CS 254 - Computer Networks

Lab 2: Investigating Application Layer Protocols with telnet and An Introduction to Socket Programming in Java

Due: Thursday, February 2 (class-time)

Purpose:

In this lab you will use `telnet` to access some simple TCP/IP services. Then you will write Java programs that use *sockets* to access those same services.

Background:

`http` and `ftp` are two of the application layer protocols that are part of the TCP/IP protocol suite. A *server* is a program that provides a *service* at a *port* on a *host* using a *protocol*. For example, a web server is a program that serves web pages using the `http` protocol at port 80. There are a number of "well-known" ports that application programs (*clients*) can connect to. An application layer protocol, such as `http` or `ftp`, is associated with each well-known port and an application program (a client) must use the correct protocol to communicate with a server via a port.

Ordinarily one uses a program (client) to make a connection to and interact with a server, e.g. a web browser or an ftp client. However, if one knows the protocol, that is, the commands recognized, etc., then one can connect to a port using *telnet* and give commands to the server and receive responses from the server that listens on that port.

Instructions:

1. Read RFC 867 about the *daytime* service/protocol. What *well-known port* provides the daytime service? Briefly describe the service/protocol.

Use `telnet` to obtain this service from `kant`. To do this open a command window and type the command

```
telnet kant <port number>
```

When you initially do this you will see an error message to the effect that the telnet command cannot be found. The reason is that Windows 7 machines are by default locked down fairly tightly. To enable telnet you need to open the **Control Panel**, select **Programs and Features**, select **Turn Windows features on or off**, and then check the **Telnet Client**. (You may need your instructor's credentials to access the setting.) Now if you return to the command window you should be able to execute the telnet command described above.

Place a copy of the results you get into your lab write-up. (Your instructor can show you how to copy text from a command window so that it can be placed into another document.)

Is this service available on the Windows server `cs-gamma`? Place a copy of the error message you get into your lab write-up.

Now see if this service is available on an unoccupied nearby lab machine. Report the error message you get.

We can fix this. Login to the unoccupied nearby lab machine and go into **Turn Windows features on or off** and check both **Simple TCPIP (i.e. echo, daytime etc.)** and **Telnet Server** services. (Again, you may need your instructor's credentials to do this.) After doing this reboot the machine. Now return to the command window in your original machine and you should be able to access the daytime service from that lab machine.

Place a copy of the results you get into your lab write-up.

2. Read RFC 862 about the *echo* service/protocol. What *well-known port* provides the echo service? Briefly describe the service/protocol.

   Use `telnet` to obtain this service from `kant`. (Use '`Ctrl+]`' then '`quit`' to exit.) Place a copy of the results you get into your lab write-up.

   Is this service available on the Windows server `cs-gamma`? Place a copy of the results you get into your lab write-up.

   Is this service available on your second lab machine? Place a copy of the results you get into your lab write-up.

3. Investigate the service that is available on port 17. What is the name of the service? What RFC specifies this service?

   Use `telnet` to obtain this service from `kant`. Place a copy of the results you get into your lab write-up.

   Is this service available on the Windows server `cs-gamma`? Place a copy of the results you get into your lab write-up.

   Is this service available on your second lab machine? Place a copy of the results you get into your lab write-up.

Background:

A *socket* is one end of a two-way connection between a program and a remote program on the network. Java provides two socket classes: `Socket` for clients and `ServerSocket` for servers. In this lab we are only interested in clients so we will use the `Socket` class.

There are five operations you can perform with a socket:

1) Create a socket using the `Socket` constructor.

2) Connect to a remote host using a socket. (This is usually done in the socket's constructor rather than as a separate step.)

3) Send data to a remote application over a socket.

4) Receive data from a remote application over a socket.

5) Close a socket.

What is important is that once a connection is made with a socket, input and output streams can be opened which can be read from or written to as with any other input or output stream, e.g. the streams used to read from or write to external files.

Instructions:

4. Obtain a copy of the Java program `LookForPorts.java`. Examine the program. It attempts to open sockets to each of ports 0 - 99 on host `kant`.

   Modify the program to obtain the host name from the user instead of having it "hardwired" in the code.

   Place a copy of the output from the program querying `kant` into your lab write-up.

   Place a copy of the output from the program querying your second lab machine into your lab write-up.

5. The `Socket` class provides a number of methods for obtaining information about the connection. If you recall from class a connection is identified by four numbers: the IP address and port number of each end of the connection. Here are the methods that let you find these numbers.

   `getInetAddress()` – gets IP address of remote host

   `getPort()` – gets port number of connection on remote host

   `getLocalAddress()` – gets IP address of local host

   `getLocalPort()` – gets port number of connection on local host

Write a Java program `GetSocketInfo` that opens a socket to a host that runs a web server. Obtain the host name (only) from the user (e.g. `www.yahoo.com`). Make sure you open the socket on port 80. Call the above functions and display the information obtained with appropriate labels. Close the socket. Catch the following exceptions (in this order):
`UnknownHostException`, `SocketException`, `IOException`.
In addition to displaying the four numbers requested, display the socket using System.out.println() (after the socket is opened).

Copy the results of two executions of the program, one for each of the hosts `www.sbu.edu` and `www.eclipse.org`. Paste these results into your lab write-up. Hand in a copy of your Java program with your lab write-up.

6. Write a Java program `DaytimeClient` that opens a socket to the *daytime* service on a host. Follow the steps:

    1) Open a socket to the host on the proper port.

    2) Create an InputStream from the socket, using the `getInputStream()` method. Wrap the InputStream in an InputStreamReader, using the InputStreamReader constructor. Finally wrap the InputStreamReader in a BufferedReader. (The benefit of using a BufferedReader to read text is that one can read a line at a time using the BufferedReader readLine() method.)

    3) Read the stream from the socket (using the BufferedReader) and display it. Note: for the daytime service you are assured that the stream will consist of a single line. Reading is a bit more involved if you are reading an indeterminate number of lines.

    4) Close everything.

    The program should display the socket information (as in the previous step) as well as the result returned by the daytime service.

    Copy the result of executing the program using the host `kant` and paste the result into your lab write-up. Hand in a copy of your Java program `DaytimeClient.java` with your lab write-up.

    Copy the result of executing the program using the remote host `cs-gamma` and add this to your lab write-up as well.

    Copy the result of executing the program using your second lab machine as the remote host and add this to your lab write-up as well.

7. Write a Java program `QoDClient` that opens a socket to port 17 on a remote host. As in the previous steps the program should display the socket information as well as the result returned by the service.

    In this program you will be required to continue reading from the buffered reader until there is no more data to read. The standard way to code this is to write a while loop that has the form:

```
while ((line = br.readLine()) != null)
{
    <process data in line>
}
```

Execute the program using the host `kant` and then execute it again using your second lab machine. Place the results of the executions into your lab write-up. Hand in a copy of your Java program `QoDClient.java` with your lab write-up.

Help Policy:

Help Policy in Effect for This Assignment: Group Project with Limited Collaboration

In particular, you may discuss the assignment and concepts related to the assignment with the following persons, in addition to an instructor in this course: any member of your group; any St. Bonaventure Computer Science instructor; and any student enrolled in CS 254.

You may use the following materials produced by other students: materials produced by members of your group.