VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

SÍŤOVÉ APLIKACE A SPRÁVA SÍTÍ

NETWORK APPLICATIONS AND NETWORK ADMINISTRATION

KLIENT IMAP S PODPOROU TLS

IMAP CLIENT WITH TLS SUPPORT

SEMESTRÁLNÍ PRÁCE

SEMESTRAL PROJECT

AUTOR PRÁCE **Jaroslav Louma**

AUTHOR

VEDOUCÍ PRÁCE **Ing. Libor Polčák, Ph.D.**

SUPERVISOR

BRNO 2024

Obsah

1. Úvod

Cieľom projektu bolo vytvoriť klienta pre protokol IMAP s podporou TLS. Klient má za úlohu stiahnuť zprávy uložené na serveri, a to do zadaného adresára. Sťahované správy sa majú ďalej filtrovať podľa príslušných parametrov. Klient musí byť implementovaný v súlade s RFC 3501 (Internet Message Access Protocol) a musí podporovať šifrovanie komunikácie pomocou TLS.

1.1 Protokol IMAP

IMAP (Internet Message Access Protocol) je protokol, definovaný v RFC 3501, ktorý slúži na prístup k správam uloženým na serveri. Tento projekt implementuje len časť klienta (nie server). Protokol umožňuje klientovi prevádzať rôzne operácie, napríklad čítať, upravovať a mazať uložené správy, avšak táto implementácia klient podporuje len základné operácie, a to prihlásenie, výber schránky, vyhľadávanie správ, stiahnutie správ/hlavičiek a odhlásenie. Samotný protokol IMAP špecifikuje možnosť šifrovania prenášaných dát protokolom TLS, pomocou explicitného príkazu `STARTTLS`, no táto implementácia využíva implicitné šifrovanie, kde sa šifrovaný kanál vytvára už pri pripájaní na server, vďaka čomu samotný protokol IMAP šifrovanie nemusí riešiť a môže fungovať vo forme plain textu.

1.2. Protokol TLS

TLS (Transport Layer Security) je kryptografický protokol, ktorý zabezpečuje šifrovanie komunikácie medzi klientom a serverom. Protokol zabezpečuje dôvernosť, autentizáciu a integritu dát, ktoré sú prenášané medzi klientom a serverom. TLS je nástupcom protokolu SSL (Secure Sockets Layer) a je štandardizovaný v RFC 5246. V implementácii sa pre prácu so šifrovaným TSL socketom využíva knižnica OpenSSL, ktorá poskytuje API pre prácu s TLS protokolom (nadväzovanie bezpečného pripojenia, overovanie certifikátov, šifrovanie a dešifrovanie dát, ...).

2. Inštalácia

Táto sekcia obsahuje informácie o spôsobe kompilácie a spustení projektu, rovnako tak všetky dodatočné informácie, pre efektívne využívanie klienta.

2.1. Kompilácia

Pre spustenie klienta je potrebné projekt najprv skompilovať, pomocou dodaného súboru Makefile. Spustenie kompilácie je možné pomocou príkazu:

```
make
```

Alternatívne ide použiť aj `make imapcl` alebo `make all`.

Daný príkaz spustí program `make`, ktorý implicitne spustí kompiláciu projektu, pomocou kompilátora `g++`. Ďalej dodaný Makefile umožňuje `make pack` pre vytvorenie `tar` archívu s kompletným projektom a `make clean` pre odstránenie všetkých súborov vytvorených počas kompilácie (vrátane vytvoreného archívu).

2.2. Spustenie

Po úspešnom prebehnutí kompilácie je výsledkom spustiteľný súbor `imapcl`, obsahujúci IMAP klienta. Tohto klienta je možné následne spustiť pomocou príkazu:

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

Kde:

- `server` - názov serveru (IP adresa, alebo doménové meno) požadovaného zdroja
  - Musí byť vždy prítomný ako prvý argument
  - Ak je zadávaný ako doménové meno, pred pripojením sa prevedie DNS rezolúcia
- `-p port` - špecifikuje číslo portu na serveri, na ktorý sa má klient pripojiť
  - V prípade vynechania sa použije predvolená hodnota **143** alebo **993** v prípade použitia parametra `-T`
- `-T` - povolene TLS šifrovanie (imaps)
  - V prípade použitia tohto parametra sa klient pripojí na port **993** ak nie je špecifikovaný iný port
  - V prípade vynechania sa použije nešifrovaná verzia protokolu
  - V prípade zlyhania overenia certifikátu serveru sa klient ukončí s chybou

