



# IMAP klient s podporou TLS

## Manuál

Autor: Adam Ližičiar (xlizic00)

18. novembra 2024

# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
<b>2</b>	<b>Hlbšia analýza problematiky</b>	<b>1</b>
<b>3</b>	<b>Používanie programu</b>	<b>1</b>
<b>4</b>	<b>Architektúra programu</b>	<b>2</b>
4.1	Hlavné riadenie programu . . . . .	2
4.2	Triedy a ich funkcionality . . . . .	2
4.3	Komunikácia klienta so serverom . . . . .	4
4.4	Spracovanie chýb a chybové kódy . . . . .	4
<b>5</b>	<b>Testovanie</b>	<b>5</b>
5.1	Jednotkové testy . . . . .	5
5.2	Systémové testy . . . . .	5
5.3	Testy pamäte . . . . .	5
<b>6</b>	<b>Literatúra</b>	<b>9</b>
<b>A</b>	<b>Prílohy</b>	<b>10</b>
A.1	Diagram prípadov použitia . . . . .	10
A.2	Diagram tried . . . . .	11
A.3	Diagram sekvencie . . . . .	12

# 1 Úvod

Hlavným zmyslom tohto programu je umožniť používateľovi čítanie emailov zo svojej emailovej schránky pomocou IMAP(S) protokolu s možnou podporou šifrovacieho protokolu TLS.

## 2 Hlbšia analýza problematiky

IMAP (Internet Message Access Protocol)[2] je protokol, ktorý sa používa na vzdialený prístup k e-mailovým správam uloženým na serveri. Je to jeden z najpoužívanejších protokolov na čítanie e-mailov a bol vyvinutý s cieľom poskytnúť synchronizovaný prístup k e-mailovým schránkam. Na rozdiel od POP3 (Post Office Protocol 3), ktorý sťahuje správy do počítača a maže ich zo servera, IMAP umožňuje užívateľom spravovať správy na serveri bez ich sťahovania, čím sa zabezpečuje prístup k e-mailu z viacerých zariadení.

IMAP protokol vykonáva textové príkazy cez sériu príkazov, v ktorej klient posiela požiadavku serveru, napríklad na získanie zoznamu správ alebo stiahnutie konkrétnej správy. Server následne odpovie s požadovanými informáciami. Tento protokol má tiež funkcie, ako je označovanie správ (napr. neprečítané, vymazané), filtrovanie a vyhľadávanie správ na serveri.

IMAPS (IMAP over SSL/TLS) je verzia IMAP protokolu, ktorá zahŕňa šifrovanie komunikácie medzi klientom a serverom prostredníctvom protokolu TLS (Transport Layer Security). TLS[3] je šifrovací protokol, ktorý zabezpečuje bezpečnú komunikáciu. Cieľom TLS je zabezpečiť, aby údaje odosielané medzi dvoma zariadeniami neboli odpočúvané, zneužívané alebo zmenené.

Pri použití IMAPS sa najprv nadviaže obyčajné (nešifrované) spojenie, a až potom sa nastaví bezpečnostné parametre na šifrovanie prenosu údajov.

IMAP(S) protokol využíva sériu príkazov, ktoré klient posiela serveru na vykonanie rôznych operácií s e-mailovými správami. Medzi základné príkazy patria: `SELECT` na výber konkrétnej poštovej schránky, `EXAMINE` na získanie informácií o poštovej schránke bez jej výberu, `FETCH` na stiahnutie konkrétnej správy alebo časti správy, `STORE` na zmenu atribútov správy (napríklad označenie ako prečítané alebo odstránené), `SEARCH` na vyhľadávanie správ podľa určitých kritérií, `LIST` a `LSUB` na získanie zoznamu poštových schránok, a `LOGOUT` na ukončenie spojenia.

## 3 Používanie programu

Program sa spúšťa s nasledujúcimi parametrami:

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

Poradie parametrov je ľubovoľné. Popis parametrov je nasledovný:

- Povinný je názov servera (IP adresa alebo doménové meno) požadovaného zdroja.
- Voliteľný parameter `-p port` špecifikuje číslo portu na serveri.
- Parameter `-T` zapína šifrovanie (`imaps`), ak nie je uvedený, použije sa nešifrovaná verzia protokolu.
- Voliteľný parameter `-c` označuje súbor s certifikátmi, ktorý sa použije na overenie platnosti certifikátu SSL/TLS predloženého serverom.
- Voliteľný parameter `-C` určuje adresár, v ktorom sa majú vyhľadávať certifikáty na overenie platnosti certifikátu SSL/TLS predloženého serverom. Výchozia hodnota je `/etc/ssl/certs`.

- Pri použití parametra `-n` sa bude pracovať iba s novými správami.
- Pri použití parametra `-h` sa budú sťahovať iba hlavičky správ.
- Povinný parameter `-a auth_file` odkazuje na súbor s autentifikačnými údajmi.
- Parameter `-b` špecifikuje názov schránky, s ktorou sa bude na serveri pracovať. Výchozia hodnota je `INBOX`.
- Povinný parameter `-o out_dir` určuje výstupný adresár, do ktorého má program uložiť stiahnuté správy.
- Parameter `-i` spustí interaktívny režim (príkazy `DONWLOADALL`, `DONWLOADNEW`, `QUIT`).

