Programming Project: Simple User Mail Access Client

Introduction

Email is one of the two main reasons people use the Internet. When someone sends you a piece of email, that letter is stored in a mailbox on a computer somewhere. When you check your mail, your mail-reading program tells you what is in your mailbox and allows you to retrieve letters from your mailbox.

How does it all work? What is a mailbox? How does your mail-reading program find out what letters are in your mailbox? How does it get letters from the mailbox for you?

In the physical world, a post office is a building that contains lots of mailboxes. In Internet land, a post office is a computer that contains lots of electronic mailboxes. In the real world, you or an agent of yours goes to the post office to examine your mailbox and collect your mail. On the Internet, a mail access program connects to your post office and examines and/or collects your mail.

For this assignment, you will write a mail access program. You can use it to list, read, and save letters in any POP3 compliant post office on the Internet.

You will be provided with a program that allows you to talk to a post-office manually, by typing commands and looking at the replies. Your job will be to use that tool to learn about the way clients talk with post offices and then to use that understanding to automate the dialog.

In the process. you will learn how Internet client-server systems work, and you will be able to reuse many of the ideas and skills you have learned earlier in this course.

The Post Office Protocol - the real world model

Consider a small town post office. Each resident has a mailbox in the post office. As letters arrive for you, the postmaster drops them into your mailbox. A mailbox is a set of letters.

When you want to check your mail, you visit the post office. Your dialog might go like this:

```
PM: Hi, welcome to the post office
YOU: I'm smith
PM: You'll need to show me some ID
YOU: my password is swordfish
PM: OK,
YOU: Please tell me the status of my mailbox
PM: OK, 3 letters containing a total of 8320 bytes
YOU: Please list the size of each letter
PM: OK
PM: message 1 has 2000 bytes
PM: message 2 has 4000 bytes
PM: message 3 has 2320 bytes
PM: end of reply
YOU: Show me the envelope for letter 2
PM: OK
PM: From: Uncle Harry
PM: Date: April 10, 1999
PM: Subject: Happy Birthday
PM: To: Jane Smith
PM: end of reply
YOU: QUIT
PM: OK
```

Sounds pretty simple. There are two players in this dialog, the client (you) and the server (the postmaster.) You make a request of the server and the server replies. Some replies consist of one line. Some replies consist of multiple lines. This simple dialog between two humans is the basis of a millions of email transactions each day.

The Post Office Protocol - You Try It

The mail system here at school speaks this protocol. Try it yourself by following this simple script. Before you do so, make sure you have some mail at your account.

```
type this                          remark
-------------------------------    -------------------------
telnet pop.MYSCHOOL.edu 110        connect to server at port 110
                                   server should say +OK
USER yourlogname                   use your logname
                                   server should say +OK
PASS yourpasword                   use your password
                                   server should say +OK
STAT                               ask for info about mailbox
                                   server says +OK, lists # and size
LIST                               request list of sizes
                                   server says +OK, lists sizes of letts
TOP 1 0                            request top of letter 1
                                   server says +OK, shows letter header
RETR 1                             retrieve letter 1
                                   server says +OK, shows letter 1
QUIT                               say goodbye
                                   server says +OK, hangs up
```

Pretty easy, eh? Just like talking to a postmaster across a counter at the post office. Notice that some answers consist of one line, and some answers consist of several lines. The answers that consist of several lines end with a line that contains a single dot.

The Post Office Protocol - Read About It

Now that you have used POP yourself, you will find RFC1939 fascinating reading. This document is the official definition of POP. It is a description of the things a client may ask of the server and the way the server is supposed to reply to those requests. Read it through, logged in to a server if you like. It is fun to try out the various commands and see how you can read your mail directly from the server without having to use a mail reader.

The mail readers out there, pine, eudora, netscape communicator, etc, conduct just this sort of dialog with your post office when you use them to read your mail.

The RFC (stands for 'request for comment') contains some examples. Read these to reinforce your own experience talking POP.

POP - Writing a Client: A General Description

By now you should be able to use telnet to connect to a POP server and read your mail. Using the LIST and TOP commands, you can determine how many letters you have, how long each is, and the sender, date, and subject of each one. It is all pretty manual, though, and not very user friendly.

The assignment is to write a curses-based mail reader that talks POP to the server and allows the user to see

a list of letters, view letters on the screen, save letters to a file, and delete letters.

The screen format is up to you, but it must include one line per letter, and each line must include the date, sender, and subject of the letter. Use the pine program to see one approach to curses-based mail readers. You are welcome to base your design on the one pine uses. If you have a better design, please use that.

The screen should display information about the mailbox. One possible presentation is:

```
==== SUMAC 1.0 ==============================================
   01 harry@isp23.net  Feb 23 2000    4390 Happy Birthday
-> 02 pres@whitehouse. Feb 28 2000   10023 Greetings
   03 bob@xyz.com      Mar  3 2000    5834 Meeting
=============================================================
 j:down, k:up, v:view, s:save, d:delete, q:quit, b:bailout
```

Put the program into cbreak, noecho mode. This will allow the user to enter commands by pressing single keys. One key moves the cursor up a line, one key moves it down a line, one will save the letter, etc.