- `-c certfile` - súbor s certifikátmi, ktorý sa použije pre overenie platnosti certifikátu SSL/TLS predloženého serverom
  - Je možné použiť iba s parametrom `-T`
- `-C certaddr` - adresár, v ktorom sa majú vyhľadávať certifikáty, ktoré sa použijú pre overenie platnosti certifikátu SSL/TLS predloženého serverom
  - Je možné použiť iba s parametrom `-T`
  - V prípade vynechania sa použije predvolený adresár `/etc/ssl/certs`
- `-n` - sťahovať iba nové správy
  - Správy sú považované za nové, ak majú nastavený príznak `\Recent`
- `-h` - sťahovať iba hlavičky správ
  - Správy sú sťahované iba s hlavičkami, bez tela správy
  - Ak je prítomný tento parameter, príznak `\Recent` sa nebude meniť (správy nebudú označené ako prečítané)
- `-a auth_file` - súbor s autentizačnými údajmi
  - Súbor musí obsahovať užívateľské meno a heslo vo formáte
  - Formát súboru:

    ```
    username = <username>
    password = <password>
    ```

- `-b MAILBOX` - názov schránky, s ktorou sa bude na serveri pracovať
  - V prípade vynechania sa použije predvolená hodnota **INBOX**
- `-o out_dir` - výstupný adresár, do ktorého sa majú uložiť stiahnuté správy

Pri akomkoľvek nesprávnom zadaní argumentov sa klient ukončí s chybou a vypíše nápovedu k použitiu.

2. Implementácia

Pre implementáciu klienta bol zvolený programovací jazyk C++. Klient je implementovaný ako konzolová aplikácia, ktorá je schopná komunikovať so serverom pomocou protokolu IMAP. Klient je schopný sťahovať správy uložené na serveri a ukladať ich do zadaného adresára.

V C++ implementácii sa pre väčšinu komponentov (tried) nenachádzajú hlavičkové súbory, každá trieda je deklarovaná aj definovaná na jednom mieste. To znemožňuje kompiláciu do objektových súborov a následné linkovanie, čím sa vždy kompile celý projekt naraz, avšak projekt nie je rozsiahly a kompilácia trvá len pár sekúnd.

Podpora šifrovania je implentovaná pomocou implicitného TLS (IMAPS), kde sa už pri pripájaní na server vytvorí bezpečný šifrovaný kanál medzi klientom a serverom, cez ktorý sa v plain texte posielajú všetky dáta protokolu IMAP, narozdiel od explicitného TLS (STARTTLS), kde šifrovaný kanál, ktorého vytvorenie zabezpečuje samotný protokol IMAP, je vytvorený až po úspešnom zahájení komunikácie a vzájomnej dohode s IMAP serverom.

2.1. Popis funkcionality

Po spustení aplikácie, program najprv prevedie kompletnú vlidáciu argumentov; pri nesprávnych argumentoch sa program končí s chybou. Validné argumenty sa ukladajú do inštancie triedy `ClientSettings`. Po úspešnej validácii větkých argumentov program spúšťa klienta pomocou dodaných nastavení. Klient ako prvé číta súbor s prihlasovacími údajmi a pri chybe sa ukončuje s chybou. Následne sa vytvorí inštancia socketu `TCPSocket` alebo `TLSSocket` podľa nastavení a ten sa pripojí na server. Po úspešnom pripojení sa klient prihlási na server pomocou prihlasovacích údajov. Po úspešnom prihlásení sa klient pokúsi vybrať schránku, s ktorou bude pracovať. Následne, po úspešnom výbere schránky sa zostavuje požiadavka `FETCH` podľa zadaných parametrov (zahŕňať len nové správy, sťahovať len hlavičky) pomocou query objektu `IMAPClient::FetchQuery`. Ak parametre vyžadujú sťahovanie len nových správ, pred odoslaním požiadavky `FETCH` sa odošle požiadavka `SEARCH`, ktorá vráti UID správ, ktoré spĺňajú zadané kritéria; tie sa následne pridávajú do query objektu. Následne sa pomocou query objektu, ak nie je prázdny, vytvorí požiadavka `FETCH`, ktorá zahŕňa všetky správy, ktoré sa majú stiahnuť. Server následne vracia odpoveď obsahujúcu všetky požadované správy, ktoré sa uložia do zadaného adresára, s názvom zahashovanej hodnoty hlavičky `message-id`. Po stiahnutí všetkých požadovaných správ (aj žiadnych) sa klient zo serveru odhlási a ukončí sa s kódom `0`.