## 4 Architektúra programu

Program je navrhnutý ako modulárny systém s využitím objektovo-orientovaného prístupu. Architektúra zahŕňa viacero tried, z ktorých má každá jasne definovanú úlohu, čím je zabezpečená prehľadnosť a jednoduchá rozširiteľnosť. Hlavné riadenie programu je implementované prostredníctvom konečného stavového automatu (viď A.1), ktorý spravuje prechod medzi stavmi, ako sú autentifikácia, výber schránky či sťahovanie správ.

Komunikácia medzi klientom a serverom prebieha podľa protokolu IMAP(S).

Pre zabezpečenie stability programu je implementovaný systém spracovania chýb. Využíva sa mechanizmus výnimiek, ktorý identifikuje rôzne druhy chýb s vlastnými chybovými kódmi.

Pri vytváraní niektorých komentárov v kódach a pri doplnení niektorých testov bol použitý nástroj Gemini od firmy Google kvôli zahrnutiu čo najväčšieho spektra rôznych testov, ktorým je program podrobený. Tento nástroj bol použitý výlučne iba na tieto dve veci a nie na tvorbu architektúry alebo funkcií v programe.

### 4.1 Hlavné riadenie programu

Hlavné riadenie programu prebieha pomocou diagramu prípadov použitia, ktorý je implementovaný v súboroch `src/classes/FiniteStateMachine/FiniteStateMachine.{cpp, hpp}`. Samotné ovládanie pomocou tohto diagramu je implementované v súbore `src/classes/IMAPClient.cpp` vďaka `while` cyklu. Vo `while` cykle sú použité podmienky `if-else` namiesto `switch-u` kvôli lepšej prehľadnosti kódu. Daný diagram je možné vidieť v prílohe A.1.

### 4.2 Triedy a ich funkcionality

Grafické zobrazenie diagramu tried je možné vidieť v prílohe A.2. Program sa skladá zo siedmich tried, ktoré majú nasledujúcu funkcionality:

- **FiniteStateMachine**  
Trieda riadi tok programu pomocou konečného stavového automatu (FSM). Obsahuje stavy programu a metódy na ich prechod, a to konkrétne stavy `INIT`, `AUTH`, `SELECT`, `DOWNLOAD`, `QUIT` a `END` (viď A.1).
- **AuthManager**  
Spravuje autentifikáciu používateľa. Zabezpečuje získanie prihlasovacích údajov a ich správne uloženie do tejto triedy. Autentifikačný súbor musí mať formát `username = xxx` a na ďalšom riadku `password = yyy`. Pri neexistujúcom súbore skončí program vyvolaním výnimky a pri nesprávnych údajoch skončí program s nulovým návratovým kódom a oznámením, že sa nepodarilo overiť identitu na strane servera.
- **ArgsParser**

```

while FSM.getState() ≠ END do
  if FSM.getState() == INIT then
    | FSM.transitionToAuth();
  else if FSM.getState() == AUTH then
    | if loginSuccessful() then
    | | FSM.transitionToSelect();
    | end
    | else
    | | FSM.transitionToEnd();
    | end
  else if FSM.getState() == SELECT then
    | if mailboxExists() then
    | | FSM.transitionToDownload();
    | end
    | else
    | | FSM.transitionToQuit();
    | end
  else if FSM.getState() == DOWNLOAD then
    | downloadEmails();
    | FSM.transitionToQuit();
  else if FSM.getState() == QUIT then
    | logout();
    | FSM.transitionToEnd();
  else
    | throwError();
  end
end

```

**Algoritmus 1:** Zjednodušený pseudokód FSM cyklu implementovaného v triede `IMAPClient`.

Zodpovedá za spracovanie argumentov príkazového riadku. Umožňuje používateľovi zadať konfiguráciu, ako je názov servera, portu, a ďalšie možnosti.

- **Message**

Reprezentuje e-mailovú správu. Obsahuje údaje ako odosielateľ, predmet, telo správy a prílohy, pričom umožňuje manipuláciu s týmito údajmi. Táto trieda obsahuje funkciu `decodeMime()`, ktorej tvorba bola inšpirovaná funkciami z knižnice `mimetic`. Funkcia dokáže zmeniť niektoré druhy MIME textov, aby boli prehľadné pre používateľa tohto programu.

- **MessageFactory**

Zodpovedá za vytváranie inštancií správ. Prijíma surové dáta zo servera a transformuje ich na objekt typu 'Message'. Po spracovaní týchto dát je každá e-mailová správa uložená do vlastného súboru, ktorý má názov vo formáte `SUBJECT_IDMESSAGE.txt`. Tento názov bol zvolený pretože je jedinečný pre akékoľvek emailové správy (vd'aka `IDMESSAGE`) a zároveň je prehľadný kvôli použitiu predmetu správy v názve. V tejto triede taktiež prebieha obsluha lokálneho súboru, ktorý obsahuje potrebné údaje pre správnu synchronizáciu dát. Po stiahnutí prvých správ vznikne v priečinku s emailami pod názvom `imapcl.log`.

- **IMAPConnection**

Spravuje spojenie so serverom pomocou protokolu IMAP(S). Umožňuje odosielať príkazy na server,

prijímať odpovede a spracúvať komunikáciu[1]. V tejto triede je vo funkcii na prijatie odpovedí zabudovaný časovač, ktorý má dĺžku 10 sekúnd a slúži na ukončenie programu v prípade, že by server prestal odpovedať.

- **IMAPClient**

Hlavná trieda, ktorá integruje všetky ostatné komponenty. Implementuje logiku programu na riadenie stavového automatu, vykonáva autentifikáciu a sťahovanie e-mailových správ.

