VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií

Technická zpráva

Klient IMAP s podporou TLS

18. listopadu 2024 Marek Tenora, xtenor02

Obsah

1 Úvod

Elektronická pošta je jedním z nejrozšířenějších způsobů komunikace na internetu. Tento projekt implementuje klienta pro protokol IMAP s podporou TLS, který umožňuje bezpečné stahování zpráv.

1.1 Protokol IMAP

Protokol **IMAP** (*Internet Message Access Protocol*), definovaný v **RFC 3501**, je standardní protokol pro přístup k elektronické poště na vzdáleném serveru. Umožňuje klientům manipulovat s e-maily přímo na serveru bez nutnosti jejich stahování do lokálního úložiště. To zahrnuje operace jako čtení, mazání, přesouvání zpráv a práci s různými mailboxy. IMAP poskytuje flexibilní způsob správy pošty, což je zvláště užitečné pro uživatele přistupující k poště z více zařízení. Klient pracuje s aktuální verzí IMAP v4. [3]

1.2 Formát e-mailových zpráv

Formát e-mailových zpráv je definován v **RFC 5322**, který specifikuje strukturu hlaviček a těla zprávy. Tento standard určuje, jak mají být e-maily formátovány, aby byly správně interpretovány různými klienty a servery. Hlavičky obsahují informace jako odesílatel, příjemce, předmět, datum odeslání a mnoho dalších, zatímco tělo obsahuje samotný obsah zprávy. [4]

1.3 Bezpečnost komunikace

Bezpečnost komunikace je zajištěna pomocí protokolu **TLS** (*Transport Layer Security*), který poskytuje šifrování dat přenášených mezi klientem a serverem. TLS zajišťuje důvěrnost a integritu přenášených informací, což je klíčové zejména při přenosu citlivých údajů, jako jsou přihlašovací informace nebo obsah e-mailových zpráv. Implementace TLS umožňuje odolat útokům, jako je odposlouchávání nebo podvržení zpráv. [1]

Pro implementaci podpory TLS v aplikaci je využita knihovna **OpenSSL**, která poskytuje nástroje pro práci s kryptografickými protokoly a algoritmy. OpenSSL nabízí širokou škálu funkcí pro šifrování, dešifrování a správu certifikátů, což usnadňuje integraci bezpečnostních funkcí do aplikace. Díky OpenSSL může aplikace podporovat různé verze TLS a zajistit kompatibilitu s většinou IMAP serverů. [2]

1.4 Cíle projektu

Cílem tohoto projektu je vytvořit jednoduchou CLI aplikaci, která umožní uživatelům bezpečně přistupovat k jejich e-mailovým schránkám prostřednictvím protokolu IMAP s podporou TLS. Aplikace podporuje více e-mailových účtů a nabízí základní funkce pro manipulaci s e-maily, jako je stahování zpráv, práce s poštovními schránkami a možnost stahovat pouze nové nebo pouze hlavičky zpráv.

2 Návod na použití

Tato kapitola poskytuje podrobné pokyny pro instalaci, kompilaci a spuštění aplikace pomocí `Makefile`. Projekt zahrnuje dva soubory `Makefile`:

- hlavní `Makefile` pro standardní sestavení aplikace,
- testovací `test_Makefile` pro spuštění testů.

Před použitím aplikace zkontrolujte, zda splňujete následující požadavky:

2.1 Požadavky a předpoklady

Pro úspěšné sestavení a spuštění aplikace je nutné mít nainstalované následující nástroje:

- **GCC a G++** – aplikace je napsaná v jazyce C++, a proto je nutné mít nainstalovaný kompilátor GCC nebo G++,
- **Make** – nástroj pro automatizaci sestavení, který je standardně předinstalován na většině distribucí Linuxu a macOS,
- **OpenSSL knihovna** – aplikace vyžaduje knihovny `libssl` a `libcrypto` pro zajištění bezpečné komunikace,
- **GoogleTest knihovna** (volitelné) – pro spuštění testů.

2.2 Rychlý start

Tato část poskytuje rychlý přehled kroků potřebných pro sestavení a spuštění aplikace. Postupujte podle následujících příkazů:

```
# Sestavení aplikace
make

# Spuštění aplikace
./imapcl <server_address> -a <auth_file> -o <output_directory>

# Spuštění testů
make -f test_Makefile run_tests
```

Základní použití aplikace vyžaduje zadání adresy serveru, autentizačního souboru a výstupního adresáře, kde se mají ukládat e-maily.

2.3 Sestavení aplikace

Pro sestavení produkční verze aplikace spusťte v terminálu následující příkaz:

```
make
```

Tento příkaz zkompiluje všechny zdrojové soubory z adresáře `src/` a vytvoří spustitelný soubor `imapcl` v adresáři `bin/`.