Pri akejkoľvek chybe od serveru, nesprávnej odpovedi alebo zlyhaní spracovania sa klient ukončuje s chybou.

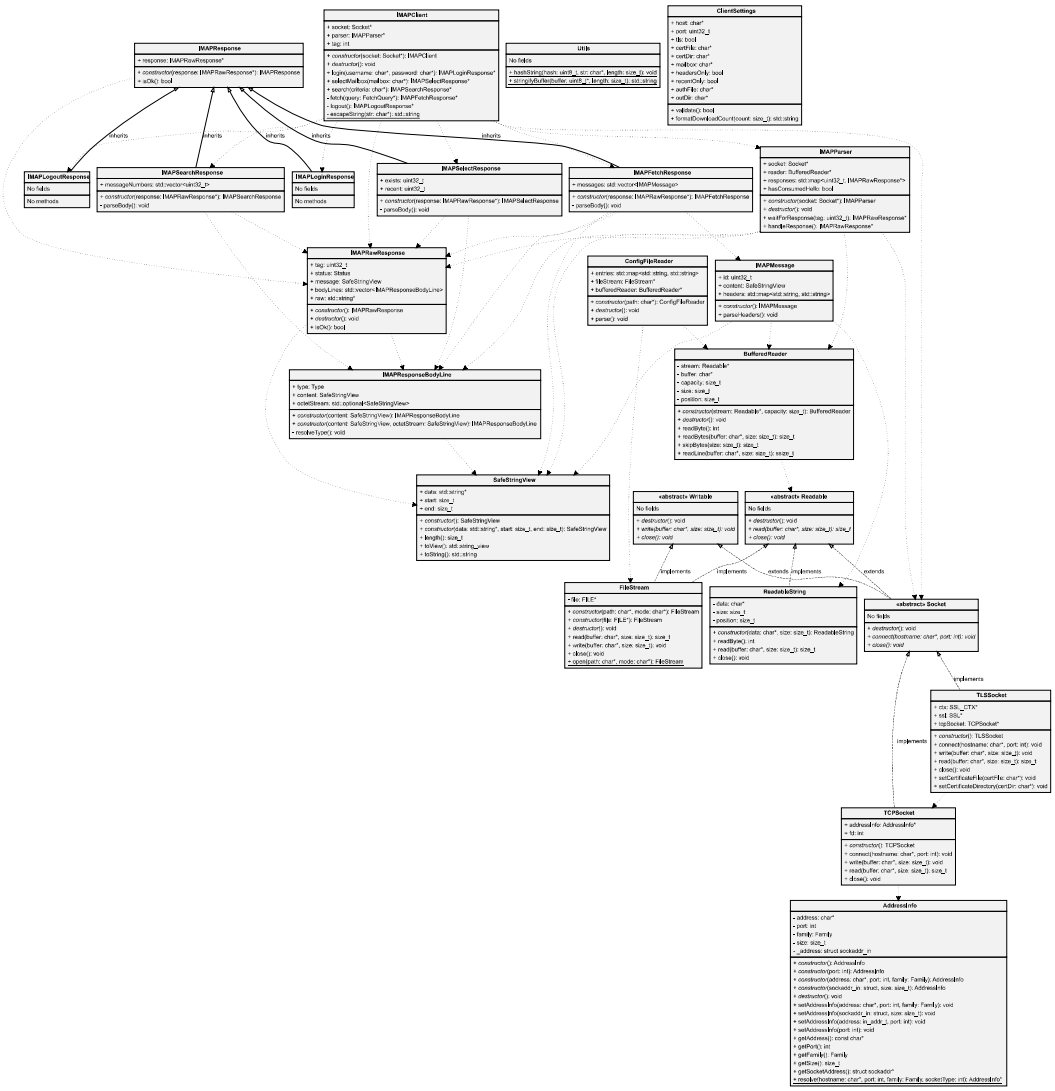
Podrobnejší popis a fungovanie jednotlivých komponentov je popísaný v časti 2.2.

