CSE 660 LAB5

Submitted By : Arumugam Thendramil Pavai

Part I Remote Procedure Call

I did part I in my virtual machine with CentOS 7

[root@localhost ~]# $man rpcgen

```
usage: rpcgen infile
      rpcgen [-abkCLNTM][-Dname[=value]] [-i size] [-I [-K seconds]] [-Y
path] infile
      rpcgen [-c | -h | -l | -m | -t | -Sc | -Ss | -Sm] [-o outfile]
[infile]
      rpcgen [-s nettype]* [-o outfile] [infile]
      rpcgen [-n netid]* [-o outfile] [infile]
options:
-a          generate all files, including samples
-b          backward compatibility mode (generates code for SunOS 4.1)
-c          generate XDR routines
-C          ANSI C mode
-Dname[=value]   define a symbol (same as #define)
-h          generate header file
-i size           size at which to start generating inline code
-I          generate code for inetd support in server (for SunOS 4.1)
-K seconds  server exits after K seconds of inactivity
-l          generate client side stubs
-L          server errors will be printed to syslog
-m          generate server side stubs
-M          generate MT-safe code
-n netid    generate server code that supports named netid
-N          supports multiple arguments and call-by-value
-o outfile  name of the output file
-s nettype  generate server code that supports named nettype
-Sc         generate sample client code that uses remote procedures
-Ss         generate sample server code that defines remote procedures
-Sm         generate makefile template
-t          generate RPC dispatch table
-T          generate code to support RPC dispatch tables
-Y path           directory name to find C preprocessor (cpp)
For bug reporting instructions, please see:
<http://www.gnu.org/software/libc/bugs.html>.
```

XDR:

The External Data Representation (XDR) is a standard for description and encoding of data. XDR uses a language to describe data formats, but the language is used only for describing data and is not a

programming language. Protocols such as Remote Procedure Call (RPC) and the Network File System (NFS) use XDR to describe their data formats [1].

XDR not only solves data portability problems, it also permits the reading and writing of arbitrary C language constructs in a consistent and well-documented manner. Therefore, it makes sense to use the XDR library routines even when the data is not shared among machines on a network.

The XDR standard does not depend on machine languages, manufacturers, operating systems, or architectures [1]. This condition enables networked computers to share data regardless of the machine on which the data is produced or consumed. The XDR language permits transfer of data between different computer architectures and has been used to communicate data between such diverse machines.

Remote Procedure Call (RPC) uses XDR to establish uniform representations for data types in order to transfer message data between machines. For basic data types, such as integers and strings, XDR provides filter primitives that serialize, or translate, information from the local host's representation to XDR's representation. Likewise, XDR filter primitives deserialize XDR's data representation to the local host's data representation. XDR constructor primitives allow the use of the basic data types to create more complex data types such as arrays and discriminated unions.

XDR are compiled as following(the component name depends on the user) [2],

```
remote$ rpcgen dir.x
remote$ cc -c dir_xdr.c
remote$ cc rls.c dir_clnt.c dir_xdr.o -o rls -lnsl
remote$ cc dir_svc.c dir_proc.c dir_xdr.o -o dir_svc -lnsl
remote$ dir_svc
```

Here running rpcgen on dir.x generates four output files: the header file, dir.h; the client stub, dir_clnt.c; the server skeleton, dir_svc.c ,and the XDR routines in the file dir_xdr.c.

rand_client.c code:

```
double
radn_prog_1(char *host)
{
      CLIENT *clnt;
      void  *result_1;
      long  initialize_random_1_arg;
      double  *result_2;
      char *get_next_random_1_arg;

#ifndef     DEBUG
      clnt = clnt_create (host, RANDOM, RANDOM_VERSION, "udp");
      if (clnt == NULL) {
            clnt_pcreateerror (host);
            exit (1);
      }
#endif      /* DEBUG */
```

```
      result_2 = get_next_random_1((void*)&get_next_random_1_arg, clnt);
      if (result_2 == (double *) NULL) {
            clnt_perror (clnt, "call failed");
      }
#ifndef    DEBUG
      clnt_destroy (clnt);
#endif      /* DEBUG */
      return *result_2;
}

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

      if (argc < 2) {
            printf ("usage: %s server_host\n", argv[0]);
            exit (1);
      }
      host = argv[1];
      //random_1 (host);
      double x;
        int i;
        printf("\n twenty random numbers ");
        for ( i = 0; i < 20; ++i ){
                x = radn_prog_1 (host);
                printf(" %f, ", x );
        }
exit (0);
}
```