### 4.3 Komunikácia klienta so serverom

V rámci implementácie triedy `IMAPClient` prebieha komunikácia klienta so serverom v jednotlivých krokoch (viď. sekvenčný diagram v prílohe A.3), ktoré zodpovedajú stavovému diagramu FSM (viď A.1). Medzi kroky použité v tomto programe komunikácie patria:

- **Prihlásenie (LOGIN)**

Klient odošle príkaz `LOGIN` s prihlasovacími údajmi (meno používateľa a heslo). Server následne odpovie, či bola autentifikácia úspešná (OK), alebo zlyhala.

- **Výber schránky (SELECT)**

Klient odošle príkaz `SELECT` s názvom e-mailovej schránky, z ktorej chce používateľ čítať e-mailovú poštu. Server odpovie, či je schránka dostupná. V prípade sťahovania neprečítaných správ je použitý príkaz `UNSEEN`, pretože stiahnutie ešte nepredpokladá, že danú správu používateľ aj naozaj prečíta, a preto ju nechá označenú ako neprečítanú.

- **Vyhľadávanie správ (SEARCH)**

Klient pomocou príkazu `SEARCH` vyhľadá správy podľa zadanych kritérií (napr. neprečítané správy, všetky správy). Server odpovie zoznamom UID správ, ktoré vyhovujú podmienkam.

- **Sťahovanie správ (FETCH)**

Po získaní UID správ klient odošle príkaz `FETCH`, aby stiahol obsah správ. Požaduje buď len hlavičky (`BODY.PEEK[HEADER]`) alebo celé telo správ (`BODY[]`). Stiahnuté správy sa spracujú a následne lokálne uložia.

- **Odhlásenie (LOGOUT)**

Po ukončení práce klient odošle príkaz `LOGOUT`, čím uzavrie spojenie so serverom. Server odpovie potvrdením ukončenia komunikácie.

Každý krok komunikácie je realizovaný prostredníctvom odoslania príkazov serveru a spracovania odpovedí. Tieto kroky zaisťujú správnu autentifikáciu, manipuláciu s e-mailovými schránkami a správami, pričom implementácia umožňuje spravovať dáta efektívne a bezpečne.

### 4.4 Spracovanie chýb a chybové kódy

Program využíva `try-catch` na spracovanie výnimiek, ktorý umožňuje identifikovať rôzne druhy chýb vznikajúcich počas behu programu. Každý typ chyby je zachytený samostatným blokom `catch`, pričom pre každú výnimku je definovaný vlastný chybový kód a správa. Každá zachytená výnimka je spracovaná s výpisom do štandardného chybového výstupu (`stderr`).

- **IMAPException**

Najvšeobecnejší druh chyby. Chybový kód: 2.

- **ArgumentsException**

Nastáva pri chybných argumentoch príkazového riadka. Chybový kód: 10.

- **AuthenticateException**

Označuje zlyhanie autentifikácie. Chybový kód: 11.

- **FileException**

Indikuje chyby spojené s internými súbormi. Chybový kód: 12.

- **ConnectionException**

Vyskytuje sa pri všetkých problémoch so sieťovým spojením, ktoré nespádajú do kategórie `BIOException` a `SSLException`. Chybový kód: 20.

- **SSLException**

Vzniká pri chybách v SSL spojení. Chybový kód: 21.

- **BIOException**

Nastáva pri chybách v nezabezpečenej komunikácii. Chybový kód: 22.

- **CommandException**

Signalizuje problém s príkazom prijatým od serveru. Chybový kód: 23.

- **MailboxException**

Reprezentuje problémy pri manipulácii s poštovými schránkami. Chybový kód: 30.

- **std::exception**

Zastrešuje všetky ostatné výnimky. Chybový kód: 1.

## 5 Testovanie

### 5.1 Jednotkové testy

Každá trieda obsahuje vlastné jednotkové testy, ktoré sú uložené v súboroch v priečinku `test/unit/...`. Dané testy sú spustené vďaka GitHub workflow-u pri každom nahratí novej verzie, a teda akákoľvek vzniknutá chyba je hneď nájdená (príklad jednotkových testov je na obrázku číslo 1).

### 5.2 Systémové testy

Systémové testy prebehli manuálnym overovaním pripojenia k serveru. Pomocou nástroja Wireshark bola analyzovaná sieťová komunikácia, vrátane TLS paketov. Na priloženom obrázku nižšie je zobrazený príklad zachyteného TLS paketu, ktorý zobrazuje zabezpečené spojenie s poštovým serverom.

Ďalším príkladom na inom obrázku je spustenie programu a overenie stiahnutia dvoch správ.

Pre účely detailnejšieho debugovania je k dispozícii debug verzia programu, ktorú je možné vygenerovať pomocou príkazu `make debug`. Výsledný spúšťateľný súbor `imapcl_debug` poskytuje podrobné informácie o priebehu programu, ako je tiež možné vidieť na ostatnom obrázku.

