

Indiana University-Purdue University Indianapolis

Department of Computer and Information Science

CSCI 53700

Fall 2017

Assignment - 3

AKHIL NAYABU
ID: 2000075395

Table of Contents

I.	Introduction.....	3
II.	RMI Concepts & Design	3
III.	Functionality and Discussion	4
IV.	Comparison between RPC and RMI.....	4
V.	Sample runs and screenshots:	6

I. Introduction

The main objective of this assignment is to build a distributed system using remote method invocation. In this assignment, we build a server and multiple clients which interact through remote method calls and provide a comparison between RMI and RPC

II. RMI Concepts & Design

RMI is an extension of RPC model with object oriented model. It allows object in one to access/ invoke a method on object running in a remote system. RMI uses stub and skeleton object for communication with remote object.

Stub:

It is an object which acts as gateway for client side. All incoming and outgoing are routed through it. Stub performs following tasks:

- Connection to remote object
- Marshals outgoing request
- Unmarshals incoming response

Skeleton:

Like stub, skeleton is a gateway for server. All incoming requests and outgoing responses go through them. Skeleton Performs following tasks:

- Processes the incoming requests
- Invokes method on actual remote object
- Returns results to client. Also marshals and unmarshals requests and responses respectively

In this assignment we have Server, Client, ServerController, ServerControllerImpl. So ServerController is the interface and ServerControllerImpl is the class where all the abstract methods in ServerController are implemented.

In our server we have create instance of this class and register (bind) the service with RMI registry.

```
ServerController controller = new ServerControllerImpl();
Naming.rebind("//tesla.cs.iupui.edu:2010/Server", controller);
```

Our client should call the registry to obtain reference to the remote object. Client will receive reference to the interface.

```
ServerController controller= (ServerController)
Naming.lookup("//tesla.cs.iupui.edu:2010/Server");
```

III. Functionality and Discussion

- A. **Get Path:** gets current working directory on which the server is running. Used "System.getProperty()" function from java.io.file library of java.
- B. **Echo:** takes a message from client passes to server and returns it back to client. Prints the returned string on client.
- C. **File Status:** takes input file name from user and checks with a file with given name exists in the current directory. Returns "File found" if it exists, "File not found" if it does not exists
- D. **Sort:** takes size and elements of list as input, passes it to server where server sorts the given list and returns resultant list which is printed on client.
- E. **Matrix Multiplication:** takes in row and column as input, to make is simple I have considered both are matrices of same dimensions. Passes matrices and dimensions to server and server computes the multiplication of these two matrices and returns the resultant matrix back which is printed on Client.

IV. Comparison between RPC and RMI

A. Quantitative Analysis :

A roundtrip comparison was made for a function on both RPC and RMI. Initially tried comparing for 1000x1000 matrix multiplication but RPC did not support array of size 1000 x 1000. So have tested with 30 x 30 matrix multiplication. The method was run 100 times and average of round trip was taken.

30 x 30:

For RPC round trip time was: 545

For RMI round trip time was: 5725

20 x 20:

For RPC round trip time was: 305

For RMI round trip time was: 2904

10 x 10:

For RPC round trip time was: 278

For RMI round trip time was: 2129

5 x 5:

For RPC round trip time was: 176

For RMI round trip time was: 1819

As we can clearly see RPC was much faster than RMI. From the results we can say that ~12 times faster than RMI. Below is graph for observations made:

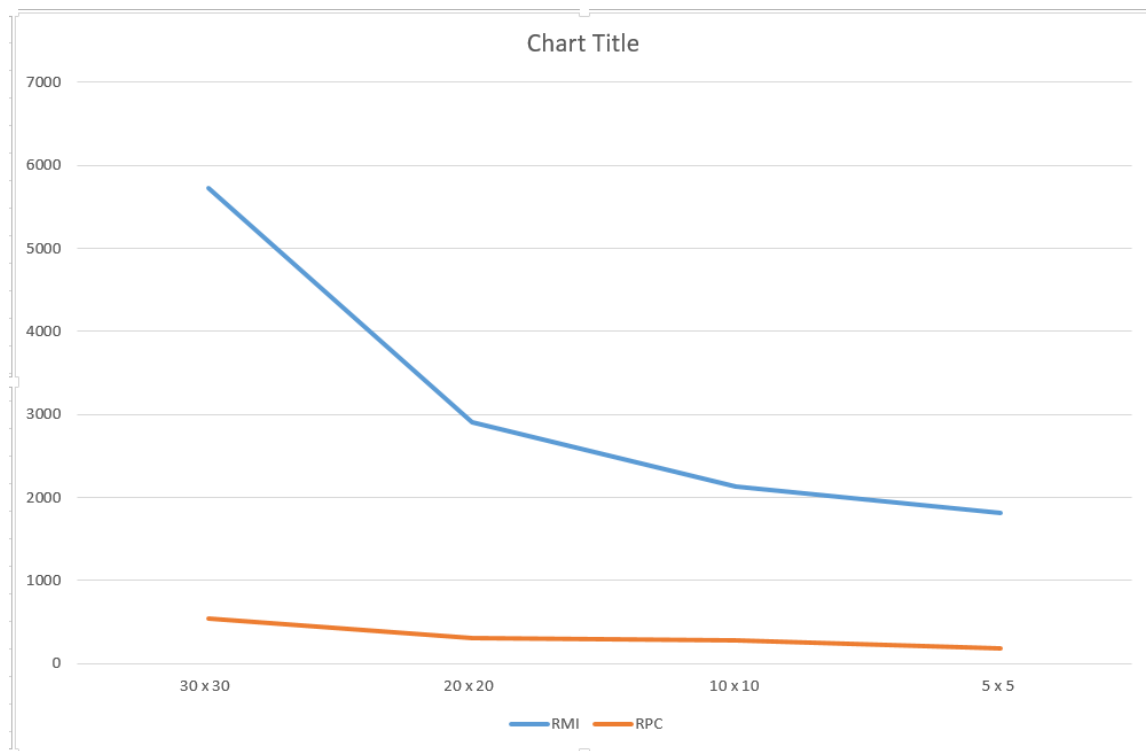


Chart (1): round trip time in microseconds

Possible reasons why RPC is so much faster than RMI in our case:

- Because it was written in C
- C does not have overhead processing like automatic garbage collection, dynamic typing and other facilities

B. Qualitative Analysis:

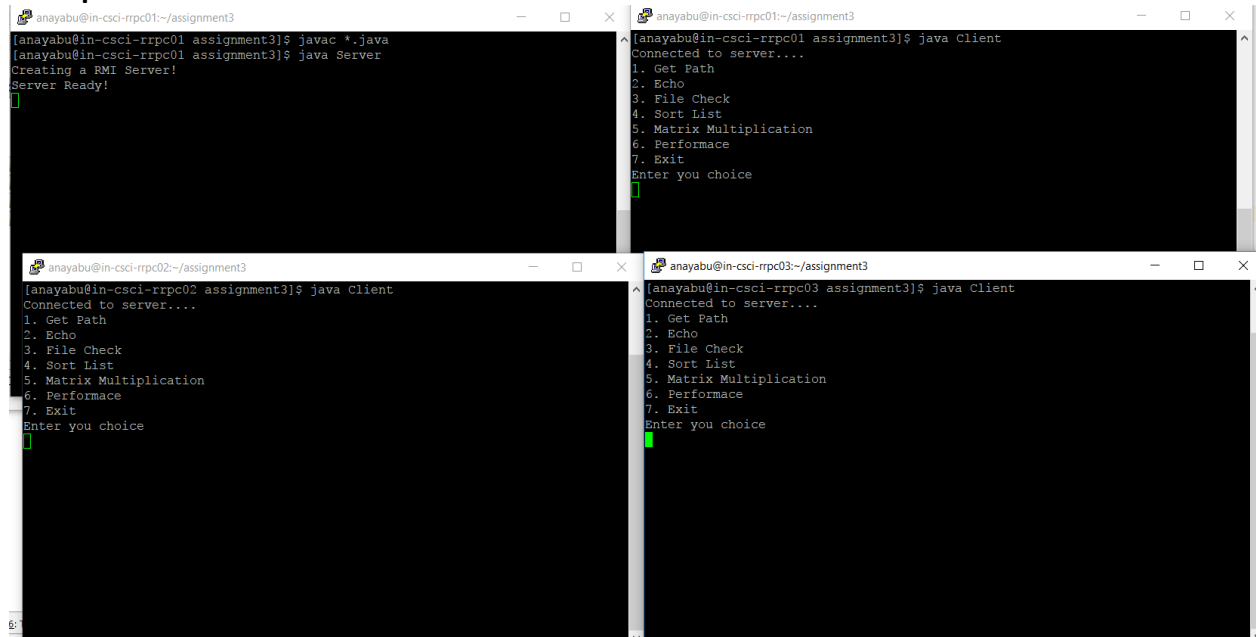
- RMI is implemented in java, hence it is object oriented hence user needs to know the object and the method of the object he needs to invoke.
In RPC we don't have any objects, it calls functions in remote machine through referential address
- RPC is looks like a local call and RMI is pretty similar too except that is passes a reference to object and method.