rand_server.c code:

```
#include "rand.h"
void *
initialize_random_1_svc(long *argp, struct svc_req *rqstp)
{
      static char*  result;
            return (void*) &result;
}

double *
get_next_random_1_svc(void *argp, struct svc_req *rqstp)
{        static double  result;
           result += 0.31;
           if ( result >= 1.0 )
```

```
            result -= 0.713;
            return &result;
}
```

5. Generating the program templates

-C is used for ANSI C mode

-a is used for generating all files, including samples

```
[root@localhost Lab5]# rpcgen -C -a rand.x

[root@localhost Lab5]# ls

Makefile.rand  rand_client.c  rand_clnt.c  rand.h  rand_server.c
rand_svc.c  rand.x

[root@localhost Lab5]# make -f Makefile.rand

cc -g    -c -o rand_clnt.o rand_clnt.c

cc -g    -c -o rand_client.o rand_client.c

cc -g     -o rand_client  rand_clnt.o rand_client.o -lnsl

cc -g    -c -o rand_svc.o rand_svc.c

cc -g    -c -o rand_server.o rand_server.c

cc -g     -o rand_server  rand_svc.o rand_server.o -lnsl

[root@localhost Lab5]# ./rand_server

[root@localhost Lab5]# ./rand_client localhost
```

Right now it doesn't return any message as there is no implementation (this part was before modification of the get_next_random_1_svc() function in rand_server.c and rand_client.c). After modification the following can be seen,

```
[root@localhost Lab5]# g++ -c rand_server.c

[root@localhost Lab5]# g++ -c rand_client.c

[root@localhost Lab5]# make -f  Makefile.rand

cc -g    -c -o rand_clnt.o rand_clnt.c

cc -g     -o rand_client  rand_clnt.o rand_client.o -lnsl

cc -g    -c -o rand_svc.o rand_svc.c
```

```
cc -g     -o rand_server  rand_svc.o rand_server.o -lnsl
```

Output:

```
[root@localhost Lab5]# ./rand_client localhost

twenty random numbers  0.310000,  0.620000,  0.930000,  0.527000,
0.837000,  0.434000,  0.744000,  0.341000,  0.651000,  0.961000,
0.558000,  0.868000,  0.465000,  0.775000,  0.372000,  0.682000,
0.992000,  0.589000,  0.899000,  0.496000,

[root@localhost Lab5]# ./rand_client mac1

twenty random numbers  0.310000,  0.620000,  0.930000,  0.527000,
0.837000,  0.434000,  0.744000,  0.341000,  0.651000,  0.961000,
0.558000,  0.868000,  0.465000,  0.775000,  0.372000,  0.682000,
0.992000,  0.589000,  0.899000,  0.496000,
```

8. **rand_server.c** Code to generate random numbers

```
#include "rand.h"
#include <time.h>
void *
initialize_random_1_svc(long *argp, struct svc_req *rqstp)
{                  static char*  result;
                 return (void*) &result;
}
double *
get_next_random_1_svc(void *argp, struct svc_req *rqstp)
{            static double  result;
      time_t now;
                struct tm *tm;
  /* Using time minute and second to generate random numbers*/
      now = time(0);
      static int i;
      if ((tm = localtime (&now)) ==NULL){
            i=1;
      }

  i = tm->tm_min + tm->tm_sec + i;

  result = (19 * i)%21;
      i+=1;
      return &result;
}
```

```
Output:
[root@localhost Lab5]# ./rand_client localhost

twenty random numbers  14.000000,  10.000000,  9.000000,  11.000000,
4.000000,  2.00000,  5.000000,  19.000000,  15.000000,  7.000000,
12.000000,  8.000000,  6.000000,  1.000000,  20.000000,  13.000000,
11.000000,  7.000000,  16.000000,  4.000000,
```

Part II Java Remote Method Invocation (RMI)

This part was done in the lab machines.

