VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IMAP Client – Manual

16th November, 2024

Tomáš Dolák

Contents

1 Introduction

The project assignment in the ISA (Network Applications and Network Administration) subject was to create IMAP4rev1 (according to RFC3501), which will be able to communicate only over TCP/IP as well as using SLL/TLS - IMAPS. The following program downloads emails from the defined server in the argument of the program call and saves them in the output directory. If the path to the output repository specified by the argument does not exist, the program creates it on this path.

1.1 IMAP in Email Services and How It's Used

1.1.1 Internet Message Access Protocol

The Internet Message Access Protocol (IMAP) is a protocol for receiving email that allows users to access their mailboxes from any device with an internet connection. IMAP was designed to provide flexibility in accessing email messages, regardless of the device or application used. This is achieved by acting as an intermediary between the email server and client, rather than directly downloading emails onto the device. However, IMAP also allows emails to be stored locally, a feature leveraged by this program to download and store emails on the user's device [1, 2].

While IMAP enhances flexibility, it does come with security considerations. By default, IMAP transmits login credentials in plain text, leaving usernames and passwords vulnerable if intercepted. This issue can be mitigated by configuring IMAP to operate over Transport Layer Security (TLS), which can be enabled in our program using the `-T` parameter, encrypting the communication between the client and server. Using TLS is a recommended practice to improve IMAP security [2, 3, 4].

1.1.2 How IMAP Works?

IMAP establishes a connection between the email client and server. In most email applications, IMAP allows users to view email headers quickly and download full messages only when selected, which conserves data. Outgoing emails, on the other hand, are sent via the Simple Mail Transfer Protocol (SMTP), which handles message delivery to the recipient [5, 4].

2 Using the `imapcl` Program

The `imapcl` program is designed to download email messages from a server via IMAP protocol, with optional support for encrypted connections via SSL/TLS. To run the program, use the following command syntax:

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

The order of parameters is flexible. Below is a description of each parameter:

2.1 Required Parameters

- `server`: IP address or domain name of the IMAP server.
- `-a auth_file`: Path to the file containing user authentication credentials.
- `-o out_dir`: Output directory where downloaded messages will be saved.

2.2 Optional Parameters

- `-p port`: Specifies the server port. The default value depends on whether the `-T` parameter is used (use port 993 for encrypted connections).
- `-T`: Enables an encrypted connection using the IMAPS protocol. If not specified, an unencrypted connection (port 143) is used.
- `-c certfile`: Path to a certificate file used to verify the validity of the server's SSL/TLS certificate.
- `-C certaddr`: Path to a directory containing certificates to verify the SSL/TLS certificate presented by the server. The default value is `/etc/ssl/certs`.
- `-n`: Only new messages – the program will work only with messages that have not yet been downloaded.
- `-h`: Download only message headers without the content.
- `-b MAILBOX`: Specifies the mailbox name on the server. The default is `INBOX`.

Example how to use the program:

```
./imapcl imap.outlook.cz -p 143 -a ~/Desktop/auth_file.txt -b INBOX \
-o ~/Desktop/email/user_inbox/
```

If program is called wrongly or with wrong arguments, the program will print a help message to the user.

2.3 Examples of Downloaded Email Filenames

If the arguments `-h` and `-n` are not used, the output filenames of downloaded emails will look like this:

```
MSG_INBOX_1717.txt
```

where `INBOX` is the mailbox and `1717` is the UID of this email.

If the arguments `-h` and `-n` are used, the output filenames of downloaded emails will look like this:

```
MSG_new_header_INBOX_1717.txt
```

where `new_header` indicates that the email contains just the header of emails that the IMAP server categorizes as new, `INBOX` is the mailbox and `1717` is the UID of this email.

If only the argument `-h` is used, the output filenames of downloaded emails will look like this:

```
MSG_header_INBOX_1717.txt
```

where `MSG_header` indicates that the file contains only the email's header, `INBOX` is the mailbox, and `1717` is the UID of this email.

If only the argument `-n` is used, the output filenames of downloaded emails will look like this:

```
MSG_new_INBOX_1717.txt
```

where `MSG_new` indicates that the file contains an email that the IMAP server categorizes as new, `INBOX` is the mailbox and `1717` is the UID of this email.

