Homework 4, Due Date*: 10:00am 03/03/2015, Cutoff Date: 10:00am 03/06/2015**

Submission: Upload all source code (.java) files to Blackboard, and java programs (.class files) to EC2 instances

***Late penalty will apply for past-due late submission; **Submission will NOT be accepted after the cutoff deadline**

Part I: Write a client program and a server program to implement the following simplified SMTP protocol based on TCP service. Please make sure your program supports multiple clients. You may choose a port and hard code it in both client and server programs.

- "**Telnet <your server's ip address> <port>**" can be used to test your SMTP server program first (especially those "505" 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 above user inputs in the following 3-phase data transfer procedure (see step 3 on server side). In each of the steps 4.2 through 4.6, display the **RTT (round-trip-time)** of each conversation in millisecond (e.g., RTT = 212.08 ms).
     a. Send the "HELO" command followed by sender's mail server domain name (e.g., xyz.com) to the SMTP server program, wait for server's response and display it on the standard output.
     b. 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. 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. Send the "DATA" command to the SMTP server program, wait for SMTP server's response and display it on the standard output.
     e. 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 3.2.
     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 3.4.
     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 3.6.
     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 3.6.
     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: Test your programs with multiple clients using the Amazon EC2 Cloud.

⚠ Please STOP your ec2 instances whenever you are NOT working on them. Please do NOT start or stop ec2 instances whose names do not contain your user name. Thank you!

(The instructor will start and use your instances and her local computer to test your programs and give credits for Parts I and II according to the test results. The latest time of your ***.class*** and ***.txt*** files on your EC2 instances is the submission time of part II.)

(continue on next page!!!)

1. MAKE a directory "**HW04**" under your home directory on each of your EC2 instances.

2. UPLOAD your ***server*** program to “HW04” on your instance in *N. VA* and your ***client*** program to “HW04” on your instance in *Sydney*.
3. TEST the server program using the TELNET client on your local computer or your EC2 instance in *Sydney*.
4. TEST the server program together with the client program on a computer and, simultaneously, on your EC2 instance in *Sydney* to test all the possible cases. Here, you need to have one server and at least two clients.
5. SAVE a file named ***testResultsClient.txt*** on your EC2 instance in *Sydney*, which captures the outputs of your ***client*** program when you test it.