Fr. Conceicao Rodrigues College of Engineering

Department of Computer Engineering

Academic Year 2022-23

Distributed Computing Lab (B.E. Computer Engineering)

LAB 2

Aim: To implement Remote Procedure Call

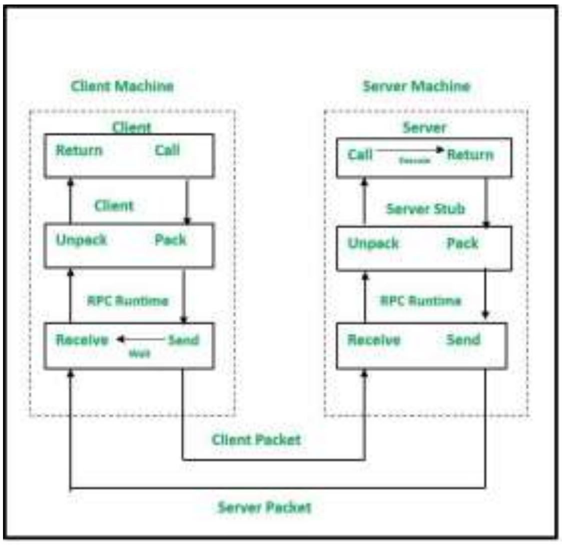
Lab Outcome:
Develop test and debug using Message-Oriented Communication or RPC/RMI based clientserver programs

Theory:

RPC is an effective mechanism for building client-server systems that are distributed. RPC enhances the power and ease of programming of the client/server computing concept. It is a protocol that allows one software to seek a service from another program on another computer in a network without having to know about the network. The software that makes the request is called a client, and the program that provides the service is called a server.

There are 5 elements used in the working of RPC:

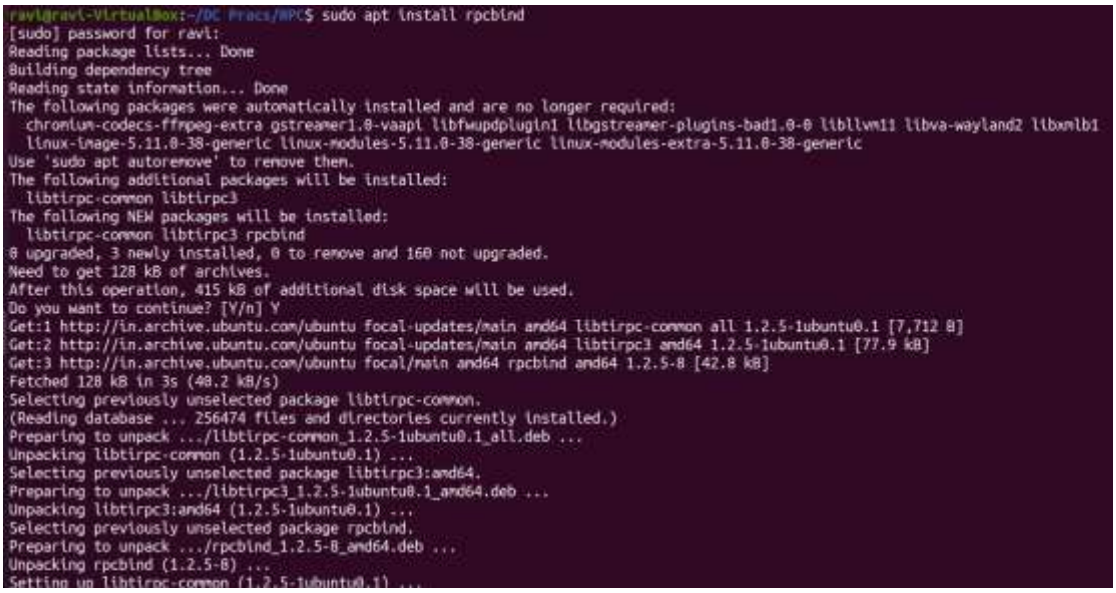
- Client
- Client Stub
- RPC Runtime
- Server Stub
- Server



