Programming Assignment 1: Socket Programming

In this assignment, you'll write a client that will use sockets to communicate with a server that you will also write. Here's what your client and server should do:

Your client should first accept an integer between 1 and 100 from the keyboard, open a TCP socket to your server and send a message containing (i) a string containing your name (e.g., "Client of John Q. Smith") and (ii) the entered integer value and then wait for a sever reply.

Your server will create a string containing its name (e.g., "Server of John Q. Smith") and then begin accepting connections from clients. On receipt of a client message, your server should

i. print (display) the client's name (extracted from the received message) and the server's name
ii. itself pick an integer between 1 and 100 (it's fine for the server to use the same number all the time) and display the client's number, its number, and the sum of those numbers
iii. send its name string and the server-chosen integer value back to the client
iv. if your server receives an integer value that is out of range, it should terminate after releasing any created sockets. You can use this to shut down your server.

Your client should read the message sent by the server and display its name, the server's name, its integer value, and the server's integer value, and then compute and the sum. The client then terminates after releasing any created sockets. As an aside (and as a check that you are doing things correctly, you should make sure for yourself that the values and the sums are correct!)

OK. You've got the basic assignment done. The last part of the assignment should be a fun, social thing to do. Team up with someone from the class (advertise on the Piazza class list if you are looking for a partner) and get your client to interact with their server or vice versa. If you've got a server up and running, you can advertise your server's services to anyone in the class (you're not allowed to charge for service!), and if you've got a client, all you need to do is interact with such an advertised server.

Note that if you've got your own client and server running, there isn't any more programming involved – just running your client and server with someone else's. This shouldn't actually be very hard at all! For this part of the assignment, you need only hand in the output from your "wide" – the client or the server, since you'll be printing out the name of the other student's client or server with whom your client or server is interacting. Recall that when we discussed RFC standards for protocols we noted that the IETF requires that two independent implementations of a protocol must interoperate; that's what you are doing here. Again, make sure to check that your numbers and sums are correct.

Please read the information posted on the class WWW site (http://gaia.cs.umass.edu/cs453_fall_2013/ pa_hand_in.htm) **about what to hand with your**

assignment. Note that a short design document is required. You should program your client and server to each print an informative statement whenever it takes an action (e.g., sends or receives a message, detects termination of input, etc.), so that you can see that your processes are working correctly (or not!). This also allows the TA to also determine from this output if your processes are working correctly. You should hand in screen shots (or file content, if your process is writing to a file) of these informative messages as well as the required output of the client and server (name strings, integer values and sums)

Programming Languages and Operating Systems

You may write your programs in Java, C++, Python, or C on any platform (Liux/unix, Mac, PC) you want; you have accounts in the edlab. Other languages are OK, but ask me first. You can do this assignment on a linux/unix computer, a Mac or a PC. See http://www.lowtek.com/sockets/ for links to tutorials on the WINSOCK API, which we haven't covered in class.

Programming the assignment in JAVA

- Our discussion in class focused on the Python API, but you can use JAVA if you prefer. Here is the JAVA-based socket section of the text that is from the 5th edition: http://gaia.cs.umass.edu/cs453_fall_2013/hw_pa_labs/K_R_sockets_in_Java.pdf Java encapsulates the concept of a client-side connection-oriented (TCP) socket with the class, `Socket`. Details about the socket class, and the other classes noted below, can be found at http://download.oracle.com/javase/1.4.2/docs/api/java/net/Socket.html
- Java encapsulates the concept of a server-side connection-oriented socket with the class, `ServerSocket`. You'll need to use the `accept()` and `close(0)` methods of this class, which are also document at this URL.
- If you're interested in a quick Java tutorial targeted specifically at socket programming, check out "Socket Programming in Java: a tutorial," by Q. Mahmoud, Javaworld, Dec. 1996, http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets_p.html. Sun's socket tutorial is at http://java.sun.com/docs/books/tutorial/networking/sockets/index.html.

Programming the assignment in C

If you choose to write your programs in C, you'll need to master the C system calls needed for socket programming. Perosnally, C is my favorite language for network programming since it gets you closest to the operating system's socket-related system calls. These include *socket(), bind(), listen(), accept(), connect(),* and *close().* For a quick text-only tutorial of socket programming specifically under Linux, see http://www.lowtek.com/sockets/. Here's another nice tutorial: http://beej.us/guide/bgnet/

Programming the assignment in Python.

The 6th edition of the text (and what we've discussed in class) uses sockets in Python. A Python socket tutorial is http://docs.python.org/howto/sockets.html

Programming notes

Here are a few tips/thoughts to help you with the assignment:

- You must chose a server port number greater than 1023 (to be safe, choose a server port number larger than 5000). If you want to explicitly choose you client-side port, also choose a number larger than 5000.
- You may need to know your machine's IP address, when one process connects to another. You can telnet to your own machine and seeing the dotted decimal address displayed by the telnet program. You can also use the UNIX nslookup command . On Windows, see the ipconfig utility. On a Mac, you can run the terminal program and use the ifconfig commend (just type in `ifconfig` or `ifconfig | grep "inet ")`.
- Many of you will be running the clients and senders on the same UNIX machine (e.g., by starting up the receiver and running it in the background, then starting up the relay in the background, and then starting up the sender. This is fine; since your using sockets for communication these processes can run on the same machine or different machines without modification. Recall the use of the ampersand to start a process in the background. If you need to kill a process after you have started it, you can use the UNIX kill command. Use the UNIX ps command to find the process id of your server
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use.

For extra credit, rewrite your server to be a concurrent server - that is a server that waits on the welcoming socket and then creates a new thread or process to handle the incoming request (i.e., carry out the username verification and message-of-the-day protocol described above). If you're doing the assignment in C, you'll need to learn about the *fork()* system call. Here is an example that will get you going: http://www.tutorialspoint.com/unix_sockets/socket_server_example.htm If you're doing the assignment in Java, see the section "Supporting Multiple Clients" in Sun's socket programming tutorial, http://java.sun.com/docs/books/tutorial/networking/sockets/clientServer.html. Here's a page that will get you started that will get you started in Python: http://ilab.cs.byu.edu/python/threadingmodule.html and http://www.eurion.net/python-snippets/snippet/Threaded%20Server.html. If you find better pages for C, Java or Python, please share with the class!