Homework 4 (100 Points) Simplified SMTP via TCP, Due Date*: 11:59pm 02/22/2023, Cutoff Date**: 11:59pm 02/24/2023

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

Submission:
1) upload and submit all your **.java file(s)** for both programs in **Canvas** (email is NOT accepted). Only ONE attempt is allowed for each student. NO Paper/Hardcopy or Email submission please!
2) upload, compile, test, and set up the **server program** (.java and .class files) on **cs3700a** under **HW04/server** and the **client program** (.java and .class files) on **cs3700b** under **HW04/client**, respectively, which will be used by the instructor for ***testing*** and ***grading*** your programs on **cs3700a** and **cs3700b**. See **task II** for more details.
3) the file named "clientOutputs.txt" under **HW04/client** on **cs3700b**. See **task II** for more details.

Recommended References:
[1] The lectures/classes ***before or on*** **the Due Date.**
[2] **Lab 1** under a "Weekly Learning Activities and Materials" module in Canvas
[3] **"TCP-based To-Upper-Case App (a Client and a Multi-Threaded Server)"** under the "**Network Programming in Java**" module in Canvas

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** in Canvas when submitting the .java files (***both team members are required to submit*** the same .java files in Canvas), and 2) put a ***team.txt*** file including both team members' names under your **HW04/** on **cs3700a** (***both team members are required to complete Task II under their own home directories on cs3700a and cs3700b***). For grading, I will ***randomly*** pick the submission in ***one team member's home directory*** on **cs3700a** and **cs3700b**, and then give both team members the ***same grade***. If the team information is *missing* in Canvas or **cs3700a**, or two or more submissions from students *not all* working in a team contain similar source code with just variable name/comment changes, it will be a violation of the Academic Integrity and students involved will receive **0** as the grade. Here are a few reminders on "What IS NOT Allowed" to hopefully avoid such violation (see Course Policies for more details).

1. Other than the example programs provided by the instructor to all students fairly, you may NOT look at code created by another person or provided by any online/offline resource (including but not limited to tutors and online/offline tutoring service) for any programming assignment/project until after you have completed and submitted the assignment yourself. You may NOT show or share your code to/with anyone other than the instructor. You may NOT ask anyone else to debug your code.

2. Students absolutely may NOT turn in someone else's code or solution with simple cosmetic changes (say, changed variable names or changed order of statements) to any homework, project, or exam.

3. It is strictly forbidden for any student to refer to code or solutions from previous offerings of this course or from any online/offline resource including but not limited to tutors and tutoring services unless it is provided by the instructor to all students fairly.

Grading: Your programs will be graded ***via*** **testing under your home directory on cs3700a and cs3700b** and points are **completely associated with how much task your programs can complete while running on cs3700a and/or cs3700b**.

1. You are highly recommended **NOT to use any IDE**, *online* or *on your computer*, to compile, debug, and test your programs. Instead, you should compile, debug, and test your programs in the command-line Linux environment on **cs3700a** and/or **cs3700b**, which is part of the learning objectives in this upper-division CS course. Being able to make your programs only work in an IDE (online or on your computer) but NOT on **cs3700a**/**cs3700b** ***cannot*** be used as an excuse for re-grading or more points either.

2. ALL the ***output messages or files*** that are REQUIRED to be ***displayed or created*** by your program in Task I are the only way to demonstrate how much task your program can complete. So, whether your program can display those messages correctly with correct information is critical for your program to earn points for the work it may complete.

3. It is also extremely important for your program to be able to correctly READ and TAKE the information from the ***input file(s)*** or the ***user inputs*** that follow the format REQUIRED in this programming assignment, instead of some different format created by yourself. Otherwise, your program may crash or get incorrect information from the input file(s) or user inputs that follow the required format.

4. Grading is NOT based on reading/reviewing your source code although the source code will be used for plagiarism check. A program that cannot be compiled or crashes while running on **cs3700a/cs3700b** will receive up to 5% of the total points. A submission of source code files that are similar to any online source code with simply cosmetic 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 (1) 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 **HW04/server** on **cs3700a** to include the commands that I need to use.

Part I (85%): Write a ***SMTP client program*** and a ***SMTP server program*** to implement the following simplified SMTP protocol based on TCP service. Please make sure your program supports multiple clients. You must use the port assigned to you and you may hard code it in both client and server programs.

- "**Telnet <your server's ip address> <port #>**" MUST be used to test your SMTP server program first (especially in the cases for those "503" responses), then your SMTP client can be tested using your SMTP server program.

