

Introduction:

The purpose of this assignment was to emphasize the principles of RPC. It required the development of a simple distributed computing environment consisting of multiple clients and a server. This system was implemented by me using C programming and also using the rpcgen utility as discussed in class.

RPC: Remote Procedure Call

Definition:

It is a paradigm to provide communication across a network between programs written in High Level Language.[1]

Meaning:

1. Distributed programming was not easy as the developers had to always consider the lower level network details for implementing programs in different distributed machines.
2. Hence to make distributed programming simplified Birrell and Nelson together proposed a paradigm called as RPC in which the remote call made is no different than a local call.
3. This leads to a higher level of abstraction and also helps remove the difficulties associated with being concerned about the lower level details of the program semantics and only focussing on the functionalities.
4. RPC greatly simplified the task of programming for distributed systems as it was similar to a local call and easy to use and understand.
5. RPC later on was implemented in various forms by different languages like Java, C, C++ etc.
6. It was considered as a major breakthrough in the area of distributed systems.

RPCGEN:

Definition:

"*rpcgen* provides programmers a direct way to write distributed applications. 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."[2]

Meaning:

Rpcgen is a paradigm which when used along with C programming, in my case, it will produce client and server stubs and skeletons i.e., by creating a .x file in which we write the remote version of the program function to be performed and then compiling with rpcgen command. Then we edit the client and server files accordingly just like we are doing a local call. The rest of the lower level details are taken care of by rpcgen. We just have to run the server and client files after editing and compiling.

Implementation of the program:

This assignment I have implemented using C programming. I have made use of the rpcgen utility as explained in class. The communication between the client and the server is hence done using RPC(rpcgen) using C programming. I learnt about rpcgen from SUN's ONC + Developer's Guide which has rpcgen programming tutorial and from the demo in class.

Logics of the program:

I have implemented the program in one folder. The client has all the 5 functions which it can by user choice can invoke the related functions as required on the server. The server has also all the 5 functions which are processed by it when the corresponding function is invoked by the client. I followed 4 steps as shown in class:

1. Make the rpcf1.x file and run rpcgen command.
2. Edit the rpcf1_server.c file.
3. Edit the rpcf1_client.c file.
4. Compile both using the Makefile.rpcf1

rpcf1.x file:

1. I have put all the variables to be invoked by each function inside structs. This is done for convenience as the chances of errors have been reduced from implementation and editing point of view. The structs are explained below:

1.1 struct echos: It contains the character array message which stores the string value of the user input for echo.

1.2 struct sorts: It contains the sa array which stores the integers to be sorted and also the variable size which is used to specify the number of integers to be sorted.

1.3 struct dirs: It contains the character arrays which store the string values of the filenames present in the directory.

1.4 struct timeint: It contains the integer values of the year, month, day, hour, min and sec.

1.5 struct arrs: It contains the integer arrays of a1, a2, b1, b2, c1 and c2.

2. Accordingly, I have made 5 program(functions) as follows:

2.1 ECHO_PROG: It contains the version ECHO_VERS and has a function get_echo which will return the type struct echos and pass in the parameter echos.

2.2 SORT_PROG: It contains the version SORT_VERS and has a function get_sort which will return the type struct sorts and pass in the parameter sorts.

2.3 readdirp_PROG: It contains the version readdirp_VERS and has a function get_readdirp which will return the type struct dirs and pass in the parameter dirs.

2.4 TIME_PROG: It contains the version TIME_VERS and has a function get_time which will return the type struct timeint and pass in the parameter timeint.

2.5 MATMUL_PROG: It contains the version MATMUL_VERS and has a function get_matmul which will return the type struct arrs and pass in the parameter arrs.

3. Each of the versions have been numbered as 1, 2, 3, 4 and 5 and their program numbers have been given differently.

4. The functions inside version are kept equal to 1 as only one function is performed when a particular service is invoked on the server by the client.

5. The rpcf1.x file is compiled using the command rpcgen -a -C rpcf1.x

6. The -a option generates all the template files and the -C compile time flag is used for ANSI C. The files generated are as follows:

6.1 rpcf1.h: It is the header file.

6.2 rpcf1_clnt: It is the client stub generated.

6.3 rpcf1_svc: It is the server stub generated.

6.4 Makefile_rpcf1: It is the makefile generated for compiling all the files.

6.5 rpcf1_xdr: It is the xdr routine file for proper data passing.

6.6 rpcf1_server: It is the server file generated by rpcgen which has to be edited.

6.7 rpcf1_client: It is the client file generated by rpcgen which has to be edited.

rpcf1 client:

There are 5 functions and a main method generated for the client. They are explained below:

1. main method:

1.1 The main method generated has been edited by me to provide the switch case functionality for the user to choose which function they want to invoke on the server. They are in the order of echo, sort, read directory contents, give current time & date and matrix multiplication. The default case is that the client enters anything other than 1 to 5 and the program exits.

1.2 The switch case is placed inside a while loop so that the users can continue to access one service after another and obviously only after one service is finished, the other service can be performed by the same client.

2. void echo_prog_1 function:

2.1 This function is called when the user selects the option by entering 1. I have added the functionality to take input from the user the message and the message is stored in the rpcgen generated echos get_echo_1_arg in which it is stored in the message character array. It is passed to the client and the value returned is caught in reslt_1 pointer of type struct echos.

2.2 If the result_1 is null i.e., the server did not return anything the call has failed which is displayed.

2.3 If the result_1 is not null then the message is printed out on the client screen.

3. void sort_prog_2:

3.1 This function is called when the user selects the option 2 by entering 2. I have added the code to take the number of integers to be sorted and then the integers into the size and sa array by using get_sort_2_arg which is of type sorts.