3 Implementation Details

The principles of object-oriented programming have been applied to the program code. The program is divided into logical classes such as the `ClientConfig` class - which takes care of the configuration of the resulting IMAP client based on the input arguments of the program, which it also processes. On the result of the configuration, either an instance of the `NonSecureImapClient` class is created, which mediates communication, classically only over TCP/IP, or an instance of the `SecureImapClient` class, which also uses SSL/TLS in addition to TCP/IP. The `SecureImapClient` class utilizes OpenSSL's **BIO** (Basic Input/Output) abstraction for handling SSL/TLS communication. The BIO object is configured to establish a secure connection with the IMAP server, handle handshakes, and ensure encrypted data transmission. Both classes `NonSecureImapClient` and `SecureImapClient` inherit basic properties from `BaseImapClient`, which provides features that are common to both derived classes, such as generating a TAG, finding the value of the current TAG or translating a hostname to an IPv4 address.[6]

3.1 Implementation of Client

The NonSecureImapClient and SecureClient classes are characterized by their very similar behavior, at the beginning when the class is instantiated they receive information like `MailBox`, `OutputDirectory`, `HeadersOnly` and `NewOnly` as input parameters. These parameters are used to define further behavior of the program and their description is given in table below.

Parameter	**Description**
MailBox	Mailbox from which emails will be downloaded
OutputDirectory	Specifies where downloaded emails will be stored
HeadersOnly	Only header of the emails will be downloaded
NewOnly	Only unseen emails will be downloaded

Then, from the user's point of view, the classes only need to call the Run method with the parameters server address, port login and password. This method interacts with the server and its behaviour is as follows.

3.2 Program's Features

3.2.1 Usage of UIDVALIDITY Value

The program also contains some of the extra features, the first of which is that the client saves a special `.uidvalidity` file with the value `UIDVALIDITY` when downloading to a specific output directory for the first time, this value is used in case the user wants to repeatedly download files to the same `output directory` and have synchronized mailboxes. On each subsequent run, the `UIDVALIDITY` value is checked to see if it is the same locally (in this `.uidvalidity` file) and on the server, if the values are different, it is a sign that the structure of the mailbox on the server has changed (i.e. the emails have been removed or moved to another mailbox, etc.) and the output directory needs to be purged and the emails downloaded again. The `.uidvalidity` file is formatted according to the rule that each `mailbox` has its corresponding line, which contains the mailbox name followed by the equal and finally the `uidvalidity value` for the given mailbox. Note that the file is not associated with the name of the IMAP server or user account, so it is recommended to download only emails from multiple mailboxes of one account to one output directory to maintain clarity and easy orientation in the application output.

An example of the ‘.uidvalidity‘ file content:

```
INBOX=1662988148
SPECIFIC_MAILBOX1=1662988155
SPECIFIC_MAILBOX2=1362922155
```

In this example:

- `INBOX=1662988148` indicates that the `uidvalidity` value for the "INBOX" mailbox is `1662988148`.
- `SPECIFIC_MAILBOX1=1662988155` indicates that the `uidvalidity` value for the "SPECIFIC_MAILBOX1" mailbox is `1662988155`.
- `SPECIFIC_MAILBOX2=1362922155` indicates that the `uidvalidity` value for the "SPECIFIC_MAILBOX2" mailbox is `1362922155`.

This is done to ensure that the locally downloaded emails are always synchronized with those on the server and to allow users to download emails from multiple mailboxes to a single output directory.

3.2.2 Creation of Output Directory

The client always needs to have a specific `output directory` to which it will download emails from the IMAP server, it can happen that the user specifies his/her own location, but the folder on the given path does not exist (for example `-o /Desktop/email/my_mailbox/`) in this case the client is able to create the folder on the given path (in this example on `/Desktop/email/my_mailbox/`) and store the emails in it.

3.2.3 Store Emails From Multiple Mailboxes Into One Output Directory

The user of `imapcl` has the possibility to download emails from several mailboxes into one `output` directory, the names of output files with saved emails are defined according to the `UID value` of the given email and according to the `mailbox` to which the email is saved. it is highly recommended not to mix emails e.g. from two IMAP servers into one output directory, in case of a mailbox name match the emails from the previous download will be deleted and replaced by the current download.

3.2.4 Gentle Download

The program also introduces a **gentle download**, i.e. emails are downloaded only if the emails are downloaded to a given output directory for the first time, or the structure of the local copy of the mailbox or the structure on the IMAP server has changed. For example, in a local folder where emails have been downloaded once before, some email(s) in the folder have been removed (perhaps by mistake) or the mailbox on the IMAP server has been modified and the `UIDVALIDITY` value has changed. The client is always in sync with the IMAP server.

