CS3205: Introduction to Computer Networks

Report: Assignment 1

Anirudh S
CS17B003

Problem statement

The task of this assignment is to implement a simple mail server and client which communicate using the TCP socket interface. The client receives commands from the user, processes the commands and sends an appropriate messages to the server. The messages travel through the TCP sockets and are received by the server. The server parses these messages and performs the necessary actions. The server sends messages, like acknowledgements, mail which is read, etc, back to the client which the latter displays to the user. Both the client and server codes are to be written in *C*.

Solution

The code development phase was divided into three parts:

1. Server
2. Client
3. Socket interface

The details about each of these phases are described in the following sections.

1 Server

The core component of the server is the command processor, which processes the messages sent by the client and executes the necessary actions. The main tasks involved are as follows.

1. **Keeping track of all the user and the current active user**
   Details of all the users are maintained in a linked list. The user id and the number of messages in the user's spool file are the details which are stored for every user. When a new user is added, a node corresponding to this user is added in the linked list. One spool file corresponding to this user is opened in the `MAILSERVER` directory (The directory is created if not already present). The current user, set using `SetUser` UI command, is kept track of by maintaining a global pointer to the node of the user in the linked list. The global variable is required in order to ensure that this information is maintained across multiple function calls (one call per client message).

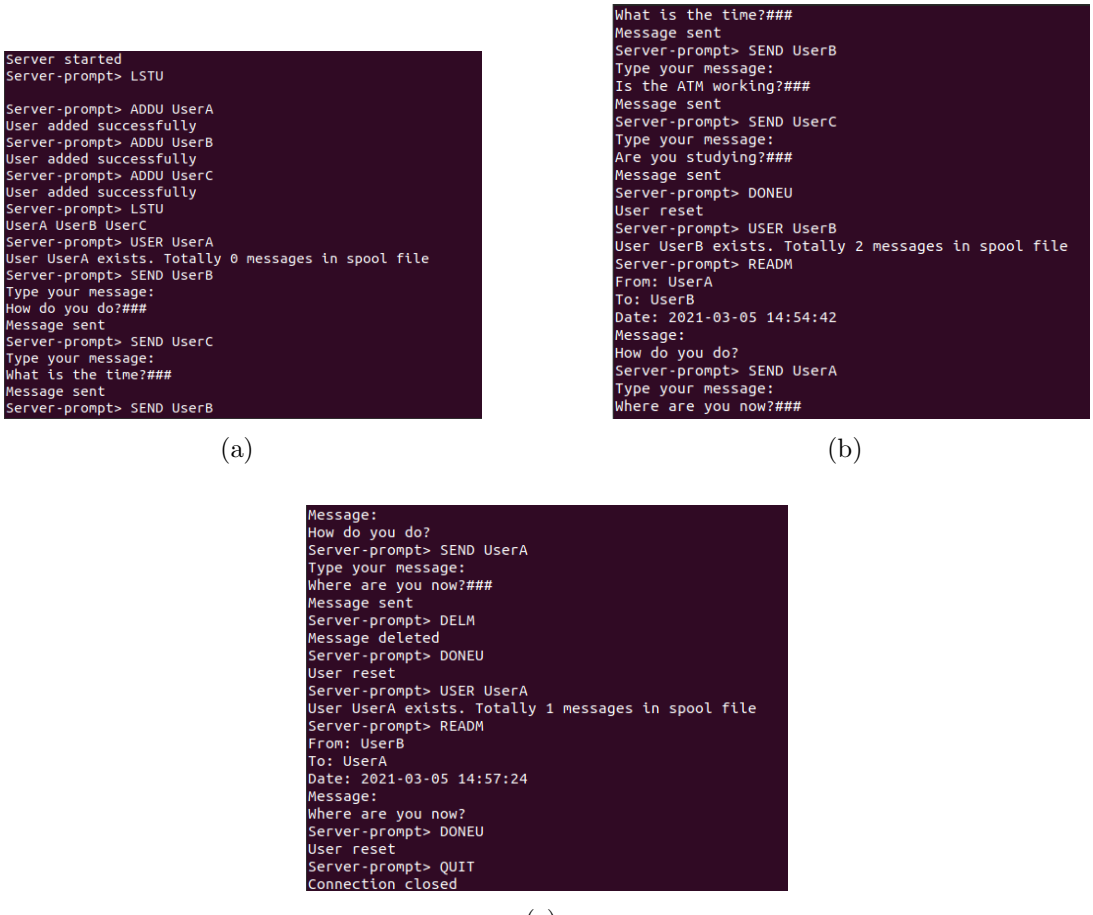
2. **File operations**
   Main file operations which are involved in this assignment are reading the messages from the spool file, writing a message to the spool file and deleting a message from the spool file.