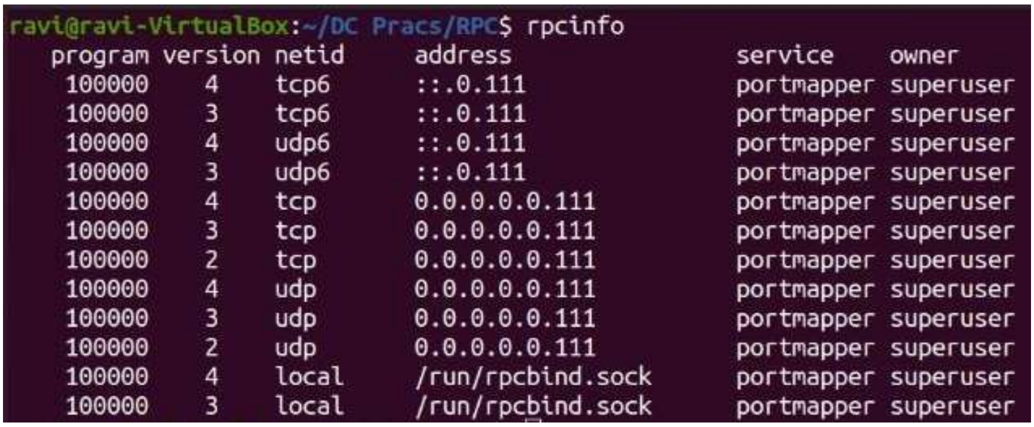
- The client, the client stub, and one instance of RPC Runtime are all running on the client machine.

- A client initiates a client stub process by giving parameters as normal. The client stub acquires storage in the address space of the client.
- At this point, the user can access RPC by using a normal Local Procedural Call. The RPC runtime oversees message transmission between client and server via the network. Retransmission, acknowledgment, routing, and encryption are all tasks performed by it.
- On the server-side, values are returned to the server stub, after the completion of server operation, which then packs (which is also known as marshalling) the return values into a message. The transport layer receives a message from the server stub.
- The resulting message is transmitted by the transport layer to the client transport layer, which then sends a message back to the client stub.
- The client stub unpacks (which is also known as unmarshalling) the return arguments in the resulting packet, and the execution process returns to the caller at this point.

Code & Output:



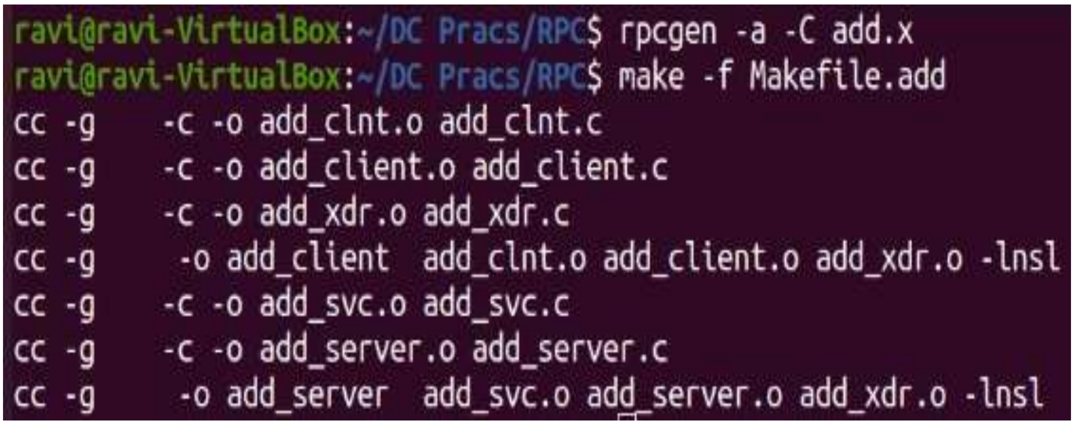
```
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ sudo apt install rpcbind
[sudo] password for ravi:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi libfwupdplugin1 libgstreamer-plugins-bad1.0-0 libllvm11 libva-wayland2 libxmlb1
  linux-image-5.11.0-38-generic linux-modules-5.11.0-38-generic linux-modules-extra-5.11.0-38-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libtirpc-common libtirpc3
The following NEW packages will be installed:
  libtirpc-common libtirpc3 rpcbind
0 upgraded, 3 newly installed, 0 to remove and 160 not upgraded.
Need to get 128 kB of archives.
After this operation, 415 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 libtirpc-common all 1.2.5-1ubuntu0.1 [7,712 B]
Get:2 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 libtirpc3 amd64 1.2.5-1ubuntu0.1 [77.9 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu focal/main amd64 rpcbind amd64 1.2.5-8 [42.8 kB]
Fetched 128 kB in 3s (40.2 kB/s)
Selecting previously unselected package libtirpc-common.
(Reading database ... 256474 files and directories currently installed.)
Preparing to unpack .../libtirpc-common_1.2.5-1ubuntu0.1_all.deb ...
Unpacking libtirpc-common (1.2.5-1ubuntu0.1) ...
Selecting previously unselected package libtirpc3:amd64.
Preparing to unpack .../libtirpc3_1.2.5-1ubuntu0.1_amd64.deb ...
Unpacking libtirpc3:amd64 (1.2.5-1ubuntu0.1) ...
Selecting previously unselected package rpcbind.
Preparing to unpack .../rpcbind_1.2.5-8_amd64.deb ...
Unpacking rpcbind (1.2.5-8) ...
Setting up libtirpc-common (1.2.5-1ubuntu0.1) ...
```



```
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ rpcinfo
   program version netid     address                service    owner
    100000    4    tcp6      ::.0.111               portmapper superuser
    100000    3    tcp6      ::.0.111               portmapper superuser
    100000    4    udp6      ::.0.111               portmapper superuser
    100000    3    udp6      ::.0.111               portmapper superuser
    100000    4    tcp       0.0.0.0.0.111          portmapper superuser
    100000    3    tcp       0.0.0.0.0.111          portmapper superuser
    100000    2    tcp       0.0.0.0.0.111          portmapper superuser
    100000    4    udp       0.0.0.0.0.111          portmapper superuser
    100000    3    udp       0.0.0.0.0.111          portmapper superuser
    100000    2    udp       0.0.0.0.0.111          portmapper superuser
    100000    4    local     /run/rpcbind.sock      portmapper superuser
    100000    3    local     /run/rpcbind.sock      portmapper superuser
```

