CS3205 - Computer Networks
Jan – May 2018
Prof. Siva Ram Murthy
Dept. of CSE, IIT Madras
Assignment 1: Simple Mail Client/Server Application Implementation
Due date: Feb. 11, 2018, 11:59 PM, On Moodle

Extension: 15% penalty for each 24-hr period;
Max. of 48-hrs past the original deadline

1 Assignment description

The objective of this assignment is to implement a simple email client, and a mail server using the Socket Interface. We will be implementing a minimal client and server in this assignment. The objective of the assignment is to familiarize yourself with writing a network server and a client.

There are two major components to the software: (i) the email server, and (ii) the email client. The communication will be based on a TCP socket.

1.1 Email server

The purpose of the email server is to respond to the requests generated by the client. The server is expected to be up and running on the remote machine, listening to a specific port.

The server software contains these major components:

(i) NETWORK INTERFACE: This piece is responsible for receiving requests from the client, and passing it to the COMMAND PROCESSOR.

(ii) COMMAND PROCESSOR: This is responsible for processing the request from the client, running appropriate system commands and passing the output generated to the NETWORK INTERFACE.

When the server is started, the default directory is called the "root" directory of the server. The user's mail spool files will be stored in this directory.

The server one spool file to each user, for storing the mails delivered to the specified user.

The user's spool file consists of multiple email records. The email record format for `userC` is as given below. Each email record is terminated by `###`.

```
From: userA
To: userC
Date: Sat Jan 28 21:38:48 IST 2017
Subject: Hello
...
Contents
...
###
```

```
From:  userB
To: userC
Date: Sat Jan 28 21:39:16 IST 2017
Subject: Hello again
...
Contents
...
###
From:  userD
To: userC
Date: Sat Jan 28 22:39:16 IST 2017
Subject: Congrats
...
Contents
...
###
```

The commands that the *server* will support are as follows:

- LSTU
  Semantics: The server will send a list of users (separated by spaces) to the client, which then prints the message.

- ADDU <userid>
  Semantics: The server will create an empty mail spool file if the specified userid is not present; otherwise, the server will reply with a message, "Userid already present" to the client.

- USER <userid>
  Semantics: If the userid is valid, the server sends an ack message to client (to be printed) that the specified user exists and has XY number of messages in his/her spool file. Also, the server will next respond to mail item requests from the specified user's file. The current user is set to this specified user.

  The mail read pointer is set to the first mail item in the user's spool file.

  If the userid is invalid, the server sends an error message to client (to be printed) that specified user does not exist.

- READM

  Semantics: If the current mail pointer of the current user is INVALID, the server sends a message "No More Mail" to the client.

  The server sends the current mail item (being pointed to) in the spool file, to the client (that prints it). The spool file's mail pointer will then move forward to the next item. If this is the last item in the spool file, then the mail pointer will go back to the first item, if exists. Otherwise, it will be set to INVALID.

  To terminate each mail message, use three special characters (###). Assume that this pattern does not appear in each mail message.

- DELM

  Semantics: If the current mail pointer of the current user is INVALID, the server sends a message "No More Mail" to the client.

  The server deletes the current mail item (being pointed to) in the spool file, and sends a confirmation message "Message Deleted". The spool file's mail pointer will then move forward to the next item. If this is the last

item in the spool file, then the mail pointer will be go back to first item, if exists. Otherwise it will be set to INVALID.

This can be verified by checking the user's spool mail file in the server directory.

- SEND <userid>

  Semantics:

  If the userid is valid, copy the message contents to the specified receiver's userid (as sent from the current user). The server will store from and To fields, and contents accordingly. The mail will be appended at the end of the receiver's spool file.

  If the userid is invalid, the server sends an error message to the client (to be printed) that the specified user does not exist.

- DONEU
  Semantics: The current user's mail spool reading activity is stopped. Any subsequent READM and DELM commands will result in error messages being sent from the server to the client. The next current user has to be set using the USER command.

- QUIT
  Semantics: The current session with the client ends. The server closes the connection with the client.

1.2 Email Client

An end user will be running the email client to communicate with the server. The client software can be partitioned into these major pieces:

(i) USER INPUT INTERFACE: The user-input interface accepts user commands, processes them, and passes appropriate data to the network interface.

(ii) NETWORK INTERFACE: The network interface is responsible for establishing the required socket(s) with the remote server. It is responsible for accepting the data provided by the USER INPUT INTERFACE above and transmitting it over the socket. It is responsible for reading the responses transmitted by the server, and either storing the data on file locally, or displaying it to the screen.

Client will wait at a prompt:
Main-Prompt>
The client software interface will then accept the following commands:

- Listusers

  The client will send the message `LSTU userid` to the server, and print the server's response.

- Adduser <userid>

  The client will send the message `ADDU userid` to the server, and print the server's response.

