Assignment 1

Computer Networks (CS 456 / CS 656)

Spring 2023

Introductory Socket Programming

Due Date: Friday, June 09, 2023, at midnight (11:59 PM)

Work on this assignment is to be completed individually

1 Assignment Objective

The goal of the 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 Assignment Specifications

2.1 Summary

In this assignment, the client will send requests over the network using sockets to the server, to check whether a string (taken as a command-line input) is a palindrome or not. The server will limit the number of messages a client can send. If the number of messages sent by the client is more than the limit (`<req_lim>`), the server will process the first `<req_lim>` messages. If the number of messages is less than or equal to the limit, all messages will be processed.

This assignment uses a two-stage communication process. In the *negotiation stage*, the client and the server negotiate a random port (`<r_port>`) for later use. The negotiation will be achieved 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 the actual data transfer.

For checking the palindrome, the server should ignore punctuation and convert characters to lowercase.

2.2 Signalling

The signalling in the assignment will be done in two stages as shown in Figure 2(a) and 2(b).

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 then send the actual requests (i.e., the strings to be checked for palindromes). 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 with a random port number <r_port>, where it will be listening for the actual requests. After receiving this <r_port>, the client closes the TCP connection with the server.

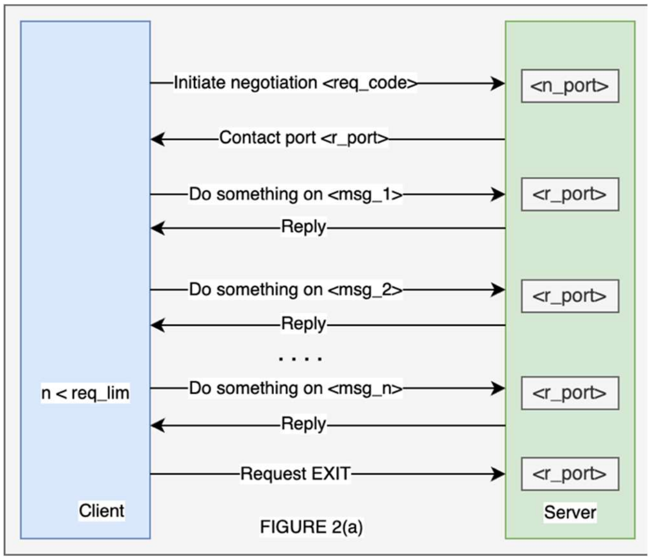

FIGURE 2(a)

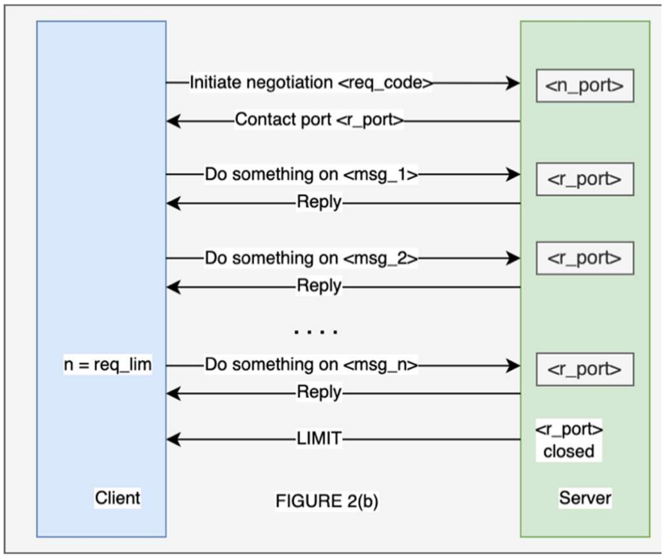

FIGURE 2(b)

Stage 2. Transaction using UDP sockets: In this stage, the client creates a UDP socket to the server on <r_port> and sends the first <msg> request, containing a string. On the server side, the server receives the string and checks if it is a palindrome or not. It then sends back the result (i.e., TRUE if the string is a palindrome, FALSE otherwise) to the client. Once received, the client prints out the result and sends the subsequent string. This process repeats for each string provided by the client, up to the specified limit <req_lim>. The UDP connection can be closed in two scenarios: (i) if the client reaches the <req_lim> the server should send 'LIMIT' to the client indicating that the <req_lim> has been reached, and then close the connection (see Figure 2(b)), and (ii) if the client has not reached the limit yet, but has sent all its messages, the client should automatically send "EXIT" to the server to close the connection (see Figure 2(a)).

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. Therefore, 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 these command line inputs: <server_address>, <n_port>, <req_code>, <msg$_1$>, <msg$_2$>, …, <msg$_n$> in the given order. There can be any number of messages from the client. If the client receives a message 'LIMIT' from the server that indicates that <req_lim> is exceeded, it should stop sending further messages and gracefully terminate the connection.

2.4 Server Program (`server`)

You should also implement a server program, named `server`. The server will take the following command-line parameters: `<req_code>` and `<req_lim>`. 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
```

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.

- To run the server, use the following command: `./server.sh <req_code> <req_lim>`. For example: `./server.sh 123 3`

  Expected server output: `SERVER_PORT=50899`

- To run the client, use the following command: `./client.sh <server_address> <n_port> <req_code> <msg_1> ... <msg_n>`

  Scenario 1:

  In this scenario, the client sends 'n' messages where 'n' is less than the request limit (which is 3 in the example above).

  - Execute the client script as follows: `./client.sh 127.0.0.1 50899 123 'level' 'hello'`
  - Expected client output: `TRUE, FALSE`

  Scenario 2:

  In this scenario, the client sends 'n' messages where 'n' is greater than or equal to the request limit (which is 3 in the example above).

  - Execute the client script as follows: `./client.sh 127.0.0.1 50899 123 'level' 'hello' 'A man, a plan, a canal: Panama' 'goose'`
  - Expected client output: `TRUE, FALSE, TRUE, Request limit reached`

3 Hints

You can use the sample codes of TCP/UDP socket programming in Python from the Chapter 2 slides.

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 settings.

- 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. However, remember you cannot share your program or work with any other student.

4 Procedures

4.1 Due Date

The assignment is due on **Friday, June 09, 2023, at midnight (11:59 PM)**.

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. You must hand in the following files/documents:

- *Source code* files.
- *Makefile* (if applicable): 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**, e.g.:

- ubuntu2004-002.student.cs.uwaterloo.ca
- ubuntu2004-004.student.cs.uwaterloo.ca

You can learn more about these servers and how to access them here:

https://uwaterloo.ca/computer-science-computing-facility/teaching-hosts

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 and understand your code).

You **will** lose points if your code is unreadable, sloppy, inefficient, or lacks internal documentation.

4.4 Evaluation

Work on this assignment is to be completed individually.

5 Additional Notes

1. 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.
2. Your code must be tested in the `linux.student.cs` environment prior to submission.
    1. Run client and server on two different `student.cs` machines
    2. Run both client and server on a single/same `student.cs` machine
3. Make sure that no additional (manual) input is required to run `server` or `client`.
4. You are expected to design a robust code that handles exceptions and does not crash abruptly.