RMI code would be much cleaner and shorter and you will feel that RMI is much better than RPC since it has a better exception handling, garbage collection, multi-threading

etc. You will have all those advantages of java over C when you choose to use RMI over RPC.

V. Sample runs and screenshots:

Multiple Client Connections:



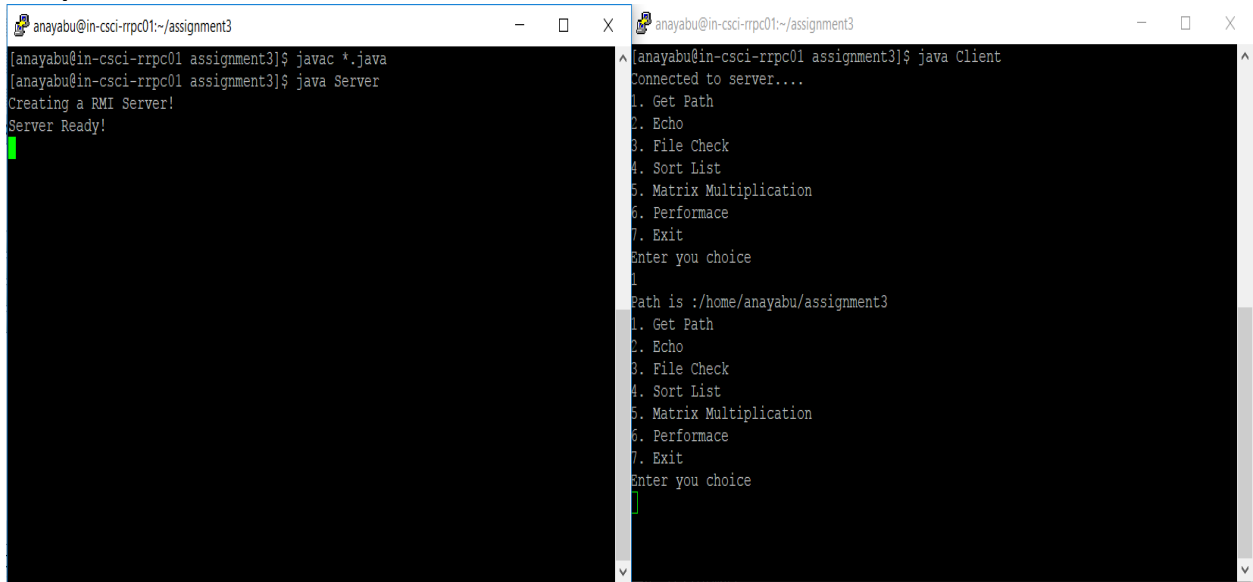
The image displays four terminal windows arranged in a 2x2 grid, illustrating the execution of a Java RMI application. The top-left window shows the server being compiled and started. The top-right, bottom-left, and bottom-right windows show multiple client instances connecting to the server and displaying a menu of operations.