3.2 If the result_1 is null after passing get_sort_2_arg then the call failed message is displayed.

3.3 If the result_1 is not null done by the else part, the result is printed out accessing the values through result_1.

4. void readdirp_prog_3:

4.1 This function is called when the user selects the option 3 by entering 3. I have added the code and it just passes the empty char arrays to the server. The get_readdirp_3_arg is of type dirs and the result_1 pointer is of type struct dirs.

4.2 If the call fails a message is displayed indicating the same by checking if result_1 is null.

4.3 The results are printed out by accessing the character arrays in result_1 pointer and they are printed until the count variable limit which tells the client as to how many filenames are there.

5. void time_prog_4:

5.1 This function is called when the user selects the option 4 by entering 4. I have added the code to print out the result returned by the server stored inside the variables in result_1 pointer which is of type struct timeint. The get_time_4_arg is of type timeint which is empty passed by the client to the server.

5.2 As usual if the call fails the result_1 is empty and if the call succeeds then the result_1 values are printed out.

6. void matmul_prog_5:

6.1 This function is called when the user selects the option 5 by entering 5. I have added the code to take input from the user first the row1 of the matrix and then row 2 and then the second matrix the first column and second column and hence used 4 arrays. they are accessed using the get_matmul_5_arg which is of type arrs. Then it is passed to the server for processing.

6.2 As usual if the result_1 is null the call failed is printed or when successful the final result of the matrix multiplication is printed out.

rpcf1 server:

At the server side, there are 5 functions created. These functions perform the operations or the services as requested by the clients and return the results back to the client. They are described below:

1. get_echo_1_svc:

1.1 This function only has a static struct of type echos called result and the value of the argp pointer is stored which is the user input.

1.2 The result is returned back to the client.

2.get_sort_2_svc:

2.1 This function has a static struct of type sorts called result which takes in the value of pointer argp. The argp pointer contains the get_sort_2_arg values which the user had entered.

2.2 These values are then accessed by the server using result variable.

2.3 The sorting program is a simple bubble sort program which will sort in ascending order the values of the integers. The highest value is bubbled up each time the function checks and compares the integer in question with its neighbor.

2.4 The final result is then stored in the sa array of the result variable and then returned to the client.

3.get_readdir_3_svc:

3.1 In this function the result is of type struct dirs which is static which is used to access the argp value from the client which is empty.

3.2 The character pointers are initialized and they are named serially to store each filename in the directory individually.

3.3 First the current directory is opened and then using if else each filename is stored one by one updating result.count each time until it is 16 as that is the limit set. The count variable is used as error handling as if there are more than 16 files then the program will not go in an unstable state.

3.4 the readdir will read from dir pointer and store into ent pointer and then from the ent we access the directory name and save into the pointer filearr1p(eg.) and then using strncpy which is string copy function we store it into the corresponding character array as per the order in the result variable.

3.5 The directory is closed after operation and if the directory cannot be opened an error message is displayed.

3.6 The result is then returned back to the client.

4.get_time_4_svc:

4.1 In this function the result is of type static struct timeint and the value of the argp pointer is stored inside result which was empty and passed by the client.

4.2 Then using the functionalities in the time.h header file we store the localtime in t1 which is of struct tm. Then the individual values of the year, month, day, hour, min, sec are stored into the respective variables in the result struct.

4.3 The result is then returned back to the client.

5.get_matmul_5_svc:

5.1 Here the result is of type static struct arrs and the client passed the input values using get_matmul_5_arg which is stored in argp pointer and the result will store the value of argp pointer.

5.2 Here the row elements are multiplied with the column elements as in matrix multiplication where for an element at (i,j) at the product the value is determined by multiplying the ith row and the jth column and the elements are multiplied in an ordered fashion. e.g., for element at the product (2,1) the 2nd row and the 1st column elements are multiplied and added. So if it is a 2x2 matrix then the (1st element of 2nd row X 1st element of 1st column)+(2nd element of 2nd row X 2nd element of 1st column) is the value of element at (2,1) in the resulting matrix.

5.3 So the final results are stored in the c1 and c2 arrays which represent the 2 rows of the resulting matrix from the product.

5.4 This is done for the first row and then for the second row and the arrays are considered accordingly.

1. a1: row 1 of matrix 1
2. a2: row 2 of matrix 1
3. b1: column 1 of matrix 2
4. b2: column 2 of matrix 2
5. c1: row 1 of matrix 3(the resulting matrix)
6. c2: row 2 of matrix 3(the resulting matrix)

So here for the first element of c1 we do [(1st element of a1 X 1st element of b1)+(2nd element of a1 X 2nd element of b1)].

5.5 The resulting matrix is then returned to the client.

Interaction Model:

The interaction model will consist of 5 clients running from different servers and also the main server running.

It is drawn below as follows as per my design:

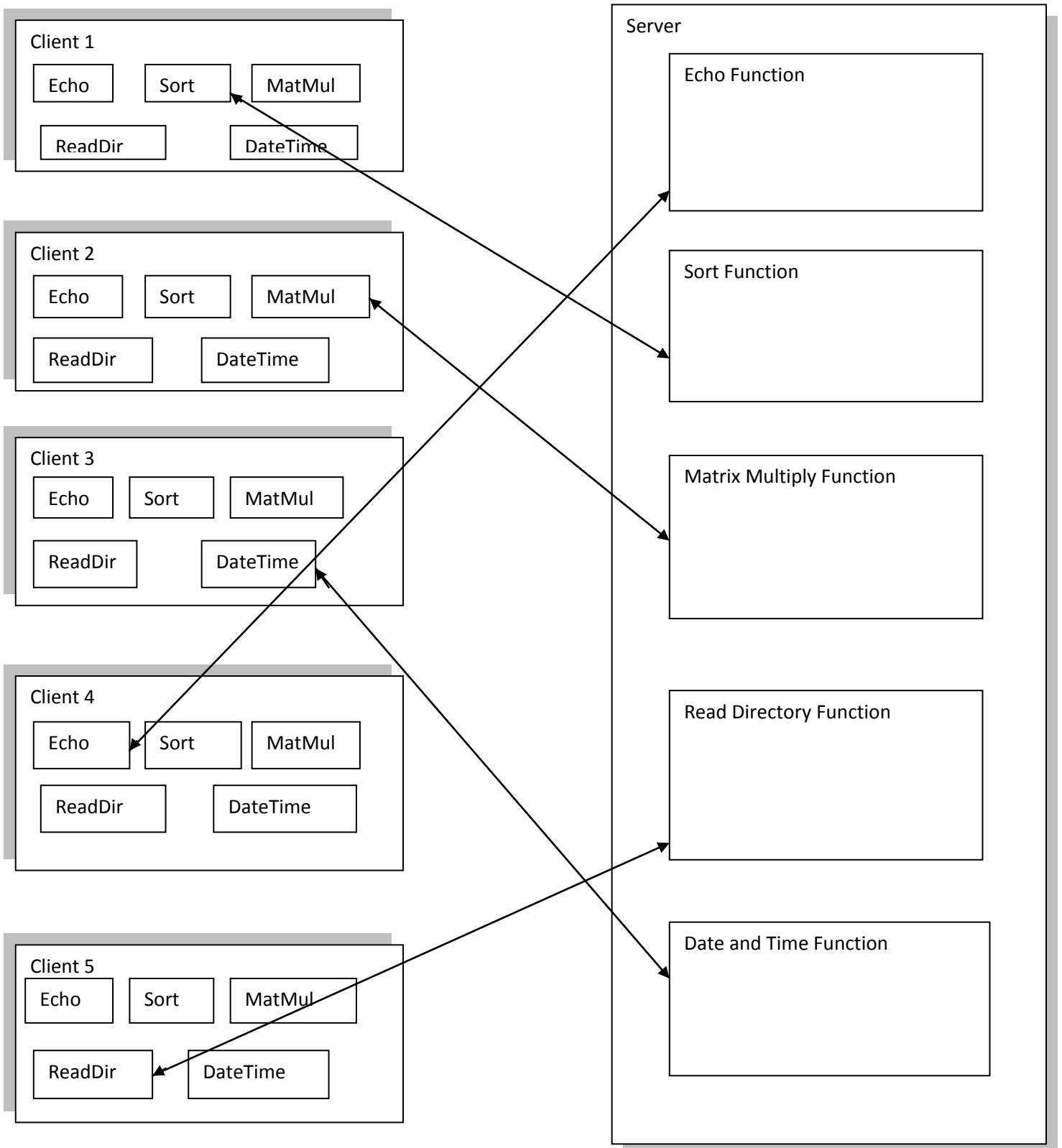


Figure 1: Interaction Diagram showing the clients invoking calls on server.

How *RPCGEN* works:

It is just an implementation of RPC Protocol.

1. First we create the specification file in which we define the program, version, their numbers and also the function types and return values and also values passed inside.

