BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

ISA course project documentation

POP3 Server

18.11.2017

Author: Jan Ondruch

Faculty of Information Technology

Brno University of Technology

Contents

1 Introduction

The content of this document is concerned with implementation of a POP3 server for a Network Applications and Network Administration class. Firstly, POP3 server technology is introduced along with its functionality and requirements for implementation. Then programming solution proposal is discussed as well as the program implementation. At last, program usage, testing and metrics are presented.

2 POP3 Server

POP3 server (only server from now on) is a program, that enables clients to connect to it and download data via POP3 protocol. POP3 server utilizes TCP socket and waits for client connections. When a client connects, the server sends a greeting. At this stage, the client enters the Authorization state and must identify itself to the server. Once it has granted access, session enters the Transaction state. The commands between the client and the server are exchanged and the Maildir directory containing e-mails can be manipulated. After the *QUIT* command is issued, the server enters the Update state, in which it safely releases resources acquired during the communication with the client. The TCP connection with the client is then closed and the server waits for more incoming connections. The whole communication process - POP3 protocol behavior, is demonstrated in the Figure 1. More detailed description of POP3 protocol can be found in the RFC 1939 standard [1].

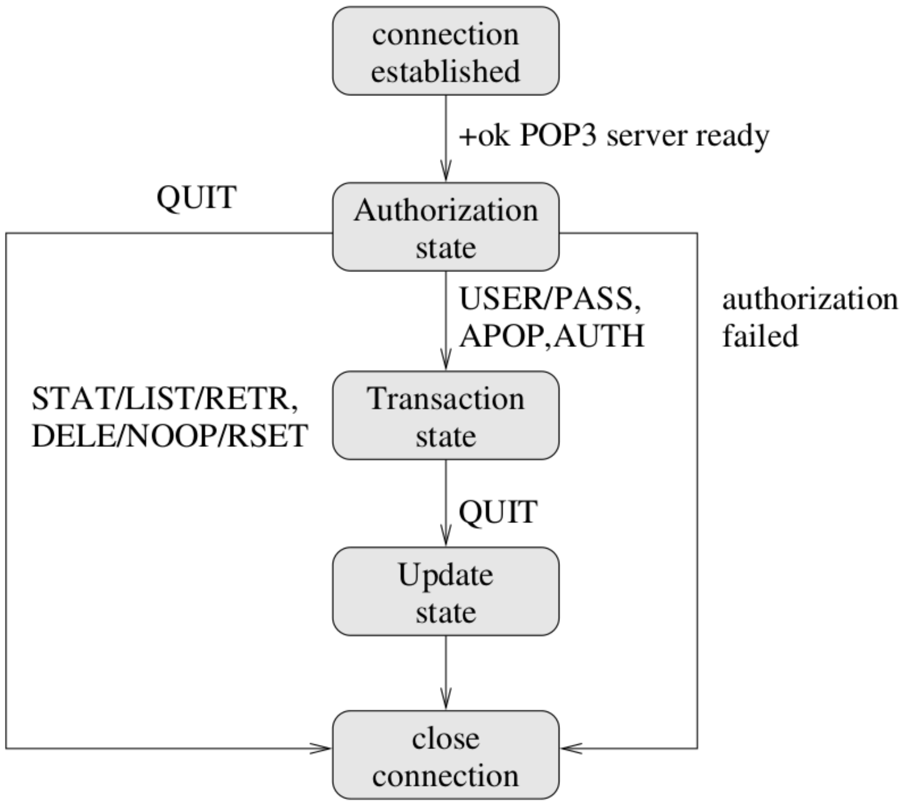

Figure 1: POP3 protocol behavior [2]

2.1 Authorization state

The first stage the client enters in is called the Authorization state. The client always finds itself at this stage, unless the server fails to establish a connection. Then, *-ERR* response is sent to the client and no further communication takes place, else the client is greeted with a welcome message *+OK POP3 server ready*. Depending on the method of authorization, *APOP*, or *USER* and *PASS* combination of commands enable the client to log in and he is given access to the maildrop. The access has to be unique and if the Maildir is being operated by another client, the authorization fails and the incoming client has to wait, until the other one releases an exclusively given lock. The server must support at least one of the aforementioned authorization methods. *APOP* enables to log in without the need to send a password in a clear form. Instead, it utilizes md5 hash function combined with other mechanisms to ensure secure access.

2.2 Transaction state

When the client is granted access to the maildrop, it can now issue various commands *STAT*, *LIST*, *RETR*, *DELE*, *NOOP*, *RSET*, *UIDL*, *QUIT* and *TOP* to communicate with the server. They enable the client, for instance, to list, read or delete e-mails. After each command, the server must send a response (again positive *+OK*, or negative *-ERR*). In case the *QUIT* command is issued, the server enters the Update state. If the client or the server failed and no *QUIT* command occurred, the server has to release the exclusive lock and does not enter the Update state.

2.3 Update state

The last part of the communication is the Update state. The client enters the stage only if the Transaction state ended successfully. The server deletes any messages marked to be deleted, releases the exclusive lock and terminates the connection.

3 Implementation

The program implementation is logically divided into server and client part, the latter being significantly larger code size-wise. Maildir directory and its functionality in *popser* is mentioned, as well. At last, the used programming language and paradigm are discussed.