```
[anayabu@in-csci-rrpc01 assignment3]$ javac *.java
[anayabu@in-csci-rrpc01 assignment3]$ java Server
Creating a RMI Server!
Server Ready!

[anayabu@in-csci-rrpc01 assignment3]$ java Client
Connected to server....
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice

[anayabu@in-csci-rrpc02 assignment3]$ java Client
Connected to server....
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice

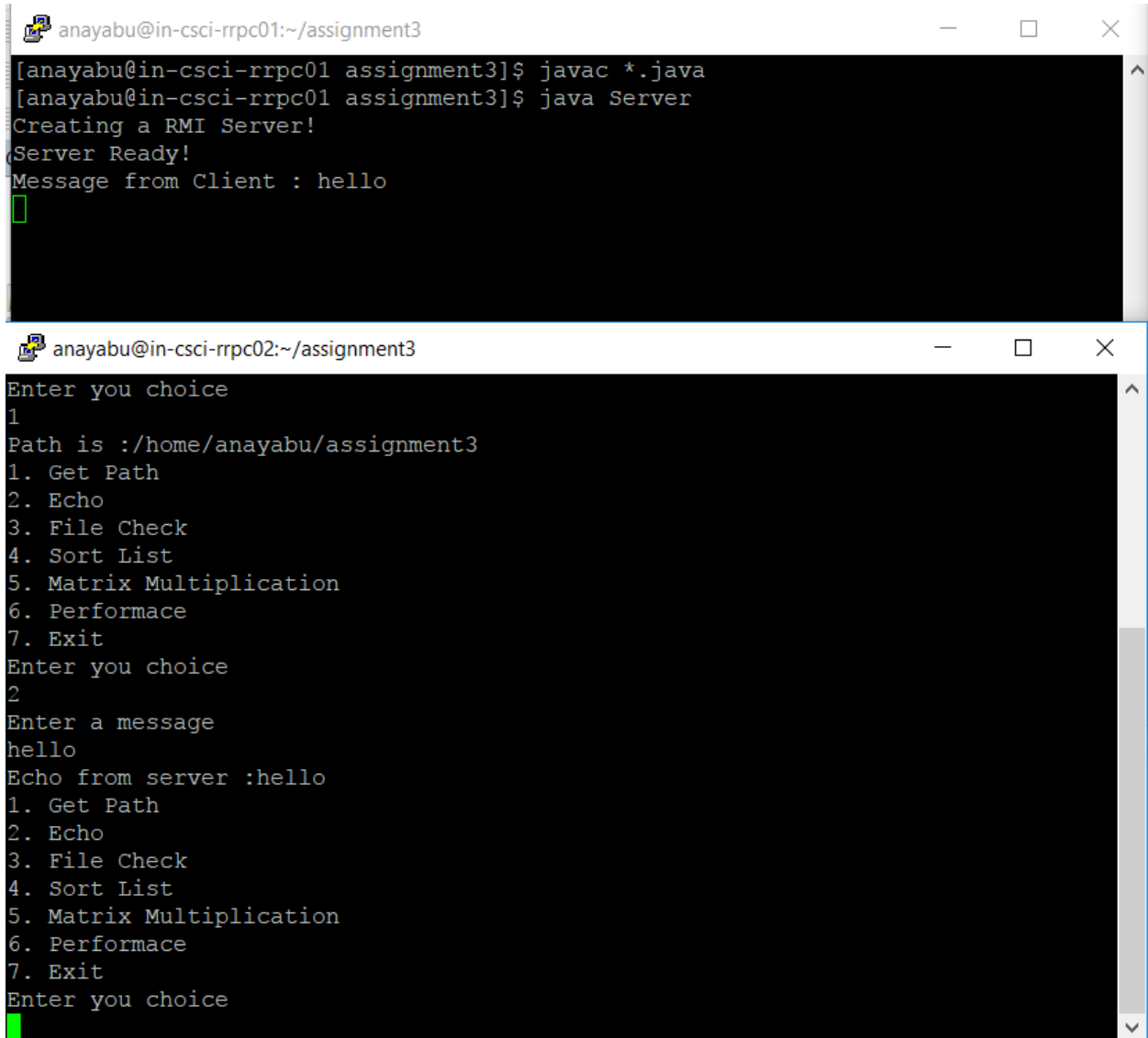
[anayabu@in-csci-rrpc03 assignment3]$ java Client
Connected to server....
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
```

Get path:

The image shows two terminal windows side-by-side. The left window shows the compilation and execution of a Java server. The right window shows the execution of a Java client that connects to the server and displays a menu of options.

```
anayabu@in-csci-rrpc01:~/assignment3
[anayabu@in-csci-rrpc01 assignment3]$ javac *.java
[anayabu@in-csci-rrpc01 assignment3]$ java Server
Creating a RMI Server!
Server Ready!

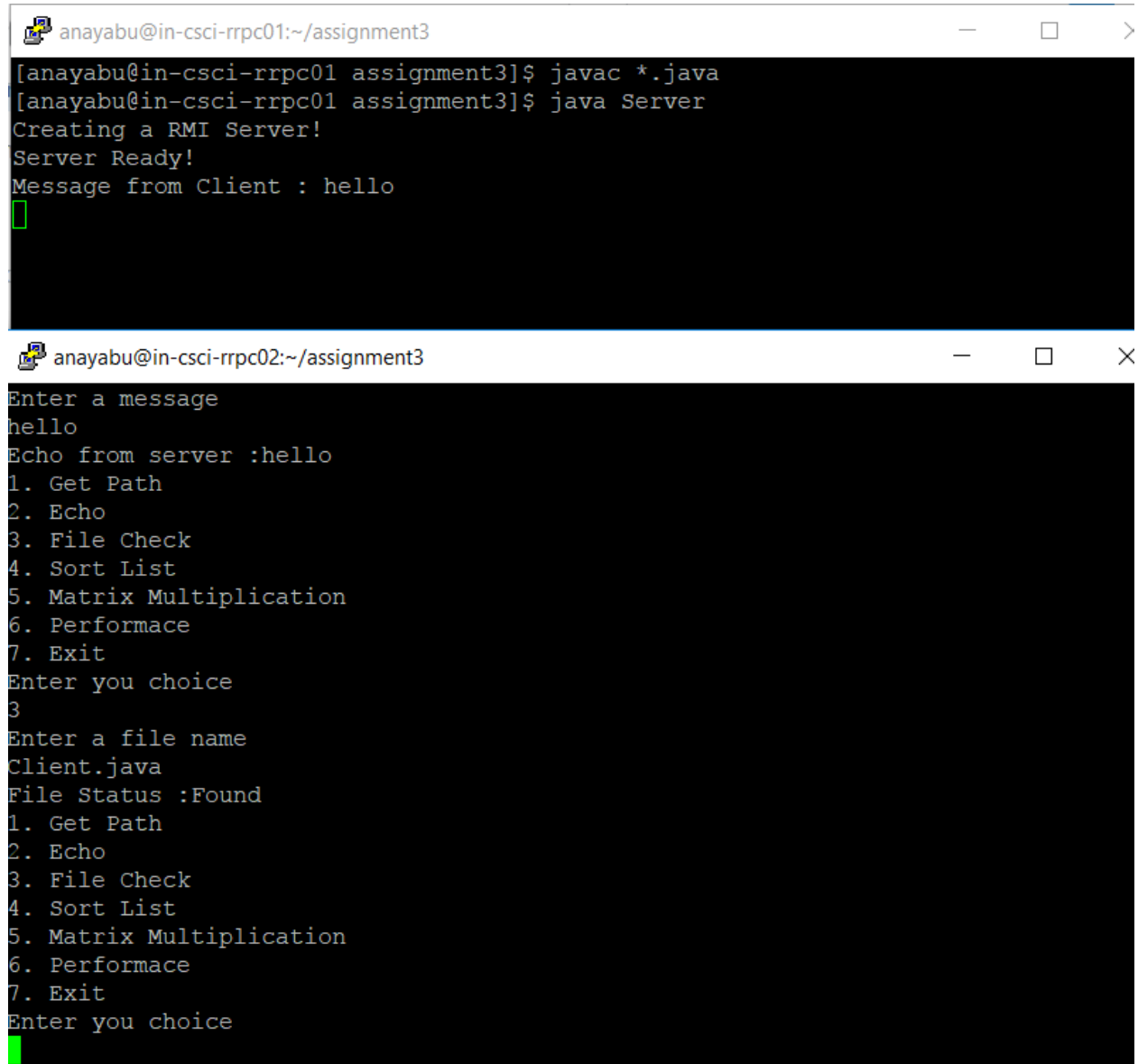
anayabu@in-csci-rrpc01:~/assignment3
[anayabu@in-csci-rrpc01 assignment3]$ java Client
Connected to server....
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
1
Path is :/home/anayabu/assignment3
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
]
```