Poznámka 1: Klient sťahuje vždy všetky vyžiadané správy; zadanie **nikde** explicitne neuvádza, že by mal klient používať cache alebo sťahovať len správy, ktoré ešte nie sú stiahnuté. Zadanie hovorí, že po ukončení programu musia byť správy v lokálnom adresári synchronizované s obsahom schránky na serveri, čo táto implementácia zabezpečuje.

Poznámka 2: Názvy súborov, do ktorých sa ukladajú správy, sú zahashované hodnoty hlavičky `message-id`.Toto riešenie bolo zvolené z dôvodu, že všetky hlavičky užitočné pre používateľa (subjekt, dátum, ...) sú generované klientami, nie servermi, tým pádom sa im nedá dôverovať. Hlavička `message-id` je unikátna pre každú správu a je generovaná serverom, čo zaručuje unikátnosť. Z dôvodu, že táto hodnota môže byť čokoľvek (každý server si môže zvoliť vlastný systém formátovania) ju nie je možné parsovať, preto sa pred použitím hodnota tejto hlavičky najprv zahashuje vlastným, 256-bitovým, *ne*-kryptografickým algoritmom.

2.2. Popis komponentov

Klient je rozdelený do niekoľkých komponentov - tried, ktoré sú zodpovedné za jednotlivé časti programu. Niektoré triedy obsahujú celkové implementácie, iné zase iba abstrakcie, ktoré sú následne implementované v triedach potomkov. V projekte je z veľkej časti využívaný vývojový vzor **Dependency Injection**, vďaka čomu je možné jednoducho využívať abstraktné triedy bez nutnosti poznania konkrétnej implementácie. Z tohto prístupu výrazne ťaží práve implementácia samotného klienta, ktorý nemusi riešiť, či beží na šifrovanom alebo nešifrovanom spojení. V projekte je utilizovaný systém výnimiek, vďaka čomu je možné jednoducho a efektívne riešiť chyby a výnimky, ktoré môžu nastať počas behu programu v rozličných komponentoch.



Obr. 1 - Diagram tried a všetkých súčastí projektu

`AddressInfo`

Trieda obsahujúca potrebné informácie o jednej konkrétnej adrese, a to IP adresu (IPv4 alebo IPv6), port, rodinu adresy a veľkosť adresy. Obsahuje niekoľko metód na nastavenie/vytvorenie inštancie tejto triedy z rôznych zdrojov (adresa ako string, adresa ako štruktúra `sockaddr`, ...). Taktiež obsahuje metódu `resolve`, ktorá je zodpovedná na DNS rezolúciu doménového mena na IP adresu.

`Readable` / `Writable`

Abstraktné triedy, ktoré definujú rozhranie pre triedy, ktoré sú schopné čítať resp. zapisovať dáta. Triedy, ktoré implementujú tieto rozhrania, musia implementovať metódy `read` resp. `write` (a `close`), ktoré sú schopné čítať resp. zapisovať dáta z/do zdroja. Tieto triedy sú implementované rôznymi ďalšími súčasťami projektu.

`Socket`

Abstraktná trieda rozširujúca triedu `Readable` a `Writable`, ktorá definuje rozhranie pre triedy, ktoré sú schopné komunikovať cez soket. Keďže sa v projekte pužíva TCP Socket len v móde klienta, triedy, ktoré implementujú toto rozhranie, musia implementovať len metódu `connect` (vynechané metódy ako `listen`, `bind`, `accept`, ...). Táto trieda je základom pre triedy `TCPSocket` a `TLSSocket`.

`TCPSocket`

Trieda, ktorá implementuje rozhranie `Socket` a je zodpovedná za komunikáciu cez TCP soket. Trieda obsahuje metódy na pripojenie sa na server, odoslanie dát, prijatie dát a zatvorenie soketu. Trieda je implementovaná pomocou triedy `Socket`, ktorá je zodpovedná za vytvorenie soketu a jeho správne nastavenie.

