CS 3251-B Computer Networks I

Fall 2018

Programming Assignment 1: Basics of Socket Programming

DUE DATE: FRIDAY SEPTEMBER 20, 11:00PM
Please work individually – no group submissions
(however, you can test your code in collaboration with other students – and we certainly encourage you to do so)

Introduction

In this assignment you will write a simple client-server application using network sockets. The first goal of the assignment is to provide you experience with basic sockets programming. The second goal is to prepare you for the second programming assignments, which will be significantly more challenging.

Assignment

You will implement two versions of the same client-server application. One version will be based on TCP. The other will use UDP. You can use Python, Java or C/C++. If you want, you can even use one programming language for the client, and another language for the server.

The application is very simple: the client sends to the server two numbers and an operand. The server replies with the result (or an error message).

Lets call the client **rmtcalc** (remote calculator) and the server **rmtcalc-srv**.

To start a TCP-based server listening at port 13001 at the local host, you would type:

rmtcalc-srv TCP 13001

Similarly, to start a UDP-based server listening at the same port, you would write:

rmtcalc-srv UDP 13001

To run the client, you must know the protocol (TCP or UDP), address (or name) of the server, and the server's port number. You can assume that the client knows these three properties of the server

because they are either "well-known" or they are somehow communicated to the client through a directory service.

For example, if the server is TCP-based, it runs at *networklab1.cc.gatech.edu* and it listens at port 13001, you should run the client by typing:

rmtcalc TCP networklab1.cc.gatech.edu 13001

Then, the client's user should be able to type arithmetic operations such as the following example:

245.678 * -272.123

At that point, the client will send this operation to the server, and it if everything works correctly, it will receive from the server the value: -66854,634

You need to support the four arithmetic operations: + - * /

Your server should be able to run correctly with any correctly implemented client– not just with your client. For this reason, we strongly encourage you to test your code in collaboration with several classmates.

We will test your implementation by running your client (server) with a server (client) other than yours.

To create such an interoperable implementation, you will need to implement precisely the following protocol:

1. The client messages should consist of the following three fields:
    a. The first 16 bytes (bytes 0-15) will be a string representation of the first number: the first character in the string (byte-0) is always the sign, while the second character (byte-1) is the most significant digit of the number. One of the 15 characters after the initial sign may be the character "." initiating the fractional part of the number.
    b. The next 16 bytes (bytes 16-31) will be the string representation of the second number (same format with the first number).
    c. Finally, one last byte at the end of the message (byte number 32) will represent the operation using one of the following characters: + - * /
2. The server's message will consist of the following two fields:
    a. The first 16 bytes will be the string representation of the result (same format as before).
    b. The next 32 bytes will be a short message that indicates either successful completion of the operation and the name of the server's author (e.g., "Offered to

you by Amy Smith's server!") or an appropriate error message ("Error: Divide by zero", "Error: Invalid operand", etc).

In both UDP and TCP, the client should be able to send multiple calculations to the server, one at a time, until the user types "quit". At that point the client should close the socket and exit.

For example, a client session may look as follows:

Please type an operation: 1.0 + 1
Result = 2

Please type an operation: -1 * 2.5
Result = -2.5

Please type an operation: 4.9 / 0.0
ERROR: "divide by zero"

Please type an operation: quit
Bye..

In the case of TCP, if the client suddenly quits or crashes, your server should not "hang" –it should be able to accept a new connection request by another client.

Your server can connect with only one client at a time – you do not need to worry about multiple concurrent clients in this assignment.

You can assume that we will format the text input as shown in the examples above. In other words, you do not need to worry about cases of mis-formatted inputs, redundant blank spaces, etc.

Please, print in an easy-to-understand format every message that the client/server sends and receives – this will help us test your code more easily.

You can (encouraged even) to use external sources as you develop this assignment (such as the socket code given in the lectures). However, **you must include a citation (text or web site) in your source code and project description if you use an external reference or existing code templates. Also see note at the end about the use of program libraries.**

Submission

You will implement two separate versions of this assignment: a client and a server. Please call these programs: rmtcalc and rmtcalc-srv (with the appropriate suffix, depending on whether you use Python, Java, C or C++).

We will test your code at the 8 network-lab machines (named "networklab1.cc.gatech.edu" through "networklab8.cc.gatech.edu"). We strongly suggest that you test your code in those machines before you submit it.

Please turn in well-documented source code, a README file, and a sample output file called Sample.txt. The README file must contain:

- Your name and email address
- Class name, date and assignment title
- Detailed instructions for compiling/running your client and server programs
- Any known bugs or limitations of your program

You must submit your program files online.

- Create a ZIP archive of your entire submission.
- Use Canvas to submit your complete submission package as an attachment.

For example, a submission may look as follows-

```
pa1.zip
  | -- pa1/
        | -- rmtcalc.py
        | -- rmtcalc-srv.py
        | -- README.txt/.pdf
        | -- sample.txt
```

Instructions for accessing networklab machines

For this assignment, we have set up several special machines to test your code that are not subject to the firewalls deployed in the GT network.

To access these machines, you need to be either on the Georgia Tech network (i.e., using GT machines or connected through the GT WiFi network) or using a Georgia Tech VPN client. Steps to install the VPN client are given by OIT (see http://anyc.vpn.gatech.edu). Make sure to start and login to the VPN client every time you plan to use the remote machines.

In order to access these special machines, you need to ssh as follows:

ssh <gt_username>@networklabX.cc.gatech.edu

where X is an integer between 1 and 8.

For transferring your code to the remote machines, you may use scp, sftp, or the more user-friendly filezilla, which has a GUI.

Grading

The grade will be split equally between the UDP and the TCP programs – 50% each. We will use the following guidelines in grading:

0% - Code is not submitted or code submitted shows no serious attempt to program the functions required for this assignment.

25% - Code is well-documented and shows genuine attempt to program required functions but does not work properly. The documentation is adequate.

50% - Code is well-documented and shows genuine attempt to program required functions. The protocol documentation is complete. The code does run but sometimes it fails to produce reasonable output or crashes.

75% - Code is well-documented. The protocol documentation is complete. The code runs correctly with properly formatted input. The program does not run correctly when we use a client or server other than yours, or when the client terminates abruptly.

100% - Code is well-documented. The protocol documentation is complete. The code runs correctly with properly formatted input. The program is resilient to input errors and client crashes. Your implementation can run correctly with other (correct) clients/servers.

Important Notes:

- Submit only source code -- no executables
- You can work locally and test on the loopback address (127.0.0.1) but ultimately you should also test on the networklab machines.
- **Using special network libraries:** Certain network libraries can make this assignment trivial – to the point that you will not learn much if you use them. For this reason, please ask (ideally on Piazza) before you use a specific library.