VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

IMAP Klient

Dokumentace Projektu

Artem Dvorychanskyi (**xdvory00**)

1 Introduction

This document provides comprehensive documentation for the IMAP client application. The application enables email retrieval using the IMAP4rev1 protocol, including downloading emails and storing them in a specified directory. This documentation covers the program's functionality, usage, implementation details, and examples.

2 Program Overview

The IMAP client is a command-line tool written in C++ using socket programming and the OpenSSL library for secure communication. It supports both encrypted (TLS/SSL) and unencrypted connections and adheres to the RFC 3501 standard.

Key features include:

- Connection to IMAP servers using secure (IMAPS) or plain protocols.
- Authentication using a credentials file.
- Fetching emails from specified mailboxes.
- Saving emails in RFC 5322 format.
- Listing all mailboxes.
- Support for optional arguments like certificate paths, mailbox selection, and new emails-only mode.

3 Command-Line Arguments

3.1 Usage Syntax

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

3.2 Arguments Description

- **server**: Hostname or IP address of the IMAP server.
- **-p port**: Optional. Specify the port number. Defaults to 143 or 993 (if `-T` is used).

- **-T**: Enables TLS encryption for secure connections.
- **-c certfile**: Optional. Path to the certificate file for SSL verification.
- **-C certaddr**: Optional. Directory containing certificates for SSL verification.
- **-n**: Only download new (unread) emails.
- **-h**: Fetch only email headers.
- **-a auth_file**: Mandatory. Path to the authentication file.
- **-b MAILBOX**: Optional. Specify a mailbox. Defaults to `INBOX`.
- **-o out_dir**: Mandatory. Output directory for saved emails.

4 Program Workflow

The IMAP client program operates in several stages, each responsible for a specific part of the email retrieval process. Below is a detailed step-by-step explanation of how the program works.

4.1 Command-Line Argument Parsing

- The program starts by parsing command-line arguments using the `createConfig` function.
- The arguments are validated for correctness, ensuring all required parameters are provided. If any mandatory arguments are missing or invalid, the program terminates with an error message.
- Configuration options are stored in a `config` structure, which includes details like the server address, port number, mailbox, output directory, and authentication file path.
- Optional parameters such as enabling TLS (`-T`), working with unread messages only (`-n`), or downloading only headers (`-h`) are also parsed and stored.

4.2 Establishing a Connection

- The program uses the `connect_to_server` or `connect_to_server_s` method of the `IMAP` class to establish a connection to the specified server.
- For secure connections (`-T`), the OpenSSL library is used to create an SSL context, load certificates, and validate the server's certificate.
- In case of any connection error, an appropriate error message is displayed, and the program terminates.

4.3 User Authentication

- The program reads the authentication file (`-a`) containing the username and password.
- The `login` method sends a `LOGIN` command to the IMAP server with the provided credentials.
- If authentication fails, the server's response is analyzed to determine the reason for failure, and the program terminates with a detailed error message.

4.4 Selecting a Mailbox

- The `select` method sends a `SELECT` command to the server to work with the specified mailbox (`-b`).
- By default, the program uses the `INBOX` mailbox unless another mailbox is explicitly specified.
- If the mailbox cannot be selected, an error message is displayed, and the program exits.

4.5 Searching for Messages

- The `search` method sends a `SEARCH` command to the server to retrieve a list of message IDs.
- If the `-n` option is used, the search is limited to unread messages (`UNSEEN`). Otherwise, all messages (`ALL`) are retrieved.
- The server responds with a space-separated list of message IDs, which are parsed for further processing.

4.6 Fetching Messages

- The program iterates over each message ID obtained from the `SEARCH` command.
- For each message, the `fetch` method sends a `FETCH` command with the `BODY[]` argument to retrieve the entire message (or just headers if `-h` is used).
- The raw email data is processed to extract headers and the message body using helper functions like `extract_headers_and_body`.
- The extracted information is saved to a file in the specified output directory (`-o`) in RFC 5322 format.

4.7 Saving Messages to Files

- Each message is saved as a separate file named using the message ID and username, ensuring uniqueness.
- The headers and body are separated by a blank line, as required by the RFC 5322 standard.
- The program ensures the output directory exists, creating it if necessary, and clears any existing files in the directory.

4.8 Logging Out and Cleaning Up

- After processing all messages, the program sends a `LOGOUT` command to the server to properly terminate the session.
- Any allocated resources, such as SSL contexts and sockets, are freed using the `finish` method.
- The program outputs the total number of downloaded messages and exits.

5 Message Processing Workflow

To handle message processing, the program performs the following operations:

- **Decoding MIME Headers**: Encoded headers (e.g., `Subject`) are decoded using Base64 or Quoted-Printable algorithms.

- **Parsing Message Body**: The body is extracted from the raw email data, and any encoded content (e.g., Base64) is decoded.
- **Formatting Headers**: Headers are rearranged and filtered to display only essential fields like `Date`, `From`, `To`, `Subject`, and `Message-ID`.

6 Testing the Application

This section provides test cases for the IMAP client application, demonstrating various command-line invocations and their expected outputs. These tests cover different functionalities and edge cases to ensure the program works as intended.

6.1 Fetching All Emails from INBOX

Command:

```
imapcl eva.fit.vutbr.cz -o maildir -a cred
```

Expected Output:

```
Downloaded 15 messages from mailbox INBOX.
```

Explanation: This command connects to the IMAP server `eva.fit.vutbr.cz`, authenticates using credentials from `cred`, and downloads all emails from the default mailbox (`INBOX`) into the directory `maildir`.

6.2 Fetching Only Email Headers

Command:

```
imapcl eva.fit.vutbr.cz -o maildir -h -a cred
```

Expected Output:

```
Downloaded headers of 15 messages from mailbox INBOX.
```

Explanation: This command downloads only the headers of all emails from the default mailbox (`INBOX`) and saves them to the directory `maildir`.

6.3 Handling Invalid Credentials

Command:

```
imapcl eva.fit.vutbr.cz -o maildir -a wrong_cred
```

Expected Output:

```
Login failed.
```

Explanation: This test checks how the program handles incorrect authentication. The program should terminate with an error message.

6.4 Fetching Emails with Missing Output Directory

Command:

```
imapcl eva.fit.vutbr.cz -a cred
```

Expected Output:

```
Error: Output directory was not specified.
```

Explanation: This test ensures the program correctly handles missing mandatory arguments (`-o` in this case).

6.5 Using Custom Certificate Paths

Command:

```
imapcl eva.fit.vutbr.cz -T -c cert.pem -C /custom/certs -o maildir -a cred
```

Expected Output:

```
Downloaded 15 messages from mailbox INBOX.
```

Explanation: This test verifies that the program correctly uses custom certificate paths for secure connections.

6.6 Saving Emails in RFC 5322 Format

Verification: After running any successful fetch command, open one of the saved email files in the output directory (e.g., `maildir`):

```
cat maildir/n_xdvory00.txt
```

Expected Content:

```
Date: Wed, 14 Sep 2016 03:54:39 -0700
From: Sender <sender@example.com>
To: Receiver <receiver@example.com>
Subject: Test Email
Message-ID: <20160914035439.03264562@example.com>

This is the email body.
```

Explanation: The saved email should follow the Internet Message Format (RFC 5322), with headers and body separated by a blank line.

7 Error Handling

Common errors and their descriptions:

- `Connection failed: <reason>`: Unable to connect to the server.
- `Login failed`: Invalid credentials.
- `Timeout while waiting for server response`: The server did not respond within the expected timeframe.