COMP30017: Operating Systems and Network Services

Project 2 2013

Due date: No later than 10:00am, Monday June 3 **Weight:** 15%

1 Project overview

The aims of this project are: to provide you with experience in writing programs that interact with each other over a network (Unix socket programming); to provide an introduction to the Post Office Protocol (POP3); and to provide you with experience in writing multi-threaded applications. As such, these aims link the *Operating Systems* and *Network Services* components together into one project.

You are required to implement in C a multi-threaded version of a mail server that implements POP3. It is important to note the simplifying assumptions in this project. You do not have to implement the SMTP. Detailed specifications are listed below.

You should complete this project with a *partner* (ie. maximum group size = 2).

Submissions that do not compile and run on the Department's student machines may receive zero marks.

2 Project specification

The Post Office Protocol (POP) is an application-layer Internet standard protocol used by local email clients to retrieve email from a remote server over a TCP/IP connection. Version 3 (POP3) being the current standard.

After connecting to the server, the client sends commands, and the server returns the status (success or failure) of the command and any information requested. Although the email clients have an option to leave mail on the mail server, it is possible to delete downloaded messages from the mail server.

Large sections of this project specification have been taken directly from the RFCs that specify the POP3 protocol. The relevant document is RFC1939 (`http://tools.ietf.org/html/rfc1939`). You should read this document carefully.

Before listing the project implementation requirements and important assumptions, key definitions are presented along with a description of the POP3.

2.1 Definitions

CRLF pair is a pair of ASCII characters – a *carriage return* followed by the *line feed* character. These two characters in combination signal the end of a line.

Octet Octet and byte are synonymous (8 bits).

Maildrop is a user's mail box. In this project, sample mail messages will be provided along with the user details.

Client The client is the program initiating a connection to the server to access a maildrop. In this project, you will simply `telnet` to your server on a nominated port.

Server The server is the program waiting for incoming connections that provides access to a maildrop.

2.2 POP3

The following details appear at: `http://tools.ietf.org/html/rfc1939`

2.2.1 Overview

Initially, the server host starts the POP3 service by listening on a port [Unix socket: SOCK_STREAM for TCP]. When a client host wishes to make use of the service, it establishes a TCP connection with the server host. When the connection is established, the POP3 server sends a greeting. The client and POP3 server then exchange commands and responses (respectively) until the connection is closed or aborted.

Commands in the POP3 consist of a case-insensitive keyword, possibly followed by one or more arguments. All commands are terminated by a `CRLF` pair. Keywords and arguments consist of printable ASCII characters. Keywords and arguments are each separated by a single `SPACE` character. Keywords are three or four characters long. Each argument may be up to 40 characters long.

Responses in the POP3 consist of a status indicator and a keyword possibly followed by additional information. All responses are terminated by a `CRLF` pair. Responses may be up to 512 characters long, including the terminating CRLF. There are currently two status indicators: positive `+OK` and negative `-ERR`. Servers **must** send the `+OK` and `-ERR` in upper case.

Responses to certain commands are multi-line. In these cases, which are clearly indicated below, after sending the first line of the response and a `CRLF`, any additional lines are sent, each terminated by a `CRLF` pair. When all lines of the response have been sent, a final line is sent, consisting of a termination octet (decimal code 046, '.') and a `CRLF` pair. If any line of the multi-line response begins with the termination octet, the line is "byte-stuffed" by pre-pending the termination octet to that line of the response. Hence a multi-line response is terminated with the five octets `CRLF.CRLF`. When examining a multi-line response, the client checks to see if the line begins with the termination octet. If so and if octets other than `CRLF` follow, the first octet of the line (the termination octet) is stripped away. If so and if `CRLF` immediately follows the termination character, then the response from the POP server is ended and the line containing `.CRLF` is not considered part of the multi-line response.

A POP3 session progresses through a number of states during its lifetime. Once the TCP connection has been opened and the POP3 server has sent the greeting, the session enters the AUTHORIZATION state. In this state, the client must identify itself to the POP3 server. Once the client has successfully done this, the server acquires resources associated with the client's maildrop, and the session enters the TRANSACTION state. In this state, the client requests actions on the part of the POP3 server. When the client has issued the QUIT command, the session enters the UPDATE state. In this state, the POP3 server releases any resources acquired during the TRANSACTION state and says goodbye. The TCP connection is then closed.

2.2.2 AUTHORIZATION state

Once the TCP connection has been opened by a POP3 client, the POP3 server issues a one line greeting. This can be any positive response.

The POP3 session is now in the AUTHORIZATION state.

The client must now identify and authenticate itself to the POP3 server. The client does this using two commands, which are only valid in the authorization state:

USER name The `USER` command has one argument – the *username*. The server shall respond with a positive status indicator `+OK`.

PASS password Once the `USER` command has been given and acknowledged, the `PASS` command shall follow. It has one argument – the *password*. At this point the server needs to check if the username and password are valid. If they are valid the server responds with a positive status, otherwise with a negative status. (See section 2.3 for more details in regard to the requirements for this project).

To terminate a session, the quit command is used:

QUIT The quit command can be given at any point during the AUTHORIZATION state. The server is to respond with a positive status and then terminate the TCP connection.

