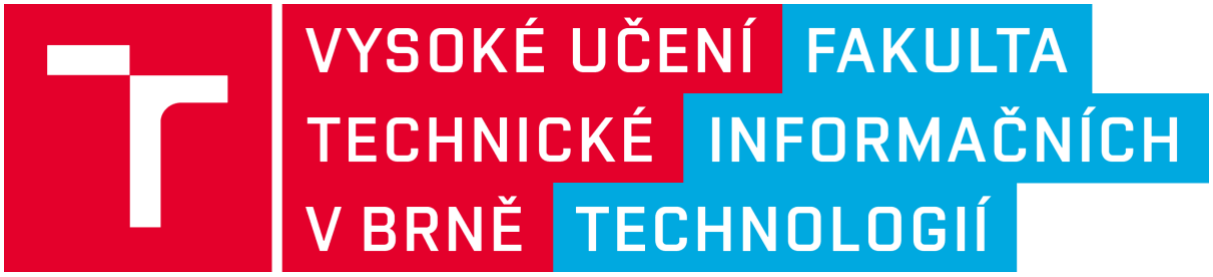

Network Applications and Network Administration (ISA)

2024/2025

Documentation of project version IMAP Client with SSL/TLS

Marek Joukl (xjoukl00)

Date: 15.11. 2024

Table of Contents

1. Introduction *3*

2. Problem analysis *3*

3. Application structure *3*

4. Implementation Description *4*

4.1. Progam structure **4**

4.2. Key Implementation Details **4**

4.2.1. Argument Parsing 4

4.2.2. IMAP Communication 4

4.2.3. State Management 4

4.2.4. Directory format 5

5. Program Usage *5*

5.1. Command-Line Arguments **5**

5.2. Execution Examples **7**

6. Testing *7*

7. Literature and resources *7*

1. Introduction

The aim of this project is to create a command-line IMAP4rev1 client capable of connecting to email servers, retrieving messages, and storing them locally. The implementation supports both standard IMAP (default port 143) and secure IMAPS connections. The program also provides flexibility through multiple command-line arguments, enabling the user to specify custom ports, download headers only, or work with specific mailboxes. It was written using C++ with OpenSSL support.

The project addresses common issues such as maintaining local synchronization with a remote mailbox, handling different mailbox states, and managing secure connections with OpenSSL. The documentation outlines the design, implementation, and testing.

2. Problem analysis

The main challenge in developing an IMAP client is ensuring seamless communication with the server according to the IMAP4rev1 protocol specification. This involves handling authentication, command parsing, message fetching, and managing mailbox states using UIDVALIDITY. Additionally, implementing secure communication with SSL/TLS for secure connections introduces complexities in certificate handling and error management.

I found these key issues to be the hardest to tackle with:

- Compliance with RFC IMAP4rev1 standards for message retrieval and mailbox management.
- Ensuring the client accurately reflects the state of the remote mailbox (new messages, deletions, etc.).
- Handling SSL/TLS negotiation and certificate validation.

3. Application structure

The application is designed using a modular approach with separate components for argument parsing, IMAP communication, SSL/TLS management, and file handling. The main control flow is managed in main.cpp, while supporting functionalities are encapsulated in separate utility files.

- Different functionalities are handled in separate files such as command parsing (arg_parser.cpp), IMAP communication (imap.cpp), secured IMAP communication (imaps.cpp), and utility function (utils.cpp)

- Local mailbox state is stored in state.txt files within each mailbox directory, tracking UIDVALIDITY and message UIDs.

- Each component includes comprehensive error handling to ensure robust operation and meaningful logging.

4. Implementation Description

4.1. Progam structure

The project consists of multiple source files, each dedicated to a specific module:

- `main.cpp`: Manages the program's control flow and high-level logic.
- `arg_parser.h/cpp`: Handles command-line argument parsing.
- `imap.h/cpp`: Contains functions for interacting with the IMAP server.
- `imaps.h/cpp`: Handles secured version on IMAP.
- `utils.h/cpp`: Utility helper functions (directory manipulation, text formatting, etc.)

4.2. Key Implementation Details

4.2.1. Argument Parsing

The ArgumentParser class provides an easy-to-use functions for retrieving command-line options and flags – functions `getOption()` and `hasFlag()`, which are called in the beginning of main. This allows users to specify custom ports, output directories, and other settings.

4.2.2. IMAP Communication

The IMAP communication is handled using a socket for standard connections and a BIO library for secure connections. Each command (e.g., SELECT, FETCH, SEARCH) follows the IMAP4rev1 (RFC 3501) format and uses appropriate response handling to ensure compliance. Both the secured and unsecured versions of the program have their own separate functions. They are structured so each function encapsulates the logic and is responsible for different part of the communication (login, authentication, fetching and storing messages, …). Each function's parameters as well as what it returns is described briefly using comments in header files.