Ak je do metódy `connect` zadaná adresa, ktorá obsahuje doménové meno, metóda použije triedu `AddressInfo` na DNS rezolúciu doménového mena. Výsledkom je zoznam adries, na ktoré sa klient môže pripojiť. Metóda sa linárne pokúša pripojiť na všetky adresy, kým sa nepripojí na jednu z nich. Ak sa nepodarí pripojiť na žiadnu adresu, metóda sa ukončí s chybou.

`TLSSocket`

Trieda, ktorá implementuje rozhranie `Socket` a je zodpovedná za komunikáciu cez TLS soket. Trieda vnútorne využíva triedu `TCPSocket` na komunikáciu cez nešifrovaný soket, pričom všetky dáta šifruje pomocou knižnice OpenSSL. Trieda obsahuje metódy na pripojenie sa na server, odoslanie dát, prijatie dát, zatvorenie soketu a nastavenia certifikátov pre overenie platnosti certifikátu serveru.

`BufferedReader`

Keďže sa v projekte vo veľkej časti vyskytuje čítanie dát a následná práca s nimi, bola implementovaná trieda `BufferedReader`, ktorá akceptuje objekt `Readable` a umožňuje čítať dáta po bytoch (jednotlivo aj naraz),preskočiť určitý počet bytov a čítať po riadkoch (ukončenie `LF`, `CR` alebo `CRLF`). Trieda si vnútorne udržuje buffer, do ktorého sa načítavajú dáta z objektu `Readable`, len v prípade, že je to potrebné, čím sa potenciálne znižuje počet systémových volaní `read` (ak ich implementácia `Readable` využíva).

`IMAPResponseBodyLine`

Trieda, ktorá reprezentuje jednen riadok IMAP správy. Obsahuje typ riadku (`UNTAGGED`, `TAGGED` alebo `CONTINUATION`), samotný obsah riadku ako `SafeStringView` a prípadný oktetový stream, ktorý obsahuje dáta riadku, ako `SafeStringView`.

`IMAPRawResponse`

Trieda reprezentujúca jednu celú odpoveď od serveru. Obsahuje pole objektov `IMAPResponseBodyLine`, ktoré reprezentujú jednotlivé riadky odpovede, tag odpovede, stavový kód a prípadnú chybovú správu.

`IMAPResponse` + Login/Select/Search/Fetch/Logout Response

Triedy reprezentujúce konkrétny typ odpovede od serveru. Triedu `IMAPResponse` rozširujú triedy `IMAPLoginResponse`, `IMAPSelectResponse`, `IMAPSearchResponse`, `IMAPFetchResponse` a `IMAPLogoutResponse`, ktoré obsahujú konkrétne informácie o odpovedi od serveru, vhodné pre daný typ požiadavky.

- `IMAPLoginResponse` - odpoveď na prihlásenie
- `IMAPSelectResponse` - odpoveď na výber schránky
  - Obsahuje informácie o počte existujúcich a nových správach
- `IMAPSearchResponse` - odpoveď na vyhľadávanie správ
  - Obsahuje zoznam čísel správ, ktoré spĺňajú zadané kritériá
- `IMAPFetchResponse` - odpoveď na stiahnutie správ
  - Obsahuje pole objektov `IMAPMessage`, ktoré reprezentujú jednotlivé správy
- `IMAPLogoutResponse` - odpoveď na odhlásenie

Každá z týchto tried obsahuje metódu `parseBody`, ktorá je zodpovedná za parsovanie odpovede od serveru a vyplnenie objektu konkrétnymi informáciami.

`IMAPMessage`

Trieda reprezentujúca jednu správu. Obsahuje id správy, obsah ako `SafeStringView` a hlavičky správy vo forme mapy. Objekty tohto typu sa dajú získať z odpovede od serveru pomocou triedy `IMAPFetchResponse`.

`IMAPParser`

Trieda, ktorá je zodpovedná za parsovanie odpovedí od serveru. Trieda sa snaží správne parsovať jednotlivé riadky odpovede, prípadné oktetové streamy a päty správ obsahujúce tag, stavový kód a chybovú správu.

Pri vytvorení treida požaduje inštanciu `Socket` reprezentujúcu aktívne spojenie so serverom, z ktorého sa dá čítať. Trieda vnútorne používa `BufferedReader` na čítanie dát z socketu. Obsahuje práve metódy `waitForResponse` a `handleResponse`.