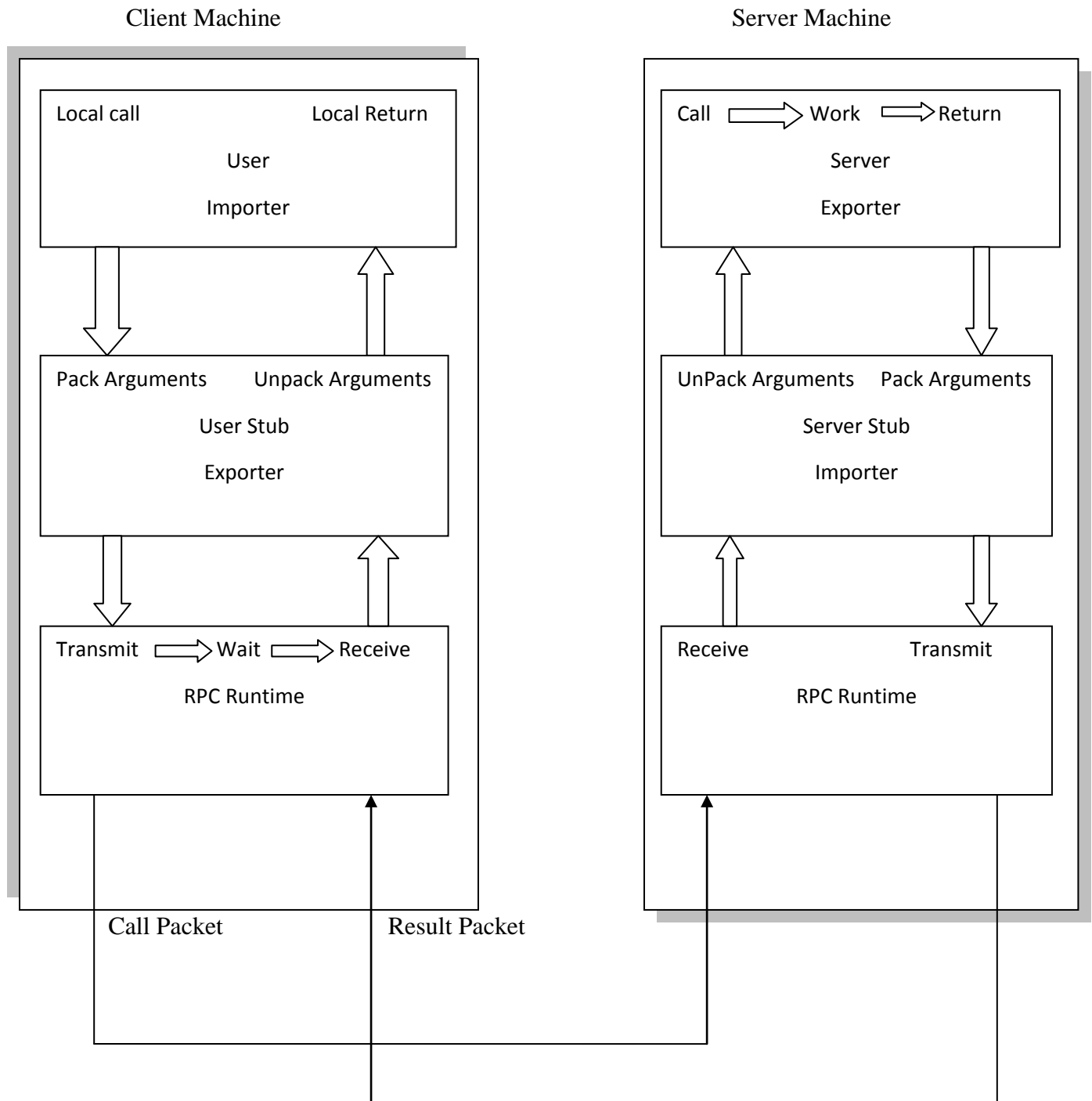


Figure 2: RPC Structure

2. When we run rpcgen then the client and server files as well as stubs, header file and an xdr file are generated.
3. The client local call first makes goes to client stub which packs the arguments and sends to the RPC runtime for transmission(e.g. The get_sort_2_arg was sent).
4. The call packet is sent over the network to the server runtime and then it is received by it.
5. The server runtime forwards to the server stub which will unpack the arguments and send to server.(e.g. The *argp was received)
6. In the server, a call is made to the concerned function which will perform some work. This is the code which I had written on the server file.
7. It will then use return (e.g. return result as in program) which will go to the server stub.
8. The server stub will pack the arguments and send to server runtime which will transmit the result packet through the network to the client runtime.
9. The client RPC Runtime will receive the result packet and send to the client stub which will unpack the arguments and send back to the client.
10. The client will then display on the screen the result(e.g. Using *result_1).

Note: Though it was a remote call, we just had to edit the client and server programs as a local call and were not concerned with the lower level details and implementation.

Failure Models:

Scenario 1:

The server crashes and the client is still running.



Figure 3: Scenario 1

1. Due to some arbitrary failure if the server crashes during the call the client may display unsuccessful call or may keep waiting.

2. If the crash happens before the call then the client will display the error message that server is not present and RPC error.
3. Thus the remote call would not be completed.

Scenario 2:

The client crashes.

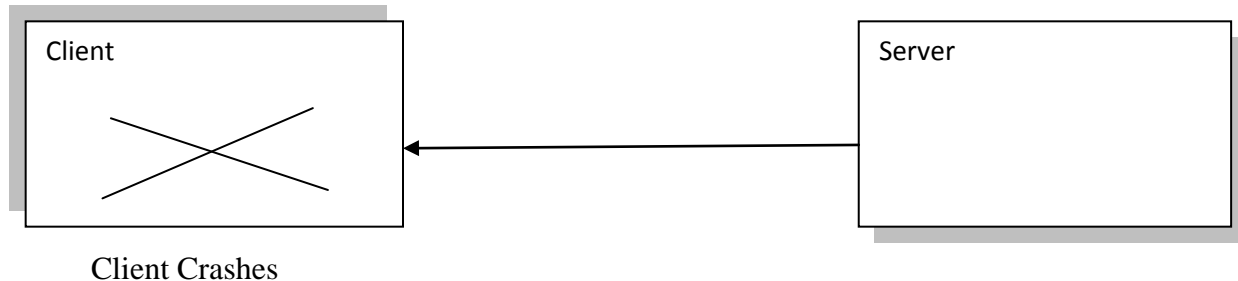


Figure 4: Scenario 2

1. If after the call to the server the client crashes, the server will not be able to send back the results and the remote call is a failure.

Scenario 3:

The network does not function properly.

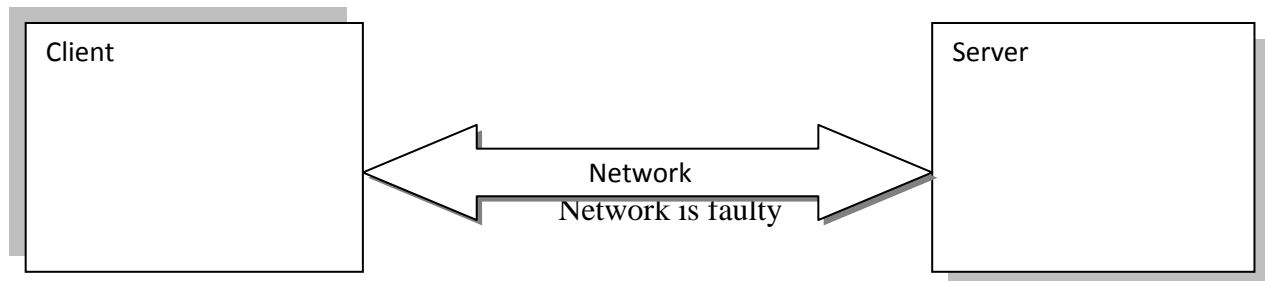


Figure 5: Scenario 3

1. In this case the network is faulty and may return a wrong result due to the error in bit transmission. Hence the result can be different from expected.
2. If the network crashes then the client will not be able to send or receive the results and the remote call would have failed.

Difficulties I faced while doing the assignment:

1. I could not implement the multi threading part on the server as I was getting errors like segmentation fault. The multithreading part I tried with the -M option in rpcgen. Though however the server is able to accept concurrent requests in my design.

2. For the matrix multiplication part, I tried to pass 2 dimensional arrays and was unsuccessful at rpcgen compiling. I tried using pointers but even that was not possible. Hence, I decided to pass a 1 dimensional arrays but perform 2 dimensional multiplication and get back the results. It works correctly as intended.

Architectural Model:

The architectural model is that of a basic client server architecture where the clients call various functions on the server.

