

Assignment 2

RPC principles - develop a simple distributed computing environment consisting of a multiple Clients and a Server

Submitted By-

Nahida Sultana Chowdhury

Student ID: 2000189256

Date: 11/16/2017

1. Introduction

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.

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.

RPCGEN is an interface generator pre-compiler for Sun Microsystems ONC RPC. It uses an interface definition file to create client and server stubs in C. creates stubs based on information contained within an IDL file. This file is written in a language called RPCL - remote procedure call language. This language closely mimics C in style, and is designed purely for defining specification to be used for ONC RPC [2].

In this assignment, we supposed to implement the RPC principles in a distributed computing environment that supports multiple clients and server. The system is based on *rpcgen* utility that supports the following functions by server to multi clients and they will concurrently request various functions:

1. Returns the current path
2. Accepts an unsorted list and returns its sorted version
3. Returns whatever a Client sends as an input
4. Accepts a file name and checks if a file is available in the current directory or not and displays a message accordingly
5. Accepts two integer matrices and returns their multiplication

2. System Overview

The Principle of RPC being close to LPC makes easy for programmer to use existing language and package. RPC structure is based on Stubs provided in Fig. 1.

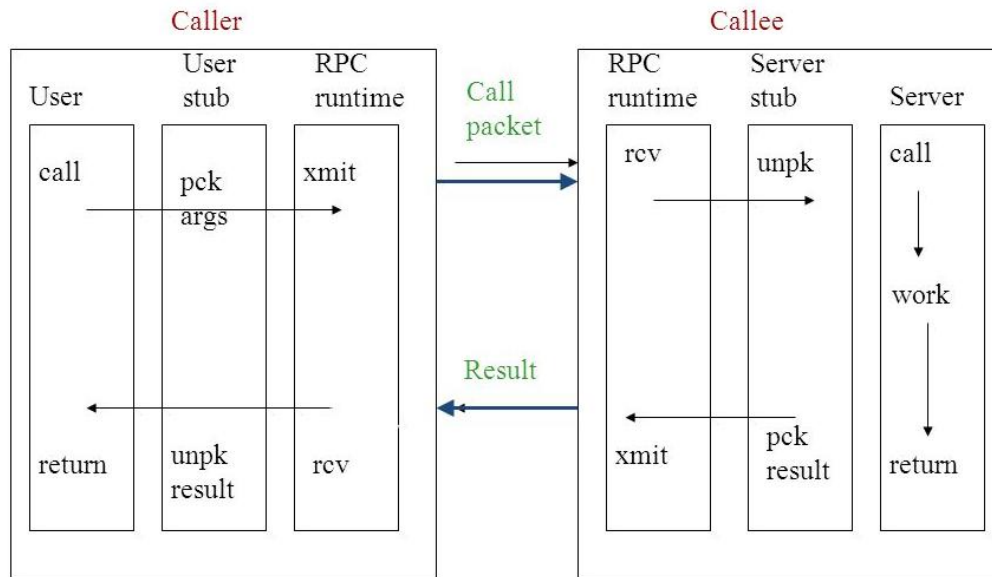


Figure 1: RPC Structure

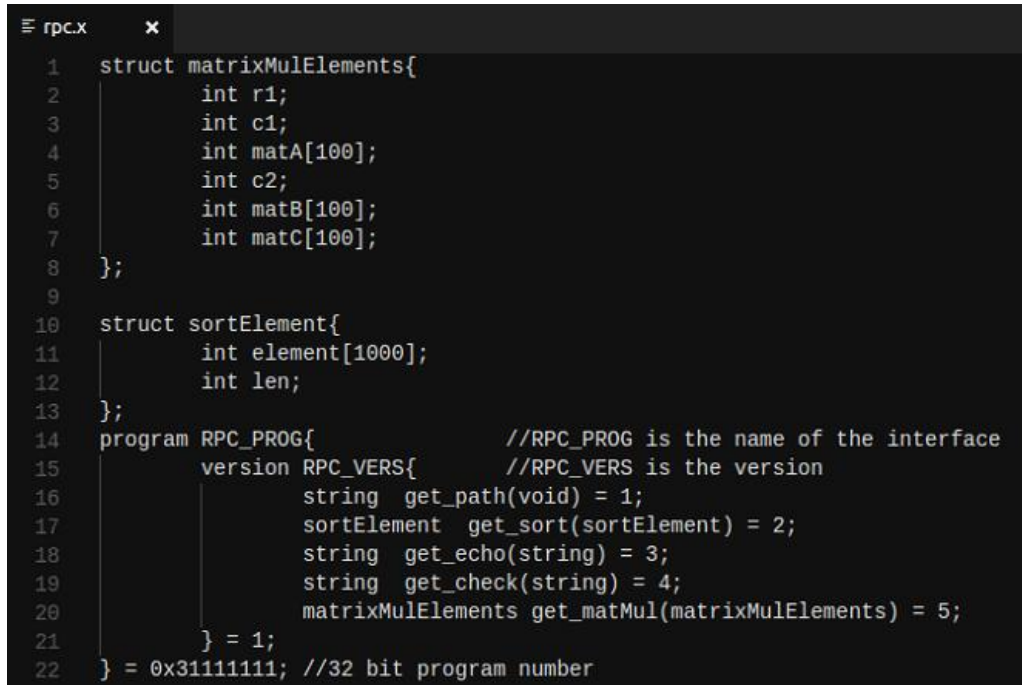
Remote Call involves 5 pieces of program are:

- User
- User Stub
- RPC Communication Package
- Server Stub
- Server

