Programming Assignment 1

CS 436/636 Winter 2024

Socket Programming

Due Date: March 1, 2024, 11:59PM

Work on this assignment is to be completed individually

1 Objective

The goal of this assignment is to gain experience with both TCP and UDP socket programming in a client-server environment (see Figure 1). You will use `Python` or any other programming language to design and implement a client program (`client`) and a server program (`server`) to communicate with each other.

Client
Server
Network

FIGURE 1

2 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, through a fixed negotiation port (<`n_port`>) of the server, a random port (<`r_port`>) for later use. Later in the transaction stage, the client connects to the server through the negotiated random port (<`r_port`>) for actual data transfer.

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 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 and continues listening on `<n_port>` for subsequent client requests. Once the server verifies the `<req_code>`, it replies 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.

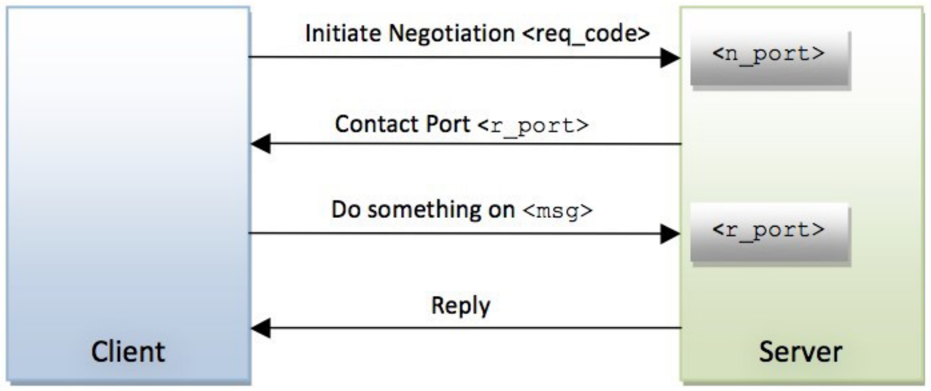

FIGURE 2

Stage 2. Transaction using UDP sockets: In this stage, the client creates a UDP socket and sends the `<msg>` containing a string to the server on `<r_port>`. 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 that 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 `<n_port>` and `<req_code>` as a command line parameters.

2.5 Example Execution

- Run server: `python server.py <n_port> <req_code>`
- Run client: `python client.py <server address> <n_port> <req_code> 'A man, a plan, a canal—panama!'`

3 Hints

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

- You can use and adapt the sample codes in the SOCKET PROGRAMMING slides.
- Use port number greater than 1024, since ports 0-1023 are already reserved for different purposes (e.g., HTTP @ 80, SMTP @ 25).
- Make sure that the `server` is running before you run the `client`.
- You must be able to run the `client` and the `server` on two different machines in the `linux.student.cs` environment.
- If both the `server` and the `client` are running in the same system, 127.0.0.1 or 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 another student.

4 Procedures

4.1 Due Date

The assignment is due on **March 1, 2024, 11:59PM**.

4.2 Hand in Instructions

Submit all your files in a single compressed file (i.e., .zip, .tar, etc.) using LEARN in the dedicated dropbox. The filename should include your username and/or student ID. You must hand in the following files / documents:

- Source code files.
- README file: This file must contain instructions on how to run your program, which `linux.student.cs` 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 `linux.student.cs` 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, or not efficient.

4.4 Evaluation

Work on this assignment is to be completed individually.

5 Additional Notes

- You must 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.
- All codes must be tested in the `linux.student.cs` environment prior to submission.
- Make sure that no additional (manual) input is required to run the `server` or `client` other than what has been specified.