- **SMTP Client Program** (NOT a TELNET client!!!):
  1. Display a message to ask the user to input the Host Name (or ip-address) of your SMTP server.
  2. Buildup the TCP connection to your SMTP server with the Host Name input by User at the given port. Catch the exception, terminate the program, and display error messages on the standard output if any. Wait for, read, and display the "220" response from the SMTP server.
  3. Display prompt messages on the standard output to ask the user to input sender's email address, receiver's email address, subject, and email contents, respectively. Since there may be multiple lines in email contents, the prompt message displayed by your Client program MUST prompt the ending signature pattern like "." on a line by itself.
  4. Use the user inputs collected above in Step 3 for the following 3-phase data transfer procedure (see step 3 on server side). In each of the following steps a. through e., display the **RTT (round-trip-time)** of each conversation in millisecond (e.g., RTT = 212.08 ms). (**hint**: the SMTP Client always sends correct command in order and that's why telnet must be used to test those out-of-order or incorrect incoming command message cases for the SMTP Server program.)
     a. Construct and send the "HELO <sender's mail server domain name>" command (e.g., HELO xyz.com) to the SMTP server program, wait for server's response and display it on the standard output.
     b. Construct and send the "MAIL FROM: <sender's email address>" command to SMTP server, wait for SMTP server's response and display it on the standard output.
     c. Construct and send the "RCPT TO: <receiver's email address>" command to the SMTP server program, wait for SMTP server's response and display it on the standard output.
     d. Construct and send the "DATA" command to the SMTP server program, wait for SMTP server's response and display it on the standard output.
     e. Construct and send the Mail message to the SMTP server. The format of this Mail message MUST follow the format detailed on the **slide titled "Mail message format"**. Wait for SMTP server's response and display it on the standard output.
  5. Display a prompt message to ask the User whether to continue. If yes, repeat steps 3 through 5. Otherwise, send a "QUIT" command to the SMTP server, display SMTP Server's response, close TCP connection, and terminate the Client program.

- **SMTP Server Program**: (server's ip or client's ip can be its dns name below.)
  1. Listen to the given port and wait for a connection request from a SMTP Client.
  2. Create a new thread for every incoming TCP connection request from a SMTP client. Send the "220" response including server's ip address or dns name to the SMTP client.
  3. Implement the following 3-phase data transfer procedure (see step 4 on client side):
     a. Wait for, read, and display the "HELO …" command from the SMTP client. If the incoming command is NOT "HELO …", sends "503 5.5.2 Send hello first" response to the SMTP client and repeat step 3.a.
     b. Send the "250 <server's ip> Hello <client's ip>" response to the SMTP client.
     c. Wait for, read, and display the "MAIL FROM: …" command from the SMTP client. If the incoming command is NOT "MAIL FROM: …", sends "503 5.5.2 Need mail command" response to the SMTP client and repeat step 3.c.
     d. Send the "250 2.1.0 Sender OK" response to the SMTP client.
     e. Wait for, read, and display the "RCPT TO: …" command from the SMTP client. If the incoming command is NOT "RCPT TO: …", send "503 5.5.2 Need rcpt command" response to the SMTP client and repeat step 3.e.
     f. Send the "250 2.1.5 Recipient OK" response to the SMTP client.
     g. Wait for, read, and display the "DATA" command from the SMTP client. If the incoming command is NOT "DATA", send "503 5.5.2 Need data command" response to the SMTP client and repeat step 3.g.
     h. Send the "354 Start mail input; end with <CRLF>.<CRLF>" response to the SMTP client.
     i. Wait for, read, and display the Mail message from the SMTP client line by line. (hint: "." is the ending signature.)
     j. Send the "250 Message received and to be delivered" response to the SMTP client.
  4. Repeat Step 3 until the "QUIT" command is read. Upon receiving "QUIT", send the "221 <server's ip> closing connection" response to the SMTP client and go to Step 5.
  5. Close all i/o streams and the TCP socket for THIS Client, and terminate the thread for THIS client.

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

Warning: to complete this part, especially when you work at home, you must first (1) **connect to GlobalProtect** using your NetID account; then (2) **connect to the virtual servers cs3700a and cs3700b** using ***sftp*** and ***ssh*** commands on a MAC computer or using software like ***PUTTY*** and ***PSFTP*** on a Windows machine. (See Lab 1 on ***how to***.)

ITS only officially 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. CREATE a directory "**HW04**" under your home directory on **cs3700a**.msdenver.edu and **cs3700b**.msudenver.edu, a subdirectory "**server**" under "**HW04**" on **cs3700a.msudenver.edu**, and a subdirectory "**client**" under "**HW04**" on **cs3700b.msudenver.edu**.
2. UPLOAD and COMPILE the ***server*** program under "**HW04/server**" on **cs3700a** and the ***client*** program under "**HW04/client**" on **cs3700b**.
3. TEST ***the server program*** running on **cs3700a**. together with ***a client program*** running on your local Windows or Mac computer and ***another client program***, simultaneously, running on **cs3700b**.msudenver.edu to test all the possible cases. For testing, the ***server program*** always has to **start running first** before starting any client program. Meanwhile, you will have to run a ***TELNET client*** on cs3700b to test your ***server program*** independently ***first*** BEFORE testing your ***client program***.
4. SAVE a file named ***testResultsClient.txt*** under "**HW04/client**" on **cs3700b**., which captures the output messages 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
cat file-name     //display the file's contents.
```
5. If you work in a team of two students, you must put a ***team.txt*** file including both team members' names under your **HW04/server** on **cs3700a** (***both team members are required to complete Task II under their own home directories on cs3700a and cs3700b***). For testing and grading, I will ***randomly pick*** the submission in one of those two *home directories*, and then give both team members the ***same*** grade for Task I.