Lab:1

Remote Procedure Call (RPC)

1. Objective

I. learn about RPC

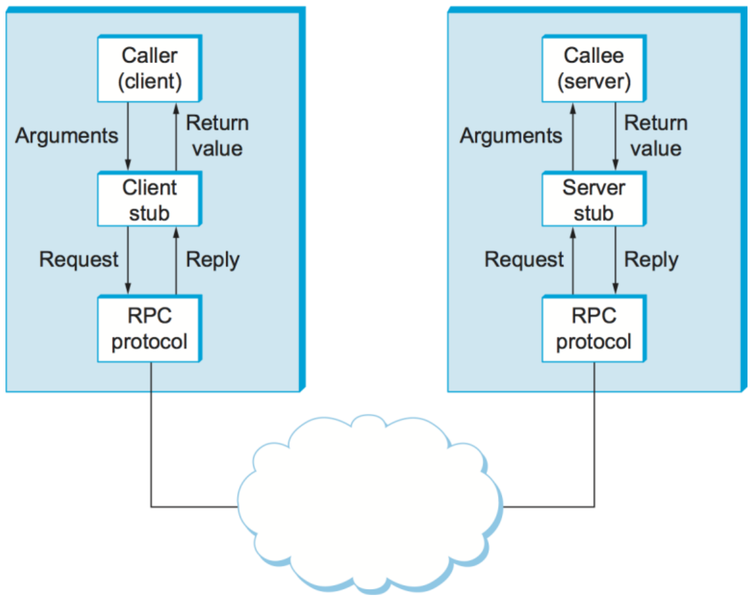
II. Implement a simple client side and server side system using java.

2. Software Used

The Java Development Kit (JDK) was used as a compiler, sublime text ide was used to write the code and the windows command prompt was used to execute the code.

3. Introduction

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details. RPC is used to call other processes on the remote systems like a local system. A procedure call is also sometimes known as a function call or a subroutine call.
RPC has the following architecture



4.Code Implementation

The following code implementation was done

Client side code

```
import java.io.*;

import java.net.*;

class client

{

public static void main(String[] args) throws Exception

{

 Socket sock = new Socket("localhost", 3000);

 BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));

 OutputStream ostream = sock.getOutputStream();

 PrintWriter pwrite = new PrintWriter(ostream, true);

 InputStream istream = sock.getInputStream();

 BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));

 System.out.println("Client ready, type and press Enter key");

 String receiveMessage, sendMessage1,sendMessage2, sendMessage3, temp;

 //while(true)

 //{

 System.out.println("\nEnter operation to perform(add,sub,mul,div)....");

 temp = keyRead.readLine();

 sendMessage1=temp.toLowerCase();


 System.out.println("Enter first parameter :");

 sendMessage2 = keyRead.readLine();


 System.out.println("Enter second parameter : ");

 sendMessage3 = keyRead.readLine();
 pwrite.println(sendMessage1);
 pwrite.println(sendMessage2);
pwrite.println(sendMessage3);

 if((receiveMessage = receiveRead.readLine()) != null)
 System.out.println(receiveMessage);
 //}
}
}
```