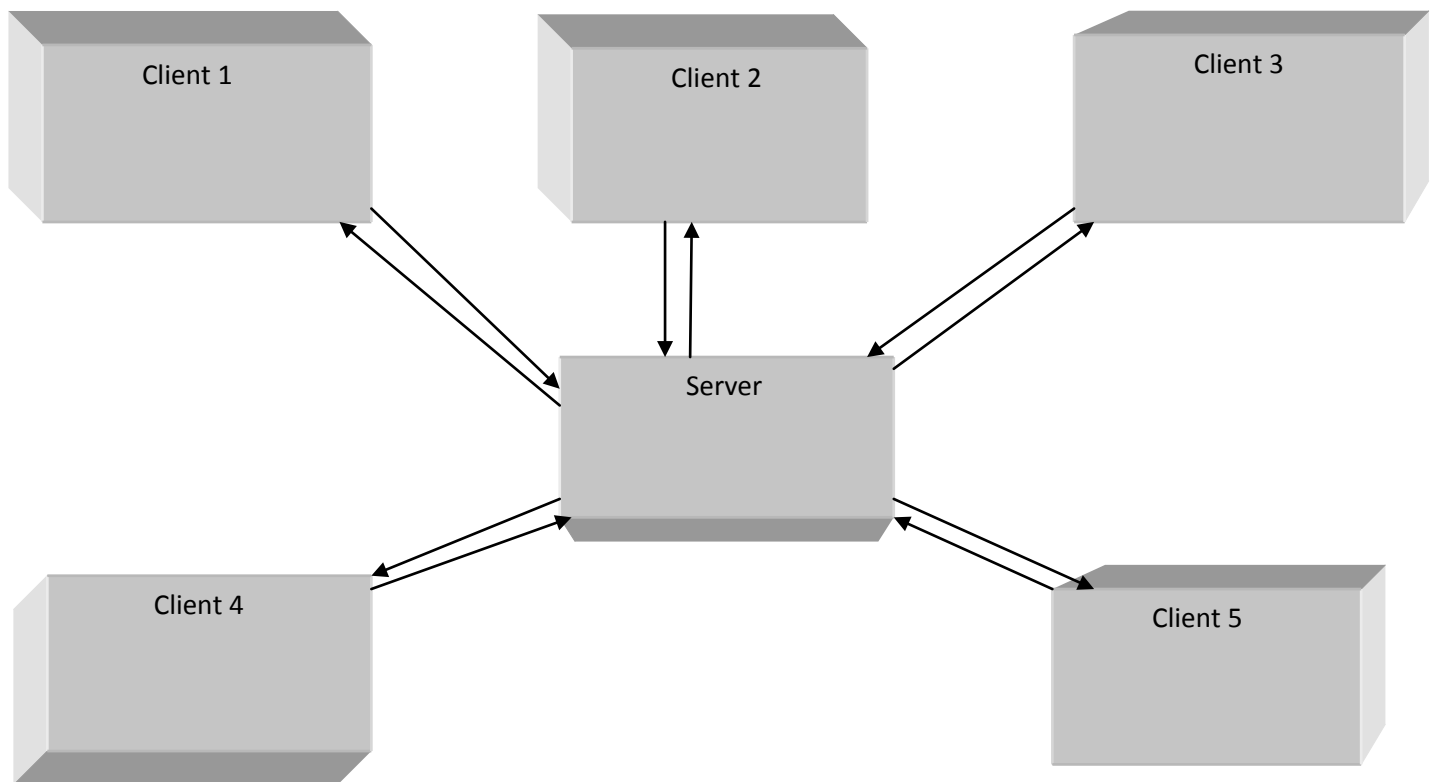


Figure 6: Architectural Model

Pros and Cons of my Design:

Pros:

1. I have kept all the functionalities in one server and the client also has all the invoking mechanisms present to call those methods. So, the server can be run on one machine and the client can be run on another machine(s) conveniently. This was achieved by incorporating all the functions in one .x file and using switch case at the client side for choosing a proper choice of function to be invoked.
2. The client after satisfaction of one request will ask for whether it wants to continue execution of the program and thus the user can call various functions one after another. This was achieved using a while loop at client end.

3. In the echo function the user can enter character as well as numbers and also alphanumeric contents as I have used strings. Hence, this flexibility has been provided.

4. In the sort function, the user can enter number count i.e., the number of integers to be sorted and then enter the numbers so the user can choose different count of numbers each time the sort function is selected.

5. In readdir function the user can read directories containing 16 files and 25 string names. Thus slightly long file names are supported.

Cons:

1. I could not implement the multithreading part of the server though the server can accept concurrent requests and it functions properly.

2. In matrix multiplication, it supports only 2X2 matrix multiplication.

3. The readdir function cannot read more than 16 files but if suppose 20 files are present it will read only 16 files.

4. In the echo function, the whitespace is the delimiter so if a user enters an input with a space the input after space will not be returned. (e.g. The user enters "Hello World!" only "Hello" would be returned).