To make your work easier, we are not requiring you to display more letters than fit on one screen. Curses stores the number of lines on the terminal in the variable LINES . You will need some lines for the header and footer, so what is left over is the space for the list of letters. It is not a lot of work to put scrolling or paging of a long list, but you are not required to do it.

The general outline of what your client has to do is

accept a hostname and username on the command line
prompt for password
login to the pop server
display a list of letters in your mailbox
accept user keys and respond
  j - move cursor down one line
  k - move cursor up one line
  v - view the selected letter
  s - save the selected letter to a file (prompt for filename)
  d - delete or undelete a letter
  q - quit (delete all marked letters)
  b - bailout (do not delete marked letters)

Getting Started

Copy all the stuff in ~COURSE/hw/sumac to your account.

You may start from scratch and design this any way you please. If you would like to experiment with a small program that does some of what you will need, try the popshell included in the stuff you copied.

To build popsh, type the command **make** . Run popsh by typing **popsh** . It will ask you for the mailhost, the username, an the password. The tty is put into noecho mode for the password.

You will then be logged in to the server. At the popsh$ prompt, you may type any POP command. Try things like STAT, LIST, TOP 1 5, etc. Each of your commands will be sent over the network to the server. The server acts on the command and sends its reply back to your client. Your client then shows the replies to you.

Each command is transmitted to the server by using the write() system call. The string is written to the

socket. The function that sends your command adds the \r\n pair to the end of each command, as specified by the RFC. The reply might consist of several lines. By using fdopen(), the program puts an fopen()-style interface on the replies. That allows the program to read lines using fgets(). Using read() for this purpose is tedious and complicated.

Read the source code to the popsh program. It shows how to handle the basics of talking to a server. You may use any or all of the code in this package.

Getting Mailbox Info - RFC822

Your program will have to display, for each letter, the sender, the date, the size, and the subject. Therefore, your program will have to obtain this information. The TOP command will display the header of any letter. Look at the headers of some letters. These headers include the name of the person who sent it, the date of the letter, and the subject (if there is one.)

The header has a well-defined format. Each line of the header has a tag, such as From:, Subject:, or Date: . The rules about how the header is structured are spelled out in rfc822.

Before you even start to worry about using curses, devise a function that will obtain for all the letters the size, from, date, and subject fields. You might want to design a struct that represents a letter and then decide on a data structure (array or linked list) to store those letters.

Using popsh or telnet, you can find the info about all the letters. The next step is to automate that process. Once you have that list of letter information, you are about one-third done.

The User Interface - Return of Curses

Now that you have the info, you need to display it for the users in a nice, clear way. Use curses to draw a nice border and then paint in one screen of a list of letters. How much of the dates do you show? What format do you expect from the header? What does the RFC say?

Once the screen is painted, you draw a ‘cursor’ pointing to letter 1. You can use any pointer you like. The ball from pong, an asterisk, an arrow, or, you can highlight the letter in reverse video. No blinking please.

If the user presses the j key, the cursor moves down. If the user presses the k key, the cursor moves up. You may make up your own keys for this, but they must be explained on the screen. Arrow keys are ok. Make sure the cursor, like the paddle in pong, does not move off the screen.

View and Save

The two important operations are viewing a letter and saving a letter.

When the user presses the ’v’ key or the Enter key, the current letter should be shown to the user. You have two choices here. First, you could write code that shows a letter the way that the more utility does - one page at a time with the option of advancing a line or page ahead. When the user is does seeing the letter, the program displays the list of letters again.

The second choice is to call the more utility as a sub-process and have it behave in the fashion of the more utility. Why write a file viewer when there already is one? All you need to do is to get the data to it. You will lose points if you do *not* use more. You may need to learn about clearok().

Saving a letter is pretty straightforward. Your program prompts for a filename, uses getstr() to read a

filename, then appends the letter to that file. Of course, you need to report any errors that may arise when opening or writing to a file.

Deleting Letters

Users may use sumac to delete letters from their mailbox. The user moves the cursor to a letter then presses the 'd' key. The line on the screen for that letter is marked with a 'D' to indicate the letter is marked for deletion. If the user presses 'd' again on that line, the 'D' is removed.

When the user presses the q key to quit from sumac, the letters marked for deletion are deleted from the mailbox. If the user decides to 'bail out' instead, the letters marked for deletion are not deleted.

Error Handling

The POP server replies with +OK and -ERR to each request. If an error occurs, your program must respond. Tell the user that an internal error has occurred and close out gracefully.

The typical problem is that the server times out and hangs up. In that case, you might want to quietly reconnect. The only problem is that the mailbox may have been modified while you were offline. It is too much trouble for this first version to try to figure out if you can leave the display alone or whether you need to tell the user that the mailbox has changed and then update the display.

Extra Credit - Resizing a Window and Paging

If the user resizes the window, there is space to show more or fewer letters. For up to five points of extra credit, add functions to your program to redraw the display when the user resizes the window. The file resize.html tells you a lot.

What if there are more letters than fit on one screen, even if you make the window really big? For up to five more points of extra credit, add functions to your program so that the user can flip through multiple screens of letters and/or scroll through a larger list than fit on the screen. You can do this yourself or you can see if curses supports scrolling windows within the screen (it does.)

What to Turn In

Turn in **(a)** your source code, **(b)** a Makefile, and **(c)** a run of a clean compile. This is curses-based program, so a script will not look good. We'll run the program.