ECHO:

The image shows two terminal windows. The top window, titled 'anayabu@in-csci-rrpc01:~/assignment3', shows the compilation and execution of a Java RMI server. The commands executed are 'javac *.java' and 'java Server'. The output shows the server creating an RMI Server, becoming ready, and receiving a message from the client: 'hello'. The bottom window, titled 'anayabu@in-csci-rrpc02:~/assignment3', shows the execution of a Java RMI client. The client prompts the user for a choice, lists seven options (1. Get Path, 2. Echo, 3. File Check, 4. Sort List, 5. Matrix Multiplication, 6. Performace, 7. Exit), and the user selects '2'. The client then prompts for a message, the user enters 'hello', and the client outputs 'Echo from server :hello'. The client then displays the same menu of options again and prompts for another choice.

```
anayabu@in-csci-rrpc01:~/assignment3
[anayabu@in-csci-rrpc01 assignment3]$ javac *.java
[anayabu@in-csci-rrpc01 assignment3]$ java Server
Creating a RMI Server!
Server Ready!
Message from Client : hello

anayabu@in-csci-rrpc02:~/assignment3
Enter you choice
1
Path is :/home/anayabu/assignment3
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
2
Enter a message
hello
Echo from server :hello
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
```


FILE STATUS:

The image shows two terminal windows. The top window, titled 'anayabu@in-csci-rrpc01:~/assignment3', shows the compilation and execution of a Java RMI server. The commands 'javac *.java' and 'java Server' are entered. The output shows 'Creating a RMI Server!', 'Server Ready!', and 'Message from Client : hello'. The bottom window, titled 'anayabu@in-csci-rrpc02:~/assignment3', shows the execution of a Java RMI client. It prompts for a message ('hello'), echoes it from the server, and then displays a menu of options: 1. Get Path, 2. Echo, 3. File Check, 4. Sort List, 5. Matrix Multiplication, 6. Performace, 7. Exit. The user enters '3' for 'File Check', prompts for a file name ('Client.java'), and receives the status 'File Status :Found'. The menu is repeated, and the user is prompted for a choice again.

```
anayabu@in-csci-rrpc01:~/assignment3
[anayabu@in-csci-rrpc01 assignment3]$ javac *.java
[anayabu@in-csci-rrpc01 assignment3]$ java Server
Creating a RMI Server!
Server Ready!
Message from Client : hello

anayabu@in-csci-rrpc02:~/assignment3
Enter a message
hello
Echo from server :hello
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
3
Enter a file name
Client.java
File Status :Found
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
```

SORT LIST:

The screenshot shows two terminal windows. The left window is the server, and the right window is the client.

Server Window:

```

anayabu@in-csci-rrpc01:~/assignment3
[anayabu@in-csci-rrpc01 assignment3]$ javac *.java
[anayabu@in-csci-rrpc01 assignment3]$ java Server
Creating a RMI Server!
Server Ready!

```

Client Window:

```

5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice
4
enter list size
5
enter list elements
5
3
2
4
1
Sorted list:
1 2 3 4 5
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice

```

Matrix Multiplication:

The screenshot shows two terminal windows. The left window is the server, and the right window is the client.

Server Window:

```

anayabu@in-csci-rrpc01:~/assignment3
[anayabu@in-csci-rrpc01 assignment3]$ javac *.java
[anayabu@in-csci-rrpc01 assignment3]$ java Server
Creating a RMI Server!
Server Ready!

```

Client Window:

```

Enter value for [1] [1]:
1
Enter values for Matrix2
Enter value for [0] [0]:
1
Enter value for [0] [1]:
1
Enter value for [1] [0]:
1
Enter value for [1] [1]:
1
Resultant Matrix:
2, 2,
2, 2,
1. Get Path
2. Echo
3. File Check
4. Sort List
5. Matrix Multiplication
6. Performace
7. Exit
Enter you choice

```