Lab2: Writing an SMTP Client

In this lab, you will develop with your partner an implementation of an SMTP client. Your implementation will not necessarily be feature-complete, but it should correctly transmit an email from a given sender address to a given recipient address, with a specified subject and the contents of a specified file.

Caution: Beware of misusing this lab

Be careful when developing and testing this code. Since you will be interacting with real-world mail servers, you want to avoid arousing too much suspicion that you are not a legitimate SMTP client. Mail servers work very hard to detect spammers, and misbehaving SMTP software, particularly when run from unusual IP addresses, raises red flags to system administrators. Too much misuse can cause your IP address to be blocked or placed on a list of believed spammers—this would hinder your ability to send normal email from your computer, and might make it hard to complete the lab.

Piazza

Please post questions to Piazza, rather than contacting the professor/TA directly (except during PSOs). Be careful not to post too fine a detail, but especially post if you find errors in the project writeup.

Submission Instructions

When complete, both lab partners will need to submit to BlackBoard. Two things will need to be submitted: (1) your implementation of mailsend.cc, and (2) your lab writeup, which should be a simple text file named README. **While implementation of mailsend.cc can be done in pairs, the lab writeup must be done individually.** You may discuss the questions with your partner, but do not write up your lab together. To submit two files to BlackBoard, it will be necessary to combine them using a tool such as “tar” or “zip”. Please use one of these two, and submit the archive of the 2 files. **NOTE: do NOT submit your entire directory, but only the two requested files.** To use tar to create an archive containing these two files, from your lab directory you can run the command:

```
$ tar cvf lab2.tar mailsend.cc README
```

The resulting lab2.tar can be submitted.

Programming Assignment

Begin from the provided tarball, untarring it . It includes a Makefile, a copy of the udns library source, and a mostly blank mailsend.cc. First you will need to build udns.

Building udns

Go into the udns subdirectory of the lab. Run these commands:

```
$ ./configure
$ make
```

At this point you should be able to confirm that there is a new libudns.a in the directory.

Implementing your SMTP client

You should only need to edit mailsend.cc. To build the client, it will suffice to run:

```
$ make
```

Running make in your lab2 directory will attempt to build mailsend including debugging flags, and linking udns.

Description

Your SMTP client is a simple, command-line tool that takes 4 command-line arguments. The first is the destination email address. The second is the source email address. The third is a subject for the email. Note–to make a multi-word subject, you will need to enclose the subject in quotes, so it is provided as a single argument. Finally, the path to a file to send as the body of the email.

An example invocation of your program (replace with applicable email addresses):

```
$ ./mailsend destination@server.com source@server2.com "A subject line" /path/to/email/body
```

You should reference the relevant SMTP RFCs to understand the protocol used for sending email. Briefly, the steps your program will need to complete are:

1. Parse the destination server, and locate the MX record (mail exchanger) for that server. The udns library provides a suitable function, and example code can be found in udns.c
2. Find the IPv4 address of the mail exchanger. I would use another udns function for this purpose, but it is not the only option.
3. Open a socket to the mail exchanger on the SMTP port (25)
4. Wait for the server to transmit a greeting (it should send code 220).
5. Send the server a “HELO” command. You need to specify some kind of hostname after your HELO; ideally this is the canonical hostname of the current host. In practice, it is often sufficient to provide any other greeting hostname
6. Wait for the server to acknowledge positively (it should send code 250).
7. Send the server a “MAIL FROM” command, and wait for the server to acknowledge with a code 250.
8. Send the server a “RCPT TO” command, and wait for the server to acknowledge with a code 250.
9. Send the server the “DATA” command, and wait for the server to instruct you to transmit the message (code 354)
10. Transmit the email headers, a blank line, then the email body. Finish the email with a line with just a dot (.) followed by a CRLF. The server should now acknowledge with a code 250.
11. Issue the “QUIT” command, which the server should acknowledge with a code 221.
12. Close the socket, and exit.

Details, requirements, and helpful notes

- Do not use formatted I/O for reading or writing to sockets or files. For this lab you should only be using the UNIX system calls open/read/write/close/connect to interact with sockets and files.
- You are welcome to use sscanf and sprintf to do buffer formatting. Also do not forget about your string and memory functions (strcmp, strstr, strchr, memmove, memcpy, etc.)
- Mail servers vary a lot in how picky they are on the formatting of things. Your client should be able to deliver an email addressed to a @gmail.com address and a @purdue.edu address.
- If an error occurs while sending the email, you need not retransmit. You must however print a helpful error message to standard error, and return a non-zero status code to the operating system.
- Make sure your SMTP client can handle
  - blank lines in an email
  - lines containing just a period (do not confuse as the end of the email)
  - lines beginning with a period (prepend a period to the line, according to the RFC)
  - lines in a file terminated with just a bare line-feed (LF) rather than a carriage-return-line-feed (CRLF) (one standard solution is to insert a CR before the LF)
  - lines that are 1000 characters long
- SMTP lines are limited to 1000 characters; but be careful, string termination, fixing bare LFs, and lines beginning with a period might make the buffer space needed longer than 1000 characters. If the input file contains lines too long, your client should print an error message and return an error status code.
- SMTP is a 7-bit ASCII protocol. You can test characters for ascii-ness using the function “isascii()”. If the input file contains any non–7-bit ASCII characters, your client should print an error message and return an error status code.
- You do not need to implement MIME or Base64 encoding. If the input file is not a valid email body, you should just given an error (and status code) describing why you cannot send the file.
- You may assume that valid command line arguments for email addresses do not include any funny characters, spaces, etc. They are simple email addresses.
- You should transmit the email to the mail exchanger with the lowest priority value. If that server is unavailable, you need not retransmit, but instead report an error (and status code).
- If no MX record exists, your mail client should fall back onto connecting to the host directly at its IP address.
- Assume IPv4 addressing.
- I have left my helper function names in the source file. You are under no obligation to use them, they are there so you can get a sense of how my implemenation works.
- Yes, I have a working implementation, so I know it can be done. It took me, working alone, approximplately 6 hours to develop the lab solution and lab.
- Your mail client should properly display To, From, and Subject headers at least. Ideally it would also show the date sent.
- SMTP servers sometimes send “spurius” 220 codes. 220 means “awaiting input”. Your client should gracefully handle such cases
- All SMTP status codes are 3 digits long.

Lab Writeup Instructions

In addition to your code, you need to submit individually written writeup (README text file). Your writeup should first detail who your team mate is, and then give a description of the functionality that works and does not work. Finally, you should answer the following questions:

1. If you were to write a spam detection software for SMTP servers, what are some things you would look for in SMTP clients that would make them appear suspicious? Why might spammers include their own SMTP client rather than using an ISP or organization provided relay?
2. Will the “From” email header necessarily match the argument in the “MAIL FROM” command? Can you think of any reasons it might not?
3. Will the “To” email header necessarily match the argument in the “RCPT TO” command? Can you think of any reasons it might not?
4. What is to prevent you from sending an email pretending to be from “obama@whitehouse.gov”?
5. SMTP is a 7-bit ascii protocol. How then are you able to send attachments such as pictures, that are not-ascii art? What impact does that have on the size of the message?

Grading and Due Date

We will soon post a rubric giving more details on grading. The due date for this assignment is Thursday, September 27, 11:59:59pm.