3.2.5 What If Server Does Sends Any Response?

In order to prevent the program from freezing due to waiting for a response from the server, a timeout is set on the sockets (20 seconds for the unsecure version of the program, 30 seconds for the secure version). These values can be configured using the macros `TIMEOUT_NON_SECURE` and `TIMEOUT_SECURE` in the ***definitions.hpp*** file located in the ***include*** folder.

3.3 Known Limitations

A user’s mailbox can often be bulky and contain a number of large emails whose size can exceed 30MB, in which case it was observed during testing that downloading the entire mailbox can take a considerable amount of time and should be taken into account.

3.4 Adjustable Definitions

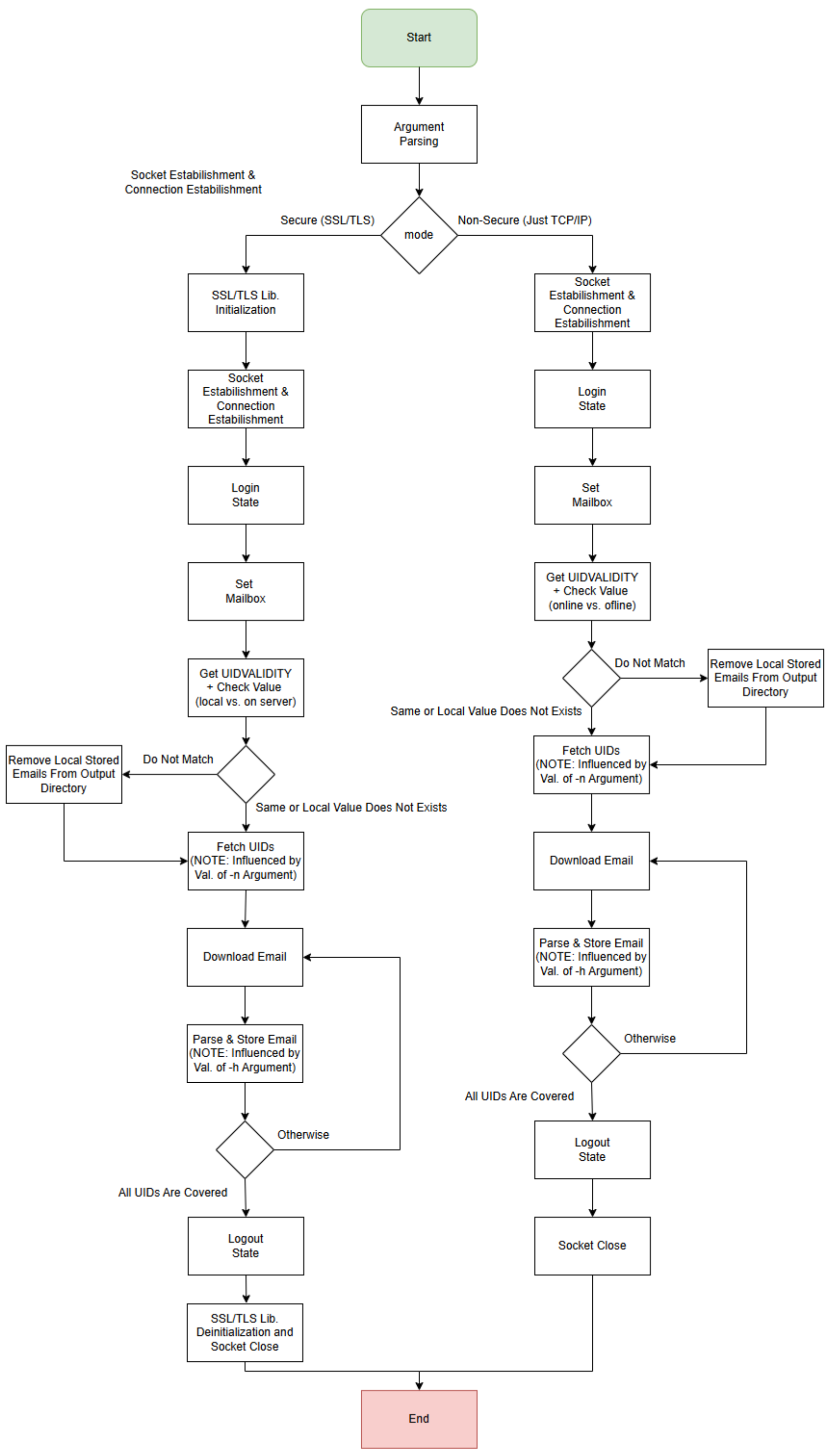
The relevant adjustable definitions are defined in the ***definitions.hpp*** file in the ***include*** folder. Some of them can be modified according to the user's needs, here is the content of the editable definitions:

- `PORT_NON_SECURE` - Default port for non-secure mode if the client does not specify one. The default value is `143`.
- `TIMEOUT_SECURE` - Timeout for receiving data from the IMAP server in secure mode, defined in seconds. The default value is `30s`.
- `DEFAULT_SSL_CERT_LOC` - Default location of SSL certificates, set to `/etc/ssl/certs`.
- `DEFAULT_MAILBOX_DIR` - Default mailbox directory, set to `INBOX`.
- `UIDVALIDITY_FILE` - Filename for storing UID validity information, set to `.uidvalidity.txt`.
- `PORT_SECURE` - Default port for secure mode if the client does not specify one. The default value is `993`.
- `TIMEOUT_NON_SECURE` - Timeout for receiving data from the IMAP server in non-secure mode, defined in seconds. The default value is `20s`.
- `OUTPUT_FILE_FORMAT` - Format for output files, set to `.log`.

3.5 Program Flow

The behaviour of the program has already been mentioned, this chapter contains a more detailed description and the program flow diagram shows the behaviour of the client in graphical form.

The program parses the program arguments at the beginning - find out how the client should be configured. Next, a client is created according to the request, which establishes a connection to the IMAP server using only `TCP/IP` or a combination of `TCP/IP` with `SSL/TLS`. If the connection is established, it sends a `LOGIN` command to the server, which should log the user in. In case of successful login, the SELECT command selects the `mailbox` from which the user wants to download emails (note: the default is `INBOX`), next, the `UIDVALIDITY` value is verified against the local copy (if the local copy does not exist in the `output directory`, it is created). Then all `UIDs` related to the mailbox are retrieved using the `FETCH` command (the collection of retrieved `UIDs` can be influenced by this parameter if the `'-n'` argument is used). After the set of `UIDs` is obtained, the email download sequence starts, the emails are downloaded one by one only - if is not yet downloaded locally into this specific output directory and the rest of the communication with the IMAP client is removed. Finally, the email is stored in the output directory specified by the `'-o'` argument and the client repeats this process with the next email with a UID from the set of pending emails.



Start
Argument Parsing
Socket Estabilishment & Connection Estabilishment
mode
Secure (SSL/TLS)
Non-Secure (Just TCP/IP)
SSL/TLS Lib. Initialization
Socket Estabilishment & Connection Estabilishment
Login State
Set Mailbox
Get UIDVALIDITY + Check Value (local vs. on server)
Do Not Match
Remove Local Stored Emails From Output Directory
Same or Local Value Does Not Exists
Fetch UIDs (NOTE: Influenced by Val. of -n Argument)
Download Email
Parse & Store Email (NOTE: Influenced by Val. of -h Argument)
Otherwise
All UIDs Are Covered
Logout State
SSL/TLS Lib. Deinitialization and Socket Close
Socket Estabilishment & Connection Estabilishment
Login State
Set Mailbox
Get UIDVALIDITY + Check Value (online vs. ofline)
Do Not Match
Remove Local Stored Emails From Output Directory
Same or Local Value Does Not Exists
Fetch UIDs (NOTE: Influenced by Val. of -n Argument)
Download Email
Parse & Store Email (NOTE: Influenced by Val. of -h Argument)
Otherwise
All UIDs Are Covered
Logout State
Socket Close
End

Figure 1: Program Flow Diagram

3.6 Classes

This section contains a visualization of the main program classes using a diagram based on the use-case diagram. The aim of these diagrams was to capture the individual classes and the methods/operations they allow.

3.6.1 ClientConfig

A class for processing user arguments, which is also responsible for storing the configuration of the future IMAP Client.

