CS3251 Computer Networks I

Spring 2019

Programming Assignment 1

Assigned: Jan. 31, 2019

Feb 14, 2019, 11:59pm

PLEASE READ THIS CAREFULLY BEFORE YOUR PROCEED

For this assignment you will design and write your own application program using network sockets. You will implement your application using TCP (SOCK_STREAM) sockets.

The application you are designing is Trivial Twitter application (or *ttweet*). In this application a *ttweet* server has room for exactly one message and is used by exactly one client. The client uploads a message to the server, then the same or another client downloads the message to read it. An uploaded message is stored at the server will overwrite an existing message if the server already has a message. A download request returns the last uploaded message or returns "Empty Message" if no message has been uploaded yet. The server is simple and can handle only one client at a time.

The server is invoked through a command line interface:

Command: *ttweetsrv <ServerPort>*

The server should run forever and not crash for any reason. The server may optionally print status reports (for example when a client attempts to upload or download and message).

The client is invoked through a command-line command with one of two modes:

- Upload Mode:

  Command: *ttweetcl -u <ServerIP> <ServerPort> "message"*

  Output: message upload successful OR error message if there are argument problems

- Download Mode:

  Command: *ttweetcl –d <ServerIP> <ServerPort>*

  Output: the last messages stored at the server with "Empty Message" if no message has been uploaded to the server OR error message if there are argument problems

The message will have 150 character limit. The client should exit gracefully without uploading a message if that limit is violated.

You will need to develop your own "protocol" for the communication between the client and the server. While you should use TCP to transfer messages back and forth, you must determine exactly what messages will be sent, what they mean and when they will be sent (syntax, semantics and timing). Be sure to document your protocol completely in the program write-up.

Your server application should listen for requests from *ttweet* clients, process the client and return the result. After handling a client the server should then listen for more requests. You are NOT required to implement a concurrent server for this assignment. The server application should be designed so that it will never exit as a result of a client user action.

Focus on limited functionality that is fully implemented. Practice defensive programming as you parse the data arriving from the other end. Do not work on a fancy GUI, but focus on the protocol and data exchange. Make sure that you deal gracefully with whatever you or the TA might throw at it.

You may also use client and server templates including the ones discussed in class. 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.**

Notes:

1. Your programs are to be written in C, C++, Python or Java.

2. **We will test your code at the shuttle machines.**
   See https://support.cc.gatech.edu/facilities/general-access-servers
   We strongly suggest that you test your code in those machines before you submit it. '

3. **For java, shuttle machines only support java 1.8 or lower. Note that there is not javac compiler in shuttle machine, so you need to upload runnable jar file to test. For python, shuttle machines only support python 2 version.**

4. We recommend that you test some existing template (for example the TCP Echo client and server code) by compiling and running them to make sure you are comfortable with the environment before attempting to work on your own code.

5. You should select a port for your *ttweet* service. We recommend something between say 13000 and 14000. It is wise to provide the server port as input to the server and not to hard code the server port in the program. Note that if your server crashes, its port number may not be usable for a short period of time afterwards (for reasons we will explain later) so it is best to choose a new port number when you restart it.

6. The server should not retain any messages from a previous run. If the server crashes or you exit the server (for example by Control C) then any messages uploaded should not reappear when your run the server again. In other words a server should always start with an Empty Message.

7. Use explicit IP addresses (in dotted decimal notation) in the client for specifying which server to contact. Do not use DNS for host name lookups.

8. Make sure that you do sufficient error handling such that a user can't crash your server. For instance, what will you do if a user provides invalid input?

9. Your server will be iterative; handling one client at a time.

10. You must test your program and convince us it works! Please provide your program output for the following test scenario. In grading your submission we will use the same test sequence (except we will use our own messages).

Test Scenario

0- (Server Not Running)
1- Run Client in upload mode -- (Output: Error Message: Server Not Found -- exit gracefully)
2- Run Client in download mode – (Output: Error Message: Server Not Found – exit gracefully)
3- Run Server (server never exits or crashes – server may print out status statements to indicate client request activity – you should specific what the server will print in the README file)
4- Run Client in download mode (Output: EMPTY Message)
5- Run Client in upload mode – upload message0 > 150 characters (Output: Error)
6- Run Client in download mode (Output: EMPTY Message)
7- Run Client in upload mode – upload message1 <= 150 characters (Output: Upload Successful)
8- Run Client in download mode (Output: message1)
9- Run Client in upload mode – upload message 2 (different from message1) <= 150 characters (Output: Upload Successful)
10- Run Client in download mode (Output: message2)
11- Run Client in upload mode – upload message3 > 150 characters (Output: Error)
12- Run Client in download mode (Output: message2)

Submission

Your final submission should include all the source code and instructions to produce 2 separate executables from the provided source code. Please call these executables: *ttweetcli, ttweetser.*

Your programs are to be written in Python, Java or C.

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.
- Names and descriptions of all files submitted.
- Detailed instructions for compiling and running your client and server programs.
- An output sample showing the result of running the test scenario above.
- Protocol description in particular describe the format of the messages exchanged between client and server (see for example how the http message format is described.
- Any known bugs or limitations of your program
- Submit only source code (no executables).
- 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
  | -- pa/
       | -- ttweetcl.py
       | -- ttweetsrv.py
       | -- README.txt/.pdf
       | -- sample.txt
```

Grading Guidelines:

We will use the following guidelines in grading:

0% - Code is not submitted or code submitted shows no 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 compile properly. The protocol documentation is adequate.

50% - Code is well-documented and shows genuine attempt to program required functions. The protocol documentation is complete. The code does compile properly but fails to run properly (crashes, or does not handle properly formatted input or does not give the correct output).

75% - Code is well-documented. The protocol documentation is complete. The code compiles and runs correctly with properly formatted input. But the program fails to behave gracefully when there are user-input errors.

100% - Code is well-documented. The protocol documentation is complete. The code compiles and runs correctly with properly formatted input. The program is totally resilient to input or network errors.