Assignment 1

CS456/656 Computer Networks

Introductory Socket Programming

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 `any programming language` to design and implement a client program (`client`) and a server program (`server`) to communicate with each other.

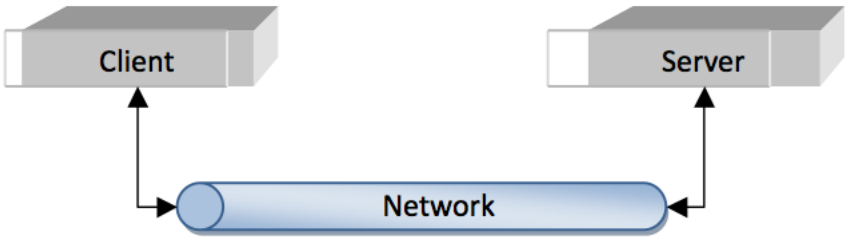

FIGURE 1

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.

2.2 Signalling

The signalling 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 from the server where it will send the actual request (i.e., the string to be reversed). To initiate this negotiation, the client sends a request code (`<req_code>`), an integer (e.g., 13), after creating the TCP connection. If the client fails to send the intended `<req_code>`, the server closes the TCP connection.

> Once the server verifies the `<req_code>`, it replies back with a random port number `<r_port>` where it will be listening for the actual request. After receiving this `<r_port>`, the client closes the TCP connection with the server.

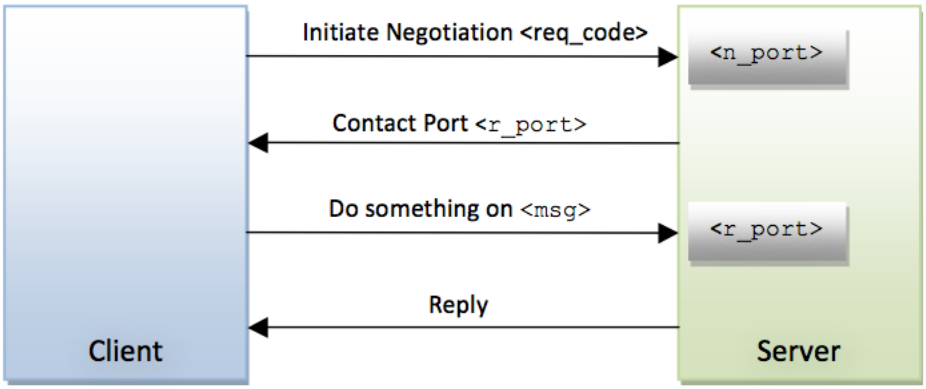

FIGURE 2

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 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 Client Program (`client`)

You should 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.

2.4 Server Program (`server`)

You should also 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 standard output (e.g. `stdout`):

```
SERVER_PORT=<n_port>
```

The server **must also** print out the `<n_port>` value in the following format into `port.txt`, a file in the same directory as the `server.sh` file (see hand-in instructions for details):

```
SERVER_PORT=<n_port>
```

The server should finish writing to `port.txt` and close the file in 3 seconds after it's started.

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

```
SERVER_PORT=52500
```

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

Note to Python users: Please do not write Python code in these two files.

3 Hints

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

Submit all your files in a single ZIP compressed file (.zip) using Assignment 1 drop box in LEARN. You must hand in the following files / documents:

- *Source code* files.
- *Makefile*: your code **must** compile and link cleanly by typing "*make*".
- *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.

Important: Your Makefile, server.sh and client.sh should not be inside any folder in your ZIP file. If you put your source code in a folder, then your Makefile, server.sh and client.sh should be able to build / call your programs without any manual intervention. You can use `unzip -l` to inspect the directory

structure inside your ZIP file. It is highly recommended to test your ZIP file with the script at https://github.com/duanqn/CS456FormatTest before submitting it.

Your implementation will be tested on the machines available in the **undergrad environment** https://cs.uwaterloo.ca/cscf/domain/CS-TEACHING/hosts

4.2 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 graders read your code).

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

4.3 Evaluation

Your assignment will be graded based on the marking scheme under Assignment 1 module in LEARN. Work on this assignment is to be completed individually.

5 Additional Notes:

1. You have to ensure that both `<n_port>` and `<r_port>` are available. Just selecting a random port does not ensure that the port is not being used by another program.
2. All codes must be tested in the `linux.student.cs` environment prior to submission.
    1. Run client and server in two different `student.cs` machines
    2. 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`.