Caller Machine has user, user-stub and RPCRuntime. Similar way *Callee Machine* has server, server-stub and RPCRuntime. User makes perfectly normal local call and it invokes corresponding procedure in the user stub. Then user stub places target procedure specifications and arguments together into packet/s. RPCRuntime in Caller machine passes them to server stub. Similar process is executed at Callee machine. Meanwhile, calling process in caller machine is suspended in waiting for the results from sever. Here, RPCRuntime is responsible for retransmission, acknowledgements, packets routing and encryption.

3. Implementation and Results

Distributed application development, using *rpcgen*, starts with designing the Interface Definition Language file. This file contains the details about the services that run on different autonomous computers in the distributed system. The file has to be saved with a **.x** extension like **rpc.x**, as in this case.

A screenshot of a code editor showing the contents of the file 'rpc.x'. The editor has a dark background with light-colored text. The file name 'rpc.x' is visible in the top left corner. The code is as follows:

```
1 struct matrixMulElements{
2     int r1;
3     int c1;
4     int matA[100];
5     int c2;
6     int matB[100];
7     int matC[100];
8 };
9
10 struct sortElement{
11     int element[1000];
12     int len;
13 };
14 program RPC_PROG{           //RPC_PROG is the name of the interface
15     version RPC_VERS{       //RPC_VERS is the version
16         string get_path(void) = 1;
17         sortElement get_sort(sortElement) = 2;
18         string get_echo(string) = 3;
19         string get_check(string) = 4;
20         matrixMulElements get_matMul(matrixMulElements) = 5;
21     } = 1;
22 } = 0x31111111; //32 bit program number
```

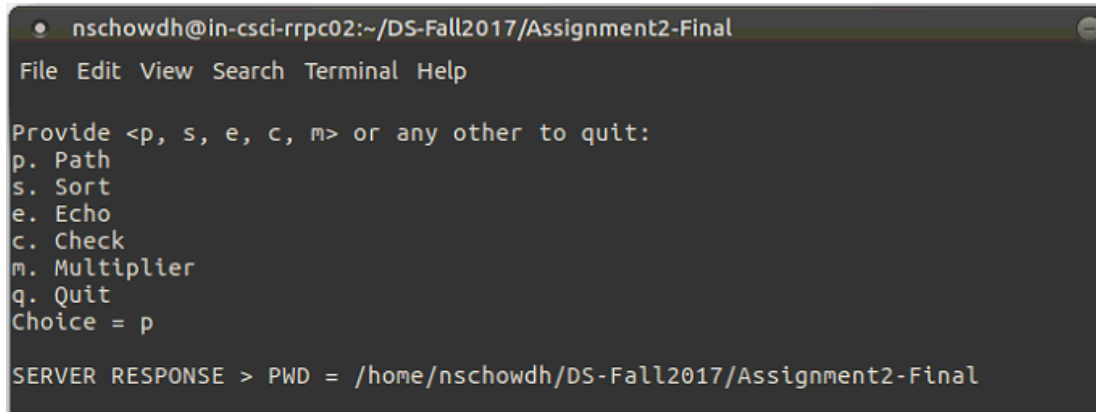
Figure 2: Interface definition file - **rpc.x**

