Socket Programming – String Converting Service – Part II

# 1. Background

In class, we used a simple example application to introduce socket programming. The behaviors of the application are as follows:

1. The client reads a line of characters (data) from its keyboard and sends data to server.

2. The server receives the data and converts characters to uppercase.

3. The server sends modified data to client; continue to run to serve other clients.

4. The client receives modified data and displays line on its screen and then exits.

We assume that both the client and the server run on the same machine.

# 2. Project Requirements

In this project, you are required to extend the behaviors of the server and the client, using. IN THIS PROJECT, YOU ARE REQUIRED TO IMPLEMENT THE SERVER AND CLIENT USING **TCP SOCKETS**. In the required implementation, the client should be able to accept multiple inputs from the user one by one until the user types in “Quit.” The server should be able to serve multiple clients. For each client, the server will use one TCP connection socket to deal with all the requests from the same client. When one client is done with all its requests, the client and the server will close the TCP connection. The server will create a new TCP connection for another client, if another client wants to connect to the server.

The implementation on the server side can be regarded as persistent, because it is using the same TCP connection socket to deal with all the requests from the same user. The TCP connection socket will be torn down only when the client is done with all the requests.

## 2.1 Persistent TCP Implementation Without Multithreading

Here is the pseudo code for this implementation, where we do not use multi-threading:

**TCPStringServerPersistent:**

Create a listening TCP socket

While (true) {

Wait for the connection from a client

While (true) {

Use the established TCP connection socket to receive a string from the client

If (the request is Quit) {

Close the connection

Break

} else {

Evaluate the result of the math expression in the string

Send back the result as a string to client using the connection socket

}

}

}

**TCPStringClientPersistent:**

Connect to the server

While (true) {

Get an input string from the standard input

Send the string to the server

If (the input string is “QUIT”) {

Break

} Else {

Receive the result from the server

Print the result as a string

}

}

Close the connection socket

Remarks: This implementation will allow the server to serve multiple clients one by one. However, the server cannot serve multiple clients at the same time, because the code is single-threaded; when the server process is stuck in the inner while loop serving one client, no other clients can be connected, although there is still a listening socket there. When one client is done, the server process will jump out of the inner while loop and go to the next iteration of the outer while loop, where it can accept the connection from a new client.

## 2.2 Persistent TCP Implementation With Multithreading

With persistent TCP implementation, if the server wants to serve multiple clients at the same time, the server needs to use multiple threads. With multithreading, the server can create a new thread to serve a new client. The main thread execution will not be stuck in a loop, and will still be able to accept new clients. Here is the pseudo code for this implementation:

**TCPStringServerPersistentMultithread:**

Define a thread that will use a TCP connection socket to serve a client

Behavior of the thread: it will receive a string from the client and convert it to an upper-case string; the thread should exit after it finishes serving all the requests of a client

Create a listening TCP socket

While (true) {

Wait for the connection from a client

Create a new thread that will use the newly created TCP connection socket to serve the client

Start the new thread.

}

**TCPStringClientPersistent:**

This should be the same as the pseudo code of TCPStringClientPersistent in Section 2.1.

# 3. Sample Code on Multi-threading

You can implement the project in any programming language you like. Examples on how to implement multi-threading can be found in the “socket programming projects” folder of Canvas Files:

Java: *MultiThreadDemo.java*, *MultiThreadDemo2.java, and MultiThreadDemo3.java*

Python: *multithread.py*, *multithread2.py, and multithread3.py*

# 4. Submission

You can use either Python or Java to complete the project. In either case, you are required to **submit four files:**

* The code for **TCPStringServerPersistent**. (preferred file name: TCPStringServerPersistent.py or TCPStringServerPersistent.java)
* The code for **TCPStringClientPersistent**. (preferred file name: TCPStringClientPersistent.py or TCPStringClientPersistent.java)
* The code for **TCPStringServerPersistentMultithread**. (preferred file name: TCPStringServerPersistentMultithread.py or TCPStringServerPersistentMultithread.java)
* The **project report**, which must include the testing you have run to verify that your code meets the requirements. You can paste what you got in the console or include some screenshots.

You are encouraged to do **an in-person demonstration** of your project to me within one week after the due date of this project.