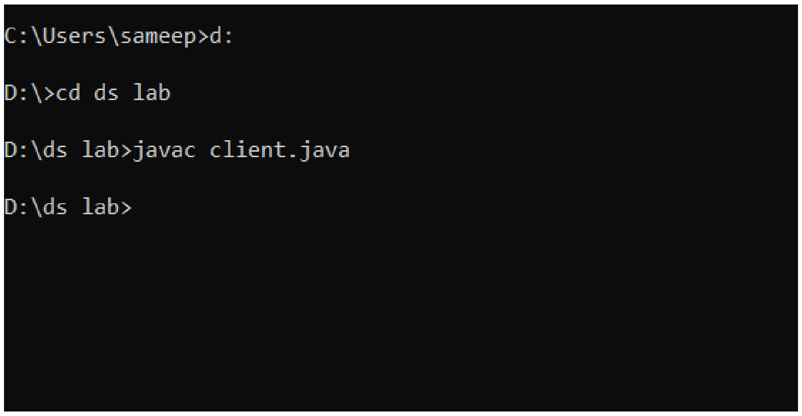
Server side code

```
import java.io.*;
import java.net.*;
class server
{
public static void main(String[] args) throws Exception
{
 ServerSocket sersock = new ServerSocket(3000);
 System.out.println("Server ready");
 Socket sock = sersock.accept( ); //establishes connection and waits for the client
 OutputStream ostream = sock.getOutputStream(); //getOutputStream returns output stream
for the given socket
 PrintWriter pwrite = new PrintWriter(ostream, true);
 InputStream istream = sock.getInputStream(); //getInputStream uses an input stream to read
data from a source.
 BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
//readsthe text from a character-based input stream
 String fun;
 int a,b,c;
 //while(true)
 //{
 fun = receiveRead.readLine();
 a = Integer.parseInt(receiveRead.readLine());
 b = Integer.parseInt(receiveRead.readLine());
 if(fun != null)
 System.out.println("Operation : "+fun);
 System.out.println("Parameter 1 : "+a);
 System.out.println("Parameter 2 : "+b);
 if(fun.compareTo("add")==0)
 {
 c=a+b;
 System.out.println("Addition = "+c);
pwrite.println("Addition = "+c);
 }
if(fun.compareTo("sub")==0)
{
c=a-b;
 System.out.println("Substraction = "+c);
pwrite.println("Substraction = "+c);
 }
 if(fun.compareTo("mul")==0)
 {
 c=a*b;
 System.out.println("Multiplication = "+c);
pwrite.println("Multiplication = "+c);
 }
 if(fun.compareTo("div")==0)
 {
 c=a/b;
 System.out.println("Division = "+c);
pwrite.println("Division = "+c);
 }
 System.out.flush();
//}
}
}
```

4. Result

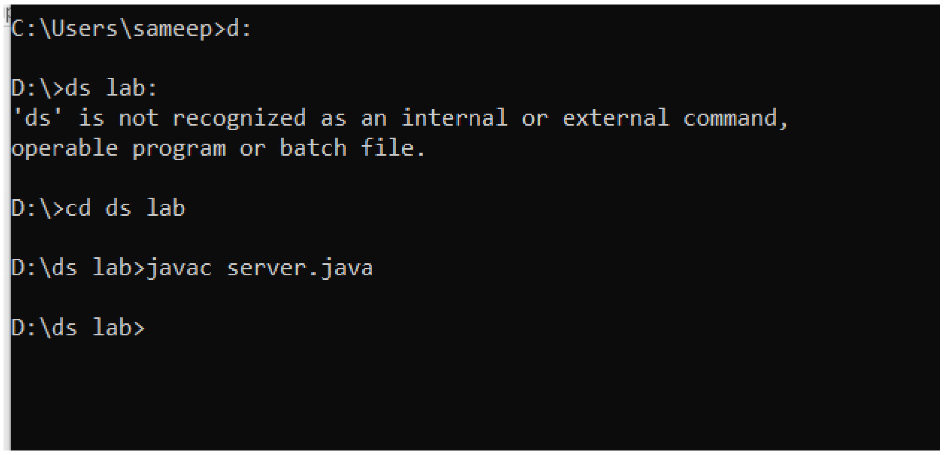
The Following results were observed when the code was executed

First the cliient side code was compiled.