Metóda `waitForResponse` akceptuje `tag` správy, na ktorú sa má čakať. Trieda si okrem iného drží aj set prijatých a **nespracovaných** odpovedí. Pri zavolaní metódy `waitForResponse` je najprv snaha nájsť správu v sete nespracovaných správ, ak taká správa existuje, odstráni sa zo setu a vráti sa volajúcemu. V opačnom prípade metóda volá metódu `handleResponse`, ktorá vracia odpovede `IMAPRawResponse`, až dovtedy, kým sa neobjaví správa s požadovaným tagom. Do toho momentu sú všetky ostatné správy ukladané do setu nespracovaných správ pre budúce spracovanie.

Metóda `handleResponse` je zodpovedná za čítanie a parsovanie správ od serveru. Metóda číta dáta z `BufferedReader` po bytoch, ukladajúc ich do `IMAPResponseBodyLine`. Ak sa nájde signatúra octet streamu, metóda prečíta definvoaný počet bytov a vytvorí `SafeStringView` ukazújúci na tieto dáta. Po prečítaní celej správy sa vytvorí `IMAPRawResponse` a pridajú sa informácie z "päty" správy (tag, stavový kód, chybová správa). Táto odpoveď sa následne vráti volajúcemu.

`IMAPClient`

Hlavná trieda projektu, ktorá je zodpovedná za komunikáciu s IMAP serverom. Trieda obsahuje metódy na prihlásenie, výber schránky, vyhľadávanie správ, stiahnutie správ a odhlásenie. Trieda akceptuje aktívny `Socket` reprezentujúci spojenie so serverom a vnútne si uchováva aktuálny tag v sekcencii (reprezentovaný ako `int`, enkodovaný ako `"a<tag>"`) a inštanciu `IMAPParser` pre parsovanie odpovedí od serveru.

`SafeStringView`, `ClientSettings`, `Utils`, `FileStream` a `ConfigFileReader`

Pomocné triedy, ktoré sú zodpovedné za rôzne časti programu.

- `SafeStringView` - trieda, ktorá reprezentuje pohľad na reťazec. Zabezpečuje bezpečný prístup k dátam, ktoré sú mimo jej rozsahu. Trieda je založená na `std::string_view` a obsahuje metódy za získanie dĺžky, podreťazca a inštancie `std::string_view` pre ostatné bežné operácie.
- `ClientSettings` - trieda, ktorá obsahuje nastavenia klienta, ako napríklad adresa serveru, port, šifrovanie, cesta k certifikátom, názov schránky, základné nastavenia pre stiahnutie správ a autentizačné údaje.
- `Utils` - trieda, ktorá obsahuje pomocné metódy na ne-kryptografické hashovanie stringov a prevod `uint8_t` buffera na hexadecimálny string.
- `FileStream` - trieda, ktorá je zodpovedná za prácu so súbormi. Trieda obsahuje metódy na otvorenie súboru, zápis dát do súboru, čítanie dát zo súboru a zatvorenie súboru. Rozširuje triedu `Readable` a `Writable`.

- `ConfigFileReader` - trieda, ktorá je zodpovedná za čítanie konfiguračného súboru. Trieda obsahuje metódy na čítanie konfiguračného súboru, získanie hodnôt z konfiguračného súboru a zistenie, či hodnota existuje v konfiguračnom súbore.

3. Testovanie

Všetky testy boli vykonané na vzdialenom serveri s nasledujúcimi špecifikáciami:

- OS: Linux loumadev 5.15.0-101-generic #111-Ubuntu SMP Tue Nov 5 20:16:58 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
- CPU: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
- NIC: Qualcomm Atheros Killer E220x Gigabit Ethernet Controller
- RAM: 32GB DDR4

3.1 IMAP Klient

Pre overenie správneho fungovania klienta boli vykonané testy, ktoré overujú správne fungovanie jednotlivých častí klienta. Testy boli vykonané na testovacom serveri, ktorý bol spustený lokálne na testovacom počítači. Testy boli vykonané na základe zadaní a ich výsledky boli porovnávané s očakávanými výsledkami.