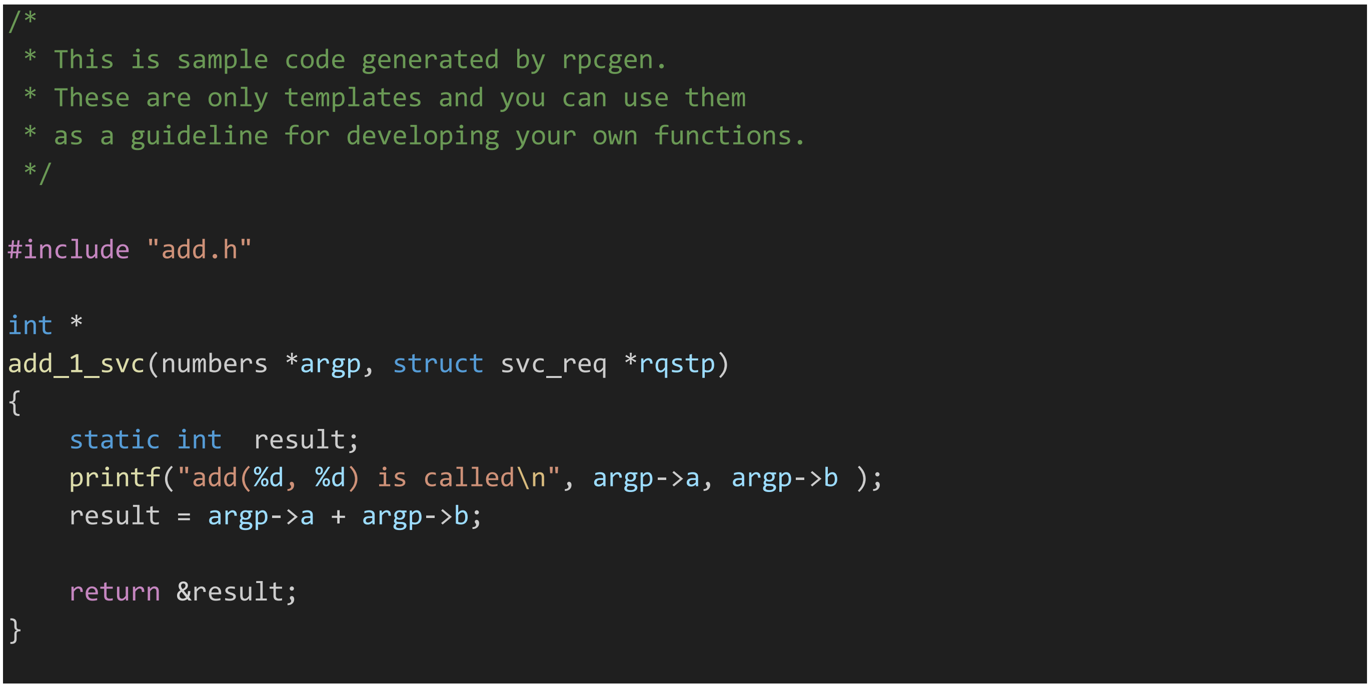
add.x

```
struct numbers{
    int a;
    int b;
};

program ADD_PROG{
    version ADD_VERS{
        int add(numbers)=1;
    }=1;
}=0x23451111;
```



```
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ rpcgen -a -C add.x
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ make -f Makefile.add
cc -g    -c -o add_clnt.o add_clnt.c
cc -g    -c -o add_client.o add_client.c
cc -g    -c -o add_xdr.o add_xdr.c
cc -g     -o add_client  add_clnt.o add_client.o add_xdr.o -lnsl
cc -g    -c -o add_svc.o add_svc.c
cc -g    -c -o add_server.o add_server.c
cc -g     -o add_server  add_svc.o add_server.o add_xdr.o -lnsl
```