Pokud chcete sestavit aplikaci v ladicím režimu, použijte příkaz:

```
make debug
```

Tento příkaz přidá příznak `-g` pro ladění a vytvoří soubor `imapcl_debug` v adresáři `bin/`.

Pro odstranění všech vygenerovaných souborů a vyčištění projektu použijte příkaz:

```
make clean
```

Tento příkaz odstraní obsah adresářů `obj/` a `bin/` a vrátí projekt do stavu před kompilací.

2.4 Použití aplikace

Aplikace je jednoduchý CLI nástroj pro připojení k IMAP serveru a manipulaci s e-maily.
Argumenty se zadávají v příkazové řádce následujícím způsobem:

```
./imapcl <server_address> [argumenty]
```

2.4.1 Povinné argumenty

- `<server_address>` – adresa IMAP serveru, ke kterému se připojujete,
- `-a <auth_file>` – cesta k autentizačnímu souboru, který obsahuje přihlašovací údaje,
- `-o <output_directory>` – cesta k adresáři, kam se budou ukládat stažené e-maily.

2.4.2 Volitelné argumenty

- `-p <port>` – specifikace portu. Výchozí hodnoty jsou 143 pro nešifrované připojení a 993 pro TLS,
- `-T` – použití protokolu TLS pro zabezpečení připojení,
  Následující parametry se používají pouze s volbou `-T`:
  - `-c <cert_file>` – cesta k souboru s certifikátem,
  - `-C <cert_directory>` – cesta k adresáři s certifikáty (výchozí hodnota je `/etc/ssl/certs`),
- `-n` – stáhne pouze nové zprávy,
- `-h` – stáhne pouze hlavičky zpráv,
- `-b <mailbox>` – název poštovní schránky, kterou chcete použít (výchozí je `INBOX`).

2.4.3 Formát souboru s autentizačními údaji

```
username = info@example.com
password = pwd123
```

2.4.4 Adresářová struktura uložených emailů

Emaily jsou rozděleny podle emailové adresy a poté dále dle jednotlivých mailových schránek. Název jednotlivých emailů je jejich unikátní identifikátor UID. Každá složka emailové schránky má uložen token validity v souboru ”uidvalidity.txt”

```
mails:
    xtenor02@vutbr.cz:
        INBOX:
            1.eml
            23.eml
            uidvalidity.txt
        Dulezite:
            4.eml
            21.eml
            uidvalidity.txt
    marektenora@gmail.com:
        INBOX:
            53.eml
            uidvalidity.txt
```

2.4.5 Příklad použití

Příklad použití:

```
./bin/imapcl mail.example.com -a auth.txt -o emails -p 993 -T -c cert.pem -
n
```

Tento příkaz se připojí k IMAP serveru `mail.example.com` na portu 993 pomocí TLS, použije autentizační soubor `auth.txt` a uloží pouze nové e-maily do adresáře `emails`. Certifikát TLS je uložen v souboru `cert.pem`.

2.5 Shrnutí příkazů

Následující tabulka shrnuje hlavní příkazy pro práci s projektem:

Úkol	Příkaz
Sestavení aplikace	`make`
Sestavení v ladicím režimu	`make debug`
Vyčištění projektu	`make clean`
Spuštění aplikace	`./imapcl <server_address>` `-a <auth_file>` `-o <output_directory>`
Spuštění testů	`make -f test_Makefile run_tests`

Tabulka 1: Shrnutí příkazů pro práci s projektem

Tento návod pokrývá všechny základní kroky potřebné k sestavení a spuštění aplikace včetně testování. Uživatelé by nyní měli být schopni aplikaci úspěšně zkompilovat, spustit a přizpůsobit svým potřebám.