1. A "remote Java object" that performs simple addition of two numbers

SumInterface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface SumInterface extends Remote {
      String sum(double a, double b) throws RemoteException;
}
```

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.net.*;

public class Server implements SumInterface {

      public Server() {}


      public String sum(double a, double b) {
        try{
            double s = a+b;
            return Double.toString(s);
        } catch ( Exception e){
            return "Unable to add two numbers";
```

```
        }
    }

    public static void main(String args[]) {

        try {
        Server obj = new Server();
        SumInterface stub = (SumInterface)
UnicastRemoteObject.exportObject(obj, 0);

        //Bind the remote object's stub in the registry
        Registry registry = LocateRegistry.getRegistry();
        registry.bind("SumInterface", stub);

        System.err.println("Server ready");

    } catch (Exception e) {

        System.err.println("Server exception: " + e.toString());
        e.printStackTrace();
    }


    }
}
```

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    private Client() {}
    public static void main(String[] args) {
    String host = (args.length < 1) ? "127.0.0.1" : args[0];
    try{
        Registry registry = LocateRegistry.getRegistry(host);
        SumInterface stub = (SumInterface)
registry.lookup("SumInterface");
        double a = Double.parseDouble(args[1]);
        double b = Double.parseDouble(args[2]);
        String sum = stub.sum(a,b);
        //String response = stub.sayHello();
        System.out.println("response: " + sum);
    } catch (Exception e) {
```

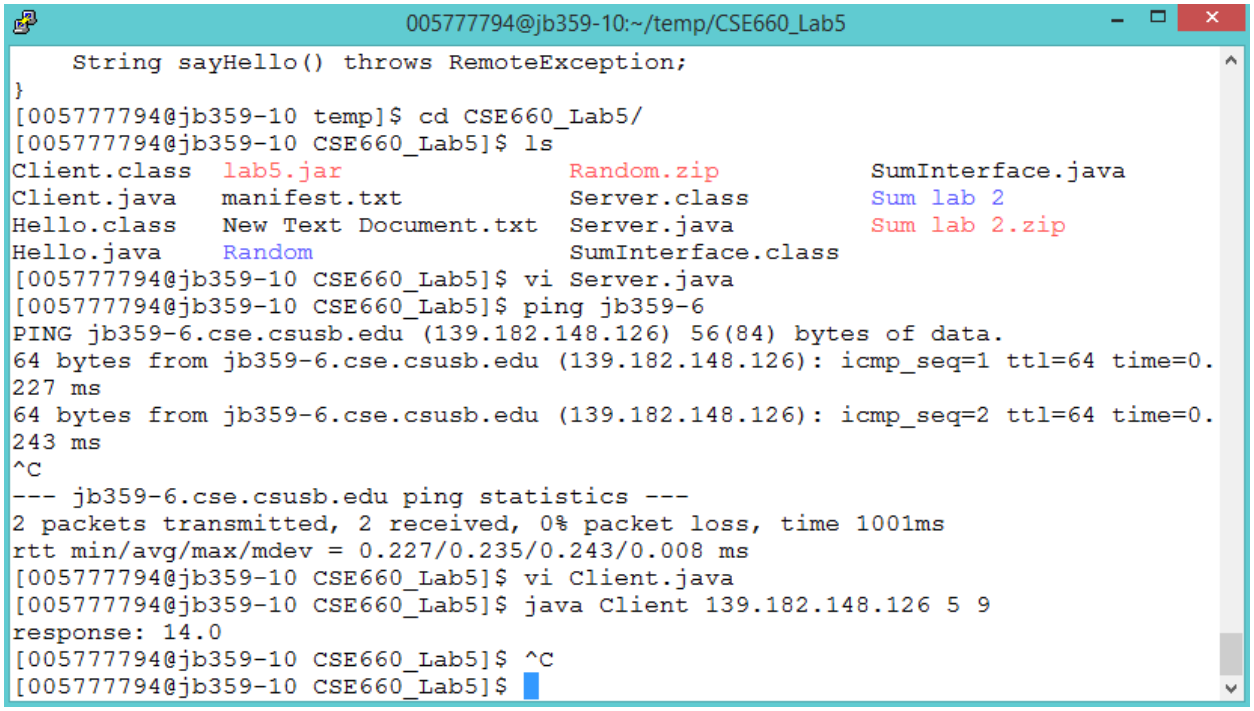
```
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Below commands I executed in Jb359-6 system (server) and I have created a jar file called lab5.jar,

```
[005777794@jb359-6 CSE660_Lab5]$ vi Server.java
[005777794@jb359-6 CSE660_Lab5]$ javac *.java
[005777794@jb359-6 CSE660_Lab5]$ jar cfm lab5.jar manifest.txt
SumInterface.class Server.class Client.class
[005777794@jb359-6 CSE660_Lab5]$ ps axuw | grep rmi
0057777+ 30453  0.2  0.4 4621296 32292 pts/0   Sl   15:03   0:00
/usr/java/jdk1.8.0_72/bin/rmiregistry
0057777+ 30576  0.0  0.0 116100  1244 pts/0    S+   15:06   0:00 grep
--color=auto rmi
[005777794@jb359-6 CSE660_Lab5]$ javac Server.java
[005777794@jb359-6 CSE660_Lab5]$ java Server
Server ready
```