Outputs:

Case 1:

Two clients are running and invoking functions on the server.

```

arvnair@sl253-rrpc02:~/myrpcf1
login as: arvnair
arvnair@10.234.140.28's password:
Last login: Wed Nov 19 09:27:02 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc02 ~]$ cd myrpcf1
[arvnair@sl253-rrpc02 myrpcf1]$ ls
client27.sh  client32.sh  rpcf1_clnt.c  rpcf1_server.o  rpcf1_xdr.o
client28.sh  Makefile.rpcf1  rpcf1_clnt.o  rpcf1_svc.c  server.sh
client29.sh  rpcf1_client  rpcf1.h      rpcf1_svc.o
client30.sh  rpcf1_client.c  rpcf1_server  rpcf1.x
client31.sh  rpcf1_client.o  rpcf1_server.c  rpcf1_xdr.c
[arvnair@sl253-rrpc02 myrpcf1]$ ./client27.sh
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
3
The filenames of the files present in the current directory are:
rpcf1_svc.o
rpcf1_clnt.c
client32.sh
client28.sh
rpcf1_svc.c
rpcf1_client.c
.
rpcf1_client.o
rpcf1_xdr.c
rpcf1_server.o
client31.sh
rpcf1_server.c
client30.sh
client27.sh
rpcf1_xdr.o
rpcf1_clnt.o
Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
5

arvnair@sl253-rrpc03:~/myrpcf1
login as: arvnair
arvnair@10.234.140.29's password:
Last login: Wed Nov 19 09:34:30 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc03 ~]$ cd myrpcf1
[arvnair@sl253-rrpc03 myrpcf1]$ ./client27.sh
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
4
now (MM-DD-YYYY HH:MM:SS): 11-19-2014 9:37:15
Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
2
Enter the number of integers to be sorted:
3
Enter the integers:
3
1
2
The array sorted in ascending order is:
1
2
3Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort

arvnair@sl253-rrpc01:~/myrpcf1
login as: arvnair
arvnair@10.234.140.27's password:
Last login: Wed Nov 19 09:26:04 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc01 ~]$ cd myrpcf1
[arvnair@sl253-rrpc01 myrpcf1]$ ./server.sh
^C[arvnair@sl253-rrpc01 myrpcf1]$
  
```

Figure 7: Case 1 Part 2 Output Screenshot

```

arvnair@sl253-rrpc02:~/myrpcf1
rpcf1_client.o
rpcf1_xdr.c
rpcf1_server.o
client31.sh
rpcf1_server.c
client30.sh
client27.sh
rpcf1_xdr.o
rpcf1_clnt.o
Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
5
Enter the 2 elements of the first row of the first matrix:
Enter element 0: 2
Enter element 1: 2
Enter the 2 elements of the second row of the first matrix:
Enter element 0: 2
Enter element 1: 2
Enter the 2 elements of the first column of the second matrix:
Enter element 0: 2
Enter element 1: 2
Enter the 2 elements of the second column of the second matrix:
Enter element 0: 2
Enter element 1: 2
Product of the two matrices is:
The first row of the first matrix: 8 8
The second row of the first matrix: 8 8
Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
4
now (MM-DD-YYYY HH:MM:SS): 11-19-2014 9:38:49
Do you want to continue?(y/n)n
[arvnair@sl253-rrpc02 myrpcf1]$

arvnair@sl253-rrpc03:~/myrpcf1
2
Enter the number of integers to be sorted:
3
Enter the integers:
3
1
2
The array sorted in ascending order is:
1
2
3Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
1
Enter something:
hello
You entered: hello
Do you want to continue?(y/n)y
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
1
Enter something:
hi
You entered: hi
Do you want to continue?(y/n)n
[arvnair@sl253-rrpc03 myrpcf1]$

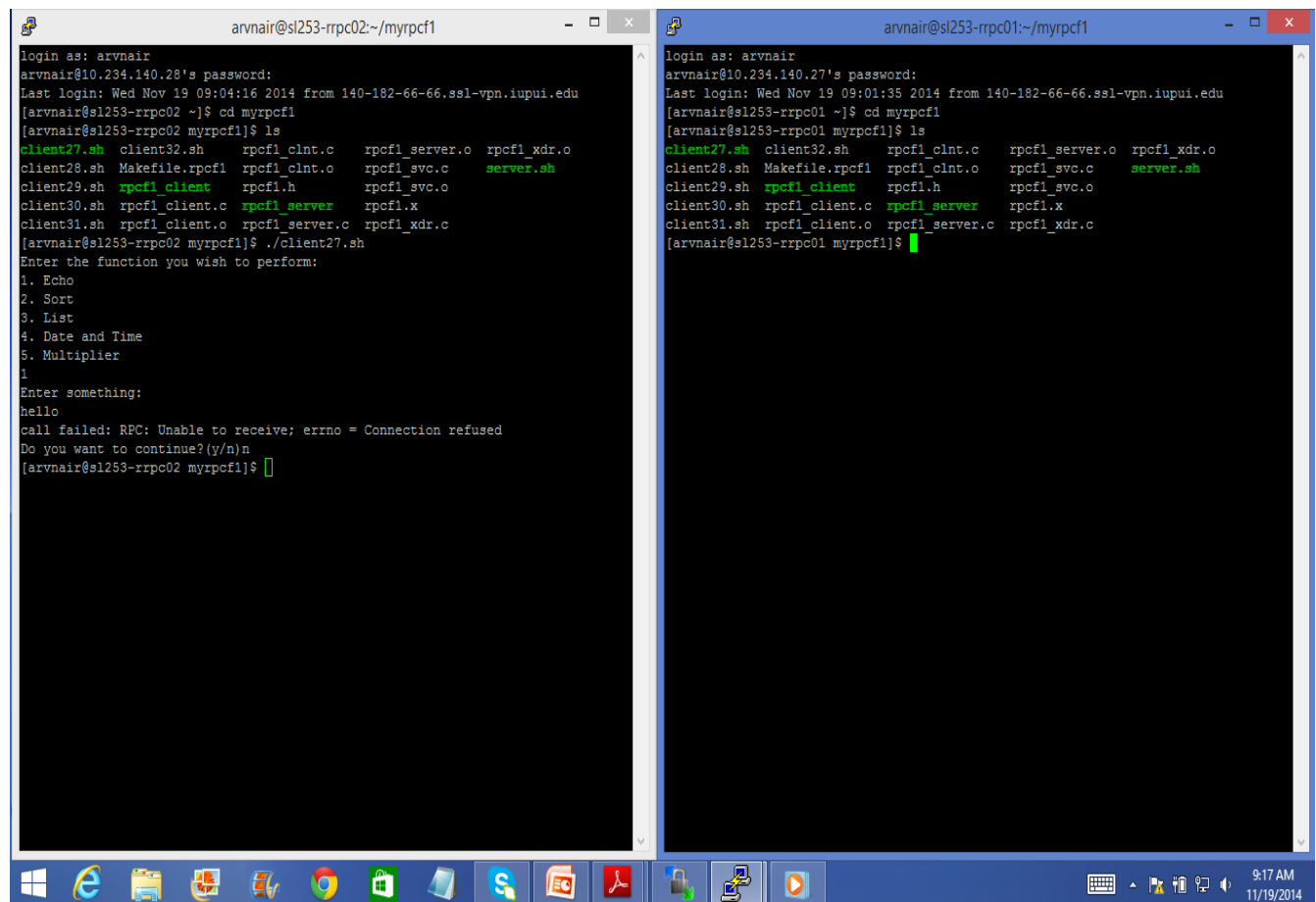
arvnair@sl253-rrpc01:~/myrpcf1
login as: arvnair
arvnair@10.234.140.27's password:
Last login: Wed Nov 19 09:26:04 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc01 ~]$ cd myrpcf1
[arvnair@sl253-rrpc01 myrpcf1]$ ./server.sh
^C[arvnair@sl253-rrpc01 myrpcf1]$
  