3 Návrh aplikace

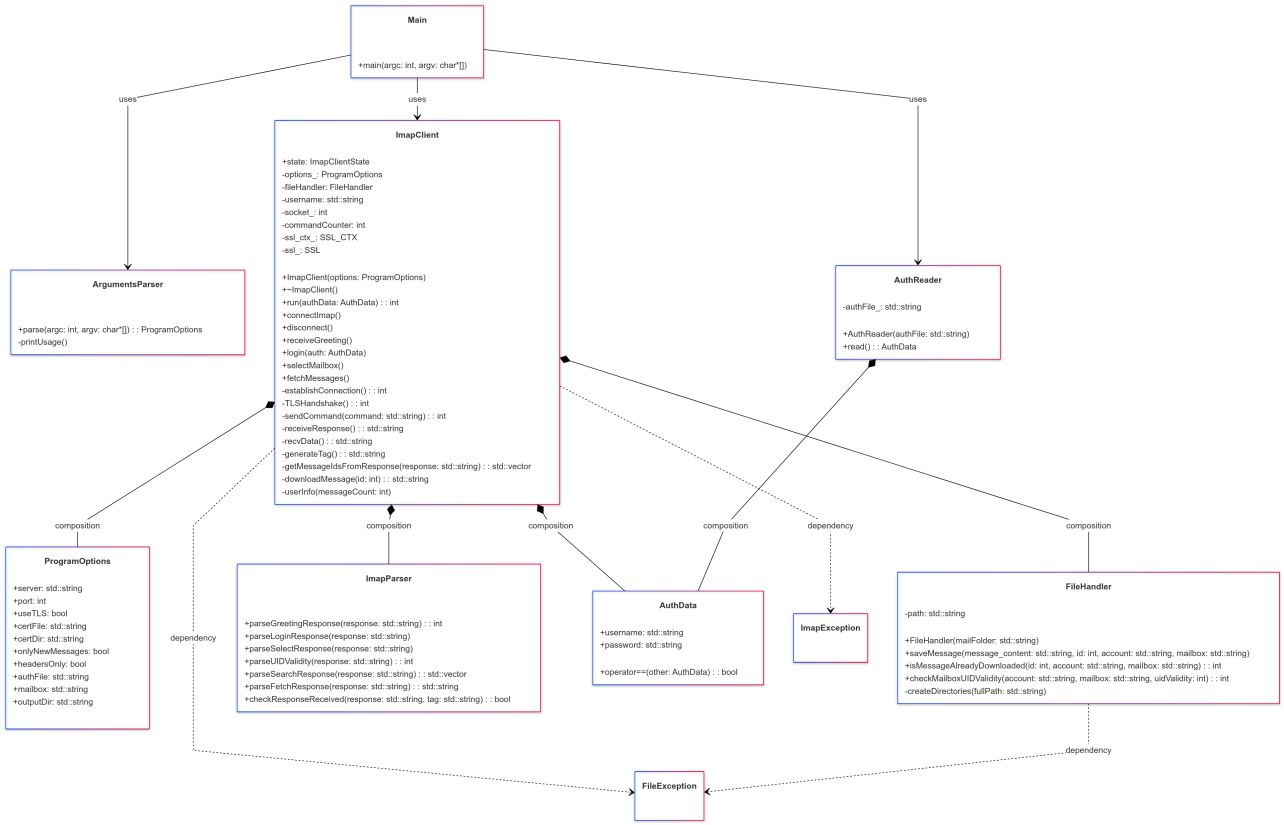
V této kapitole naleznete popis návrhu aplikace, včetně její architektury, použitých tříd a jejich vzájemné interakce.

3.1 Architektura aplikace

Stručný přehled architektury aplikace. Popis hlavních modulů a komponent, jejich úloh a vztahů.

3.2 Diagram tříd

Diagram tříd znázorňuje statickou strukturu aplikace, vztahy mezi třídami a jejich atributy a metody.



Obrázek 1: Diagram tříd

3.3 Popis tříd

Detailní popis jednotlivých tříd, jejich odpovědností, veřejných metod a atributů.

3.3.1 Třída `ArgumentsParser`

- **Účel**: Zpracování a validace vstupních argumentů příkazové řádky.
- **Hlavní metody**:
  - `parse(int argc, char* argv[])` – Parsování argumentů.
  - `printUsage()` – Zobrazení nápovědy pro použití programu.

3.3.2 Třída `ImapClient`

- **Účel**: Třída `ImapClient` je zodpovědná za připojení k IMAP serveru, autentizaci uživatele, výběr poštovní schránky, stahování e-mailů a řízení celkového chodu IMAP klienta.
- **Hlavní metody**:
  - `ImapClient(options: ProgramOptions)` – Konstruktor, inicializuje klienta s daným nastavením aplikace.
  - `~ImapClient()` – Destruktor, zajišťuje správné uvolnění zdrojů při zániku objektu.
  - `run(authData: AuthData): int` – Hlavní metoda, která řídí průběh připojení, autentizace a stahování e-mailů.

- `connectImap()` – Navázání spojení s IMAP serverem.
- `disconnect()` – Odpojení od IMAP serveru a uvolnění zdrojů.
- `receiveGreeting()` – Přijetí uvítací zprávy od serveru po navázání spojení.
- `receiveResponse()` – Přijetí ostatních zpráv ze serveru.
- `recvData()` – Naslouchá a přijímá jednotlivá data ze serveru a vrací je po 4096Bé.
- `sendCommand(string command)` – Odeslání příkazu na server.
- `login(auth: AuthData)` – Přihlášení uživatele pomocí poskytnutých autentizačních údajů.
- `selectMailbox()` – Výběr poštovní schránky (například `INBOX`).
- `fetchMessages()` – Stažení e-mailových zpráv z vybrané schránky.

3.3.3 Třída `AuthReader`

- **Účel**: Třída `AuthReader` zajišťuje čtení souboru s přihlašovacími údaji.
- **Hlavní metody**:
  - `ImapClient(string authFile)` – Konstruktor, nastaví cestu k auth souboru.
  - `read()` – čte přihlašovací údaje a ukládá je do struktury AuthData.