I executed the below in the client machine JB359-10

```
[005777794@jb359-10 CSE660_Lab5]$ vi Server.java
[005777794@jb359-10 CSE660_Lab5]$ ping jb359-6
PING jb359-6.cse.csusb.edu (139.182.148.126) 56(84) bytes of data.
64 bytes from jb359-6.cse.csusb.edu (139.182.148.126): icmp_seq=1
ttl=64 time=0.227 ms
64 bytes from jb359-6.cse.csusb.edu (139.182.148.126): icmp_seq=2
ttl=64 time=0.243 ms
^C
--- jb359-6.cse.csusb.edu ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.227/0.235/0.243/0.008 ms
[005777794@jb359-10 CSE660_Lab5]$ vi Client.java
[005777794@jb359-10 CSE660_Lab5]$ java Client 139.182.148.126 5 9
response: 14.0
```



2. Implementation and testing a remote object that provides an interface to a "random number generator."

RandomInterface.java

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;

public interface RandomInterface extends Remote {
      String generateRandom(int n) throws RemoteException;
      String generateRandomBounds(int min, int max) throws
RemoteException;
}
```

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.util.Random;
import java.rmi.server.UnicastRemoteObject;
import java.net.*;

public class Server implements RandomInterface {

      public Server() {}


      public String generateRandom(int n) {
        try{
            Random rand = new Random();
            String response ="";
            for(int i=0;i<n;i++) {
                  response += Integer.toString(rand.nextInt(n) + 1) + "
";
            }
            return response;
        } catch ( Exception e){
            return "Unable to add two numbers";
        }
      }

      public String generateRandomBounds(int min, int max) {
        try{
            int range = (max - min) + 1;
            String response ="";
            for(int i=0;i<range;i++) {
                  response += Integer.toString((int)(Math.random() *
range) + min) + "  ";
            }
            return response;
```

```
        } catch ( Exception e){
            return "Unable to add two numbers";
        }
      }

      public static void main(String args[]) {

            try {
            Server obj = new Server();
            RandomInterface stub = (RandomInterface)
UnicastRemoteObject.exportObject(obj, 0);

            //Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("RandomInterface", stub);

            System.err.println("Server ready");

      } catch (Exception e) {

            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
      }
      }
}
```

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

      private Client() {}

      public static void main(String[] args) {

      String host = (args.length < 1) ? "127.0.0.1" : args[0];
      try{
            Registry registry = LocateRegistry.getRegistry(host);
            RandomInterface stub = (RandomInterface)
registry.lookup("RandomInterface");
            int n  = Integer.parseInt(args[1]);
            String response = "";
            if(args.length > 2) {
```

```
                int min = Integer.parseInt(args[1]);
                int max = Integer.parseInt(args[2]);
                response = stub.generateRandomBounds(min, max);
            }
            else {
                response = stub.generateRandom(n);
            }
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();