- SetUser <userid>

  The client sends the message `USER userid` to the server, and prints the server's response.

  The client will then print a sub-prompt as follows:

  Sub-Prompt <userid>

  and wait for subsequent commands, specific to this user. These commands are:

  1. Read

     The client will send the message `READM` to the server, and print the server's response.

2. Delete

   The client will send the message `DELM` to the server, and print the server's response.

3. Send <receiverid>

   The client prints a prompt, "Type Message:". The user will type a mail message terminated by `###`. The client then sends the message `SEND receiverid <message>` to the server, and prints the server's response.

4. Done

   The client will send the message `DONEU` to the server, and print the server's response.

- Quit

  The client will send the message `QUIT` to the server, print the server's response, and close the TCP session.

The role of the client interface is thus to accept the user commands, parse the commands, and generate the appropriate Email Server commands that will be passed to the server.

2 Sample Session

Assume that you have created the files emailclient.c and emailserver.c and the corresponding executables in your Assignment1 directory. Please use a suitable unique port number for your server. The directory Assignment1/MAILSERVER will be the "root" directory for the server purposes.

(Note: You can test for the following cases – (i) server and client running on the same machine. You can invoke the client as: emailclient localhost <port>; (ii) server and client running on different machines.)

```
% cd Assignment1
% ./emailserver 25678 &                -- Server is running on say, dcf15

% cd Assignment1
% ./emailclient dcf15 25678           -- Client running on another host
.... Server's initial response.
Main-Prompt> Listusers
.... Server's response.
Main-Prompt> Adduser UserA
...
Main-Prompt> Adduser UserB
...
Main-Prompt> Adduser UserC
...
Main-Prompt> Listusers
.... Server's response.
Main-Prompt> SetUser UserA
...
Sub-Prompt-UserA>Read
... (Error)
Sub-Prompt-UserA>Send UserB
Type Message: How do you do?###
Sub-Prompt-UserA>Send UserC
Type Message: What is the time?###
Sub-Prompt-UserA>Send UserB
Type Message: Is the ATM working?###
```

```
Sub-Prompt-UserA>Send UserC
Type Message: Are you studying?###
Sub-Prompt-UserA>Done
Main-Prompt> SetUser UserB
...
Sub-Prompt-UserB> Read
...
Sub-Prompt-UserB> Send UserA
Type Message: Where are you now?###
...
Sub-Prompt-UserB> Delete
...
Sub-Prompt-UserB> Done
...
Main-Prompt> SetUser UserA
...
Sub-Prompt-UserA>Read
...
Sub-Prompt-UserA>Done
...
Main-Prompt> Quit
...
%
```

3 What to Submit

Name your assignment directory as Assignment1 (Note: ALL UPPERCASE)

Once you are ready to submit, change directory to the directory above Assignment1, and tar all files in the directory with the command:

`tar czf XYxyBabc-Assignment1.tgz Assignment1`

where XYxyBabc refers to your roll number.

The directory should contain the following files:

- Client Software Files
- Server Software files
- Makefile
  Typing command 'make' at the UNIX command prompt, should generate all the required executables.
- a README file containing what port number to use, and instructions to compile, run and test your program. README file should be written such that a TA must be able to run your code without your presence/help.
- a REPORT which consists snapshots (or copy pasted) of different sessions that are carried out. This can also include extra/new things that you tried during the experiment (This gives you extra points). And, give a brief summary as to what you learnt in this experiment and how much beneficial you feel the experiment was.

Make sure that all files have been correctly tar-red with the appropriate command. Then, submit the *tgz* file via Moodle.

4 Help

1. WARNING ABOUT ACADEMIC DISHONESTY: Do not share or discuss your work with anyone else. The work YOU submit SHOULD be the result of YOUR efforts. Any violation of this policy will result in an automatic ZERO on the assignment, a 'U' in the course, and other academic penalties.

2. Ask questions EARLY. Do not wait until the week before. This assignment is quite time-consuming.

   Please make use of the office hours of the TA to the fullest extent. If you need to meet him during other times, please send him an email for appointment.

3. Implement the solutions, step by step. Trying to write the entire program in one shot, and compiling the program will lead to frustration, more than anything else.

   For example, implement the server COMMAND PROCESSOR first, without any network interface. Make sure that textual input generates the correct server responses.

   Then, implement the client USER INPUT INTERFACE to accept user level commands and generate commands that the server will understand.

   Now, implement the network sockets at both ends, and test **ONE COMMAND AT A TIME** between client and server.

4. Demonstration of code execution to the TAs MUST be done using the students code uploaded on Moodle.

5 Grading

- Server: 35 points
- Client: 35 points
- Report: 20 points
- Viva voce: 10 points

NO MAKE FILE: -10 points;
NO README: -10 points;