Edits to the original doc:

1. **If password incorrect, print on STDOUT "Wrong Passwd\n" and close connection.**
2. **See Phase-4 client-side**

Socket Programming Assignment: Email Application

Due Date: March 31st , 2019 : 11pm

In this lab, we will implement a simplified version of PoP3 email protocol between a client and a server. You may want to read up on POP3 (which we will cover later) but this is not strictly required. Essentially the email server has access to a password file and a directory structure where messages(emails) of the different users are stored under individual user folders. Each client contacts the server, shares its login credentials which are verified by the server. After successful client login, a client can determine the number of messages it has stored at server and then retrieve a few of these messages. In our simplified version, messages simply correspond to files of different types: txt, pdf, jpg etc

This lab is to be done individually. Every single code of line should be your own. Note evaluation of this lab will be via an autograded script.

The program can be written in phases, with increasing levels of functionality in each step.

You will require some files and folder structure for this lab. The same is available as a zip for download under helper files. NOTE: The content within the provided files may not be the same as one used by us during auto-evaluation but the structure will be the same. We will have lot many more messages of different sizes (KB to MB) and types (mp4, png etc).

Use loopback interfaces for initial testing (i.e. all processes run on the same machine). Once it works fine, you should run the processes on different machines and make it work. In our grading, we will run some processes on same and some on different machines.

Phase-1: Simple login and and logout [6 marks]

1. Server called SimpleEmailServerPhase1.cpp. Note the server needs to handle only one client in phase-1.
   a) 2 command line arguments: portNum, passwdfile
      i. Print usage on stderr and exit with exit-code 1, if wrong number of command line arguments are given

ii. Print appropriate error message on stderr and exit with exit-code 2, if bind on given port fails

iii. Print appropriate error message on stderr and exit with exit-code 3, if given passwdfile not present or not readable

b) Server should listen on given port for incoming connection from client

i. STDOUT on successful bind "BindDone: portNum\n"

ii. STDOUT on successful listen "ListenDone: portNum\n"

iii. STDOUT on incoming connection "Client: ipaddr:port\n" where ipaddr and port are the client-side info

c) After forming incoming connection from client, read a null-terminated string from socket (null is also sent on the socket by the client). If it is of the format "User: user-name Pass: passwd", parse the string; compare the user-name and password with what is present in the passwdfile.

i. Unable to parse string, print on STDOUT "Unknown Command\n" and close connection.

ii. If invalid user (user not found in the passwd file), print on STDOUT "Invalid User\n" and close connection.

iii. If password incorrect, print on STDOUT "Wrong Passwd**\n**" and close connection.

iv. If login successful, print on STDOUT "Welcome user-name\n", where user-name is the login user-name and send the same message to the client as well.

d) After forming incoming connection from client, read a null-terminated string from socket. If it is of the format "quit", print on STDOUT "Bye user-name\n" and close connection

i. Unable to parse string, print on STDOUT "Unknown command\n" and close connection.

2. Client called SimpleEmailClientPhase1.cpp

a) 3 command line arguments: serverIPAddr:port, user-name, passwd

i. Print usage on stderr and exit with exit-code 1, if wrong number of command line arguments are given

ii. Print appropriate error message on stderr and exit with exit-code 2, if connection to server fails

b) Form a TCP connection to the given serverIPaddr:port

i. STDOUT on successful connection "ConnectDone: serverIPAddr:port\n"

c) Send null terminated string “User: user-name Pass: passwd” over the TCP connection.

   i. Print on STDOUT any message received from server

d) Send null terminated string “quit” over the TCP connection.

e) Client can exit after this

3. Marking scheme:

   a) Correct exit codes at server: 1

   b) Correct exit codes at client: 1

   c) Correct Login and Logout: 4

Phase-2: List Command and In-sequence Multi-Client [3+1=4 marks]

1. Server called SimpleEmailServerPhase2.cpp, similar to phase1, except for addition of another input argument called folder-path and command called “LIST”
   a) Server now takes 3 command line arguments: portNum, passwdfile, user-database, where user-database specifies the folder where user messages are stored (see the tar.gz file for the structure; user-database is the top most directory, under this many folders named after the user-names will be present, which in turn contain a number of files of different types (.txt, .pdf, .jpg etc) that correspond to messages of the user.)
      i. In addition to earlier error messages, Print appropriate error message on stderr and exit with exit-code 4, if unable to access the given user-database (top-most directory).
   b) After forming incoming connection from client and successful login of client, read a null-terminated string. If it is of the format “LIST”, traverse the user-database to determine the number of messages present for this user (the tar.gz will have this information, essentially the number of files under a given user-name folder).
      i. print on STDOUT “ user-name: No of messages N \n”, where user-name is the login user-name, N is the number of messages; send the same message to the client as well.
      ii. If incorrect format of string from client, STDOUT “Unknown Command\n” and close the client connection
      iii. If user-name folder not present or readable, STDOUT “user-name: Folder Read Fail\n”, and close the client connection

   c) Server to run continuously and accept other connections from other clients after interaction with a given client. Note server need not handle simultaneous clients yet in this phase; only one after the other.