3.1 Server establishment

When the program is launched, arguments validity is checked in an `arguments` object. Afterwards the object instance is passed to the server object's method `ConnectSocket()` and `AcceptClients()`, which take care of the connection. TCP *BSD sockets* are utilized and as the program specification requires, the server socket is implemented in a non-blocking fashion. Concurrency is ensured by threads from *pthread.h* library. In case the arguments are invalid or the server fails to establish a connection, the program outputs an error message and terminates. The program receives new client connections with the use of accept function and creates a new thread for each along with a new socket `newsockfd`. This process runs indefinitely in a loop, until a `SIGINT` signal is received and the program is gracefully shut down.

3.2 Client thread

Each of the client threads operates independently. After the client connects, the time stamp is generated and welcome message is sent. Communication is handled in a loop containing `select()` function with a timeout set to 10 minutes. In case a client is inactive for longer than 10 minutes, the connection is terminated. Since `select()` is unable to handle timeout once the `recv()` function is initiated, `setsockopt()` setting `SO_RCVTIMEO` flag has to be called beforehand. Only then the program is fully able to detect possible client inactivity. As `recv()` receives new data, `communicate()` function containing a final-state automaton is called to return a valid response to the client‘s message. Exclusive access to Maildir is carried out using mutex. The mutex is always released (if acquired first) to avoid deadlock. When the client terminates the connection, the corresponding client socket is closed.

3.3 Maildir

Maildir information is stored in a helper file named *popser.log*. *Popser.dupl.log* is used for additional manipulations. *Popser.log* is structured as a table, enabling the program to search within entries with a row-column key. Each entry consists of an absolute file (message) path, unique identifier, size and a flag marking if the message was labelled as deleted. In case *-r* flag is set, both helper files are deleted, but only if resetting was successful – if an error occurs and mails were not moved back to the new folder. Removing the helper files would mean loosing the track of the additional information, which is essential for proper server restoration. The naming of the messages along with more information on how to operate a Maildir was inspired by D. J. Bernstein‘s Maildir guide [3]. The program expects the messages to be in a valid Internet Message Format specified by the RFC 5322 standard [4]. However, it must ensure, that proper message size is counted and if the *RETR* command is issued, that the message is sent in the specified format. This is due to inconsistent end of line representation across various file systems, that may handle end of line occurrences differently.

3.4 Language and programming paradigm

The program is implemented in the C++ programming language. It utilizes standard library header files for C and C++, header files for socket implementation such as *sys/socket.h* or library for threads *pthread.h*. The object oriented approach was partially used to parse the input arguments as well as to put the server into operation. The rest of the program follows the imperative programming pattern. This is due to function creating new threads, which I was unable to incorporate into a class. A possible solution would be to use a wrap struct object, or to use a static class method.

4 Installation and usage

In order to start *popser*, it must be installed first and run with a specific set of options.

4.1 Installation

To make the program work, the user executing *popser* must have read and write permissions granted across the directory containing the runnable binary file.

Execution of the tool is possible right after the provided Makefile is run. The executable file name is *popser*. Any further manipulation within the executable binary file's directory is highly discouraged as it may disrupt the functionality of the program.

`make`	Runs the script and outputs an executable binary file *popser.*
`make clean`	Cleans files created by the make execution.

4.2 Usage

Options and parameters for program usage:

```
./popser [-h] [-a PATH] [-c] [-p PORT] [-d PATH] [-r]
```

-h|--help	Prints help message and exits.
-a	Path to the authentication file.
-c	Accepts non-encrypted login password method (optional parameter).
-p	Runs application on this port.
-d	Path to the Maildir folder.
-r	Resets the Maildir (optional parameter).

Clients can then connect via Telnet to the given port.

5 Limitations

The program fully implements the RFC 1939 specification apart from the section 8, does not support encryption (TSL/SSL) and omits the *TOP* command. In case the *TOP* command is given, the server will issue a negative answer. The username in the authorization file does not permit any spaces and follows the syntax of classic e-mail address username.

6 Testing

Different parts of the program were tested by various means. Validity of the authorization file, as well as of given options, arguments and the server‘s responses were checked by an automated script [5]. Behavior of POP3 protocol along with the server responses was compared with *Dovecot* POP3 Server [6]. The tool was tested on *Linux Ubuntu 14.04 LTS*, *Linux Ubuntu 16.04 LTS* and *CentOS Linux 7* operating systems. Both the script and *Dovecot* reference comparison were successful without any differences in outputs.

A Code Metrics

Number of files:	12
Lines of code:	2488

References

[1] J. Myers, M. Rose and M. Carnegie, "Post Office Protocol - Version 3," 17 November 2017. [Online]. Available: https://tools.ietf.org/html/rfc1939.

[2] P. Matoušek, Síťové aplikace a jejich architektura, Brno: VUTIUM, 2014.

[3] D. J. Bernstein, "cr.yp.to.," 17 November 2017. [Online]. Available: https://cr.yp.to/proto/maildir.html.

[4] P. Resnick, Ed., "Internet Message Format," 17 November 2017. [Online]. Available: https://tools.ietf.org/html/rfc5322.

[5] P. Rusiňák, "Testovací skript na POP3 server v ISA," 18 November 2017. [Online]. Available: https://github.com/atepr/isa-pop3-server-tests.

[6] Dovecot, "https://wiki.dovecot.org/POP3Server," 17 November 2017. [Online]. Available: https://wiki.dovecot.org/POP3Server.