3.3.4 Třída `FileHandler`

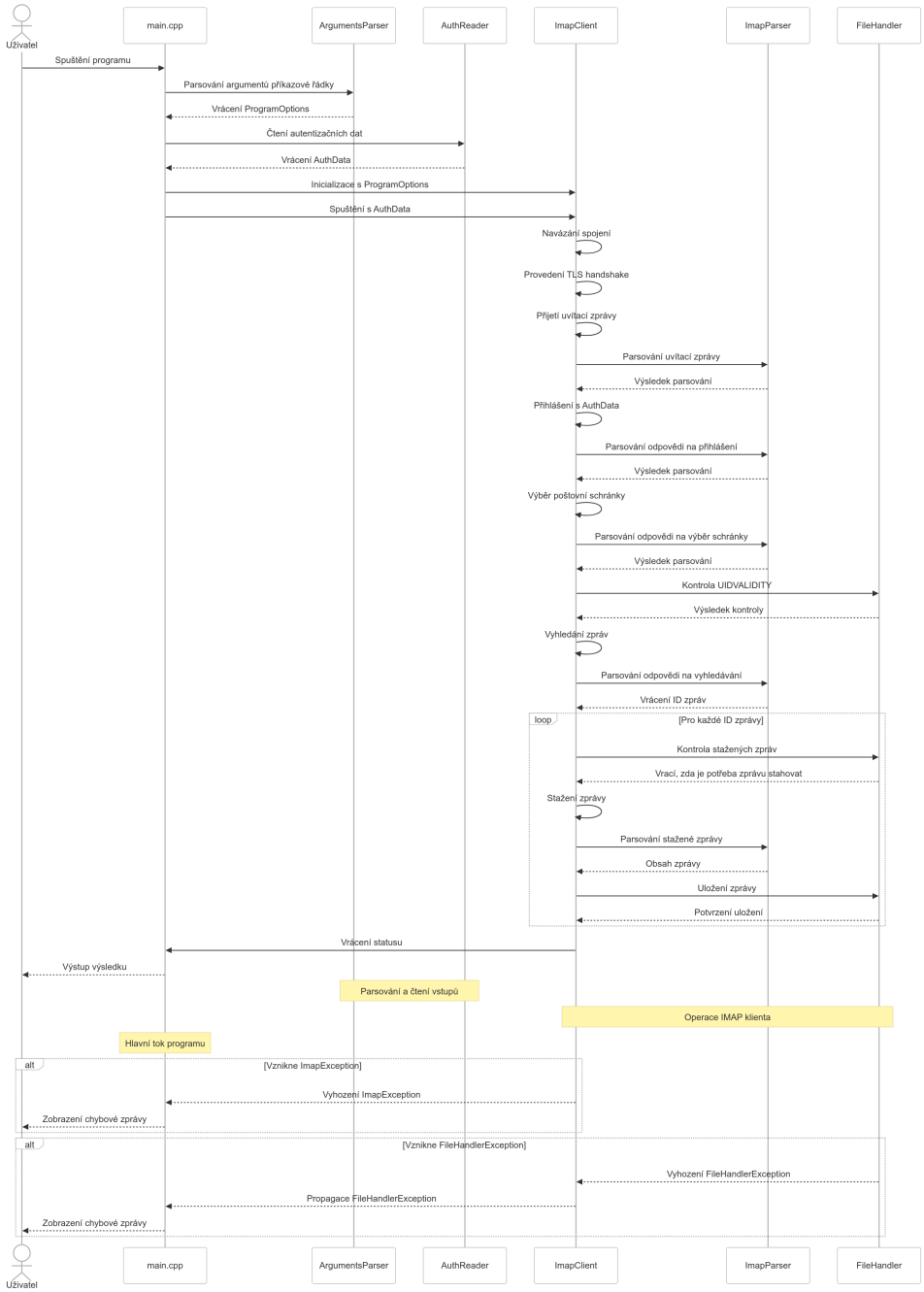
- **Účel**: Třída `FileHandler` je zodpovědná za správu ukládání zpráv, kontrolu, zda byly zprávy již staženy, a kontrolu konzistence schránky pomocí UIDVALIDITY hodnoty.
- **Hlavní metody**:
  - `saveMessage(string message_content, int id, string account, string mailbox)` – Ukládá obsah zprávy do souboru podle daného ID, názvu účtu a názvu schránky. V případě chyby vyhodí výjimku `FileException`.
  - `isMessageAlreadyDownloaded(int id, string account, string mailbox)` – Kontroluje, zda zpráva s daným ID již byla stažena. Vrací `0`, pokud nebyla stažena, `1`, pokud byla stažena celá, `2`, pokud byl stažen pouze hlavičkový soubor, a `-1` při chybě.
  - `checkMailboxUIDValidity(string account, string mailbox, int uidValidity): int` – Kontroluje UIDVALIDITY hodnotu schránky a ověřuje její konzistenci. Pokud hodnota nesouhlasí, všechny zprávy ve schránce jsou vymazány a nastaví se nová UIDVALIDITY hodnota. Vrací `0`, pokud hodnota souhlasí, `1`, pokud nesouhlasí, `2`, pokud schránka ještě neexistuje, a `-1` při chybě.