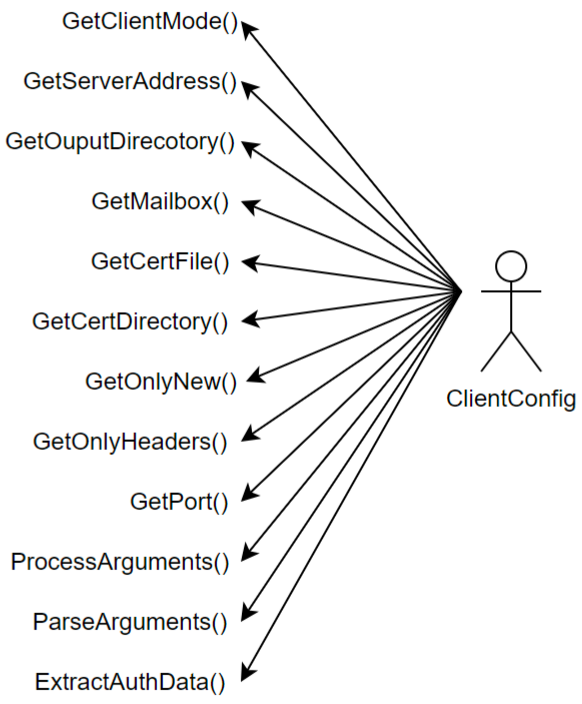


Figure 2: Class ClientConfig and ClientConfig Methods

3.6.2 ImapClients

Here it may be seen that both `SecureImapClient` and `NonSecureImapClient` have a common ancestor, which is `BaseImapClient` - providing common methods for both successors. On the basis of the configuration, an instance of the `SecureImapClient` or `NonSecureImapClient` class is created, it is responsible for taking care of the whole process of communication with the IMAP server and storing the results.

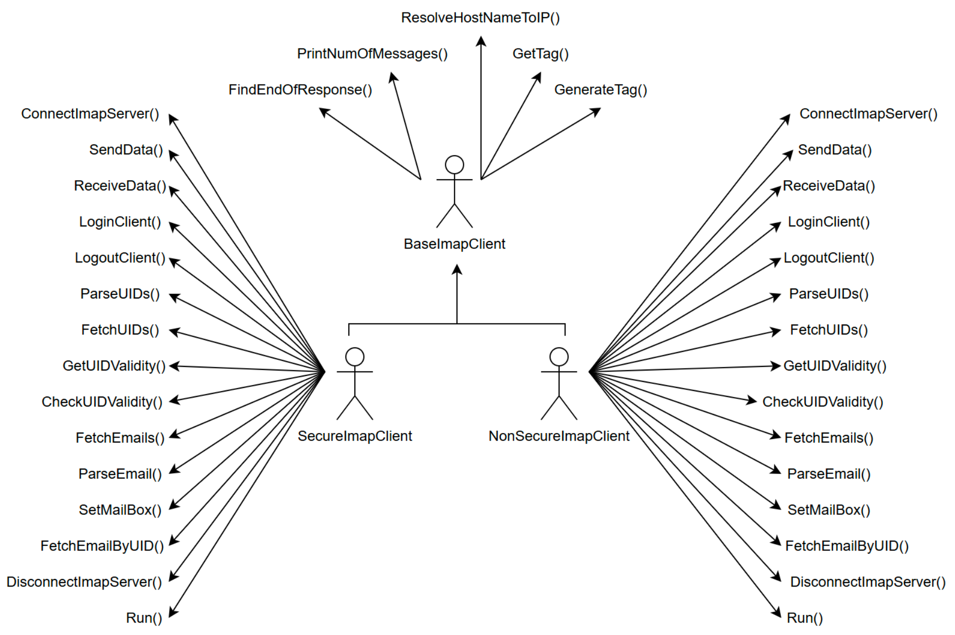


Figure 3: ImapClient's Classes and Methods

4 Testing

The `imapcl` program has also been tested. The testing was performed in two ways, the first one was tested using the local implementation of the IMAP server, the second way was chosen to test on foreign IMAP servers provided by the school such as eva or merlin.

4.1 Testing on Local IMAP Server

To ensure the correct operation of the program, `imapcl` was tested using a local implementation of the IMAP server. The local IMAP server is available in the repository in the tests folder under the name `imap_server.py`. The server is implemented in Python using the `os`, `socket`, `threading`, `subprocess`, `time`, and `signal` libraries and handles all requests from the client implementation such as `LOGIN`, `FETCH` (fetch support involves responding to specific requests such as `'tag' UID 'x' FETCH BODY[HEADER]` or `'tag' UID 'x' FETCH BODY[TEXT]` where `'tag'` is specific tag of the request and `'x'` is UID of the email), `SELECT`, and `LOGOUT`.