        }
}
}
```

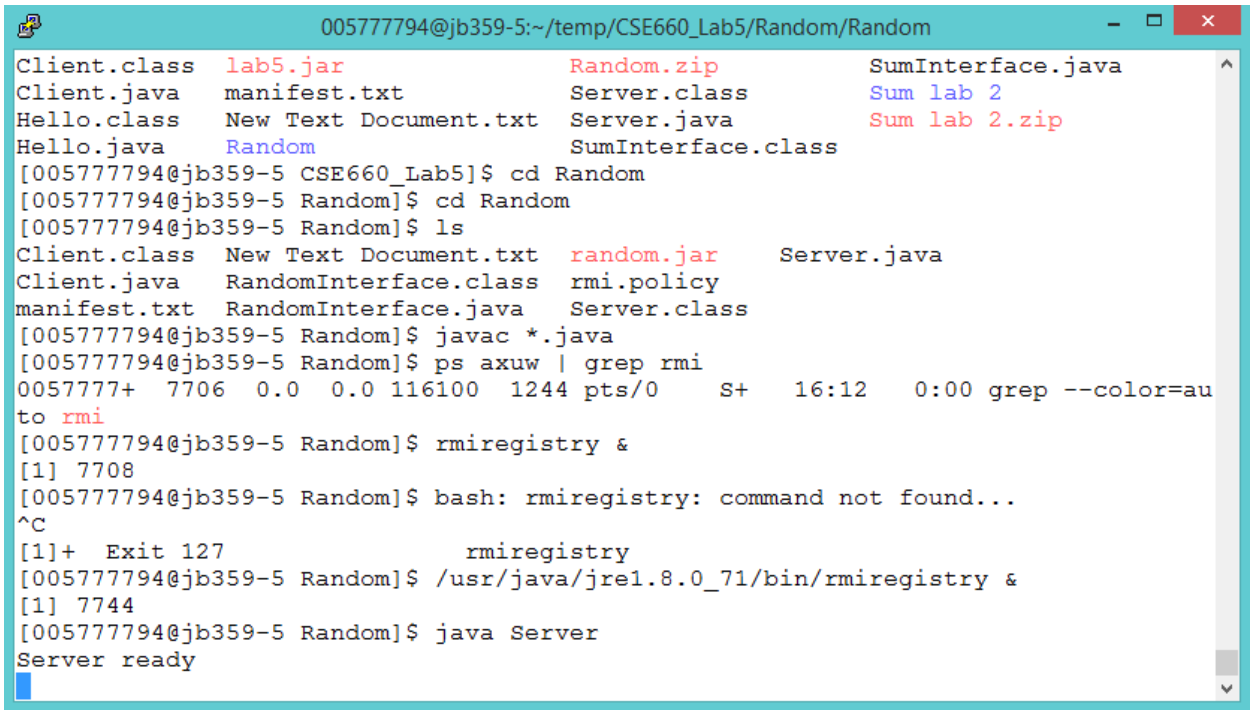
I executed the following commands in jb359-5 machine (server)

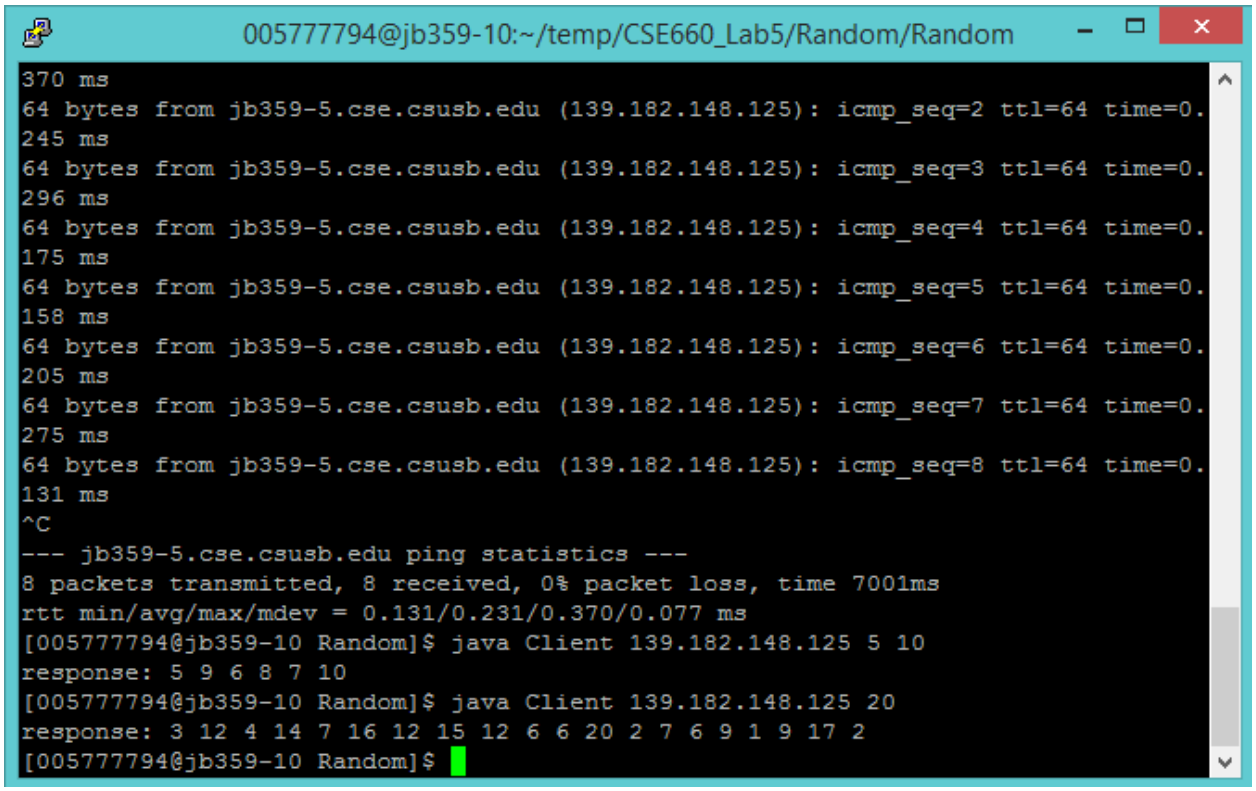
```
[005777794@jb359-5 Random]$ javac *.java
[005777794@jb359-5 Random]$ ps axuw | grep rmi
0057777+  7706  0.0  0.0 116100  1244 pts/0    S+   16:12   0:00 grep
--color=auto rmi
[005777794@jb359-5 Random]$ rmiregistry &
[1] 7708
[005777794@jb359-5 Random]$ bash: rmiregistry: command not found...
^C
[1]+  Exit 127                rmiregistry
[005777794@jb359-5 Random]$ /usr/java/jre1.8.0_71/bin/rmiregistry &
[1] 7744
[005777794@jb359-5 Random]$ java Server
Server ready
```

I executed the following commands in jb359-10 machine (client)

