HW #3: Simplified HTTP on TCP, Due Date*: 11:59pm 02/18/2020, Cutoff Date**: 11:59pm 02/20/2020

Submission: (1) Upload all source code (.java) files to both Blackboard and the Virtual Servers (cs3700a and cs3700b), AND (2) set up and test java programs (.class files) on cs3700a and cs3700b (See Part II for details)

****Submission will NOT be accepted after the cutoff deadline**

An option of peer programming: You may choose to work on this assignment individually or in a team of two students. If you choose to work in a ***team of two students***, you **must** 1) **add** both team members' first and last names as the **comments** on Blackboard when submitting the .java files (both team members are required to submit the same .java files on Blackboard), and 2) put a ***team.txt*** file including both team members' names under your HW3/ on cs3700a (both team members are required to complete Task II under both home directories on cs3700a). For grading, I will randomly pick the submission in one of the two *home directories* of the team members on **cs3700a** and **cs3700b**, and then give both team members the same grade. If the team information is *missing* on Blackboard or cs3700a, two or more submissions with similar source codes with just variable name/comment changes will be considered to be a violation of the Integrity described in the course policies and both or all will be graded as 0.

Grading: Your programs will be graded via testing and points are associated with how much task it can complete. A program that cannot be compiled or crashes while running will receive up to 5% of the total points. A submission of .java files that are similar to any online Java program with only variable name changes will receive 0% of the total points.

Programming in Java is highly recommended, since all requirements in this assignment are guaranteed to be supported in Java. If you choose to use any other language such as Python or C/C++, it is YOUR responsibility, before the cutoff deadline, to (2) set up the compiling and running environment on cs3700a and cs3700b, (2) make sure that I can run/test your programs in your home directory on cs3700a and cs3700b, and (3) provide a README file under HW3/ on cs3700a to include the commands that I need to use.

Part I (85%): Write a ***client*** program and a ***server*** program to implement the following simplified HTTP protocol based on TCP service. Please make sure your program supports multiple clients. The webpage file CS3700.htm is provided. You may choose a port like 5678 and hard code it in both client and server programs. (Hints: for testing both programs on the same computer, you may want to put client program and server program at two different directories. Do NOT hard code the file name "CS3700.htm"!)

- The ***ending signature*** of the ***entity body*** in the ***HTTP response message*** for the case "200 OK":
  - When the HTTP Server program sends a HTTP response message out, it pads four continuous blank lines, i.e., "\r\n\r\n\r\n\r\n", to the end of the .htm file as the ending signature of the entity body.
  - When the HTTP Client program read the HTTP response message line by line, it counts the number of **continuous** lines that are empty strings "" (NOT a null string). Once such number reaches 4, the entity body has been fully received.
  - It is assumed that the .htm file does not include such ending signature (in the real practice, length of the entity body may be included in the head line and used to determine the end of an entity body).
- **HTTP Client Program:**
  1. Display messages on the standard output to ask the user to input the DNS name/ip of your HTTP server.
  2. Buildup the TCP connection to your HTTP server with the Host Name input by User at the given port, and display the **RTT of establishing this TCP connection** in millisecond (the difference between the moments right before and after creating the socket object). Catch the exception, terminate the program, and display error messages on the standard output if any.
  3. Display message**s** on the standard output to ask the user to input the HTTP method type, name of the htm file requested, HTTP Version, and User-Agent, **respectively (separately please!)**. (hint: all inputs can be strings.)
  4. Use the above inputs from user to construct ONE HTTP request message and send it to the HTTP server program over the TCP connection. Your HTTP request message only needs to include the following lines. (Hint: At the **end of each line including the last empty line**, a **"\r\n"** is needed. The correctness of the format will be checked by the instructor.):
     ```
     The request line (hint: the URL should include a '/' in front of the htm file name)
     The header line for the field "Host:"
     The header line for the field "User-Agent:"
     <empty line>
     ```
  5. Receive and interpret the HTTP response message from the HTTP Server program **line by line**, display the **RTT** (File Transmission Time may be included) **of HTTP query** in millisecond (the difference between the moment right before HTTP request is sent and the moment right after HTTP response is received) as a single line (e.g., RTT = 1.089 ms), display the **status line** and **header lines** of the HTTP response message on the standard output, and save the data in the **entity body** to a .htm file to local directory if there is any. (Hint: (a) When one empty string "" (NOT a null string!) is read the FIRST TIME, it indicates the header lines are over and the entity body is going to start next line if the case is "200 OK".)
  6. Display a message on the standard output to ask the User whether to continue. If yes, repeat steps 3 through 6. Otherwise, close all i/o streams, TCP connection, and terminate the Client program.
- **HTTP Server Program**:
  1. Listen to the given port and wait for a connection request from a HTTP Client.
  2. Create a new thread for every incoming TCP connection request from a HTTP client.
  3. Read, display to the standard output, and interpret incoming HTTP request message line by line