(a) **Reading messages:** The file pointer of the current active user (called *read pointer*) and the number of messages read from spool file are stored as a global variables for reasons mentioned previously. Initially the read pointer is `NULL`. When the user first tries to read, the file is opened in `read` mode and assigned to the read pointer. The text is read till a `###` is encountered and this text is returned to the client. The number of messages read is incremented. If this number becomes equal to the total number of messages in the spool file, the read pointer is reset to the beginning and the counter is set to 0.

(b) **Writing messages:** When a send request is sent, the spool file of the receiver is opened in `append` mode. The `From` user is obtained using the current user and the `To` user is sent by the client. The date and time is calculated from the computer using `time.h` library. After these fields are written, the message sent by the client is written to the file and closed. The number of spool messages is incremented in the node of the receiver.

(c) **Deleting messages:** The deletion of the message currently pointed to by the pointer happens as follows. The original spool file is opened in `read` mode and a temporary file is opened in `write` mode. The number of messages read till now are copied to the temporary file from the spool file. One message (the message to be deleted) is skipped and the remaining messages are copied. The original spool file is then deleted and the temporary file is then renamed as required.

As initially the socket interface is still not set up, the commands were read from the standard input. A sample run is shown in Figure 1.



(a) (b) (c)

Figure 1: Working of server command processor

```
Main-Prompt> Listusers
LSTU
Main-Prompt> Adduser UserA
ADDU UserA
Main-Prompt> Adduser UserB
ADDU UserB
Main-Prompt> Adduser UserC
ADDU UserC
Main-Prompt> Listusers
LSTU
Main-Prompt> SetUser UserA
USER UserA
Sub-Prompt-UserA> Read
READM
Sub-Prompt-UserA> Send UserB
Type message:
How do you do?###
SEND UserB How do you do?###
Sub-Prompt-UserA> Send UserC
Type message:
What is the time?###
SEND UserC What is the time?###
Sub-Prompt-UserA> Send UserB
Type message:
Is the ATM working?###
SEND UserB Is the ATM working?###
Sub-Prompt-UserA> Send UserC
Type message:
Are you studying?###
SEND UserC Are you studying?###
Sub-Prompt-UserA> Done
DONEU
Main-Prompt> SetUser UserB
USER UserB
Sub-Prompt-UserB> Read
READM
Sub-Prompt-UserB> Send UserA
Type message:
Where are you now?###
SEND UserA Where are you now?###
Sub-Prompt-UserB> Delete
DELM
Sub-Prompt-UserB> Done
DONEU
Main-Prompt> SetUser UserA
USER UserA
Sub-Prompt-UserA> Read
READM
Sub-Prompt-UserA> Done
DONEU
Main-Prompt> Quit
QUIT
```

Figure 2: Client-user interface working with the prepared messages displayed

2 Client

The main operations performed by the client are taking commands from the user and interacting with the server. The client has a prompt for the user to input the commands. All the valid commands are parsed and converted to an equivalent server commands. These commands are then sent to the server using the socket interface. Then it receives a message from the server which is displayed to the user. To display the sub-prompt, the current active user is maintained using a global variable (to maintain the state across communications to the server). It is set to the user id whenever a `SetUser` command is provided by the user. If the server returns an error, the current active user is reset and the main prompt is displayed instead. A sample run using the example from the assignment problem sheet is shown in Figure 2. The output is the message which is sent to the server.

