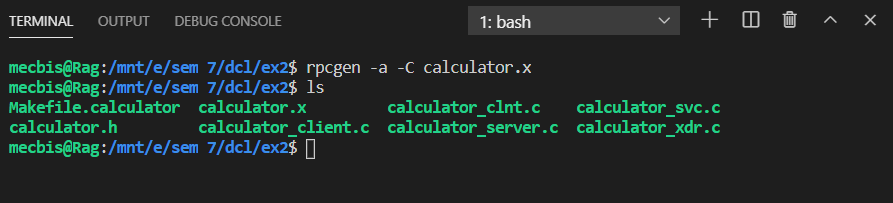
**Step 2:** Check if the package is working by using rpcinfo command



**Step 3:** Create a file with .x extension for the calculator program.

****

**Step 4:** Use the rpcgen command to generate the required files according to the calculator.x file

****

**Step 5:** Once the editing is done for the calculator\_client.c and calculator\_server.c, they are compiled using the make command.

**CALCULATOR\_SERVER.C:**

#include "calculator.h"

int \*

calculator\_1\_svc(numbers \*argp, struct svc\_req \*rqstp)

{

    static int  result;

    printf("The given choice is: %c\n",argp->c);

    printf("The given numbers are: %d %d\n",argp->a,argp->b);

    if(argp->c=='+') {

        result = argp->a + argp->b;

    }

    else if(argp->c=='-') {

        result = argp->a - argp->b;

    }

    else if(argp->c=='\*') {

        result = argp->a \* argp->b;

    }

    else {

        result = argp->a / argp->b;

    }

    return &result;

}

**CALCULATOR\_CLIENT.C:**

#include "calculator.h"

void

calculator\_prog\_1(char \*host,int num1,int num2,char operator)

{

    CLIENT \*clnt;

    int  \*result\_1;

    numbers  calculator\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, CALCULATOR\_PROG, CALC\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif  /\* DEBUG \*/

    calculator\_1\_arg.a = num1;

    calculator\_1\_arg.b = num2;

    calculator\_1\_arg.c = operator;

    result\_1 = calculator\_1(&calculator\_1\_arg, clnt);

    if (result\_1 == (int \*) NULL) {

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

    }

    else

    {

        printf("The result is: %d\n",\*result\_1);

    }

#ifndef DEBUG

    clnt\_destroy (clnt);

#endif   /\* DEBUG \*/

}

int

main (int argc, char \*argv[])

{

    char \*host;

    if (argc < 5) {

        printf ("usage: %s server\_host\n", argv[0]);

        exit (1);

    }

    host = argv[1];

    int num1 = atoi(argv[2]);

    int num2 = atoi(argv[3]);

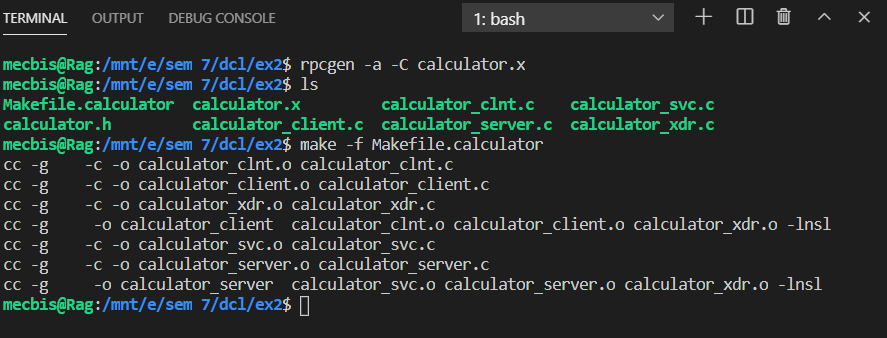
    char operator = argv[4][0];

    calculator\_prog\_1 (host,num1,num2,operator);

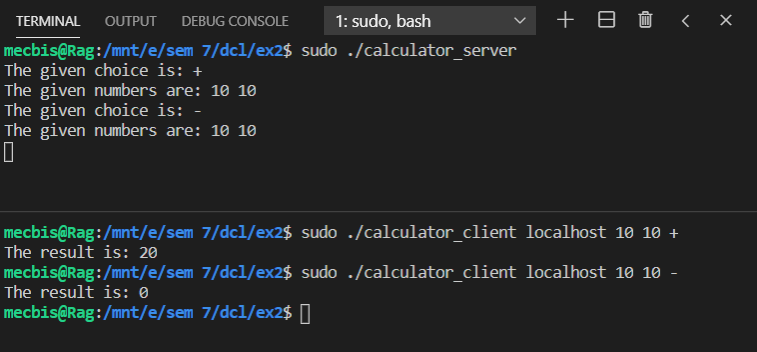
exit (0);

}

**Output:**

****

**Step 6:** Run the calculator\_server and calculator\_client to perform the calculator operations.

****

**Result:**

The Remote Procedure Call is understood successfully with the help of a program using rpcgen.