The next step is to compile the *rpc.x* file *rpcgen*. There are several compilers flag exists that passed with *rpcgen* during compile time. However the command that we used for our experiment is “**\$rpcgen -C -a rpc.x**”; where **-C** option is used to generate the ANSI-C code and **-a** option generate the all templates file. It creates the C files for both client (*rpc_client.c*) and server (*rpc_server.c*) applications; their corresponding stub files (*rpc_clnt.c*, *rpc_svc.c*); *rpc.h* file and Makefile (*Makefile.rpc*).

From the client side clients are free to request any function for any number of times to execute to server. The loop will continue until the client doesn't want to terminate. In this application to **quit** client should provide 'q' or any other character excluding 'p', 's', 'e', 'c', and 'm'. The server will listen always, will never terminate. To force terminate server please press **clt+c**.

a) Function 1: Returns the current path

Client will request to server to get the current working directory and in return server will return the full pathname of the current directory. The client side screenshot is given below:



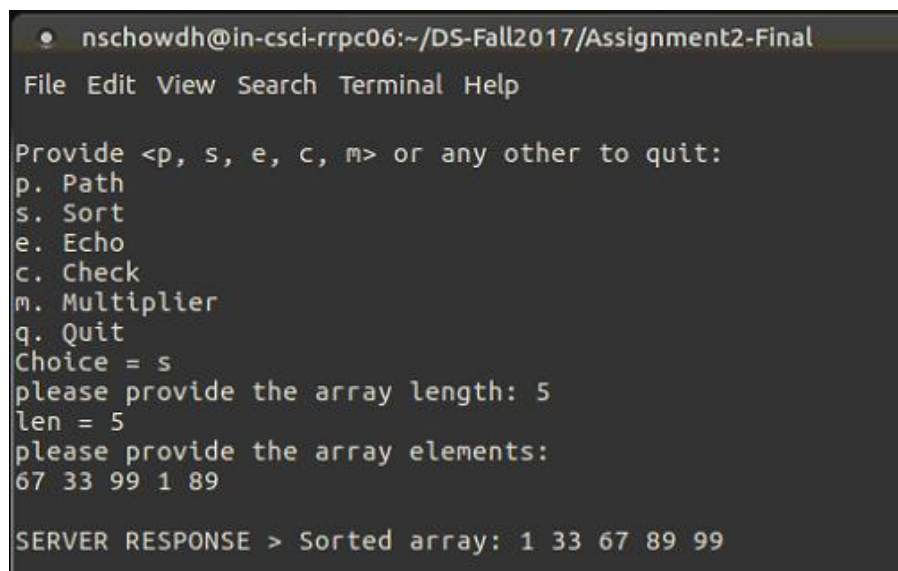
```
nschowdh@in-csci-rrpc02:~/DS-Fall2017/Assignment2-Final
File Edit View Search Terminal Help

Provide <p, s, e, c, m> or any other to quit:
p. Path
s. Sort
e. Echo
c. Check
m. Multiplier
q. Quit
Choice = p

SERVER RESPONSE > PWD = /home/nschowdh/DS-Fall2017/Assignment2-Final
```

