Socket Programming: Number Exchange

Assigned: Oct. 10, 2023
Due: Oct. 17, 2023 @2355
Version: 0.1.1

In this assignment, you'll write a client that will use sockets to communicate over TCP with a server that you will also write. Your client and server will exchange the sequence of messages shown in Fig. 1.

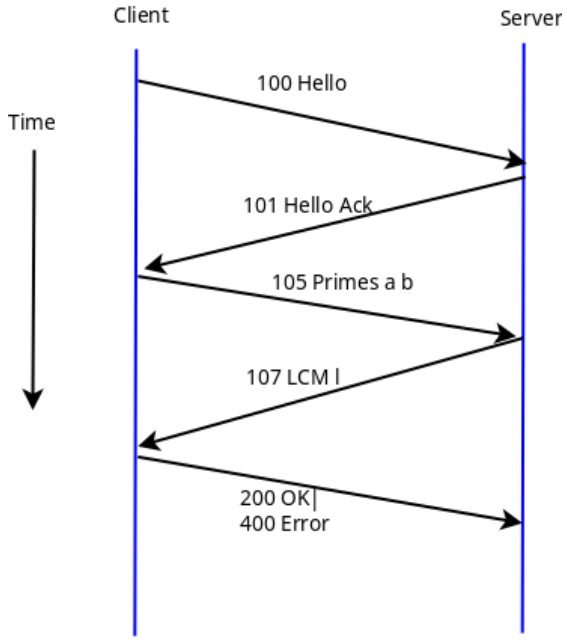

Figure 1: Message sequence exchanged between client and server

Description:

In this assignment, you will "simulate" a client sending prime numbers to a server. You will write a client that will communicate with a server that you will also write. On start-up, each client will print out a string saying: "Client of *Joan A. Smith*" and then it will open a TCP socket to your server and send a message (a string of characters) to your server containing the string "100 Hello". Upon receipt of the "Hello" message, the server will acknowledge the client's hello message as shown above. The server's response will contain the string "101 Hello Ack". Upon receipt of this message, the client will prompt the user for two prime numbers each between 1031 and 6397, which it will then send to the server in the "105 Primes *a b*" message, where *a* and *b* represent the prime numbers that the user entered. The server will acknowledge this message with a "107 LCM *l*" string, where *l* represents the lowest common multiple (LCM) of the two prime numbers that were supplied by the client. If the server-computed LCM matches that computed by the client, then the client will send the "200 OK" message to the server, otherwise, it will send a "400

Error" message to the server. After sending any one of these messages, the server will close the connection socket that it has created for the client. The client will also close its connection to the server. The server and client will then terminate. If at any time the server receives a message from the client that is unexpected, the server will respond with a "500 Bad Request" message.

Part A: Student Implementation

Server Implementation

Your server will create a string containing its name (e.g., "*Server of Joan A. Smith*"). As indicated above, the server will wait until the client says hello with a "100 Hello" message. The rest of the server operation is as described above.

Client Implementation

Your client will create a string containing its name (e.g., "*Client of Joan A. Smith*"). The client will then send a "100 Hello" message to the server. The rest of the client operation is as described above.

Additional details:

Note that a short design document is required. You should program each process to print an informative statement whenever it takes an action (e.g., sends or receives a message, detects termination of input, etc.), so that you can see that your processes are working correctly (or not!). This also allows the grader to also determine from this output if your processes are working correctly. You should hand in screen shots of these informative messages. Your document should also discuss how you tested the program to ensure that it meets the specification above.

Programming Notes

Here are a few tips/thoughts to help you with the assignment:

- You must choose a server port number greater than 1023 (to be safe, choose a server port number larger than 5000).
- Starter code is provided for you on VLE.
- You **must** write your programs in Python3. Details about the socket module can be found at http://docs.python.org/3/library/socket.html and Socket Basics file posted on OurVLE.
- Many of you will be running the clients and senders on the same UNIX machine (e.g., by starting up the receiver and running it in the background, then starting up the relay in the background, and then starting up the sender. This is fine since you are using sockets for communication these processes can run on the same machine or different machines without modification. Recall the use of the ampersand to start a process in the background. If you need to kill a process after you have started it, you can use the UNIX `kill` command. Use the UNIX `ps` command to find the process id of your server.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use.
- Do not open up a separate socket for each message that you send. Doing so is poor programming style, and your grade will be affected negatively for doing so.

Part B: Using a Large Language Model (Optional: up to 10 extra-credit points)

Use a Large Language Model (LLM) such as ChatGPT to generate code to solve the project specification from Part A. Submit your solution, the LLM-based one, and in your write-up comment on the differences between them. Note, you must still accept the client and server parameters from the command line.
NB: You will not receive the extra-credit points if your solution to Part A is non-functional.

Rules

- **The project is designed to be solved independently**.
- **You may not share submitted code with anyone.** You may discuss the assignment requirements or your solutions away from a computer and without sharing code, but you should not discuss the detailed nature of your solution. If you develop any tests for this assignment, you may share your test code with anyone in the class. Please **do not put any code from this project** in a public repository or in **a private repository that you share with other students**.
- All students whose code is shared will get zeros.

What to turn in

1. You will hand in the code for the client and server implementations along with screen shots of a terminal window, verifying that your programs actually carry out the computation that is specified.
2. A separate (typed) document of a page or so, describing the overall program design, a description of "how it works," and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
3. A separate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
4. A key part of the testing of real-world protocols is an interoperability test. In this course, you will simulate interoperability by running your client against a classmate's server, and your server against a different classmate's client. Take screenshots of each of these tests.
5. There is nothing in the specification that ensures that the numbers are indeed prime. Describe how to improve your implementation to check that the numbers are indeed prime, and how you will deal with the cases where the numbers are not prime. You are allowed to introduce a new status code and message to communicate to the client that the number is not prime.

Grading

- Program listing
    - Works correctly as specified in assignment sheet – 35 points
    - Contains relevant in-line documentation – 5 points
- Design document
    - Description – 5 points
    - Thoroughness of test cases – 3 points
    - Interoperability tests – 2 points