CIS 457 - Lab 3: Working with TCP and UDP sockets programming

Due by: 9/25/2018
Total Points: 12 Points
Submission format: one hardcopy report per a group of 2 students

Lab Objectives:

The purpose of this lab is to:

- Practice with the basic TCP and UDP sockets programming
- Be familiar with the java.net, java.io and java.util network programming packages

Notes:

- Copy the server and the client sides of the applications studied in this lab to a folder in your home directory.
- Ignore any Exception handling related issues. We focus on the concepts related to network programming more than the Java language.
- Run both client and server programs on the same machine.
- Hit "CTRL + C" to close a client or a server session.

Question 1 **(3 points)**

Compile the given Java programs TCPEchoClient and TCPEchoServer such as follows:

javac TCPEchoServer.java and
javac TCPEchoClient.java

Then, test the application by sending a simple text after running the server and the client such as follows (use 2 separate terminals):

java TCPEchoServer *PortNumber* (In our example use port # 8888)
java TCPEchoClient localhost "Hi World" 8888

Then, answer the following questions:

a) Describe briefly, what this application is doing.
b) What happens if you use different port number at the client side to connect to the server from what the server uses?
c) Suppose you run TCPClient before you run TCPServer. What happens?
d) Test your app 3 times by starting 3 clients connections from the same terminal and record the port numbers that the client used. Are they the same in each connection and why?
e) Use the ps command with the appropriate switch to determine the Process ID for your application.
f) Use the netstat command with the appropriate switch to determine the TCP connection state for this process
g) Provide a screen capture for the testing process that shows the application's output in the 2 terminals.

Question 2 **(3 points)**

Compile and run the given two java programs TimeClient and TimeServer. You can run this app such as follows:

@ The Server Side
java TimeServer 1234
@ The Client Side
java TimeClient localhost 1234

a. Describe briefly, what this application is doing.
b. What is the return type of the read () method in this program?

c. Why is there a single stream attached to either the client or the server side in this application?
d. What is the type of I/O streams that is attached to the I/O sockets?
e. What are the limitations of using this type of I/O stream?
f. Provide a screen capture for the testing process that shows the application's output in the 2 terminals.

Question 3 **(2 points)**

Now, compile and run the same Time Server application using the two java programs TimeClient2 and TimeServer2.

a. How different is this application's implementation from the first one?
b. What is the type of I/O stream that is attached to the sockets?
c. In TimeClient2 and TimeServer2, what do these lines do?

```
DataInputStream in = new DataInputStream(new
BufferedInputStream(server.getInputStream()));

DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(client.getOutputStream()));
```

d. Which application implementation (TimeClient and TimeServer) or (TimeClient2 and TimeServer2) is recommended and why?

Question 4 **(2 points)**

Compile the Java programs UDPClient and UDPServer. Run the server using the command java UDPServer and run the client using the command java UDPClient. Then test the application by sending a simple text. Make sure to change the "hostname" in the UDPClient file to the proper name.

a) Provide a screen capture for the testing process that shows the application's output in the 2 terminals.
b) Suppose you run UDPClient before you run UDPServer. What happens?

Suppose that in the UDPClient.java we replace the line

```
DatagramSocket clientSocket = new DatagramSocket();
with
DatagramSocket clientSocket = new DatagramSocket(5432);
```

c) Will it become necessary to change UDPServer.java? and why?

Question 5 **(2 points)**

One of the classes within the *java.net* package is called InetAddress, which handles Internet addresses both as host names and IP addresses.

a) Compile the given IPFinder.java program and describe its functionality.
b) Modify the given *IPFinder.java* program so that it takes a list of host names from the command line and prints the host name and the IP address (s) for each host specified in the user input. The program should be able to retrieve the IP address of the local machine. (**Hint**: you need to use the *getAllByName(), getHostName() and getHostAddress()* methods to complete this part). Feel free to use the supplied code (listed below) and fill in the missing info **or** create your own one from scratch. The program output should look as it is shown in the figure below.
c) Provide a screenshot and the program code for the testing process. You can use any two domains that you may wish for the testing process.

```
C:\0\SD-8-31-2016\A1\Course Mats\CIS457\Labs\F-16\Lab03\Sol>java IPFinder2 www.gvsu.edu www.yaho
Local Hostname and its IP address: ELSAIDM-58E53A/192.168.1.107
Host IP address:
        192.168.1.107
Local Host Name:
        ELSAIDM-58E53A
www.gvsu.edu:
        www.gvsu.edu
        148.61.6.9
www.yahoo.com:
        www.yahoo.com
        98.138.252.30
        98.139.180.149
        98.138.253.109
        98.139.183.24
```

```
import java.net.*;  // for InetAddress

public class IPFinder2 {

    public static void main(String[] args) {

   // Get name and IP address of the local host
   try {
Step 1:    InetAddress address = InetAddress.?; // Gets the local host's IP address in object format

     System.out.println("Local Host:");
Step 2:    System.out.println("\t" + address.?);//  Gets the host name for this IP address.

   } catch (UnknownHostException e) {
    System.out.println("Unable to determine this host's address");
   }

   for (int i = 0; i < args.length; i++) {
    // Get name(s)/address(es) of hosts given on command-line
    try {

    // Create an array of InetAddress instances for the specified host
Step 3:    InetAddress[] addressList = InetAddress.?;

     System.out.println(args[i] + ":");

    // Print the first name and all associated IP addresses.  Assume array contains at least one entry.
Step 4:                System.out.println("\t" + addressList[0].?);
                       for (int j = 0; j < addressList.length; j++)
Step 5:                        System.out.println("\t" + addressList[j].?);

     } catch (UnknownHostException e) {
      System.out.println("Unable to find address for " + args[i]);
     }
    }
   }
}
```