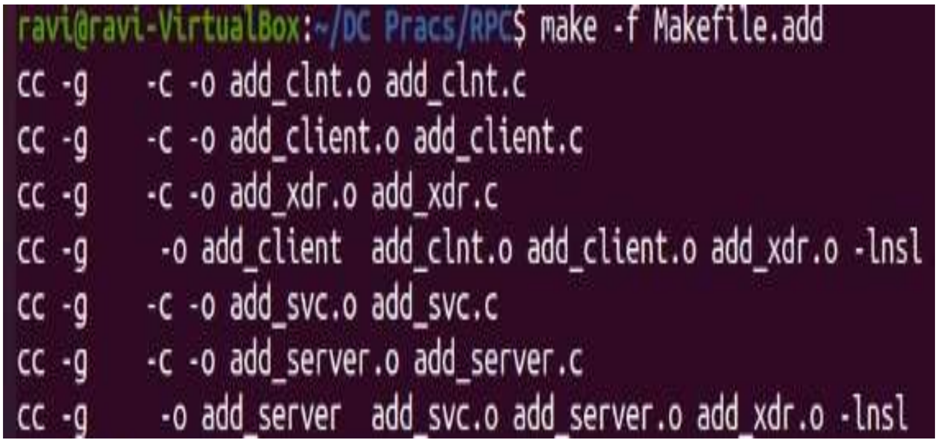
add_server.c



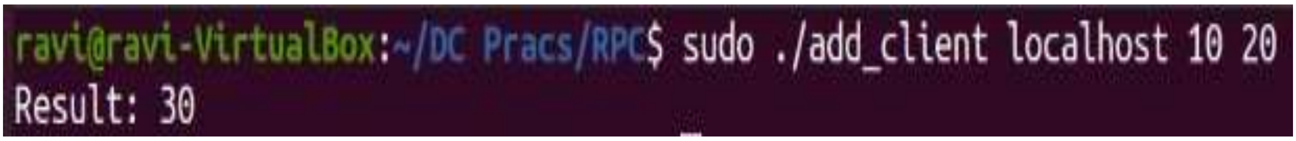
```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "add.h"

int *
add_1_svc(numbers *argp, struct svc_req *rqstp)
{
    static int  result;
    printf("add(%d, %d) is called\n", argp->a, argp->b );
    result = argp->a + argp->b;

    return &result;
}
```

add_client.c

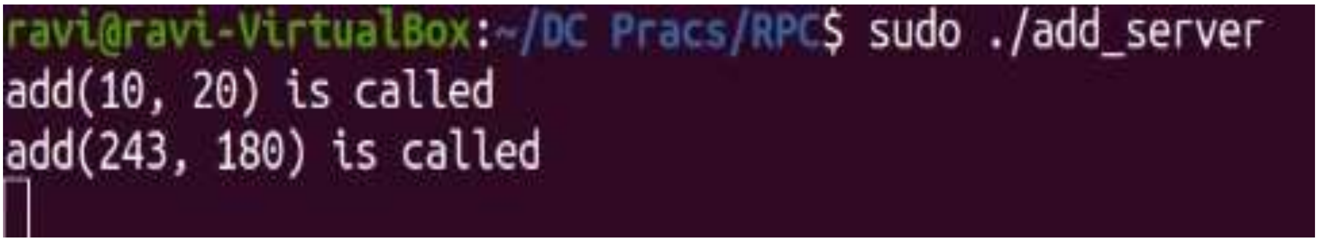
```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "add.h"


void
add_prog_1(char *host,int x, int y)
{
    CLIENT *clnt;
    int  *result_1;
    numbers add_1_arg;

#ifndef DEBUG
    clnt = clnt_create(host, ADD_PROG, ADD_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif  /* DEBUG */
    add_1_arg.a = x;
    add_1_arg.b = y;
    result_1 = add_1(&add_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    else{
        printf("Result: %d\n", *result_1);
    }
#ifndef DEBUG
    clnt_destroy (clnt);
#endif   /* DEBUG */
}


int
main (int argc, char *argv[])
{
    char *host;

    if (argc < 4) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    add_prog_1 (host, atoi(argv[2]), atoi(argv[3]));
exit (0);
}
```



```
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ make -f Makefile.add
cc -g    -c -o add_clnt.o add_clnt.c
cc -g    -c -o add_client.o add_client.c
cc -g    -c -o add_xdr.o add_xdr.c
cc -g     -o add_client  add_clnt.o add_client.o add_xdr.o -lnsl
cc -g    -c -o add_svc.o add_svc.c
cc -g    -c -o add_server.o add_server.c
cc -g     -o add_server  add_svc.o add_server.o add_xdr.o -lnsl
```



```
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ sudo ./add_client localhost 10 20
Result: 30
```



```
ravi@ravi-VirtualBox:~/DC Pracs/RPC$ sudo ./add_server
add(10, 20) is called
add(243, 180) is called
```

Conclusions:

In conclusion, the remote procedure call (RPC) is a powerful technology that enables communication between processes running on different machines in a networked environment. The experiment performed on RPC in C language has demonstrated its ability to enable distributed computing across different machines.

Postlab Questions:

1. In which category of communication, RPC be included?
2. What are stubs? What are the different ways of stub generation?
3. What is binding?
4. Name the transparencies achieved through stubs