b) Function 2: Accepts an unsorted list and returns its sorted version

Form the client side user will provide the input array that will ask user to provide the length of array and the elements. Then the array sends to server. Server accepts the unsorted array and after applying sorting mechanism then again sends back to client. Here bubble sort is implemented for sorting on server side. The client side screenshot is given below:



```
nschowdh@in-csci-rrpc06:~/DS-Fall2017/Assignment2-Final
File Edit View Search Terminal Help

Provide <p, s, e, c, m> or any other to quit:
p. Path
s. Sort
e. Echo
c. Check
m. Multiplier
q. Quit
Choice = s
please provide the array length: 5
len = 5
please provide the array elements:
67 33 99 1 89

SERVER RESPONSE > Sorted array: 1 33 67 89 99
```

c) Function 3: Returns whatever a Client sends as an input

Take input from user and send it to server and server response the same sent by client side. The client side screenshot is given below:

```
nschowdh@in-csci-rrpc02:~/DS-Fall2017/Assignment2-Final
File Edit View Search Terminal Help

Provide <p, s, e, c, m> or any other to quit:
p. Path
s. Sort
e. Echo
c. Check
m. Multiplier
q. Quit
Choice = e
Provide echo msg: RPC principles

SERVER RESPONSE > Echo msg: RPC principles
```

d) Function 4: Accepts a file name and checks if a file is available in the current directory

From the client side user will provide a file name to server and server will check if that file is available in the current directory or not and displays a message accordingly. Such if the file exists then server will return “File Exists” and otherwise “File Doesn’t Exist”. The client side screenshot is given below for both cases:

```
nschowdh@in-csci-rrpc02:~/DS-Fall2017/Assignment2-Final
File Edit View Search Terminal Help

Provide <p, s, e, c, m> or any other to quit:
p. Path
s. Sort
e. Echo
c. Check
m. Multiplier
q. Quit
Choice = c
Provide file name: rpc.x

SERVER RESPONSE > File Exists

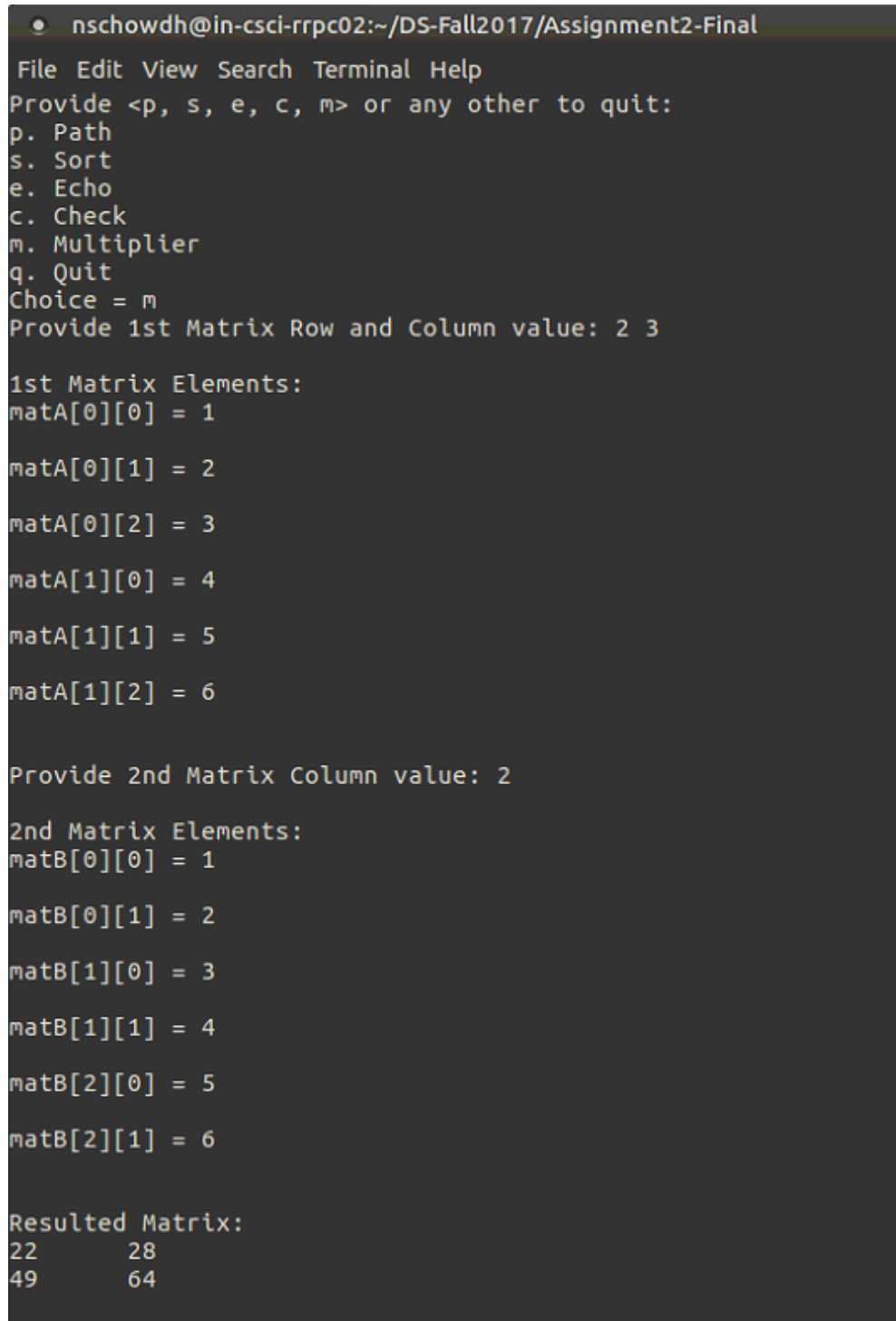
Provide <p, s, e, c, m> or any other to quit:
p. Path
s. Sort
e. Echo
c. Check
m. Multiplier
q. Quit
Choice = c
Provide file name: test.txt

SERVER RESPONSE > File Doesn't Exist
```

