CS 6250 Computer Networks

Fall 2019

Programming Assignment

Assigned: Oct 17, 2019

Oct 31, 2019, 11:59pm

PLEASE READ THIS CAREFULLY BEFORE YOUR PROCEED

1. Overview

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). It allows user to publish their messages and also read other’s messages. For more details, please read below.

2. Details

First, a (perhaps obvious) note on command syntax for this instruction:

- “$” precedes Linux commands that should be typed or the output you print out to the stdout.
- “>” shows the input to the program through stdin
- “-” shows the program’s output in stdout

2.1 Twitter Client

When first starting your client it should accept 3 command line args: the server ip, port and a username you would like to use. An example can be seen below. Please follow this order of arguments exactly.

```
$ ./ttweetcli   <ServerIP>   <ServerPort>   <Username>
```

After you run the above command to start your client it should use system standard input to get user input and output corresponding result through system standard output. Here **username**

only consists of alphabet characters(upper case + lower case) and numbers and you could assume the maximum username won't exceed 64 chars. Username should be unique for each client. After client start, it should contact the server to check if the username has been taken. If so, the client should exit directly and notify the current user. The same username could only be used after the previous client logout.

Your client should be able to handle the following commands from stdin and print feedback for the operation(like success or the reason failed). **There will be one whitespace to separate each argument.**

- **tweet** "<150 char max tweet>"
  - The tweet should be uploaded to the server and server should add this message to other user's mailbox. **If user isn't online, you don't need to store this message for them.**
  - You should only allow [1,150] character tweets, you are expected to handle the illegal messages (length illegal).
  - We ensure message will be inside a pair of "".("" shouldn't be considered as the length of message.) **Message itself doesn't have "", but could contain other common ascii symbols**.
- **timeline**
  - The client will output all the unread tweets in its mailbox stored by server since the last time the user has run the **'timeline'** command.
  - The output should contain the following 3 pieces of information:

```
<current_client_username> from <sender_username>:  "<tweet_message>"
```

- **exit**
  - Clean up any necessary state and then close the client gracefully.

2.2 Twitter Server

Your server should take in the port number as its only command line arg like so:

```
$ ttweetsrv   <Port>
```

Your server should listen to the local machine (like localhost or 127.0.0.1). Your server will receive uploaded tweets from a client and enables other clients to get the message. **Your server does not need to store these tweets in the file system.** Whenever we start server, it should be the initial state.

2.3 Example Program Flow

The below example assumes there is a ttweetsrv running at 127.0.0.1 port 8080.

Client 1

```
$ ./ttweetcli   127.0.0.1   8080   Matt                  # Connect to server
-username legal, connection established            # program output
```

Client 2

```
$ ./ttweetcli   127.0.0.1   8080   Raf                   # Connect to server
-username legal, connection established            # program output
>tweet “Hello from client 2”    # Tweet to the server
-success.
```

Client 1

```
>timeline                                   # Print all the unread messages
-Matt from Raf: “Hello from client 2”
>timeline                                   # No new tweets to show
-No new message
>tweet “Hello from client 1”
-success
```

Client 3

```
$ ./ttweetcli   127.0.0.1   8080   Raf           # Same username
-user already login, shutdown client
```

Client 2

```
>timeline                                   # Print all unread tweets
-Raf from Matt: “Hello from client 1”
>exit                               # Exit the client. Close gracefully.
-Bye bye
```

Client 1

```
>Command: exit                                   # Exit the client. Close gracefully.
-Bye bye
```

3. Program Formats

You need to implement two separate programs using C/C++ or Python or Java: one for the server (**ttweetsrv.x**) and another for the client (**ttweetcli.x**). Note that, here "**ttweetsrv.x**" could be **ttweetsrv.c**, **ttweetsrv.cpp**, **ttweetsrv.py**, or **ttweetsrv.java**, it depends on which language you choose. Same rule applies to "**ttweetcli.x**".

- For running command, we will run each language as:
    - Java(both cases are ok)
        - java ttweetsrv <port_number>
        - java -jar ttweetsrv.jar <port_number>
    - C/C++
        - ./ttweetsrv <port_number>
    - Python(both cases are ok)
        - python2 ttweetsrv.py <port_number>
        - python3 ttweetsrv.py <port_number>
- Your server program should accept **one** required input parameter: the port number. For example, if we want the server to listen to port number 8090, we will start your program by doing:
    - ./ttweetsrv 8090
- Your client program should accept **three** required input parameters: the server ip, server port, and the chosen username (we ensure the username are different for different client instances). For example, if the server ip is 127.0.0.1, port number is 8090 and desired username is Matt then we will start your program by doing:
    - ./ttweetcli 127.0.0.1 8090 Matt

4. Notes

1. **Please strictly conform to the input & output formats. We use automated scripts to test your program. If you don't adhere to the formats, our testing may fail to show that your program works.**
2. Please output necessary information in the output to help us identify the current client's identity.
3. We don't have a high requirement for the concurrency of your program (we won't test this) but you still need to handle data races under multithreading environment.
4. For error handling, we require 2 states. The first one is **before the user successfully login**, these errors including the wrong IP address, invalid port number, duplicate usernames...etc, **your client program should exit gracefully with error message.** The second state is **after a successful user login**. These errors including invalid message format or unknown command, **your program should print the error message but keep running for the next command.**

5. For input format, we will only test for the client.But your server should report error when the port number has been taken by other program.
6. **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.
7. **For java, shuttle machines only support java 1.8 or lower.**
8. **For python, shuttle machines only support python 2.6 & 3.4 version.**
9. Use explicit IP addresses (in dotted decimal notation) in the client for specifying which server to contact. Do not use DNS for hostname lookups.
10. You must test your program and convince us it works! Please provide your program output for the following test scenario in section 2.3

5. 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.

6. Deliverables

- You need to submit one compressed folder as ZIP file containing
  - ttweetsrv.x
  - ttweetcli.x
  - Makefile
  - README
  - Sample output for section 2.3

- In the README please give a high level description of your implementation ideas.
  - Please include your name and GT ID, username.
  - Your design for the server and client
- Note that, We will compile your program by just doing "make", if you use C/C++/Java.
- Please comment your source code.

7. Useful Links

Sockets Links : For Sockets Programming the following on-line reference will be useful:

- "Beej's Guide to Network Programming" The current version uses the latest ipv4/v6 addressing. The simpler (and older version) is used here
- A Tutorial http://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf
- A couple of videos
  - 1. https://www.youtube.com/watch?v=LtXEMwSG5-8
  - 2. https://www.youtube.com/watch?v=mStnzIEprH8
- Python tutorial on socket programming: https://realpython.com/python-sockets/
- Some Templates:
- See this link for IPv4-only templates .
- See this link for IPv6 templates and more modern error checking .
- Yet more sockets programming templates from Doug Comer
- For java: https://cs.lmu.edu/~ray/notes/javanetexamples/