```
[005777794@jb359-10 Random]$ ping jb359-5
PING jb359-5.cse.csusb.edu (139.182.148.125) 56(84) bytes of data.
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=1
ttl=64 time=0.
370 ms
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=2
ttl=64 time=0.
245 ms
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=3
ttl=64 time=0.
296 ms
```

```
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=4
ttl=64 time=0.
175 ms
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=5
ttl=64 time=0.
158 ms
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=6
ttl=64 time=0.
205 ms
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=7
ttl=64 time=0.
275 ms
64 bytes from jb359-5.cse.csusb.edu (139.182.148.125): icmp_seq=8
ttl=64 time=0.
131 ms
^C
--- jb359-5.cse.csusb.edu ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7001ms
rtt min/avg/max/mdev = 0.131/0.231/0.370/0.077 ms
[005777794@jb359-10 Random]$ java Client 139.182.148.125 5 10
response: 5 9 6 8 7 10
[005777794@jb359-10 Random]$ java Client 139.182.148.125 20
response: 3 12 4 14 7 16 12 15 12 6 6 20 2 7 6 9 1 9 17 2
```





I have successfully completed all parts of this lab.

References:

For XDR:
[1]https://www.ibm.com/support/knowledgecenter/en/ssw_aix_61/com.ibm.aix.progcomc/ch4_xdr.htm
[2] https://users.cs.cf.ac.uk/Dave.Marshall/C/node34.html