Assignment 1

Computer Networks (CS 456)

Fall 2012

Introductory Socket Programming

Due Date: Thursday, October 18th 2012, at midnight (11:59 PM)

Work on this assignment is to be completed individually

1 Assignment Objective

The goal of this assignment is to gain experience with both TCP and UDP socket programming in a client-server environment (Figure 1). You will use `Java` to design and implement a client program (`client`) and a server program (`server`) to communicate between themselves.

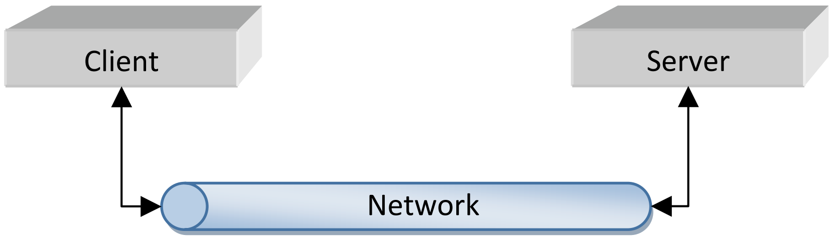

FIGURE 1

2 Assignment Specifications

2.1 Summary

In this assignment, the client will send requests to the server to reverse strings (taken as a command line input) over the network using sockets.

This assignment uses a two stage communication process. In the *negotiation stage*, the client and the server negotiate on a random port (`<r_port>`) for later use through a fixed negotiation port (`<n_port>`) of the server. Later in the *transaction stage*, the client connects to the server through the selected random port for actual data transfer.

Note: The assignment description in the following assumes an implementation in `Java`.

2.2 Signaling

The signaling in this project is done in two stages as shown in Figure 2.

Stage 1. **Negotiation using TCP sockets**: In this stage, the client creates a TCP connection with the server using `<server_address>` as the server address and `<n_port>` as the negotiation port on the server (where the server is listening). The client sends a request to get the random port number on the server where it will send the actual request. For simplicity, the client will send an integer (e.g., 13) to initiate negotiation.

Once the server gets this request in the negotiation port, it will reply back with a random port number `<r_port>` where it will be listening for the actual request.

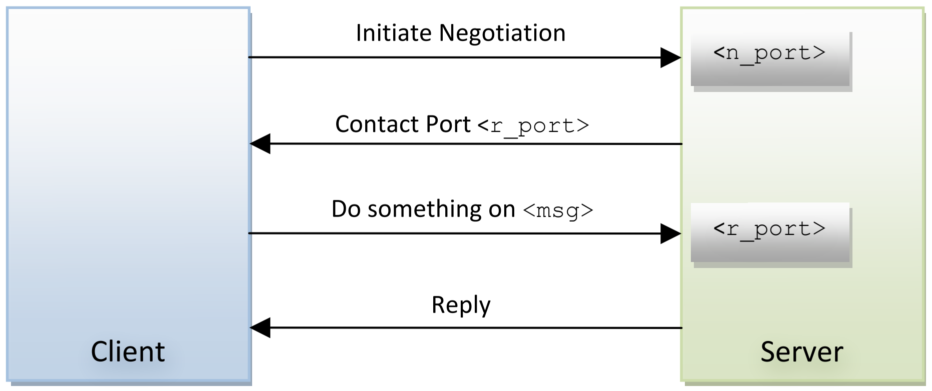

FIGURE 2

Stage 2. **Transaction using UDP sockets**: In this stage, the client creates a UDP socket to the server in `<r_port>` and sends the `<msg>` containing a string. On the other side, the server receives the string and sends the reversed string back to the client. Once received, the client will print out the reversed string.

2.3 Client Program (`client`)

You should implement a client program, named `client`. It will take the command line inputs: `<server_address>`, `<n_port>`, and `<msg>` in the given order.

2.4 Server Program (`server`)

You should also implement a server program, named `server`. There are no inputs to the server. Note that, the server **must** print out the `<n_port>` value that it is using so that the information can be used in the client program.

2.5 Example Execution

- On **host1**: `server`
- On **host2**: `client <host1/server address> <n_port>` "A man, a plan, a canal—Panama!"

3 Hints

You can use the sample codes of TCP/UDP socket programming in Java from the Module 2 slides (pages 94-112).

Below are some points to remember while coding/debugging to avoid trivial problems.

- Use port id greater than 1024, since ports 0-1023 are already reserved for different purposes (e.g., HTTP @ 80, SMTP @ 25)
- If there are problems establishing connections, check whether any of the computers running the server and the client is behind a firewall or not. If yes, allow your programs to communicate by configuring your firewall software.
- Make sure that the server is running before you run the client.
- Also **remember** to print the `<n_port>` where the server will be listening and make sure that the client is trying to connect to that same port for negotiation.

- If both the server and the client are running in the same system, 127.0.0.1 (i.e., localhost) can be used as the destination host address.
- You can use help on Java network programming from any book or from the Internet, if you properly refer to the source in your programs. But remember, you cannot share your program or work with any other student.

4 Procedures

4.1 *Due Date*

The assignment is due on Thursday, October 18th 2012, at midnight (11:59 PM).

4.2 *Hand in Instructions*

Submit all your files in a single compressed file (.zip, .tar etc.) using LEARN in dedicated Dropbox.

You must hand in the following files / documents:

- *Source code* files.
- *Makefile*: your code **must** compile and link cleanly by typing "*make*" or "*gmake*".
- *README* file: this file ***must*** contain instructions on how to run your program, which undergrad machines your program was built and tested on, and what version of *make* and *compilers* you are using.

Your implementation will be tested on the machines available in the **undergrad environment**.

4.3 *Documentation*

Since there is no external documentation required for this assignment, you are expected to have a reasonable amount of internal code documentation (to help the markers read your code).

You **will** lose marks if your code is unreadable, sloppy, inefficient, or not modular.

4.4 *Evaluation*

Work on this assignment is to be completed individually.