If the user successfully authenticated, the session enters the TRANSACTION State. If authentication failed, the client is to send another USER and PASS commands to retry authentication.

2.2.3 TRANSACTION state

Once the client has successfully identified itself to the POP3 server, the POP3 session is now in the TRANSACTION state. The client may now issue any of the following POP3 commands repeatedly. After each command, the POP3 server issues a response. Eventually, the client issues the `QUIT` command and the POP3 session enters the UPDATE state.

The following POP3 commands are valid in the TRANSACTION state:

STAT The POP3 server issues a positive response with a line containing information for the maildrop. This line is called a "drop listing" for that maildrop.

In order to simplify parsing, all POP3 servers are required to use a certain format for drop listings. The positive response consists of `+OK` followed by a single space, the number of messages in the maildrop, a single space, and the size of the maildrop in octets.

Note that messages marked as deleted are not counted in either total.

Note that additional '.' characters that may need to be inserted must not be counted when calculating the size of messages.

LIST [msg]

Arguments A message-number (optional), which, if present, may **not** refer to a message marked as deleted.

Discussion If an argument was given, the POP3 server issues a positive response with a line containing information for that message. This line is called a "scan listing" for that message.

If no argument was given, the POP3 server issues a positive response, then the response given is multi-line. After the initial `+OK`, for each message in the maildrop the POP3 server sends a response – a line containing information for that message. This line is also called a "scan listing" for that message. If there are no messages in the maildrop, then the POP3 server responds with no scan listings – it issues a positive response followed by a line containing a termination octet and a `CRLF` pair.

A scan listing consists of the message-number of the message, followed by a single space and the exact size of the message in octets. Each individual scan listing ends with a `CRLF` pair.

The message number is simply a unique identifier (counter) based on the displayed list of messages.

Note that additional '.' characters that may need to be inserted must not be counted when calculating the size of messages.

Note that messages marked as deleted are not listed.

RETR msg

Arguments A message-number (required), which, if present, may **not** refer to a message marked as deleted.

Discussion The response given is multi-line, the first line being a positive status response ending in a `CRLF` pair. The POP3 server then sends the message corresponding to the given message-number, being careful to byte-stuff the termination character as described previously. The multiline response is ended with a termination character ('.') and a `CRLF` pair.

DELE msg

Arguments A message-number (required), which, if present, may **not** refer to a message marked as deleted.

Discussion The POP3 server marks the message as deleted. Any future reference to the message-number associated with the message in a POP3 command generates an error. The POP3 server does not actually delete the message until the POP3 session enters the UPDATE state.

The POP3 server then responds with a positive status.

NOOP The POP3 server does nothing, it merely replies with a positive response.

RSET If any messages have been marked as deleted by the POP3 server, they are unmarked. The POP3 server then replies with a positive response.

QUIT Enter the UPDATE State.

2.2.4 UPDATE State

When the client issues the `QUIT` command from the TRANSACTION state, the POP3 session enters the UPDATE state.

If the client issues the `QUIT` command from the AUTHORIZATION state, the POP3 session terminates but does **not** enter the UPDATE state.

If a session terminates for some reason other than a client-issued `QUIT` command, the POP3 session does **not** enter the UPDATE state and **must** not remove any messages from the maildrop.

Upon receving the `QUIT` command the POP3 server removes all messages marked as deleted from the maildrop. If this was successfull the server will send a positive response. If a error was encountered when attempting to deleting a messages, a negative response is to be sent.

Under no circumstances may the server remove any messages not marked as "deleted". Whether the removal was successful or not, the server then closes the TCP connection.

2.3 MailDir

The fundamental approach used for storing email on your POP3 server is based on the **MailDir** convention. That is, each individual email message is stored in its own file. All the messages in a particular maildrop are stored in an individual user's folder.

For this project the traditional folder structure is simplified. There is a root folder, which contains a folder for each user. Inside each user's folder are the email files. Each user folder also contains a special file named `pass`. This file contains the password for the user – a string of characters without a new line character. Passwords are at most 20 characters long.

A sample **MailDir** folder containing multiple user folders (with corresponding password files `pass` and sample email messages) is available on the LMS.

2.4 Implementation Requirements

2.4.1 Requirement A

You **must** use the skeleton code supplied and provide appropriate implementations for the functions described in the header files. These files are available on the LMS.

Do not modify these header files or create additional header files. The automated testing used during the marking process will compile your code with special versions of some of the modules.

Your executable program **must** be called `pop3`.

The command line arguments used when testing your POP3 server are in order:

```
./pop3 [port] [MailDir path] [log file]
```

where:

`[port]` is the port number on which the server should be listening for new connections.
`[MailDir path]` is the path to the root **MailDir** folder. This path must end with the '/' character.
`[log file]` is the path and file name of the log file.

2.4.2 Requirement B

Your POP3 server should provide implementation details for each of the commands listed above in the AUTHORIZATION, TRANSACTION and UPDATE states.

A sample text file recording a `telnet` session interacting with a POP3 server is available on the LMS.