3.3.5 Třída `ImapParser`

- **Účel**: Třída `ImapParser` je zodpovědná za parsování odpovědí od IMAP serveru a extrakci důležitých informací, jako jsou stav připojení, přihlašovací stav, ID zpráv a obsah zpráv.
- **Hlavní metody**:
  - `parseGreetingResponse(string response)` – Parsuje uvítací zprávu od serveru a identifikuje její stav.
  - `parseLoginResponse(string response)` – Parsuje odpověď na přihlášení a ověřuje úspěšnost přihlášení.

- `parseSelectResponse(string response)` – Parsuje odpověď serveru při výběru poštovní schránky.
- `parseUIDValidity(string response)` – Extrahuje hodnotu UIDVALIDITY z odpovědi.
- `parseSearchResponse(string response)` – Parsuje odpověď na vyhledávací příkaz a extrahuje ID nalezených zpráv.
- `parseFetchResponse(string response)` – Extrahuje obsah zprávy z odpovědi serveru.
- `checkResponseReceived(string response, string tag)` – Kontroluje, zda odpověď serveru obsahuje specifikovaný tag.

3.4 Sekvenční diagram



Obrázek 2: Sekvenční diagram

4 Implementace

V sekci je popsaná implementace složitějších metod a způsobů použitých v projektu.

4.1 Čtení argumentů

Před samotným čtením se nastaví některé výchozí hodnoty nastavení. Poté se přečte první argument který musí být vždy adresa serveru. Po přečtení adresy serveru se pomocí while smyčky a getopt() čtou zbývající

argumenty.

4.1.1 Nastavení portu

Protokol IMAP používá dva různé výchozí porty podle toho, zda je připojení šifrované (TLS), nebo ne. Pokud argument `-p` není uveden (port tedy není explicitně zadán), nastaví se port na konci funkce na:

- 143 pro nešifrované připojení,
- 993 pro připojení s použitím TLS.

Na začátku je port nastaven na hodnotu `-1`, aby bylo možné na konci čtení rozpoznat, zda uživatel port definoval.

4.2 Čtení autentizačního souboru

Autentizační soubor obsahuje přihlašovací údaje, které jsou nutné pro úspěšné přihlášení k IMAP serveru. Proces čtení probíhá následujícím způsobem:

1. Ověření, zda soubor existuje
2. Soubor je čten řádek po řádku. Každý řádek obsahuje klíč a hodnotu, oddělené znakem =.
3. Klíč a hodnota jsou následně odděleny a z obou částí jsou odstraněny přebytečné mezery.
4. Proces se zastaví, pokud jsou obě hodnoty `username` a `password` nalezeny.

Pokud se při zpracování objeví neplatný formát nebo chybí některý z povinných údajů (`username` nebo `password`), je vyvolána výjimka, která upozorní na chybu v autentizačním souboru.

4.3 Řízení IMAP klienta

Řízení IMAP klienta má na starost stavový automat který jede v nekonečné smyčce. Automat se spouští ve funkci ImapClient::run()

4.3.1 Stavy

- **Disconnected**: Klient není připojen k IMAP serveru.
- **ConnectionEstablished**: Spojení se serverem bylo úspěšně navázáno.
- **NotAuthenticated**: Klient je připojen, ale není autentizován.
- **Authenticated**: Klient byl úspěšně autentizován.
- **SelectedMailbox**: Byla vybrána poštovní schránka pro další operace.
- **Logout**: Klient ukončuje komunikaci a odpojuje se od serveru.

4.4 Připojení k IMAP serveru

Před samotnou komunikací je potřeba ji vůbec navázat. Implementace zahrnuje vytvoření socketu, navázání spojení a případné zabezpečení pomocí TLS.