3 Socket Interface

The code for the socket interface is largely borrowed from the reference[1] provided in the assignment slides. Initially, the server creates a socket and bind to a particular port. Then it passively listens for the client. On the client side, a socket is created and connection is established using the IP address and the port number of the server. After successful connection, a message is printed. Next a call to function `prepare_command` is made which prompts the user, reads the command, processes it and return the command message to be sent to the server. This message is sent to the server using the socket interface. The server receives this command and it is sent to the `process_command` function. This function processes the commands and returns a message which is to be displayed by the client.

The server takes the port number it must utilise as a command line argument. For all the illustrations shown in the report the port number 8080 was utilised. The command utilised to run the server is as follows.

```
$ ./emailserver 8080 &
```

The client accepts the IP address and port of the server as command line arguments. The port number must be the same as the one provided to the server. For example, the command used to connect to the server through `localhost` is as follows.

```
$ ./emailclient 127.0.0.1 8080
```

The set up was done with the above commands and the example provided in the problem statement was executed. The output is as shown in the Figure 3.

Next, the error handling capacity of the server is demonstrated. For this, the client is connected to the server using the local IP address of the server as given by the following command.

```
$ ./emailclient 192.168.0.129 8080
```

The different errors are: sending, reading or deleting from the main prompt, reading or deleting when there are no files, providing a non-existent user id to `SetUser` or `Send` and repeatedly adding an existent user. The output of this is shown in Figure 4.

NOTE: The local IP address varies according to the computer on which the server is hosted. The IP address can be found using the links given in the references for Linux[2] and for Windows/Mac[3].