2.4.3 Requirement C

Your POP3 server must be multi-threaded, that is it must be able to handle up to 100 concurrent sessions from seperate users.

2.4.4 Requirement D

The POP3 server is required to log key events to a file – the `[log file]` listed as a command line argument. This is to be done by the *log_write* function in the provided skeleton code, the function comment also describes the formatting to be used. This function takes four arguments from which the log entry is constructed. The first argument is the *module*, this should indicate which part of the program is creating the log entry. This can be *Main*, *POP* or *MailDir*.

Table 1: Log file entries

Module	**P1**	**P2**	**P3**
Main	POP3 Server Started	“”	“”
Main	New Connection	“”	“”
MailDir	[user name]	GetList	“”
MailDir	[user name]	Delete	[file name]
MailDir	[user name]	GetFile	[file name]
POP	[user name]	Authenticated	“”
POP	[user name]	Auth failed	“”
POP	No Auth	USER	[argument]
POP	No Auth	PASS	[argument]
POP	No Auth	QUIT	“”
POP	[user name]	TRANSACTION STATE	“”
POP	[user name]	[command]	[argument]
POP	[user name]	UPDATE STATE	“”
POP	[user name]	Closing Connection	“”

Table 1 lists the expected events that should be logged with the arguments to *log_write* that should be used.

When logging POP commands, they should be given in all upper case. The entries for the TRANSACTION and UPDATE states should be created as the POP3 session enters the respective state. In the AUTHORIZATION state “No Auth” should be used for argument *p1* before the client has been authenticated.

Additional important events may be logged as well, such as the creation of a new thread. However the log file should not be filled with ‘debugging’ information.

A sample log file is available on the LMS.

2.5 Important assumptions

The following assumptions can be made:

- AUTHORIZATION and input issues
  - The input from the client will be valid, hence extensive validation is not required. However the username and password must be checked / verified.
- **Maildir** issues
  - While a user session is active, the files in the **Maildir** will not be altered by another program. However, when a particular user is not connected, changes to the user’s directory may occur.
  - The file name in the **MailDir** is a unique identifier for the email.
  - The message numbers given as arguments will be valid.
  - The email files on the disk have the correct `CRLF` (new line format with the CRLF pair).
- Multi-theading issues
  - A particular user will at most have one connection open to the server.

3 Using SVN and Submission details

Just as you did in project 1, you must make use of a SVN repository when developing your solution.

You will use either your own individial project 2 repository, or your group repository. We are in the process of seeting up group repositories. Further information will be posted on the LMS.

You should add the skeleton code (and Makefile) to your subversion repository for the project. As you modify source code, it should be added/checked-in to your repository.

Only **one submission** is required from each group. Please include both your **name** and **login_id** and your partners **name** and **login_id** in a comment at the top of each file.

Note: you must not modify the header files or create additional header files. The automated testing used during the marking process will compile your code with special versions of some of the modules.

4 Assessment details

This project is worth 15% of your final mark for the subject.

4.1 Available marks

Limited automatic testing of your solution will be conducted. This includes testing the basic functionality of your `pop3` server and verifying that your implementation can correctly process multiple requests. Additional "manual" testing will be conducted to verify that your program is working as expected.

Your submission will be tested and marked with the following criteria:

2 marks – **Maildir** processing correct

2 marks – Socket implementation correct

2 marks – Log file correctly reflects the command sequence / transactions

6 marks – POP3 commands function as expected

- AUTHORIZATION
- TRANSACTION
- UPDATE

3 marks – for coding style: program structure; use of modules; use of appropriate data structure, naming conventions etc., elegancy of code.

- Efficiency and clarity of code are both important.

4.2 Bonus marks

A maximum of **2 bonus** marks are available for server implementations that include additional features – beyond the minimal requirements listed above. Possible features that you might consider including are:

- Implement additional commands from the RFCs. Some of these are very simple to do, needing only a couple of lines of code.
- Include additional/simple error handling, responding to invalid commands, invalid arguments, recovering from input that is too long.
- Handling of SIGTERM to allow **Maildir** operations (*eg* deleting messages) to complete before closing all the connections and the socket.
- You could implement APOP authorisation from RFC 1460 – it's basically a way to use MD5 with a unique ID to avoid sending the password in plain text.

Important notes:

- If you attempt to collect the bonus marks, you must clearly identify what you have done in a `README` file. This `README` file will be submitted along with all source files.
- Bonus marks allocated will be commensurate with the complexity of the extensions applied.
- If the code you submit related to the bonus material causes problems, and possible generates incorrect output for the other implementation requirements, you run the risk of being penalized.

4.3 Final comments

Extension policy: If you believe you have a valid reason to require an extension you must contact the head tutor or the lecturer at the earliest opportunity, which in most instances should be well before the submission deadline. Requests for extensions are not automatic and are considered on a case by case basis. You may be required to supply supporting evidence such as a medical certificate.

Plagarism policy: You are reminded that all submitted project work in this subject is to be your own individual work. Automated similarity checking software will be used to compare submissions. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student concerned.

Using SVN is an important step in the verification of authorship.

Questions about the project specification should be directed to the LMS discussion forum.