4.4.1 Inicializace připojení

Metoda `connectImap()` je zodpovědná za navázání spojení se serverem. Nejprve volá funkci `establishConnection()`, která vytvoří socket a připojí se k IMAP serveru na základě zadané adresy a portu. Pokud je v nastavení povoleno použití TLS, provede se zabezpečení spojení pomocí metody `TLSHandshake()`.

```
void ImapClient::connectImap() {
    int conn = establishConnection();
    if (conn != 0){
        throw ImapException("Failed to establish connection");
    }

    if (options_.useTLS) {
        int hs = TLSHandshake();
        if (hs != 0){
            throw ImapException("Failed to establish TLS connection");
        }
    }
    state = ImapClientState::ConnectionEstabilished;
}
```

4.4.2 Navázání socketového spojení

Funkce `establishConnection()` zajišťuje vytvoření socketu a navázání spojení se serverem. Rozlišuje, zda je adresa serveru ve formě IP adresy nebo doménového jména, a podle toho použije vhodné metody pro získání potřebných informací.

```
int ImapClient::establishConnection() {
    // Nastavení parametrů pro getaddrinfo()
    struct addrinfo hints, *res;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;        // IPv4 nebo IPv6
    hints.ai_socktype = SOCK_STREAM;    // TCP socket

    // Získání informací o serveru
    int status = getaddrinfo(
        options_.server.c_str(),
        std::to_string(options_.port).c_str(),
        &hints, &res);
    if (status != 0) {
        throw ImapException("Failed to get server IP address");
    }

    // Vytvoření socketu
    socket_ = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (socket_ < 0) {
        freeaddrinfo(res);
        throw ImapException("Failed to open socket for connection");
    }
```

```
    // Navázání spojení
    if (connect(socket_, res->ai_addr, res->ai_addrlen) == -1) {
        close(socket_);
        freeaddrinfo(res);
        throw ImapException("Failed to connect to server");
    }

    freeaddrinfo(res);
    return 0;
}
```

4.4.3 Zabezpečení pomocí TLS

Pokud je povoleno použití TLS, metoda `TLSHandshake()` se stará o zabezpečení spojení. Vytvoří SSL kontext, načte certifikáty (pokud jsou specifikovány), vytvoří SSL objekt a provede TLS handshake.

```
int ImapClient::TLSHandshake() {
    // Inicializace SSL kontextu
    ssl_ctx_ = SSL_CTX_new(SSLv23_client_method());
    if (!ssl_ctx_) {
        throw ImapException("Failed to create SSL context");
    }

    // Načtení certifikátů, pokud jsou specifikovány
    if (!options_.certFile.empty() || !options_.certDir.empty()) {
        if (SSL_CTX_load_verify_locations(ssl_ctx_,
        options_.certFile.empty() ? nullptr : options_.certFile.c_str(),
        options_.certDir.empty() ? nullptr : options_.certDir.c_str()) != 1) {
            throw ImapException("Failed to load SSL certificates");
        }
    }

    // Vytvoření SSL objektu a přiřazení socketu
    ssl_ = SSL_new(ssl_ctx_);
    SSL_set_fd(ssl_, socket_);

    // Provedení TLS handshake
    if (SSL_connect(ssl_) <= 0) {
        throw ImapException("Failed to establish TLS connection");
    }

    return 0;
}
```

4.4.4 Greeting

Po navázaní spojení se serverem je potřeba přijmout Greeting zprávu ze serveru. Tato zpráva je specifická tím, že nevrací žádný tag. Proto je přijímání Greeting zprávy řešeno odděleně od klasické odpovědi serveru.

4.5 Odesílání a přijímání zpráv

Komunikace mezi klientem a IMAP serverem je realizována pomocí odesílání příkazů a přijímání odpovědí. Implementace se zaměřuje na spolehlivost přenosu a zahrnuje mechanismy pro zotavení v případě, že server neodpoví nebo dojde k chybě v komunikaci.

4.5.1 Odesílání příkazů

Odesílání příkazů na server je zajištěno metodou `sendCommand()`, která přidává na konec každého příkazu znaky CRLF, jak vyžaduje IMAP protokol. Pokud je použito šifrované spojení (TLS), data jsou odesílána pomocí SSL_write(), jinak se používá standardní send().

```
int ImapClient::sendCommand(const std::string& command) {
    // Přidání CRLF na konec příkazu
    std::string full_command = command + "\r\n";

    ssize_t bytes_sent;
    if (ssl_) {
        bytes_sent = SSL_write(ssl_, full_command.c_str(), full_command.length());
    } else {
        bytes_sent = send(socket_, full_command.c_str(), full_command.size(), 0);
    }
    if (bytes_sent < 0) {
        throw ImapException("Failed to send command to server");
    }

    return 0;
}
```

4.5.2 Přijímání odpovědí

Přijímání odpovědí od serveru je realizováno metodou `receiveResponse()`. Tato metoda opakovaně volá `recvData()`, která přijímá data ze socketu, a kontroluje, zda byla přijata kompletní odpověď pro daný příkaz.