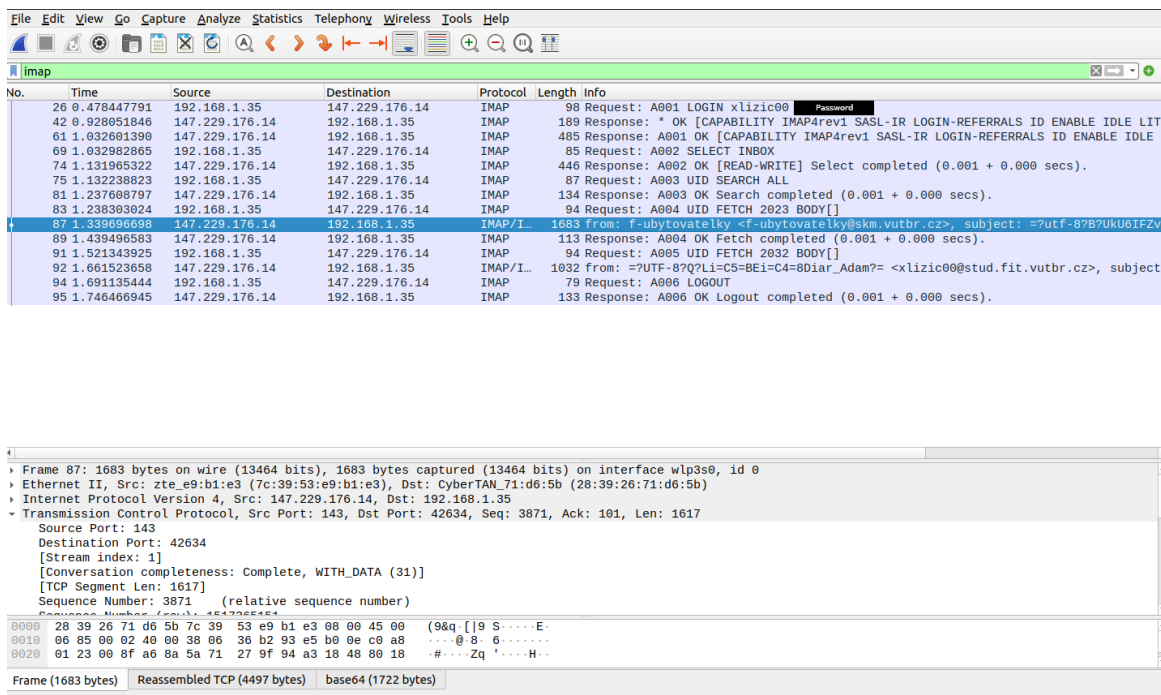
### 5.3 Testy pamäte

Sú vykonávané vďaka nástroju Valgrind a GitHub workflow-u. Ten pri každom nahratí novej verzie vykoná príkaz `make valgrind`, ktorý spustí Valgrind analýzu. Následne prebehne kontrola, ktorá zaistí, či výsledný súbor obsahuje reťazec `ERROR SUMMARY: 0 errors from 0 contexts`, teda či je všetka práca s pamäťou správna. Výstup valgrind testov je možné vidieť na obrázku nižšie.

```
Run Tests 6s

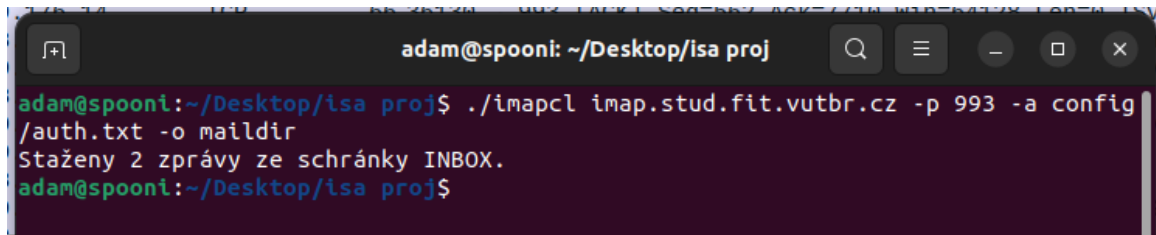
1 ▶ Run make test
4 g++ -std=c++20 -Wall -Wextra -Werror src/main.cpp -o imapcl -lssl -lcrypto
5 g++ test/unit/_init_.cpp -o test/unit/run -lgtest -lgtest_main -lssl -lcrypto
6 ./test/unit/run
7 [=====] Running 36 tests from 5 test suites.
8 [-----] Global test environment set-up.
9 [-----] 10 tests from ExceptionTest
10 [ RUN      ] ExceptionTest.ArgumentsExceptionTest
11 [       OK ] ExceptionTest.ArgumentsExceptionTest (0 ms)
12 [ RUN      ] ExceptionTest.AuthenticateExceptionTest
13 [       OK ] ExceptionTest.AuthenticateExceptionTest (0 ms)
14 [ RUN      ] ExceptionTest.BIOExceptionTest
15 [       OK ] ExceptionTest.BIOExceptionTest (0 ms)
16 [ RUN      ] ExceptionTest.CommandExceptionTest
17 [       OK ] ExceptionTest.CommandExceptionTest (0 ms)
18 [ RUN      ] ExceptionTest.ConnectionExceptionTest
19 [       OK ] ExceptionTest.ConnectionExceptionTest (0 ms)
20 [ RUN      ] ExceptionTest.FileExceptionTest
21 [       OK ] ExceptionTest.FileExceptionTest (0 ms)
22 [ RUN      ] ExceptionTest.MailboxExceptionTest
23 [       OK ] ExceptionTest.MailboxExceptionTest (0 ms)
24 [ RUN      ] ExceptionTest.SSLEnvironmentTest
25 [       OK ] ExceptionTest.SSLEnvironmentTest (0 ms)
26 [ RUN      ] ExceptionTest.IMAPExceptionTest
27 [       OK ] ExceptionTest.IMAPExceptionTest (0 ms)
28 [ RUN      ] ExceptionTest.IMAPExceptionTestOwnReturnCode
29 [       OK ] ExceptionTest.IMAPExceptionTestOwnReturnCode (0 ms)
30 [-----] 10 tests from ExceptionTest (0 ms total)
31
32 [-----] 9 tests from ArgsParserTest
33 [ RUN      ] ArgsParserTest.ValidMinimalArguments
```

Obr. 1: Zobrazenie jednotkových testov na GitHube.