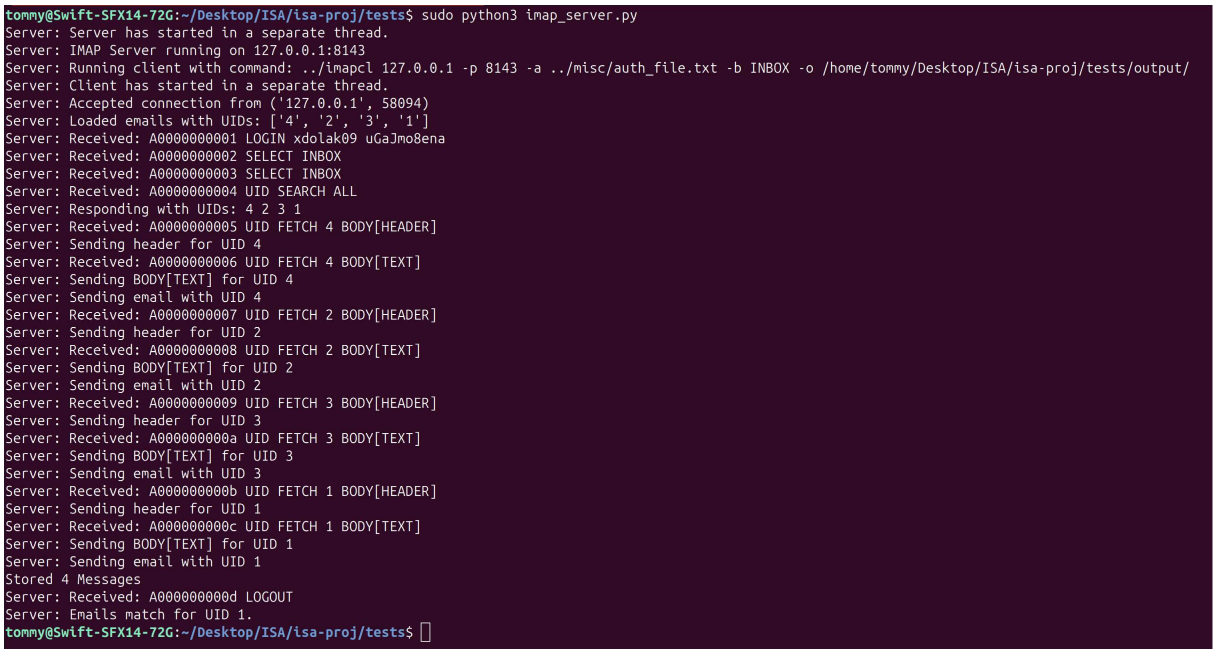
The tests folder also contains the test_emails folder with sample emails that are sent to the IMAP client, at the same time these emails are used to compare what was sent with what was received on the IMAP client, the contents should be the same - the result of the comparison is displayed in the output of the script `imap_server.py`.

The local IMAP server was used for basic testing of the application functionality, and at a later stage of development it was used to simulate errors and unusual conditions that can occur during communication between client and server.

4.1.1 Implementation Details of Local IMAP Server

To run the test correctly it is necessary to configure the parameters client_path (containing the name of the binary file with the program), output_dir (where the output files will be stored) and client_args (arguments with which the tested program will be called), the local IMAP server uses the address 127.0.0.1 and runs by default on port 8143.

Here can we be seen the output of the test, which ended with success, the test during its run logs to the terminal information about what is currently happening.



```
tommy@Swift-SFX14-72G:~/Desktop/ISA/isa-proj/tests$ sudo python3 imap_server.py
Server: Server has started in a separate thread.
Server: IMAP Server running on 127.0.0.1:8143
Server: Running client with command: ../imapcl 127.0.0.1 -p 8143 -a ../misc/auth_file.txt -b INBOX -o /home/tommy/Desktop/ISA/isa-proj/tests/output/
Server: Client has started in a separate thread.
Server: Accepted connection from ('127.0.0.1', 58094)
Server: Loaded emails with UIDs: ['4', '2', '3', '1']
Server: Received: A0000000001 LOGIN xdolak09 uGaJmo8ena
Server: Received: A0000000002 SELECT INBOX
Server: Received: A0000000003 SELECT INBOX
Server: Received: A0000000004 UID SEARCH ALL
Server: Responding with UIDs: 4 2 3 1
Server: Received: A0000000005 UID FETCH 4 BODY[HEADER]
Server: Sending header for UID 4
Server: Received: A0000000006 UID FETCH 4 BODY[TEXT]
Server: Sending BODY[TEXT] for UID 4
Server: Sending email with UID 4
Server: Received: A0000000007 UID FETCH 2 BODY[HEADER]
Server: Sending header for UID 2
Server: Received: A0000000008 UID FETCH 2 BODY[TEXT]
Server: Sending BODY[TEXT] for UID 2
Server: Sending email with UID 2
Server: Received: A0000000009 UID FETCH 3 BODY[HEADER]
Server: Sending header for UID 3
Server: Received: A000000000a UID FETCH 3 BODY[TEXT]
Server: Sending BODY[TEXT] for UID 3
Server: Sending email with UID 3
Server: Received: A000000000b UID FETCH 1 BODY[HEADER]
Server: Sending header for UID 1
Server: Received: A000000000c UID FETCH 1 BODY[TEXT]
Server: Sending BODY[TEXT] for UID 1
Server: Sending email with UID 1
Stored 4 Messages
Server: Received: A000000000d LOGOUT
Server: Emails match for UID 1.
tommy@Swift-SFX14-72G:~/Desktop/ISA/isa-proj/tests$ 
```