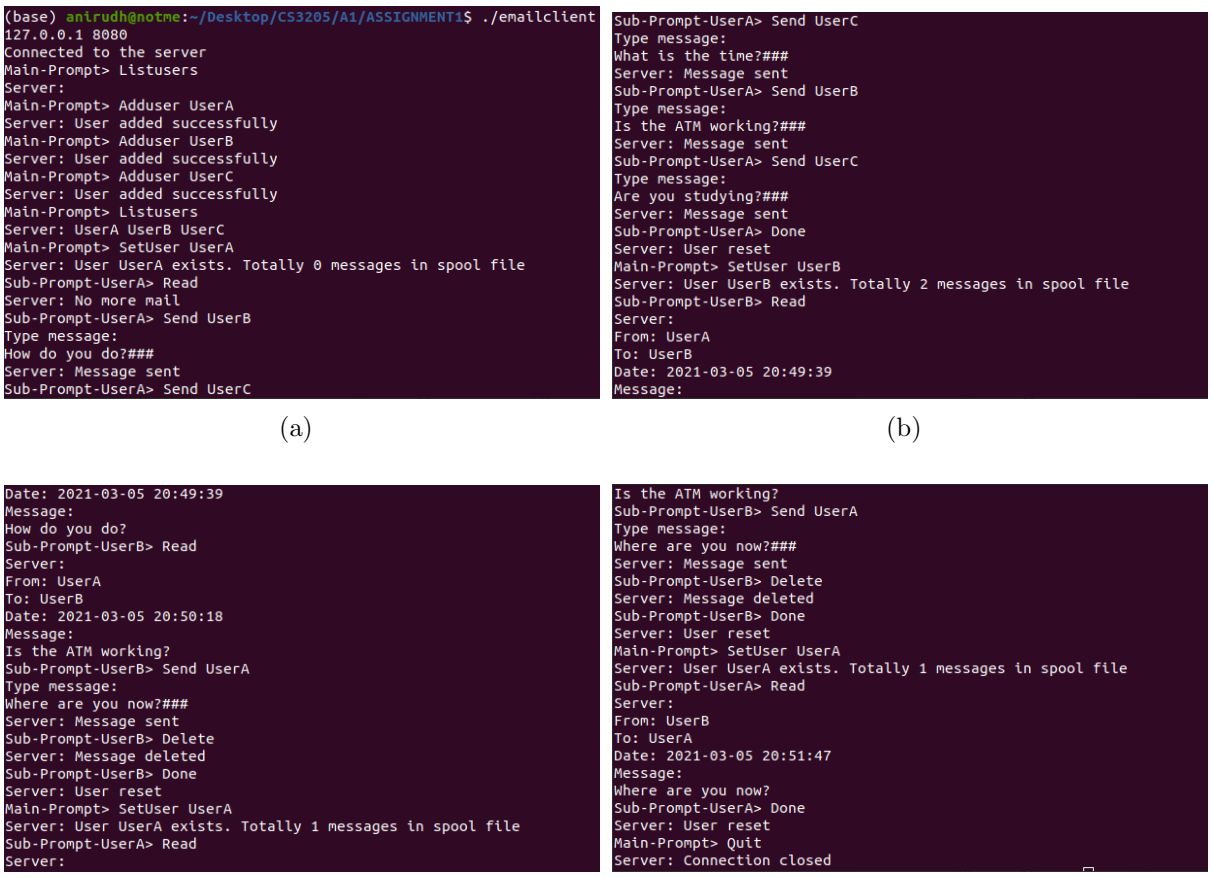


Figure 3: Trial run of the complete setup

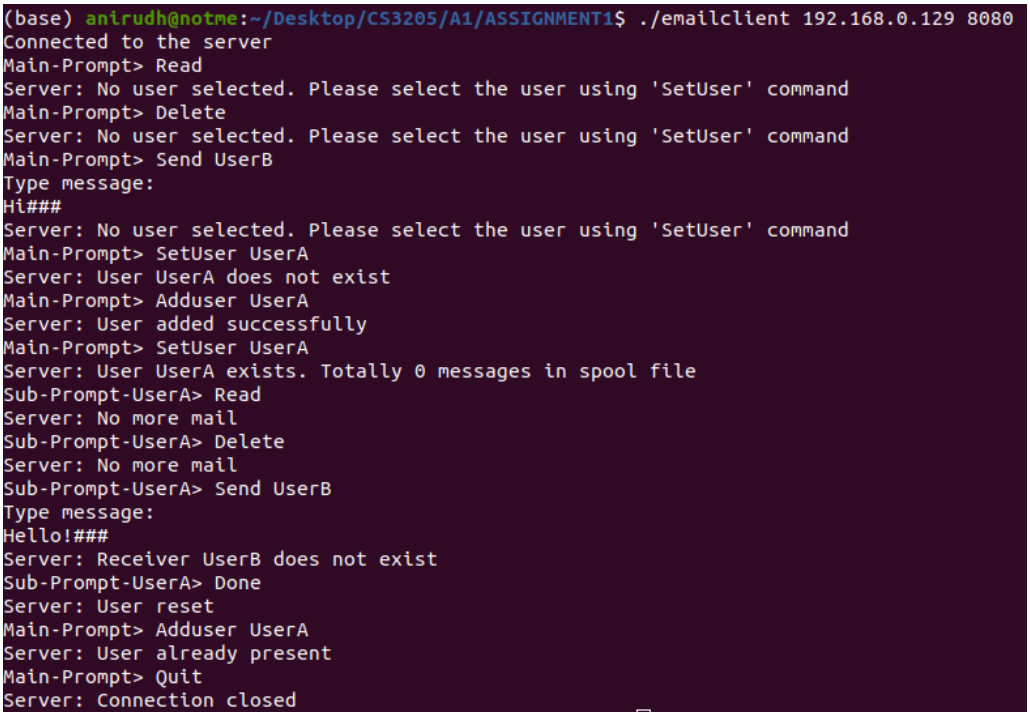


Figure 4: Error handling by the server

4 Summary

The process of implementing a mail server and a mail client communicating through the TCP socket interface provided an in-depth understanding of the application layer programming in networks. This assignment was also a great exercise in socket programming. It also helped a great deal in cementing the theoretical concepts learnt in the theory classes. All in all, this assignment was very beneficial in providing us with practice of practical implementation, while at the same time increase our depth of understanding in these essential computer networks topics.

References

[1] TCP client-server implementation in C. URL: https://www.geeksforgeeks.org/tcp-server-client-implementation-in- c/

[2] How to Find/Get your IP Address in Linux. URL: https://linuxize.com/post/how-to-find-ip-address-linux/

[3] Find Your IP Address on Windows or Mac URL: https://www.avg.com/en/signal/find-ip-address