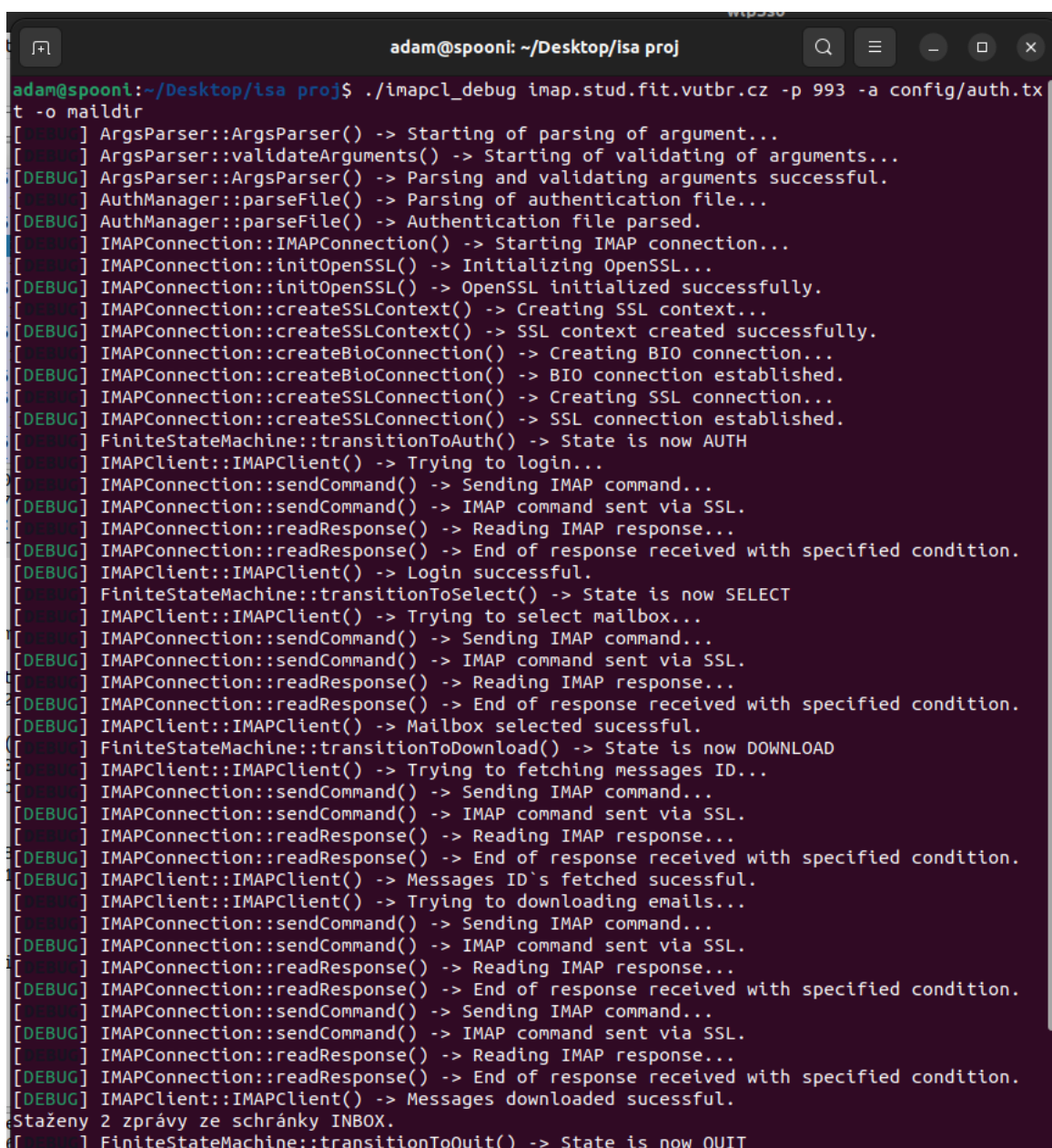
Obr. 2: Príklad nezabezpečenej IMAP komunikácie v programe Wireshark.





