Computer Networks
CS3205
Assignment 1

Simple Mail Client/Server Application Implementation

Soumith Basina - CS18B005

Date Submitted: March 5th, 2021

Instructor .... Prof. Siva Ram Murthy

1 Description

The objective of this assignment is to implement a simple email client, and a mail server using the Socket Interface. There are two major components to the software: (i) the email server, and (ii) the email client. The code for both of them are in emailserver.c and email-client.c respectively. The communication will be based on a TCP socket.
We will look at how the commands are implemented in both of them.

2 Example Sessions

Client Session:

```
$ ./emailclient 127.0.0.1 8080
Socket created successfully
Connected to the Server
Main-Prompt> Adduser abc
User created successfully!
Main-Prompt> Adduser abc
User already exists, pick a unique name
Main-Prompt> Adduser def
User created successfully!
Main-Prompt> SetUser abc
```

```
User exists, setting current user to abc
You have 0 messages
Sub-Prompt-abc> Read
No mails found
Sub-Prompt-abc> Delete
No mails to delete
Sub-Prompt-abc> Send def
Enter the Subject: hey
Enter the Message:
hello
line1
line2
###
Message delivered successfully
Sub-Prompt-abc> Dome
Illegal command
Sub-Prompt-abc> Done
User "abc" closed
Main-Prompt> SetUser def
User exists, setting current user to def
You have 1 messages
Sub-Prompt-def> Read
From: abc
To: def
Date: Fri Mar  5 10:43:50 IST 2021
Subject: hey
Contents
hello
line1
line2
Sub-Prompt-def> Delete
Mail has been deleted succesfully
Sub-Prompt-def> Read
No mails found
Sub-Prompt-def> Done
User "def" closed
Main-Prompt> Listusers
def abc
Main-Prompt> Quit
Quitting...
```

Server Session:
The server prints all the commands it receives

```
$ ./emailserver 8080
Socket created successfully
Socket bind successful
Server listening
Server accepted the client
```

```
ADDU abc
ADDU abc
ADDU def
USER abc
READM
DELM
SEND def
DONEU
USER def
READM
DELM
READM
DONEU
LSTU
QUIT
```

3 Implementation

The email server acts as the backend for the application. The server stores the mails by storing them in a spool file for each user. Those spool files are stored in the `MAILSERVER` subdirectory. It is responsible for modifying the spool files depending on the command.
The email client acts as a command processor, rejecting any illegal commands and forwarding the corresponding valid commands to the server. The client and the server communicate through sockets that are created in the beginning of the code files.

There are 8 commands that can be given to the client and the server.

3.1 Listusers / LSTU

This command is used to list the users currently present in the server. This is done by printing the filenames of the spool files in the `MAILSERVER` subdirectory. Using `opendir()` and `readdir()` functions in `dirent.h` include file, we can read the filenames in the directory and send them to the client. The client prints out the received names until it encounters `##END##`, which signifies the end of the list of names.

3.2 Adduser <userid>/ ADDU <userid>

This command is used to add the user to the server if not present, or display an error otherwise. Using `"wx"` mode in `fopen`, we can create a new file if it doesn't exist and returning `NULL` instead of overwriting if it exists. We send `##OK##` if the user is created successfully or `##ERR##` if user already exists, to the client. The filename of the created files is the user id itself.

3.3 Quit / QUIT

This closes the sockets in both server and client and breaks out of the main loop to exit.

3.4 SetUser <userid>/ USER <userid>

This command is used to ”login” to the given user id if it exists, or display an error otherwise. Using `"r+"` mode in `fopen`, we can open the file if it exists, returning `NULL` otherwise. If it succeeds, we then count the number of mails in the spool file, rewind the spool pointer to the beginning and then send the number to the client to be printed there. If the client receives `##ERR##`, it prints an error message. Otherwise, it receives a number which is the number of mails in the spool file. The message is printed along with the number of mails and then the program enters an infinite loop inside. It then can only accept the following 4 commands.

3.5 Read / READM

This command is used to read the mail at which the mail pointer is pointing and move the pointer to the next mail. If there are no mails, an error is displayed. The server reads the spool file and sends it line by line to the client. If it sees `"###"`, the mail pointer is incremented and breaks off the loop. If it is at the end of the file, it rewinds to the beginning before breaking. The client prints the lines it received until it receives the end of the mail.

3.6 Delete / DELM

This command is used to delete the mail at which the mail pointer is pointing and move the mail pointer to the next mail. If it was the last mail, rewind the pointer to the beginning. If there are no mails, an error is displayed. The server creates a temporary file, copy all the contents of the spool file except the current mail, delete the original mail and rename the new file.

3.7 Send <userid>/ SEND <userid>

This command is used to send the mail to the given user id. If the given user id doesn’t exist, an error is displayed. The client first takes a line for subject, sends it to the server. Then it can take multiple lines of message until `"###"` is given as input which then breaks out of the loop. All the lines are then printed concurrently to the spool files. Time and date is printed using `asctime` and `localtime` in the `time.h` include file. The end of the mail is marked by `"###"` in the spool file.

3.8 Done / DONEU

This command closes the spool file pointer in the server and exits from the sub prompt infinite loop.

4 Summary

This experiment made me understand basic socket programming in C, sending and receiving data through sockets, basic usage of Makefile and have an understanding of how an email service might work . This has been helpful in having a solid grasp on the basics of Networks.