V případě, že odpověď není kompletní, metoda `receiveResponse()` se pokouší data přijímat znovu, až do vyčerpání maximálního počtu pokusů (`maxTries`). Pokud ani poté není odpověď kompletní, je vyvolána výjimka.

```
std::string ImapClient::receiveResponse() {
    std::string currentTag = generateTag();
    std::string response;
    bool received = false;
    int maxTries = 1000;

    while (!received && maxTries > 0) {
        std::string recved = recvData();
        response += recved;

        // Kontrola, zda byla přijata kompletní odpověď
        if (ImapParser::checkResponseReceived(response, currentTag)) {
            received = true;
```

```
        }
        maxTries--;

        if (maxTries == 0) {
            throw ImapException("Failed to receive complete response from server");
        }
    }

    commandCounter++;
    return response;
}
```

4.5.3 Přijímání dat

Metoda `recvData()` se stará o skutečné přijímání dat ze socketu. Používá funkci select(), která čeká na dostupnost dat ke čtení s nastaveným časovým limitem. Pokud je použito TLS, kontroluje se dostupnost dat pomocí SSL_pending().

V případě, že během časového limitu nejsou přijata žádná data, je vyvolána výjimka s informací o timeoutu. Pokud dojde k chybě při čtení ze socketu nebo při SSL komunikaci, metoda se pokusí opakovat čtení, případně vyvolá výjimku.

```
std::string ImapClient::recvData() {
    char buffer[4096];
    int bytes_received = 0;
    const int timeout_seconds = 30;

    while (true) {
        fd_set read_fds;
        FD_ZERO(&read_fds);
        FD_SET(socket_, &read_fds);
        int max_fd = socket_ + 1;

        struct timeval timeout;
        timeout.tv_sec = timeout_seconds;
        timeout.tv_usec = 0;

        int select_result;
        if (ssl_) {
            // Pro SSL kontrolujeme, zda jsou data k~dispozici
            if (SSL_pending(ssl_) > 0) {
                select_result = 1; // Data jsou k~dispozici
            } else {
                select_result = select(max_fd, &read_fds, NULL, NULL, &timeout);
            }
        } else {
            // Pro nešifrované spojení použijeme přímo select
            select_result = select(max_fd, &read_fds, NULL, NULL, &timeout);
        }

        if (select_result > 0) {
            if (FD_ISSET(socket_, &read_fds)) {
```

```
                if (ssl_) {
                    bytes_received = SSL_read(ssl_, buffer, sizeof(buffer) - 1);
                } else {
                    bytes_received = recv(socket_, buffer, sizeof(buffer) - 1, 0);
                }

                if (bytes_received > 0) {
                    buffer[bytes_received] = '\0';
                    return std::string(buffer);
                } else if (bytes_received == 0) {
                    // Spojení bylo uzavřeno ze strany serveru
                    throw ImapException("Server closed connection");
                } else {
                    // Zpracování chyb při čtení
                    // ...
                }
            }
        } else if (select_result == 0) {
            // Vypršel časový limit
            throw ImapException("Timeout while waiting for data from server");
        } else {
            // Chyba funkce select()
            // ...
        }
    }
}
```

4.5.4 Zotavení při stahování zpráv

Při stahování zpráv více zpráv může nastat situace, kdy server neodpoví nebo dojde k přerušení komunikace. V takovém případě se dotaz jednou opakuje, pokud ani tentokrát nedostane odpověď, vyvolá vyjímku.

