UNIVERSITY OF DELAWARE
DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

CISC 650 / CPEG 651 / ELEG 651: Computer Networks II

Fall Semester, 2017 Professor: Adarsh Sethi

PROGRAMMING PROJECT 1

This project is an individual assignment and must be done independently by each student.

Your task in this project is to design and implement a client and a server that communicate via TCP to accomplish the task described below. You may use either C or C++ for your programming. You may also use the example client-server programs supplied in the class as a starting point for your programs.

Client Request and Server Response

The client sends a request to the server containing a letter and an integer. The letter may be either the letter 'C' or the letter 'N' (upper-case only). The integer will be a 16-bit unsigned integer in the range 1-65535.

The server responds by first echoing the same letter and same integer back to the client. If the letter was a 'C', the server then randomly generates a sequence of lower-case letters (in the range 'a' - 'z') and transmits them to the client. The number of letters generated and transmitted is equal to the value of the unsigned integer that was sent by the client. If the letter was an 'N', the server randomly generates a sequence of 16-bit unsigned integers and transmits them to the client. In this case, the number of integers generated and transmitted is equal to the value of the unsigned integer that was sent by the client.

Server

When the server starts, it prints out the port number it will be listening on and then waits for a client to establish a connection with it. When a client has connected, it then waits for the letter and the unsigned integer to be received. The server then echoes the letter and integer back to the client. Next, it starts to generate the sequence of letters or integers, as the case may be. The server uses a buffer of size 100 bytes which it fills with the letter/integer sequence. When the buffer is filled, the server transmits its contents to the client. After each such transmission, the server waits for a random amount of time before generating another sequence to fill the buffer; this random wait is achieved by generating yet another random number (in the range 100 - 999), multiplying it by 1000, and counting down from this number to zero in a loop that does nothing else. Once the wait is over, the next part of the sequence is generated to fill the buffer and then transmitted to the client. Thus, all of these transmissions to the client will have 100 bytes in them, except possibly for the very last one which may be shorter.

The server counts and keeps track of the following quantities (these don't include the initial letter and number being echoed):

- the total number of letters or integers transmitted
- the number of separate transmissions (each *send* counts as a separate transmission)
- the total number of bytes transmitted (as measured by the sum of the values returned by each *send* operation)

In addition, the server maintains a checksum of all the letters/integers transmitted. If it is transmitting letters, the letters are all added up after being cast as 8-bit unsigned integers. If it is transmitting integers, then they are all added as 16-bit unsigned integers. In each case, any carry or overflow is ignored.

When the server has finished transmitting its entire response to the client, it prints a message on its standard output including the three counts and the checksum computed as described above. It then closes the connection and returns to wait for a new connection request.

When you are finished with your program, you will terminate the server with a *Control-C*.

Client

When the client starts, it prompts the user to type in the hostname and port number of the server. Next, it prompts the user to enter either the letter 'C' or the letter 'N' and also an integer in the range 1-65535. The client should check each user input to ensure it conforms to the rules, rejecting it and asking the user to re-enter it if it does not.

Now the client establishes a connection to the server and sends the letter and the number. Then it waits for the response to come from the server. When the initial letter and unsigned integer is received from the server, the client prints them on the standard output.

Next, the client waits for the stream of random letters or integers to come. It uses a buffer of size 80 bytes to read in the incoming data. After each read operation, the client waits for a random period of time using the same procedure used by the server, and then tries to read the next incoming data.

The client should **NOT** print any of the letters/integers received in the input stream of data. Instead, the client maintains the following counts (these don't include the initial echoed letter and number):

- the total number of letters or integers received
- the number of separate receive operations (each *recv* counts as a separate operation)
- the total number of bytes received (as measured by the sum of the values returned by each *recv* operation)

In addition, the client also computes a checksum of all the letters/integers received (in the same way as described for the server). When the client has finished receiving the entire response from the server, it prints a message on its standard output including the three counts and the checksum computed as described above. It then closes the connection and prompts the user to see if the user would like to send another request to the server. If the user responds positively, the whole operation is repeated, otherwise the client exits.

Things to watch out for:

You should note that TCP is a byte-stream protocol that does not preserve user message boundaries. Thus, the server may send a long message to the client in a single operation, but the client may receive it in multiple pieces. Conversely, multiple messages sent individually may be delivered all together in a single read operation. For this reason, the client may need to perform the read operation in a loop until the entire expected message has been received. The client program supplied to you in class does not have this functionality, so you must add this in. Also, you should be aware of the sizes of your data items (8-bit/16-bit etc.) and ensure that a 16-bit quantity is indeed transmitted as 16 bits (2 bytes) and not as something of a different size.

See the document "Project Hints" for additional tips for your project.

Submission:

Put your client and server programs in the same directory along with the Makefile. Make sure there are no other files in this directory. In two different windows, create two scripts, one for the client and one for the server.

In the first window, start the server script. Now do a long listing of the directory (using *ls -l*); next do *make clean* which will delete the executables; then do another *ls -l*; then compile the client and server programs using the Makefile (*make all*); and then do another long listing. Now run the server. In the second window, start the client script. Do a long listing of the directory, and then run the client.

Test the functionality of the client and the server by giving various commands through the user interface. The test session should include at least two different connections between the client and the server *in the same run of the client*, and at least two different executions of the client *during the same run of the server*. Test out both the character and integer cases, giving both small and large counts (with one count being at least 5000). Also, show error-checking of the user inputs by the client.

Finally, exit the scripts. Make sure the script files are stored in the same directory as your programs. Zip this directory and submit the zipped file as your submission in Sakai. The file that you submit should have a *.zip* extension. The submission must be made in the Assignments tool on Sakai. No paper submission is required for this project.

Grading:

Your programs will be graded on the following criteria:

Correctness: 60 %
Testing and Proper Output: 20 %
Readability and Documentation: 20 %

Deadline for submission: Monday October 16, 11 pm.

The Late Policy for assignments stated in the Course Policies handout will apply. It is to be understood that computers and computing facilities are subject to failure, overload, unavailability, etc., so it is your responsibility to plan and execute your project work sufficiently in advance so as to meet the due date, and these will not serve as excuses for making exceptions to that policy.