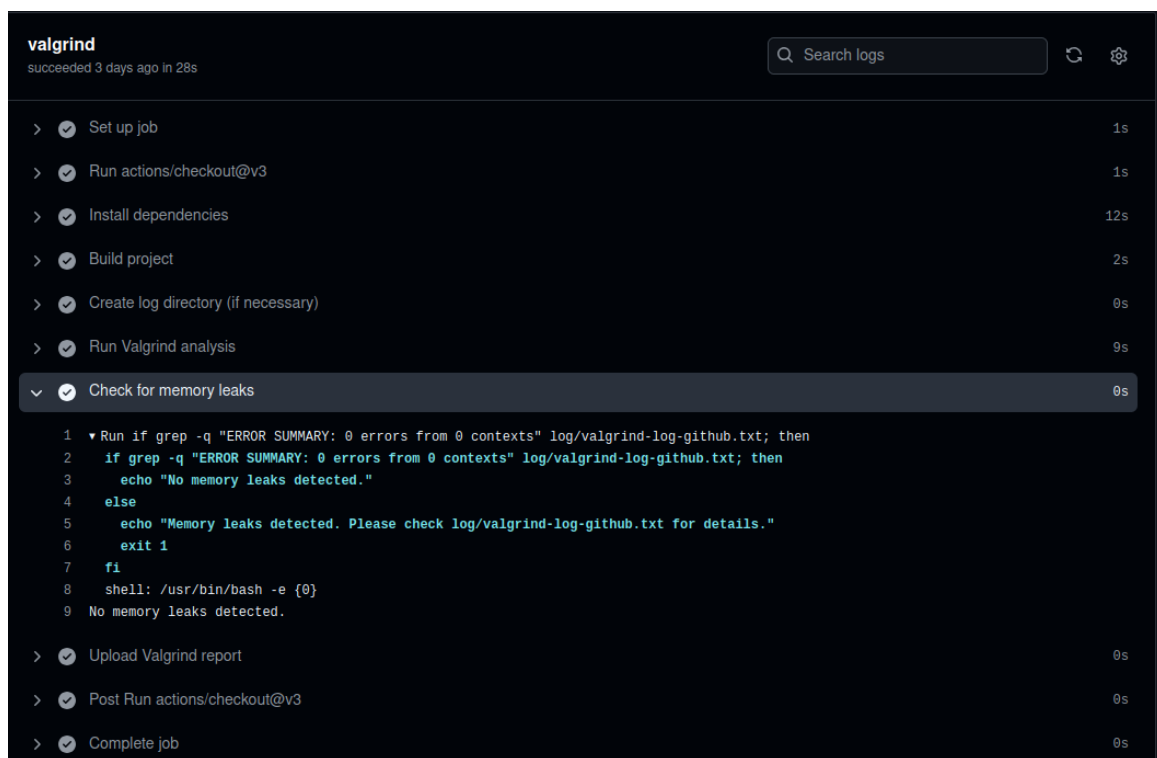
```
adam@spooni: ~/Desktop/isa proj
adam@spooni:~/Desktop/isa proj$ ./imapcl imap.stud.fit.vutbr.cz -p 993 -a config/auth.txt -o maildir
Stáženy 2 zprávy ze schránky INBOX.
adam@spooni:~/Desktop/isa proj$
```

Obr. 3: Obyčejné spustenie programu.



```
adam@spooni: ~/Desktop/isa proj
adam@spooni:~/Desktop/isa proj$ ./imapcl_debug imap.stud.fit.vutbr.cz -p 993 -a config/auth.txt -o maildir
[DEBUG] ArgsParser::ArgsParser() -> Starting of parsing of argument...
[DEBUG] ArgsParser::validateArguments() -> Starting of validating of arguments...
[DEBUG] ArgsParser::ArgsParser() -> Parsing and validating arguments successful.
[DEBUG] AuthManager::parseFile() -> Parsing of authentication file...
[DEBUG] AuthManager::parseFile() -> Authentication file parsed.
[DEBUG] IMAPConnection::IMAPConnection() -> Starting IMAP connection...
[DEBUG] IMAPConnection::initOpenSSL() -> Initializing OpenSSL...
[DEBUG] IMAPConnection::initOpenSSL() -> OpenSSL initialized successfully.
[DEBUG] IMAPConnection::createSSLContext() -> Creating SSL context...
[DEBUG] IMAPConnection::createSSLContext() -> SSL context created successfully.
[DEBUG] IMAPConnection::createBioConnection() -> Creating BIO connection...
[DEBUG] IMAPConnection::createBioConnection() -> BIO connection established.
[DEBUG] IMAPConnection::createSSLConnection() -> Creating SSL connection...
[DEBUG] IMAPConnection::createSSLConnection() -> SSL connection established.
[DEBUG] FiniteStateMachine::transitionToAuth() -> State is now AUTH
[DEBUG] IMAPClient::IMAPClient() -> Trying to login...
[DEBUG] IMAPConnection::sendCommand() -> Sending IMAP command...
[DEBUG] IMAPConnection::sendCommand() -> IMAP command sent via SSL.
[DEBUG] IMAPConnection::readResponse() -> Reading IMAP response...
[DEBUG] IMAPConnection::readResponse() -> End of response received with specified condition.
[DEBUG] IMAPClient::IMAPClient() -> Login successful.
[DEBUG] FiniteStateMachine::transitionToSelect() -> State is now SELECT
[DEBUG] IMAPClient::IMAPClient() -> Trying to select mailbox...
[DEBUG] IMAPConnection::sendCommand() -> Sending IMAP command...
[DEBUG] IMAPConnection::sendCommand() -> IMAP command sent via SSL.
[DEBUG] IMAPConnection::readResponse() -> Reading IMAP response...
[DEBUG] IMAPConnection::readResponse() -> End of response received with specified condition.
[DEBUG] IMAPClient::IMAPClient() -> Mailbox selected successful.
[DEBUG] FiniteStateMachine::transitionToDownload() -> State is now DOWNLOAD
[DEBUG] IMAPClient::IMAPClient() -> Trying to fetching messages ID...
[DEBUG] IMAPConnection::sendCommand() -> Sending IMAP command...
[DEBUG] IMAPConnection::sendCommand() -> IMAP command sent via SSL.
[DEBUG] IMAPConnection::readResponse() -> Reading IMAP response...
[DEBUG] IMAPConnection::readResponse() -> End of response received with specified condition.
[DEBUG] IMAPClient::IMAPClient() -> Messages ID's fetched successful.
[DEBUG] IMAPClient::IMAPClient() -> Trying to downloading emails...
[DEBUG] IMAPConnection::sendCommand() -> Sending IMAP command...
[DEBUG] IMAPConnection::sendCommand() -> IMAP command sent via SSL.
[DEBUG] IMAPConnection::readResponse() -> Reading IMAP response...
[DEBUG] IMAPConnection::readResponse() -> End of response received with specified condition.
[DEBUG] IMAPConnection::sendCommand() -> Sending IMAP command...
[DEBUG] IMAPConnection::sendCommand() -> IMAP command sent via SSL.
[DEBUG] IMAPConnection::readResponse() -> Reading IMAP response...
[DEBUG] IMAPConnection::readResponse() -> End of response received with specified condition.
[DEBUG] IMAPClient::IMAPClient() -> Messages downloaded successful.
Stáženy 2 zprávy ze schránky INBOX.
[DEBUG] FiniteStateMachine::transitionToQuit() -> State is now QUIT
```

Obr. 4: Spustenie debugovacích informácií.