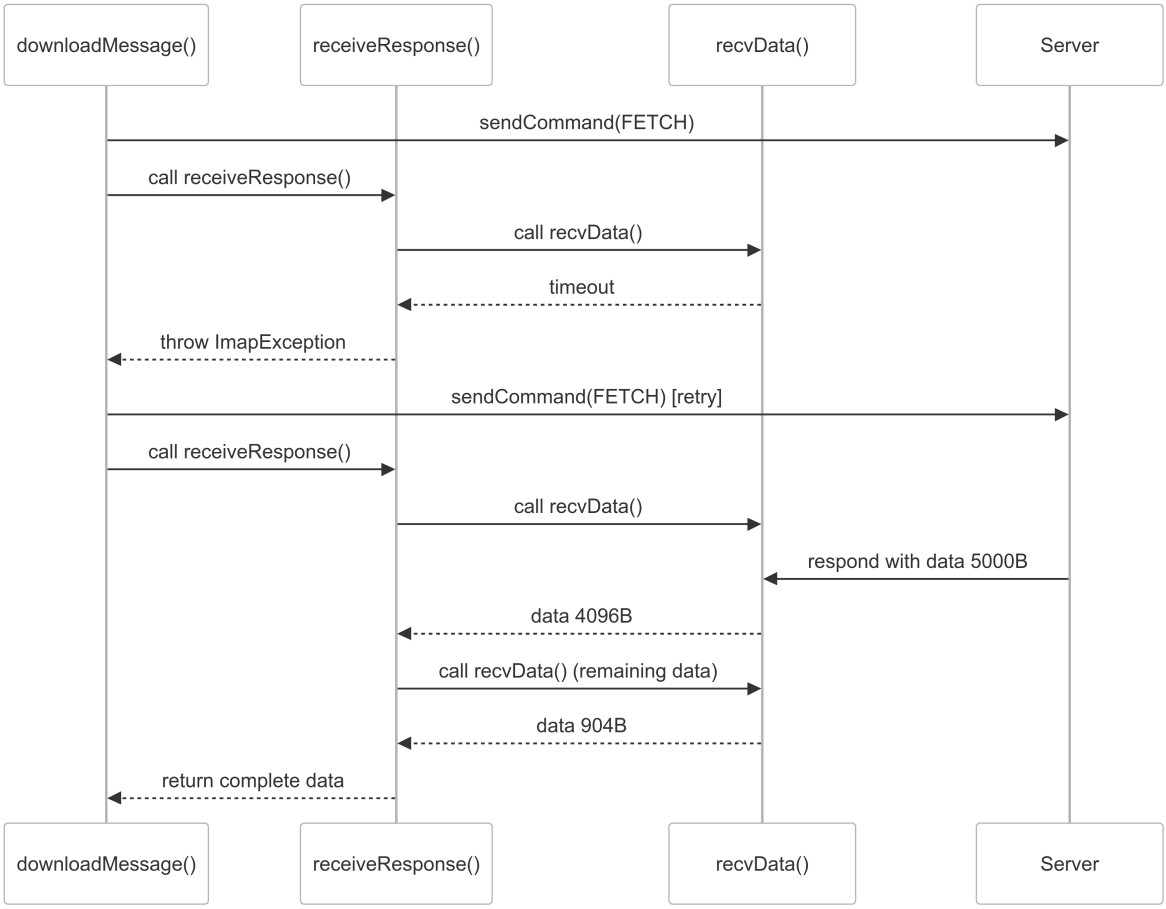
```
std::string ImapClient::downloadMessage(int id) {
    std::ostringstream command;
    if (options_.headersOnly) {
        command << generateTag() << " UID FETCH " << id << " BODY[HEADER]";
    } else {
        command << generateTag() << " UID FETCH " << id << " BODY[]";
    }

    // Odeslání příkazu FETCH
    if (sendCommand(command.str()) != 0){
        throw ImapException("Failed to send FETCH command");
    }

    std::string response;
    try {
        // Přijetí odpovědi od serveru
        response = receiveResponse();
    } catch (const ImapException& e) {
        // Pokud se nepodařilo zprávu stáhnout, pokusí se znovu
```

```
        std::cerr << "Failed to download message from server. Retrying one more tim
        if (sendCommand(command.str()) != 0){
            throw ImapException("Failed to send FETCH command");
        }
        response = receiveResponse();
    }
    std::string message = ImapParser::parseFetchResponse(response);

    return message;
}
```



Obrázek 3: Sekvenční diagram zotavení

5 Testování

5.1 Google Test

K testování byla použita knihovna Google Test. Pro každou třídu jsem vytvořil několik testů které kontrolují zda program funguje správně. Všechny testy jsou uloženy ve složce /tests.

5.1.1 Spuštění testů

Pro spuštění testů je k dispozici `test_Makefile`. Spusťte následující příkaz:

```
make -f test_Makefile run_tests
```

Tento příkaz spustí všechny definované testy, jejichž výstup bude zobrazen v terminálu. [**?**]

5.2 Debugger

Použití debuggeru bylo klíčové při testování aplikace. Díky němu jsem byl schopen rychle nalézt proč se mi kód zasekl v nekonečných smyčkách, číst hodnoty proměnných za běhu a kontrolovat správný chod programu.

5.3 Wireshark

Pro kontrolu komunikace jsem využil wireshark v kombinaci se spuštěním programu bez TLS. Díky tomu jsem mohl číst komunikaci. Wireshark určitě umožňuje číst i šifrovanou komunikaci pokud bych do něj vložil správné klíče, ale na takovou úroveň jsem se zatím nedostal.

5.4 Lokální server

Pro kontrolu funkčnosti vlastních certifikátů při použití argumentů -c nebo -C jsem využil self-signed certifikát a lokální server na který jsem se klientem připojil.

Reference

[1] Eric Rescorla: RFC 8846: The Transport Layer Security (TLS) Protocol Version 1.3. [Online]. Dostupné: `https://datatracker.ietf.org/doc/html/rfc8446`.

[2] Kenneth Ballard: Secure programming with the OpenSSL API. [Online]. Dostupné: `https://developer.ibm.com/tutorials/l-openssl/`.

[3] Mark Crispin: RFC 3501: Internet Message Access Protocol - Version 4rev1. [Online]. Dostupné: `https://datatracker.ietf.org/doc/html/rfc3501`.

[4] Pete Resnick: RFC 5322: Internet Message Format. [Online]. Dostupné: `https://datatracker.ietf.org/doc/html/rfc5322`.