4.2.3. State Management

The program tracks the local state of each mailbox in state.txt files, which are created in the corresponding mailbox directories. Each `state.txt` file includes:

- **HeadersOnly**: Displays if only headers are downloaded or whole messages.
- **UIDVALIDITY**: Ensures the state consistency of the mailbox.
- **UIDs**: Lists the UIDs of downloaded messages.

Before downloading new messages from the server, after receiving response for sending `SEARCH` command, the program first compares checks HeadersOnly parameter. That is used for situation when only headers are present and user wants to download whole messages or

otherwise. If so, different content is stored locally than the user desires and, therefore, the messages are redownloaded adequately (only headers or whole messages). Next the UIDVALIDITY and UIDs stored in the `state.txt` are compared to the response. If both are equal, no new downloads need to be made. If UIDVALIDIY changes, it means that mailbox has changed and, therefore, the program downloads all new messages leading to synchronized mailbox. If only some UIDs are changed, the program downloads only the missing ones. This makes the program more efficient and saves resources.

4.2.4. Directory format

The scheme below displays how are directories and files are structured in `out_dir` specified by the user. Each imap server has its own directory containing directories for specific mailboxes. Each mailbox contains download messages as well as `state.txt`. Names of files of messages are using UIDs for easier manipulation.

```
out_dir/
└── server1/
    └── INBOX/
        ├── message_uid_1.eml
        ├── message_uid_2.eml
        └── .state
```

The format of downloaded emails is formatted using `formatToRFC5322()` function that removes trailing IMAP commands (OK UID FETCH completed etc.) and rearranges headers according to RFC 5322.

The program also counts on user to create appropriate output directories according to his email addresses. So, for instance, if the user has two email addresses on the same imap server, he needs to create separate directories and store his emails accordingly.

5. Program Usage

Start by running command **`make`** for compiling the project.

5.1. Command-Line Arguments

./imapcl server [-p port] [-T [-c certfile] [-C certaddr]] [-n] [-h] -a auth_file [-b MAILBOX] -o out_dir

Run the program with options:

- -p: Specify the server port (default 143 for unsecured connection).
- -T: Use SSL/TLS for secure connections.
- -c: Specify the path to the certificate file.
- -C: Specify the certificate directory.
- -n: Download new messages only.
- -h: Fetch headers only.
- -a: Specify the authentication file.

- -b: Specify the mailbox to interact with.
- -o: Specify the output directory for storing emails (default INBOX).
- server: the address of the IMAP server

5.2. Execution Examples

Download all messages from server:

- ./imapcl -a auth_file -o maildir imap.seznam.cz

Download only headers of new messages using secured connection on port 993:

- ./imapcl -a auth_file -o maildir imap.seznam.cz -T -p 993 -n -h

6. Testing

Testing was done using imap.pobox.sk server for unsecured connection (port 143) and imap.seznam.cz for secured connection (port 993).

The program was tested with multiple command-line inputs to ensure that the program recognized all options correctly. Tests included valid inputs, such as -p for custom port selection and -T for SSL/TLS, as well as incorrect inputs to confirm that the program handles errors, like missing required arguments. Examples:

Test case	Input	Output
Valid input with SSL	./imapcl imap.seznam.cz -T -a auth_file -o out_dir -p 993	Downloaded 1 message from mailbox INBOX
Missing required argument	./imapcl -T -a auth_file -o out_dir -p 993	Usage: ./imapcl server [-p port] [-T] -a auth_file -o out_dir
Invalid port number	./imapcl -T -a auth_file -o out_dir -p abc	Error: The specified port is not a valid number.
New messages only with headers	./imapcl imap.seznam.cz -n -h -a auth_file -o out_dir -p 993 -T	Downloaded 1 new message from mailbox INBOX
Invalid flag	./imapcl imap.seznam.cz -a auth_file -o out_dir -p 993 -T	Error:Unexpected arguments provided.

7. Literature and resources

I. OpenSSL Documentation: https://www.openssl.org/docs/
II. RFC 3501: Internet Message Access Protocol - Version 4rev1 (https://www.rfc-editor.org/rfc/rfc3501)
III. RFC 5322: Internet Message Format (https://www.rfc-editor.org/rfc/rfc5322)
IV. OpenSSL BIO (https://developer.ibm.com/tutorials/l-openssl/)
V. IMAP Client best practises: (https://github.com/dovecot/imaptest/wiki/ClientImplementation)