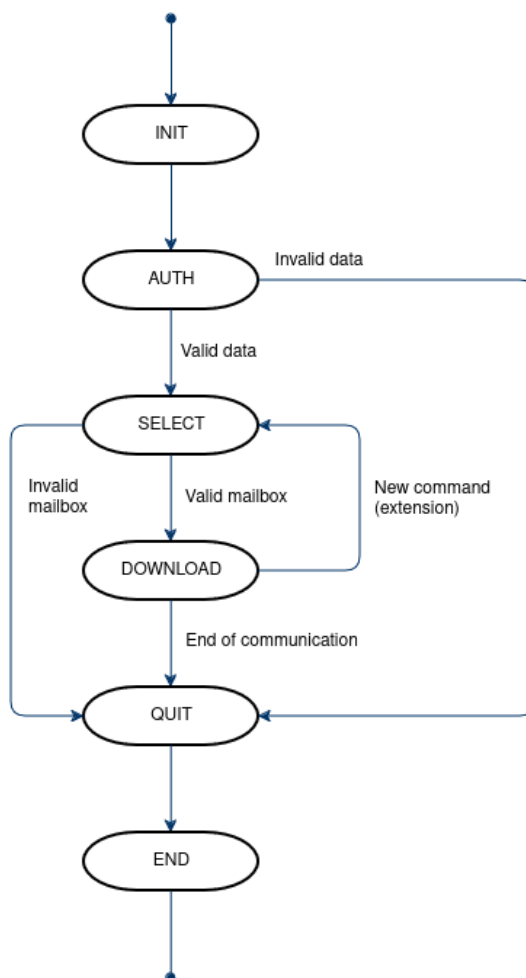
Obr. 5: Zobrazenie výstupu valgrind testu na GitHube.

## 6 Literatura

- [1] Ballard, K.: Secure programming with the OpenSSL API. 2018, [Navštívené 17.11.2024].
- [2] Crispin, M.: INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501, Březen 2003, doi:10.17487/RFC3501. Dostupné z: <https://www.rfc-editor.org/info/rfc3501>
- [3] Rescorla, E.; Dierks, T.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Srpen 2008, doi:10.17487/RFC5246. Dostupné z: <https://www.rfc-editor.org/info/rfc5246>