2. Client called SimpleEmailClientPhase2.cpp, similar to phase1, except for the addition of the LIST Command.
   a) On successful server connection and login, send null terminated string "LIST" and print on STDOUT any message received from server.
   b) Remaining behaviour same as phase1 client i.e. send Quit after List.
3. Marking scheme:
   a) Correct behaviour implementing LIST functionality from server: 2
   b) Correct behaviour implementing LIST functionality from client: 1
   c) Server supports many clients, one after another: 1

Phase-3: Message Transfer [6 Marks]

1. Server called SimpleEmailServerPhase3.cpp, similar to phase2, except for addition of another command called "RETRV".
   a) After forming incoming connection from client and successful login of client, read a null-terminated string from socket. If it is of the format "RETRV M", where M refers to the id of the message, the server picks the file M.xxx from the relevant user-name directory (xxx can be txt, pdf, jpg etc) and transfers the same to the client. Note: It is assumed that the id of the message is same as the file name.
      i. If incorrect format of string from client, STDOUT "Unknown Command\n" and close the client connection.
      ii. If user-name folder not present/readable or message of that id not present, STDOUT "Message Read Fail\n", and close the client connection.
      iii. print on STDOUT " user-name: Transferring Message M \n", where M is the message id and start transfer of message to client.
2. Client called SimpleEmailClientPhase3.cpp, similar to phase2, except for a) the addition of two other arguments, one of which is a list and another a folder and b) the RETRV Command.
   a) 5 command line arguments: serverIPAddr:port, user-name, passwd, list-of-messages, local-folder where list of messages is a list of numbers separated by comma (e.g. 3, 5, 9 or 4 or 1, 8) and local-folder is the folder where the downloaded messages will be stored.

i. Apart from earlier error messages, print usage on stderr and exit with exit-code 3, if the list does not parse to a list of numbers.

ii. Delete any existing local folder with the same name and create a new one (we want this folder empty each time client runs). Exit with error-code 4, if unable to create/access the local folder.

b) On successful server connection and login, retrieve messages with message ids as specified in the list via a loop within which you send the null terminated string "RETRV X" and print on STDOUT, "Downloaded Message X" once download of the message is complete. The downloaded message needs to be stored in the relevant local folder. For example, if message list if 3, 8; you will send RETRV 3 first, once 3.xxx is downloaded, you will send RETRV 8. (When coding, you should be sure to check (manually i.e.) that the downloaded file is the exact same one as present in the server, a hash of the files via md5sum is useful here. If they don;t match that means there is some bug in your file transfer.)

c) Remaining behaviour same as phase2 client i.e. send Quit after all the RETRVs.

3. Marking scheme:

a) Correct behaviour from server: 3

b) Correct behaviour from client: 3

Phase-4: Simultaneous Multiple Clients [4 marks]

1. Server called SimpleEmailServerPhase4.cpp, similar to phase3, except that it should support at least 2 simultaneous clients

a) You *do not* have to use multiple threads (or multiple processes) for this; in fact I recommend strongly that you instead use the select system call appropriately

2. Client called SimpleEmailClientPhase4.cpp, similar to phase3, except the following

a) It should take a sixth command-line argument: interval (specified in seconds)

i. After authentication, it should sleep for the specified interval before sending the RETRV command. The purposes of this feature of the client is to be able to easily test multiple simultaneous clients at the server. While one sleeps, others login and retrieve files.

3. Marking:

a) Correct server implementation: 4

<u>General Instructions:</u>

1. This lab is to be done individually.
2. You are **not allowed to exchange code snippets or anything** across groups. In case of any doubts, you are allowed to consult any Internet resource, books, or the instructor. While it is alright to read on general socket programming on the Internet, and look at code examples, this should be only for the purpose of understanding. Do all the coding yourself, do not copy or cut-paste code from any website. You can use only socket programming for these projects. Do not use any sophisticated libraries downloaded from the web. Please understand the spirit behind all these and follow them strictly.
3. You can use C/C++ for coding, **no other language**. Although Java/Perl and other languages may provide a better interface, C/C++ are bare-bones, and hence appropriate for a networking course.
4. Give sufficient time for testing the code, do not cram everything to the last day. Remember that in industry, people spend more time testing than coding!
5. Commenting and indenting are important. Comment *during/before* coding, not at the end. For every variable/function you should have a comment. Use appropriate code comments for explaining the logic where necessary.
6. Pay attention to the variable names, function names, file names, directory names, etc. Make sure they are intuitive.
7. Refer to http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html for more instructions on coding conventions. Although this link is for Java, many of the principles are applicable for any programming language.

<u>Submission guidelines</u>

Will be via bodhitree; more details later.