Figure 4: Output of imap_server.py

Testcase	Expected Output	Program Output
Server does not responds to `LOGIN` command.	Program should exit with `BAD_RESPONSE` error code.	Program ends with `BAD_RESPONSE` error code.
Server sends different end of response string (e.g. not `TAG OK FETCH Completed` but `TAG OK fetch`).	Client should handle this kind change and continue.	Client successfully proceeds and completes its download.
Server sends different `.uidvalidity` value for mailbox.	Output directory should be cleared and emails should be downloaded again.	Output directory is cleared and emails are downloaded again.
Server does not respond at all.	Client should successfully end and print error message on stderr.	Client successfully ends and prints error message on stderr.
Stress test, sending three emails, size of each of them 33MB.	Emails successfully downloaded.	Emails were successfully downloaded. Note: Speed was not optimal.
Sent an email whose size slightly exceeds the receiving buffer (the end of the email is split into two parts)	No freezing of the program and email successfully downloaded.	No freezing of the program and email successfully downloaded.

4.2 Testing on Foreign IMAP Servers

As a complementary method, testing on foreign IMAP servers provided by the school, such as eva or merlin, was chosen to ensure compatibility outside the local environment and to demonstrate the usability of the program.

Testcase	Server	Expected Output	Program Output
Compilation	eva	Successful compilation.	Successful compilation.
Auth. file parsing	eva	Auth. file successfully parsed	Auth. file successfully parsed.
Calling `./imapcl outlook.office365.com -p 143 -a PATH/auth_file.txt -b INBOX -o PATH/output/`	eva	Downloaded whole emails (including header and body), note: number emails on server was 1739.	Downloaded 1739 from server with headers and bodies.
Calling `./imapcl outlook.office365.com -p 143 -a PATH/auth_file.txt -b INBOX -o PATH/output/` with changed value of `.uidvalidity` file	eva	Downloaded whole emails (including header and body) and updated value in `.uidvalidity` file, note: number emails on server was 1739	Downloaded 1739 from server with headers and bodies. Updated valued of `.uidvalidity` file.

Testcase	**Server**	**Expected Output**	**Program Output**
Calling `./imapcl outlook.office365.com -p 143 -a PATH/auth_file.txt -b INBOX -o PATH/output/` and path `PATH/output/` does not exists	eva	Downloaded whole emails (including header and body) and created output directory in `PATH/output/` file, note: number emails on server was 1739	Downloaded 1739 from server with headers and bodies. in output directory `PATH/output/`.
Calling `./imapcl outlook.office365.com -T -a PATH/auth_file.txt -b INBOX -o PATH/output/` with changed value of `.uidvalidity` file	eva	Downloaded whole emails (including header and body) and updated value in `.uidvalidity` file thru SSL/TLS on port 993, note: number emails on server was 1739	Downloaded 1739 from server with headers and bodies. Updated valued of `.uidvalidity` file. Used SSL/TLS on default port 993.
Compilation	merlin	Successful Compilation	Successful Compilation
Auth. file parsing	merlin	Auth. file successfully parsed	Auth. file successfully parsed.
Calling `./imapcl eva.fit.vutbr.cz -p 143 -a PATH/auth_file.txt -b INBOX -o PATH/output/`	merlin	Downloaded whole emails (including header and body), note: number emails on server was 2349.	Downloaded 2349 from server with headers and bodies.
Calling `./imapcl eva.fit.vutbr.cz -p 143 -h -a PATH/auth_file.txt -b INBOX -o PATH/output/`	merlin	Downloaded email only headers, note: number emails on server was 2349.	Downloaded 2349 email headers from server.
Calling `./imapcl eva.fit.vutbr.cz -p 143 -n -a PATH/auth_file.txt -b INBOX -o PATH/output/`	merlin	Downloaded email only new emails (headers + bodies), note: number emails on server was 56.	Downloaded 56 new emails from server with headers and bodies.
Calling `./imapcl eva.fit.vutbr.cz -p 143 -n -a PATH/auth_file.txt -b BOX1 -o PATH/output/`	merlin	Two emails should be downloaded and `UIDVALIDITY` of `BOX1` should be stored into `.uidvalidity.txt`	Two emails were downloaded and `UIDVALIDITY` of `BOX1` were saved into `.uidvalidity.txt`
Calling `./imapcl eva.fit.vutbr.cz -T -p 993 -n -a PATH/auth_file.txt -b INBOX -o PATH/output/`	merlin	For SSL cetificates should be used default location - `/etc/ssl/certs`.	Used default location for SSL certificates, `/etc/ssl/certs`.