```

Figure 8: Case 1 Part 2 Output Screenshot

1. Server: The server is running at 10.234.140.27.
2. Client 1: The client 1 is running at 10.234.140.28 and calls the following functions on server:
 - 2.1 List : Lists all filenames in current directory.
 - 2.2 Multiplier: performs the matrix multiplication on a 2X2 matrix.
 - 2.3 Date and Time: Returns the current date and time.
3. Client 2: The client 2 is running at 10.234.140.29 and calls the following functions on server:
 - 3.1 Date and Time: Returns the current date and time.
 - 3.2 Sort: Sorts the input numbers in ascending order.
 - 3.3Echo: Returns the message the user sends to server via console input (option chosen twice).

Case 2:

The server is not running and the client invokes a function.



```
login as: arvnair
arvnair@10.234.140.28's password:
Last login: Wed Nov 19 09:04:16 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc02 ~]$ cd myrpcf1
[arvnair@sl253-rrpc02 myrpcf1]$ ls
client27.sh  client32.sh  rpcfl_clnt.o  rpcfl_server.o  rpcfl_xdr.o
client28.sh  Makefile.rpcfl  rpcfl_clnt.o  rpcfl_svc.c  server.sh
client29.sh  rpcfl_client  rpcfl.h      rpcfl_svc.o
client30.sh  rpcfl_client.o  rpcfl_server  rpcfl.x
client31.sh  rpcfl_client.o  rpcfl_server.c  rpcfl_xdr.c
[arvnair@sl253-rrpc02 myrpcf1]$ ./client27.sh
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
1
Enter something:
hello
call failed: RPC: Unable to receive; errno = Connection refused
Do you want to continue?(y/n)n
[arvnair@sl253-rrpc02 myrpcf1]$
```

```
login as: arvnair
arvnair@10.234.140.27's password:
Last login: Wed Nov 19 09:01:35 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc01 ~]$ cd myrpcf1
[arvnair@sl253-rrpc01 myrpcf1]$ ls
client27.sh  client32.sh  rpcfl_clnt.o  rpcfl_server.o  rpcfl_xdr.o
client28.sh  Makefile.rpcfl  rpcfl_clnt.o  rpcfl_svc.c  server.sh
client29.sh  rpcfl_client  rpcfl.h      rpcfl_svc.o
client30.sh  rpcfl_client.o  rpcfl_server  rpcfl.x
client31.sh  rpcfl_client.o  rpcfl_server.c  rpcfl_xdr.c
[arvnair@sl253-rrpc01 myrpcf1]$
```

Figure 9: Case 2 Output Screenshot

Server: Here the server is not running.

Client: The client is running on 10.234.140.28 and tries to call the echo function on server at 10.234.140.27 . The client enters the message but gets the error unable to receive and thus RPC has failed.

Case 3:

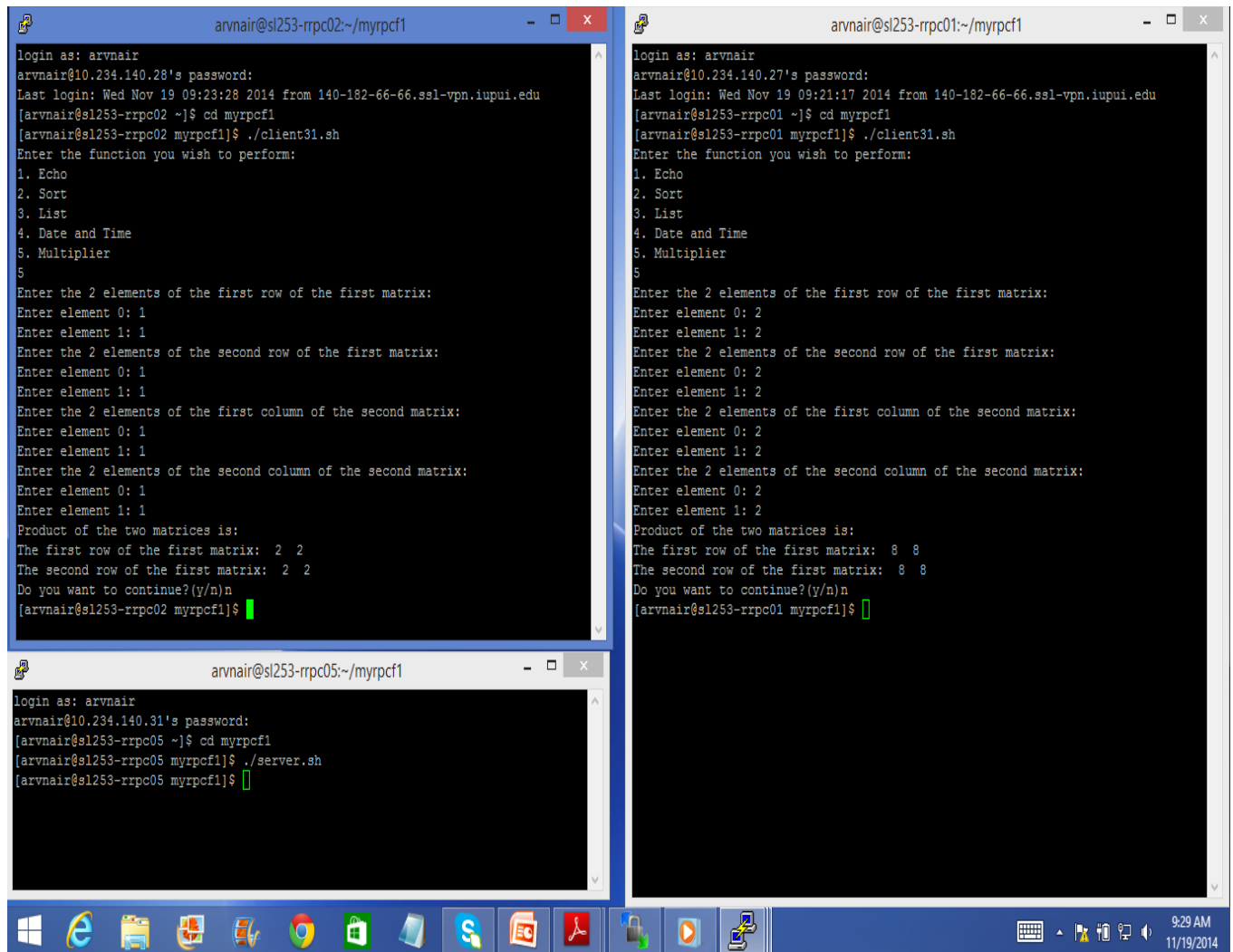
2 clients concurrently access a function on the server.

Server: The server is running on 10.234.140.31.

Client 1: The client 1 is running on 10.234.140.27 and calls the multiplier function.

Client 2: The client 2 is running on 10.234.140.28 and calls the multiplier function.

Both the clients simultaneously input the matrix and get the answer or result specific to their query. Hence there are no concurrency issues found in this case.



The image displays three terminal windows from a Windows desktop environment. The top-left window, titled 'arvnair@sl253-rrpc02:~/myrpcf1', shows a client running './client31.sh'. It prompts for a function (5. Multiplier) and matrix elements, resulting in a product of 2 2. The top-right window, titled 'arvnair@sl253-rrpc01:~/myrpcf1', shows a client running './client31.sh' with function 5 and matrix elements 2, resulting in a product of 8 8. The bottom window, titled 'arvnair@sl253-rrpc05:~/myrpcf1', shows a server running './server.sh' which receives the request from the bottom-left client. The Windows taskbar at the bottom shows the time as 9:29 AM on 11/19/2014.

```
login as: arvnair
arvnair@10.234.140.28's password:
Last login: Wed Nov 19 09:23:28 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc02 ~]$ cd myrpcf1
[arvnair@sl253-rrpc02 myrpcf1]$ ./client31.sh
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
5
Enter the 2 elements of the first row of the first matrix:
Enter element 0: 1
Enter element 1: 1
Enter the 2 elements of the second row of the first matrix:
Enter element 0: 1
Enter element 1: 1
Enter the 2 elements of the first column of the second matrix:
Enter element 0: 1
Enter element 1: 1
Enter the 2 elements of the second column of the second matrix:
Enter element 0: 1
Enter element 1: 1
Product of the two matrices is:
The first row of the first matrix:  2  2
The second row of the first matrix:  2  2
Do you want to continue?(y/n)n
[arvnair@sl253-rrpc02 myrpcf1]$
```

```
login as: arvnair
arvnair@10.234.140.27's password:
Last login: Wed Nov 19 09:21:17 2014 from 140-182-66-66.ssl-vpn.iupui.edu
[arvnair@sl253-rrpc01 ~]$ cd myrpcf1
[arvnair@sl253-rrpc01 myrpcf1]$ ./client31.sh
Enter the function you wish to perform:
1. Echo
2. Sort
3. List
4. Date and Time
5. Multiplier
5
Enter the 2 elements of the first row of the first matrix:
Enter element 0: 2
Enter element 1: 2
Enter the 2 elements of the second row of the first matrix:
Enter element 0: 2
Enter element 1: 2
Enter the 2 elements of the first column of the second matrix:
Enter element 0: 2
Enter element 1: 2
Enter the 2 elements of the second column of the second matrix:
Enter element 0: 2
Enter element 1: 2
Product of the two matrices is:
The first row of the first matrix:  8  8
The second row of the first matrix:  8  8
Do you want to continue?(y/n)n
[arvnair@sl253-rrpc01 myrpcf1]$
```

```
login as: arvnair
arvnair@10.234.140.31's password:
[arvnair@sl253-rrpc05 ~]$ cd myrpcf1
[arvnair@sl253-rrpc05 myrpcf1]$ ./server.sh
[arvnair@sl253-rrpc05 myrpcf1]$
```

Figure 10: Case 3 Output Screenshot

References:

1. *Implementing Remote Procedure Calls*, Andrew D. Birrell and Bruce Jay Nelson, Xerox Palo Alto Research Center.[1]
2. <https://docs.oracle.com/cd/E19683-01/816-1435/6m7rrfn7k/index.html>. [2]
3. Class notes and slides on RPC ppt and rpcgen.
4. *DISTRIBUTED SYSTEMS Concepts and Design Fifth Edition*, George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair, Addison Wesley Publications.