Assignment 1

Computer Networks (CS 456), Fall 2018

Introductory Socket Programming

Due Date: Wednesday, October 3, 2018, at midnight (11:59 PM)

Use piazza for all communication

Work on this assignment is to be completed individually

1 Assignment Objective

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

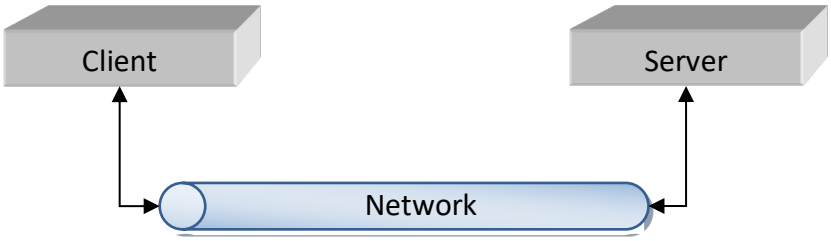

FIGURE 1

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 negotiation port (`<n_port>`) of the server. Later in the *transaction stage*, the client connects to the server through the selected random port (`<r_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 UDP sockets**: In this stage, the server creates a UDP socket with `<n_port>` and start listening on this port. The client creates a UDP socket 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 (i.e., the string to be modified). To initiate this negotiation, the client sends a request code (`<req_code>`), an integer (e.g., 13), through the

UDP socket. If the client fails to send the intended <req_code>, the server does nothing. Once the server verifies the <req_code>, the server creates a TCP socket with a random port number <r_port> where it will be listening for the actual request in the transaction stage. It then uses the UDP socket to reply back with <r_port>. After receiving this <r_port>, the client sends another UDP message to the server to confirm <r_port>. The server then acknowledges that the received <r_port> at the client is correct using a UDP message. After receiving the acknowledgement from the server, the client closes the UDP socket with the server.

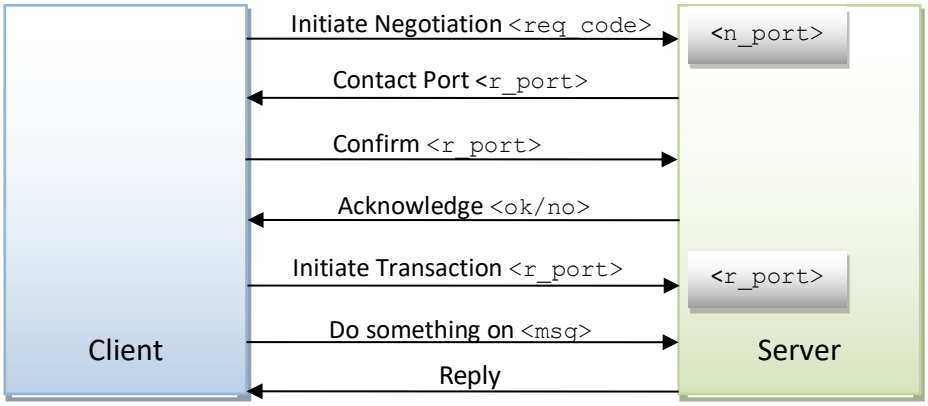

FIGURE 2

Stage 2. **Transaction using TCP sockets**: In this stage, the client creates a TCP connection to the server at <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 prints out the reversed string and exits. Note that the server should continue listening on its <n_port> for subsequent client requests. For simplicity, we assume, there will be only one client in the system at a time. So, the server does not need to handle simultaneous client connections.

2.3 Server Program (`server`)

You will implement a server program, named `server`. The server will take <req_code> as a command line parameter. The server **must** print out the <n_port> value in the following format as the first line in the stdout:

```
SERVER_PORT=<n_port>
```

For example, if the negotiation port of the server is 52500, then the server should print:

```
SERVER_PORT=52500
```

The server should also print out the randomly generated <r_port> value and the <msg> string it receives from the client in the following format:

```
SERVER_TCP_PORT=34566
SERVER_RCV_MSG='A man, a plan, a canal—Panama!'
```

2.4 *Client Program (`client`)*

You will implement a client program, named `client`. It will take four command line inputs: `<server_address>` , `<n_port>`, `<req_code>`, and `<msg>` in the given order.
The client should print out the reversed <msg> string it receives from the server in the following format:

```
CLIENT_RCV_MSG='!amanaP—lanac a ,nalp a ,nam A'
```

2.5 *Example Execution*

Two shell scripts named **server.sh** and **client.sh** are provided. Modify them according to your choice of programming language. You should execute these shell scripts which will then call your client and server programs.

- Run server: `./server.sh <req_code>`
- Run client: `./client.sh <server address> <n_port> <req_code>` 'A man, a plan, a canal—Panama!'

3 Hints

You can use the sample codes of TCP/UDP socket programming in Python from Section 2.7 in the textbook.

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 Wednesday, October 3, 2018, 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.
- Modified server.sh and client.sh scripts.

Your implementation will be tested on the machines available in the **undergrad environment (accessible via *linux.student.cs.uwaterloo.ca*)**.

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 or unmodular.

4.4 Evaluation

Work on this assignment is to be completed individually.

5 Additional Notes:

1. You have to test both `<n_port>` and `<r_port>` are not occupied. Note that just selecting a random port does not ensure that the port is not being used by other program.
2. All codes must be tested in undergrad environment prior to submission.
    a. Run client and server in two different student.cs machines
    b. Run both client and server in a single student.cs machine
3. Make sure that no additional (manual) input is required to run any of the server or client.