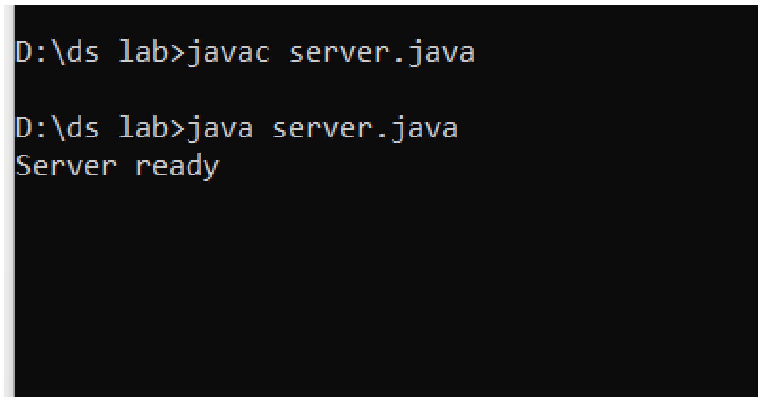
```
C:\Users\sameep>d:

D:\>cd ds lab

D:\ds lab>javac client.java

D:\ds lab>
```

Then the server side code was compiled.



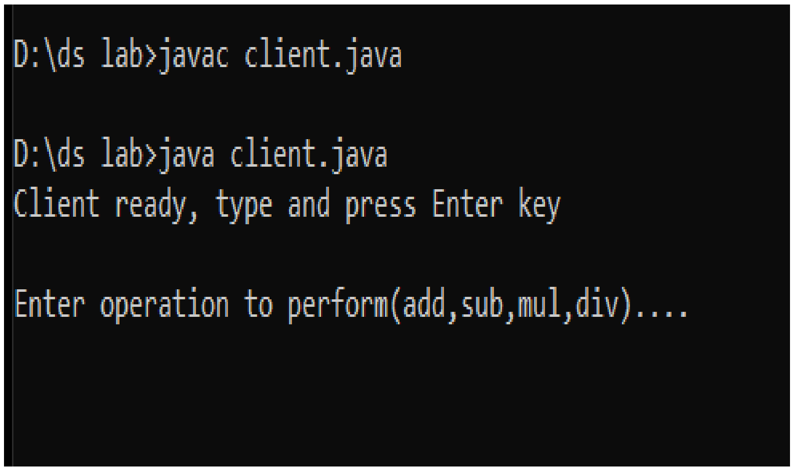
```
C:\Users\sameep>d:

D:\>ds lab:
'ds' is not recognized as an internal or external command,
operable program or batch file.

D:\>cd ds lab

D:\ds lab>javac server.java

D:\ds lab>
```

Then in a different terminal the server side code was executed.



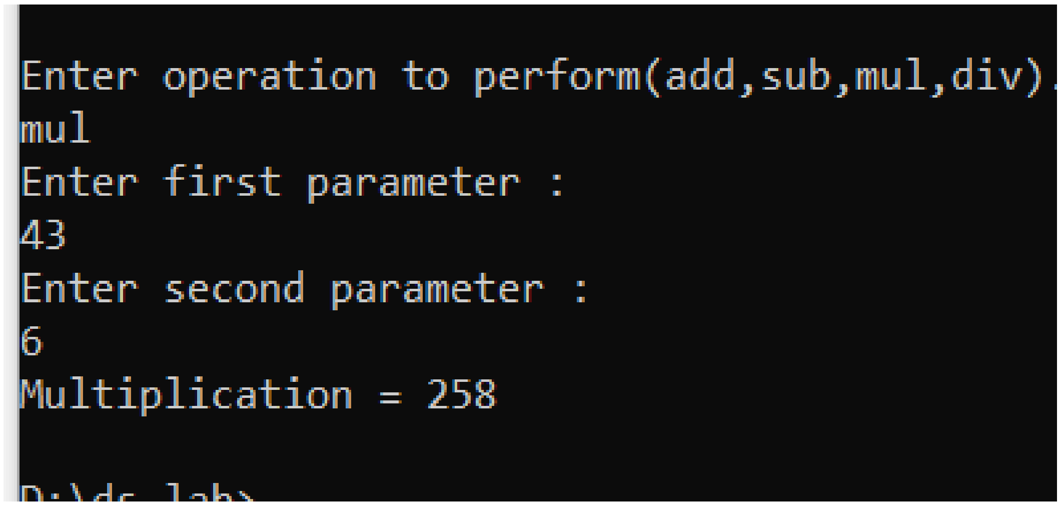
```
D:\ds lab>javac server.java

D:\ds lab>java server.java
Server ready
```

The client side code was also executed in a different terminal.



