Socket Programming: Vote Counter

Assigned: Oct. 4, 2020
Due: Oct. 12, 2020 @2355
Version: 0.1.2

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.

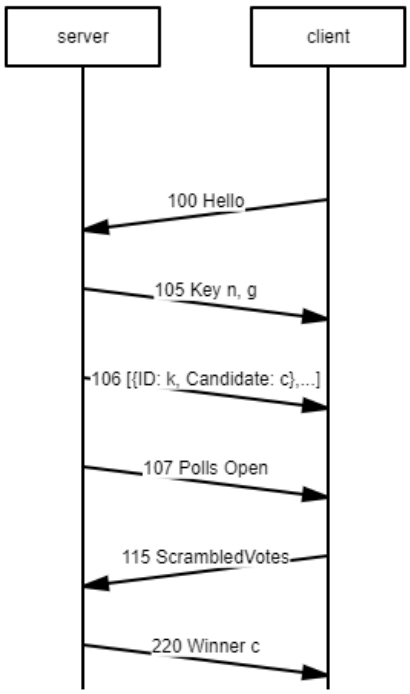

Figure 1: Message sequence exchanged between client and server

Description:

In this assignment, you will "simulate" an election process. 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 send three messages back-to-back. The first message will contain the string "105 Key *n*, *g*", where *n* and *g* are integers. The second message will contain the string "106 [{ID: *k*, Candidate: *c*},…]". The string within the curly braces is an object representing one candidate. For the purposes of this project, you will have exactly two objects in the "*106* …" message. The third message is "107 Polls Open". Upon receipt of this message, the client can begin collecting votes for the candidates locally. After the votes have been collected, the client(s) will send its (their) votes to the server using the "115 …" message. However, these votes will be

scrambled with a formula that will be provided for you in the code. The server will then pick the candidate with the most votes, and print that winner to the console, and send the "220 …" message to the client. After sending this message, the server will close the connection socket that it has created for the client. The server and client will both terminate.

Server Implementation

Your server will create a string containing its name (e.g., "*Server of Joan A. Smith*") and then prompt the user for the first names of two candidates. The server will prompt for two prime numbers, *p*, and *q*, each between 251 and 499. Next, compute the product of the two primes, and store that in a variable, *n*. Ensure that the greatest common divisor (gcd) of *n*, and *(p-1)(q-1)* is 1. If not, prompt for a new pair of prime numbers. Also compute the lowest common multiple (lcm) of *(p-1)* and *(q-1)*. Call this lcm, lambda. Define the function, $L(x)$, where $L(x) = \frac{x-1}{n}$. Pick a random integer, g, between 1 and n^2. Send n and g in the "105 …" message described above, and keep lambda on the server.

For the ID, assume that each candidate will have no more than 255 votes. Thus, you can set one ID to be 256, i.e., 2^8, and the other to be 65536 i.e., 2^{16}. In general, if one candidate can receive no more than 2^k votes, you should set the candidate IDs to 2^k and i.e., 2^{2k}.

Client Implementation

Your client will create a string containing its name (e.g., "*Client of Joan A. Smith*"). After receiving the names of the two candidates, the client will prompt the user for the candidate to vote for. Upon receiving the name, the client will pick a random number, r, in the range $0 < r < n$. The scrambled vote is $g^i r^n \bmod n^2$, where *i* is the candidate ID. Store this result in an array. When you are ready to send votes to the server, you will multiply all these votes modulo n^2, and send the result of this calculation to the server.

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 OurVLE.
- 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.

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.

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 – 5 points