Test vstupných argumentov

- Testovanie správneho spracovania nevalidných vstupných argumentov.

```
$ ./imapcl
> Argument Error: Missing host

$ ./imapcl random.host
> Argument Error: Missing auth file

$ ./imapcl random.host -a invalid.file
> Argument Error: Missing output directory
```

```
$ ./imapcl random.host -a invalid.file -o /tmp
> Client Error: Failed to open file 'invalid.file' (2)

$ ./imapcl random.host -a invalid.file -o /tmp -p 100000
> Argument Error: Invalid port '100000'

$ ./imapcl random.host -a invalid.file -o /tmp -T -c invalid.file
> Client Error: Failed to load certificate file

$ ./imapcl random.host -a auth.ini -o /tmp
> Client Error: Failed to resolve host address 'random.host' (0)

$ ./imapcl 1.1.1.1 -a auth.ini -o /tmp
> Client Error: Connection timed out after 30000ms
```

Test prihlásenia

```
$ ./imapcl imap.zoho.eu -T -a auth.incorrect.zoho.ini -o /tmp
> Client Error: Request failed ([AUTHENTICATIONFAILED] Invalid credentials(Failure))
```

Test výberu schránky

```
$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -b INVALID
> Client Error: Request failed ([TRYCREATE] Folder INVALID does not exist)

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -b INBOX
> Downloaded 6 messages from mailbox INBOX.
```

Test stiahnutia správ

```
$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp
> Downloaded 6 messages from mailbox INBOX.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -b INBOX
> Downloaded 6 messages from mailbox INBOX.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -b SPAM
> Downloaded 1 message from mailbox SPAM.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -h
> Downloaded 6 message headers from mailbox INBOX.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -n
> No new messages to download from mailbox INBOX.
```

- Po prijatí novej správy:
- (agument `-h` nemení príznak `\Recent`)

```
$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -h -n
> Downloaded 1 new message header from mailbox INBOX.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -h -n
> Downloaded 1 new message header from mailbox INBOX.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -n
> Downloaded 1 new message from mailbox INBOX.

$ ./imapcl imap.zoho.eu -T -a auth.zoho.ini -o /tmp -n
> No new messages to download from mailbox INBOX.
```

Test prístupu bez TLS

```
$ ./imapcl imap.centrum.sk -a auth.centrum.ini -o /tmp
> Downloaded 4 messages from mailbox INBOX.
```

Test vlastného certifikátu

- Vygenerovanie certifikátu a následné spustenie OpenSSL serveru na porte `1234`:

```
openssl s_server -accept 1234 -key key.pem -cert cert.pem
> ACCEPT
```

- Spustenie klienta s vlastným certifikátom:

```
imapcl localhost -p 1234 -a auth.ini -o /tmp -T -c cert.pem
```

- OpenSSL server prijíma:

```
> a1 LOGIN "<username>" "<password>"
```

3.2 Hashovacia funckia

Projekt obsahuje vlastnú implementáciu hashovacej funckie, ktorá síce nemusí byť kryptograficky bezpečná, ale musí mať dostatočný rozptyl distribúcie generovaných hashov. Pre overenie správneho fungovania hashovacej funkcie boli vykonané testy, ktoré kontrolujú počet kolízií pri rôznych vstupoch.

Keďže hashovacia funckia je použitá pre hashovanie skôr kratších vstupov, testy boli prevádzané na reťazcoch s dĺžkou 16 až 64 znakov.

Testy ukázali, že daná hashovacia funkcia je dostatočná na požadovanú úlohu.

Test kolízií v náhodnom reťazci

- Pre testovanie bol zvolený náhodný reťazec dĺžky 16 až 64 znakov, s obsahom hodnot 32 až 126 (ASCII hodnoty).
- Opakovanie testu: 10M krát.
- Nájdených kolízií: 0

Test kolízií v náhodnom reťazci so spoločnými prefixami

- Pri testovaní sa vygeneruje náhodný reťazec - prefix, dĺžky 1 až 16 znakov, v rozsahu ASCII hodnôt, následne sa generuje zvyšných 16 až 48 znakov náhodne, rovnako v rozsahu ASCII hodnôt.
- Opakovanie testu: pre 3000 rôznych prefixov, 3000 náhodných reťazcov. (9M testov)
- Nájdených kolízií: 0

Literatúra

1. RFC 3501 - INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1
2. RFC 5322 - Internet Message Format
3. C++ Reference
4. OpenSSL 3.4 Documentation
5. IANA: Service Name and Transport Protocol Port Number Registry
6. Zoho Mail: IMAP Access