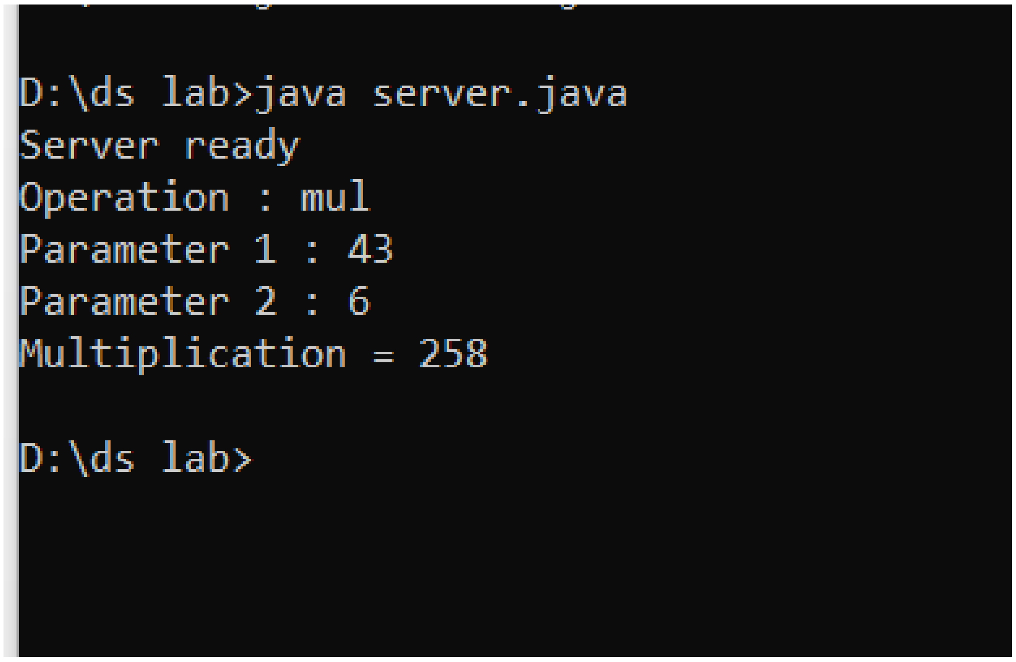
```
D:\ds lab>javac client.java

D:\ds lab>java client.java
Client ready, type and press Enter key

Enter operation to perform(add,sub,mul,div)....
```

Then the inputs were given to client side.



```
Enter operation to perform(add,sub,mul,div)
mul
Enter first parameter :
43
Enter second parameter :
6
Multiplication = 258

D:\ds lab>
```

Operation on the server was also observed.



```
D:\ds lab>java server.java
Server ready
Operation : mul
Parameter 1 : 43
Parameter 2 : 6
Multiplication = 258

D:\ds lab>
```

5. Discussion

We implemented a simple RPC using java and observed the following limitations:

- It is not a standard as it can be implemented in different ways.
- There is no flexibility in RPC for hardware architecture. It is only interaction based.
- Context switching increases scheduling costs.
- RPC does not solve the most of the distribution creation problems
- This mechanism is highly vulnerable to failure as it involves a communication system, another machine, and another process.

6. Conclusion

Hence, in this lab we implemented a simple RPC system using java which was used to take inputs from the client side and perform arithmetic operations on the given inputs using the given instructions from the server side which returned the result to the client. Observing this system we knew about various advantages and drawbacks of RPC.