Testcase	Server	Expected Output	Program Output
Calling `./imapcl eva.fit.vutbr.cz -T -p 993 -n -a PATH/auth_file.txt -b BOX1 -o PATH/output/` with stored all emails from mailbox and right value of `UIDVALIDITY`	merlin	Zero emails should be downloaded	Zero emails downloaded.
Calling `./imapcl eva.fit.vutbr.cz -p 143 -h -n -a PATH/auth_file.txt -b INBOX -o PATH/output/`	merlin	Downloaded email only new emails (only headers), note: number emails on server was 56.	Downloaded 56 headers of new emails from server.
Calling `./imapcl eva.fit.vutbr.cz -g 143 -h -n -a PATH/auth_file.txt -b INBOX -o PATH/output/`	merlin	Wrong argument `-g`, help on terminal should be printed.	Printed help to the user on terminal.
Calling `./imapcl eva.fit.vutbr.cz -p 143`	merlin	A small number of arguments for calling the program. The error message should be displayed to the user.	Displayed the error message that informs user a small number of arguments for calling the program that were used.

5 Return Codes

The program is designed in such a way that in case of an error or failure, the user is always informed about the situation that has occurred. In case of success the program always returns the return value 0 (respectively `SUCCESS`), in other cases the program gives the return values according to the table below.

Name	Data Type	Value	Description
OUTPUT_DIR_NOT_CREATED	int	-7	Output Directory Could Not Be Created
UIDVALIDITY_FILE_NOT_FOUND	int	-6	.uidvalidity File Not Found
UIDVALIDITY_FILE_ERROR	int	-5	Error With .uidvalidity File (Invalid Format, Out of Range)
CREATE_CONNECTION_FAILED	int	-4	Failed to Create Connection With IMAP Server
SSL_CERT_VERIFICATION_FAILED	int	-3	SSL Certificate Verification Failed

FETCH_EMAIL_FAILED	int	-2	Fetching Email By UID Failed
SUCCESS	int	0	Operation Was Successful
NO_IP_ADDR_FOUND	int	1	No IPv4 Address Was Found
PARSE_ARGUMENTS_FAILED	int	2	Parsing Program's Arguments Failed
PARSE_CREDENTIALS_FAILED	int	3	Parsing Credentials Failed
SERVER_UNKNOWN_RESPONSE	int	4	Server Sent an Unknown Response
TRANSMIT_DATA_FAILED	int	5	Transmission of Data Failed
RECEIVE_DATA_FAILED	int	6	Reception of Data Failed
RESPONSE_NOT_FOUND	int	7	Expected Server's Response Was Not Found
PARSE_BY_REGEX_FAILED	int	8	Parsing of Regular Expression Failed
NON_UIDS_RECEIVED	int	9	No UIDs Were Received From The IMAP Server
CONTINUE_IN_RECEIVING	int	10	Continue Receiving More Data
UNDEFINED_STATE	int	11	Undefined State Encountered
UID_VALIDITY_ERROR_IN_RECV	int	14	Unable to Receive UIDVALIDITY From The Server
REMOVAL_OF_EMAILS_FAILED	int	15	Failed to Remove Emails When UIDVALIDITY Does Not Match
BAD_RESPONSE	string	"Bad Response :("	Error During Receiving of Server's Response

References

[1] Internet Engineering Task Force. Internet message access protocol - version 4rev1. [online], 2024. [cited 2024-10-28].

[2] Cloudflare. What is imap? [online], 2024. [cited 2024-10-28].

[3] Microsoft Support. What is the difference between pop and imap? [online], 2024. [cited 2024-10-28].

[4] EmailLabs. Email protocols: Learn the difference between imap, pop3, and smtp. [online], 2024. [cited 2024-10-28].

[5] TechTarget. Imap (internet message access protocol). [online], 2024. [cited 2024-10-28].

[6] IBM Developer. Secure programming with the openssl api. [online], 2024. [cited 2024-10-28].