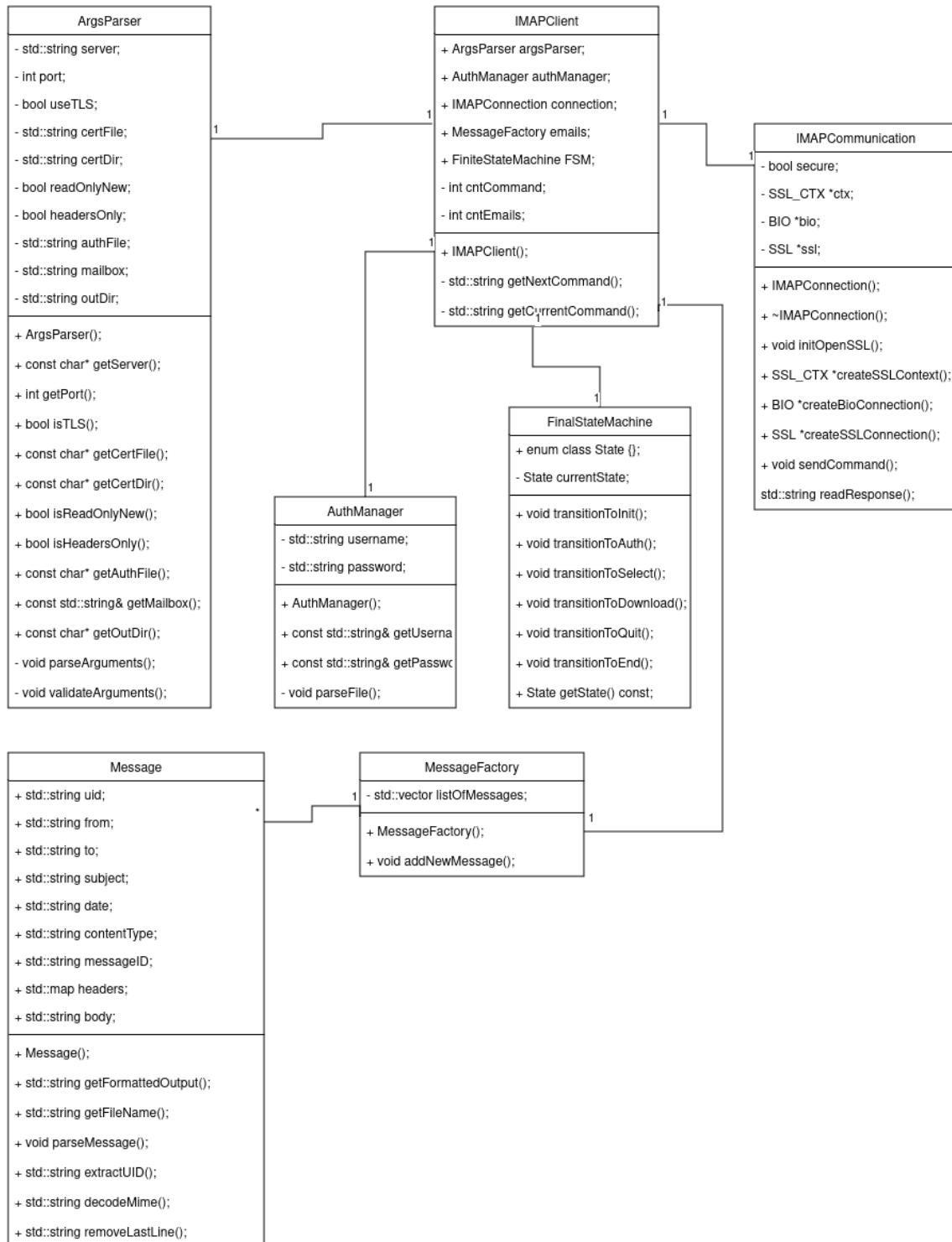
## A Prílohy

### A.1 Diagram prípadov použitia



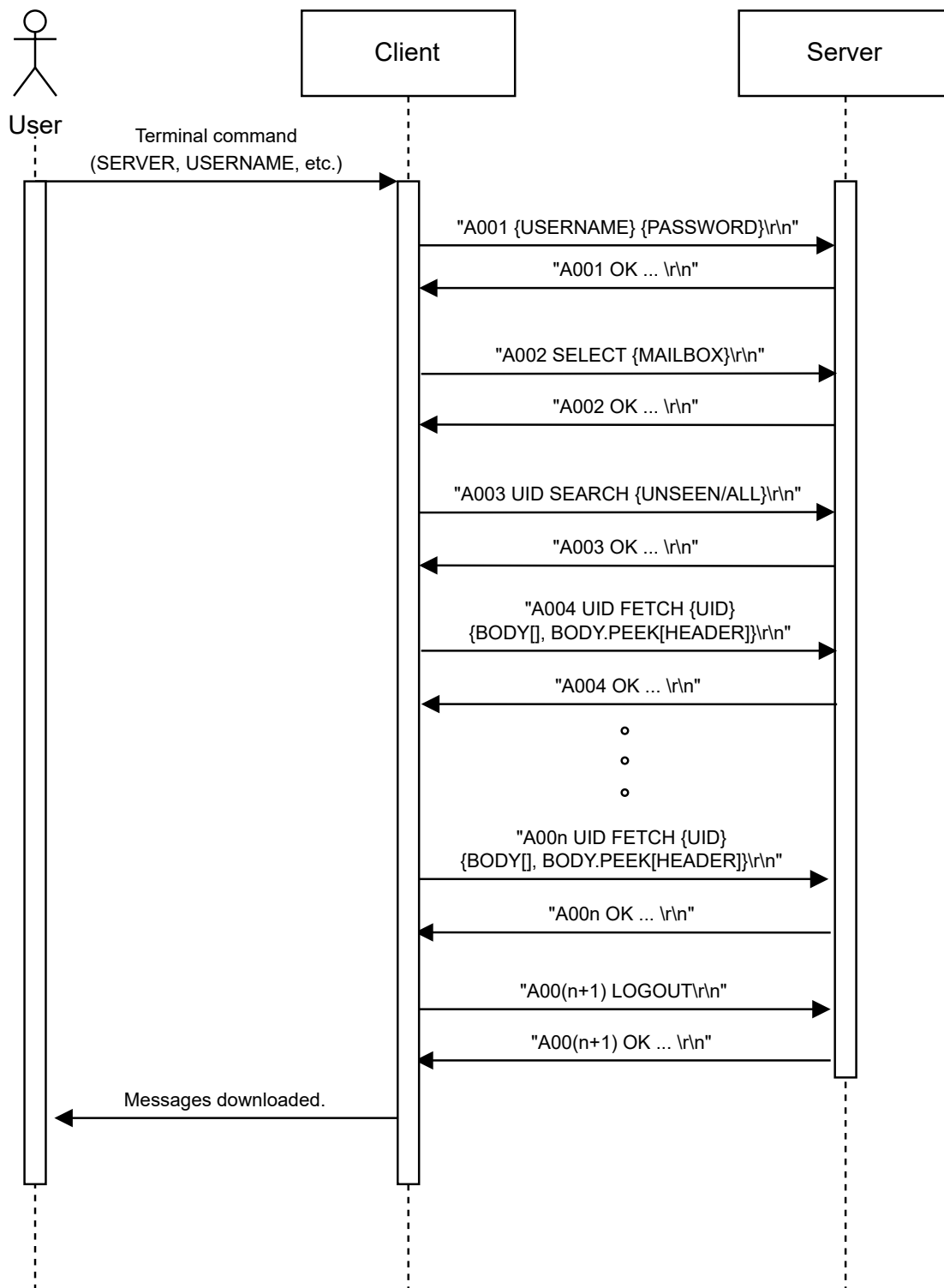
Obr. 6: Diagram prípadov použitia

## A.2 Diagram tried



Obr. 7: Diagram tried

### A.3 Diagram sekvencie



Obr. 8: Diagram sekvencie: jednoduchá komunikácia používateľa so serverom pomocou klienta.