- If the method given in the request line is NOT "GET", it is a "400 Bad Request" case
- If the file specified in the URL does not exit/cannot be open, it is a "404 Not Found" case
- Otherwise, it is a "200 OK" case

4. According to the case discovered above, construct ONE HTTP response message and send it to the HTTP client program over the TCP connection. Your HTTP response message needs include the following lines using the HTTP message format. (Hint: At the **end of each line including those empty lines**, a **"\r\n"** is needed.)
```
The status line
The header line for the field "Date:"
The header line for the field "Server: ", you may use any value of your choice
<empty line>
Data read from the requested HTML file line by line … (hint: for the 200 OK case only)
<empty line>
<empty line>
<empty line>
<empty line>
```
5. Repeat Step 3 through 4 until a null is read. (Hint: this happens when THIS client closes the TCP connection.)
6. Close all i/o streams and the TCP socket for THIS Client, and terminate the thread for THIS client. (Hint: when this happens, the parent thread is still alive doing Steps 1 and 2 forever, unless the Server process is killed/terminated.)

Part II (15%): Test your programs on the Virtual Servers in the cloud and your laptop/home computer.

Warning: to complete this part, especially when you work at home, you must first (1) **connect to GlobalProtect** using your NetID account (please read "**how to connect to GlobalProtect …**" at https://msudenver.edu/vpn/); then (2) **connect to the virtual servers cs3700a and cs3700b** using ***sftp*** and ***ssh*** command on MAC/Linux or ***PUTTY*** and ***PSFTP*** on Windows.

ITS only supports GlobalProtect on MAC and Windows machines. If your home computer has a different OS, it is your responsibility to figure out how to connect to cs3700a and cs3700b for programming assignments and submit your work by the cutoff deadline. Such issues cannot be used as an excuse to request any extension.

1. MAKE a directory "**HW3**" under your home directory on **cs3700a**.msdenver.edu and **cs3700b**.msudenver.edu, a subdirectory "**server**" under "**HW3**" on **cs3700a.msudenver.edu**, and a subdirectory "**client**" under "**HW3**" on **cs3700b.msudenver.edu**.
2. UPLOAD and COMPILE the ***server*** program under "**HW3/server**" and the ***client*** program under "**HW3/client**" on the VMs.
3. TEST ***the server program*** running on **cs3700a**.msudenver.edu together with ***a client program*** running on your laptop or lab computer and ***another client program***, simultaneously, running on **cs3700b**.msudenver.edu to test all the possible cases.
4. SAVE a file named ***testResultsClient.txt*** under "**HW3/client**" on **cs3700b**.msudenver.edu, which captures the outputs of your ***client*** program when you test it. You can use the following command to redirect the standard output (stdout) and the standard error (stderr) to a **file** on UNIX, Linux, or Mac, and view the contents of the file
```
java prog_name_args | tee testResultsClient.txt //copy stdout to the .txt file
       //if you want, you may also use "script" command instead of the "tee" command
       //to write both stdin and stdout into testResultsClient.txt while testing your
       //client program on cs3700a.  For how to use "script", see
       // https://www.geeksforgeeks.org/script-command-in-linux-with-examples/
       //or Google "script command in Linux" if the above link is broken.
cat file-name     //display the file's contents.
```