e) Function 5: Accepts two integer matrices and returns their multiplication

Take two matrices from user and send it to server and then server response the result of multiplication.

The client side screenshot is given below:



```
nschowdh@in-csci-rrpc02:~/DS-Fall2017/Assignment2-Final
File Edit View Search Terminal Help
Provide <p, s, e, c, m> or any other to quit:
p. Path
s. Sort
e. Echo
c. Check
m. Multiplier
q. Quit
Choice = m
Provide 1st Matrix Row and Column value: 2 3

1st Matrix Elements:
matA[0][0] = 1

matA[0][1] = 2

matA[0][2] = 3

matA[1][0] = 4

matA[1][1] = 5

matA[1][2] = 6

Provide 2nd Matrix Column value: 2

2nd Matrix Elements:
matB[0][0] = 1

matB[0][1] = 2

matB[1][0] = 3

matB[1][1] = 4

matB[2][0] = 5

matB[2][1] = 6

Resulted Matrix:
22      28
49      64
```

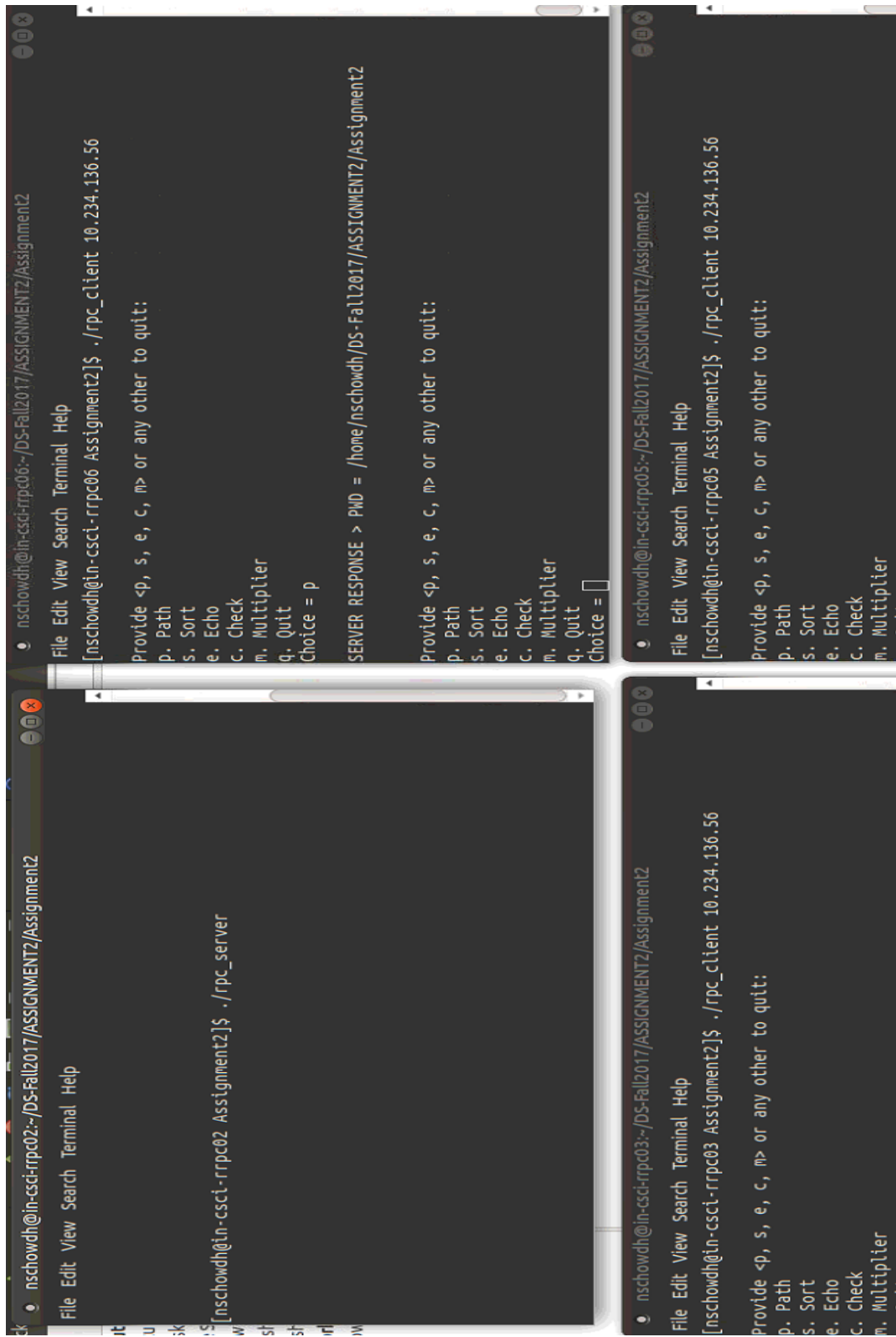


Figure 3: Multiple Clients

4. Conclusion

Server procedures can be written in any language that observes procedure-calling conventions. These procedures are linked with the server stub produced by *rpcgen* to form an executable server program. Client procedures are written and linked in the same way.

PROS:

- You don't have to worry about getting a unique transport address (picking a unique port number for a socket on a machine). The server can bind to any available port and then register that port with an RPC name server. The client will contact this name server to find the the port number that corresponds to the program it needs. All this will be invisible to the programmer.
- The system can be independent of transport provider. The automatically-generated server stub can make itself available over every transport provider on a system, both TCP and UDP. The client can choose dynamically and no extra programming is required since the code to send and receive messages is automatically generated.
- Applications on the client only need to know one transport address: that of the name server that is responsible for telling the application where to connect for a given set of server functions.
- The function-call model can be used instead of the send/receive (read/write) interface provided by sockets. Users don't have to deal with marshaling parameters and then parsing them out on the other side.

CONS:

- Context switching increases scheduling costs
- PC does not solve the most of the distribution creation problems
- RPC is only interaction based. This does not offer any flexibility in terms of hardware architecture.

References:

1. Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014), Introduction to Distributed Systems, Arpaci-Dusseau Books
2. RPCGEN: <https://en.wikipedia.org/wiki/RPCGEN>