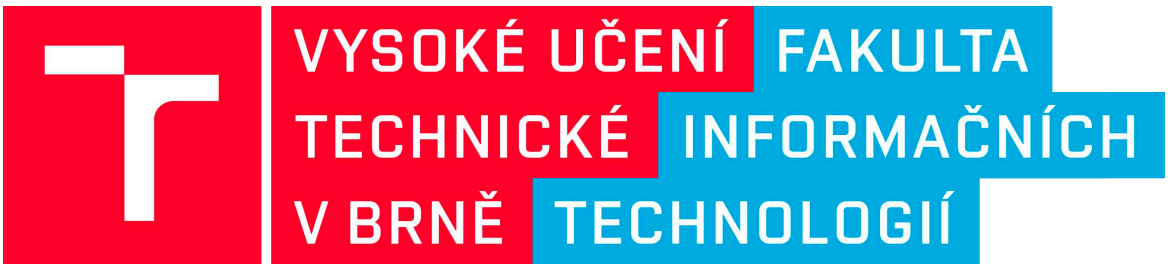
Vysoké Učení Technické v Brně



Projekt ISA - IMAP klient

Autor: Jan Pawlus, xpawlu00@stud.fit.vutbr.cz

Obsah

1 Úvod do problematiky

Mým úkolem bylo vytvořit IMAP klienta, který bude umět stahovat e-maily na základě zadaných parametrů, a ukládat výstupy podle standardu `RFC5322`. Součástí je také implementace zabezpečeného `TLS` spojení. Protokol IMAP je dobře popsán v `RFC3501`, nebyl tedy až takový problém porozumět komunikaci s ním. Při vývoji nastalo pár otázek, jako například jak implementovat funkci pro ukládání e-mailů či jak správně otevřít a zakončit komunikaci se serverem.

2 Návrh aplikace

Klienta jsem se rozhodl implementovat v jazyku `C++`, neboť oproti `C` obsahuje hned několik výhod, kterým kraluje hlavně zahrnutí různých knihoven (mnou používaný `regex`) do standardu `C++11` nebo rozšíření standardních datových typů - namátkou typy `vector` a `string`. Dále například objektový přístup, který jsem využil, můj klient je tedy třída s atributy a metodami. Tímto a použitím hlavičkového souboru, definujícím atributy a metody, se kód stává přehlednějším.

Návrh funkčnosti je jednoduchý - klient musí ověřit parametry, soubor s autentizačními údaji (případně certifikát), připojit se k serveru, provést autentizaci, stáhnout zprávy do zadané složky a odhlásit se. Při jakékoliv chybě je nutno aplikaci ukončit a informovat uživatele (případně vypsat nápovědu k použití programu).

3 Popis implementace

3.1 Start

Jako první je třeba zpracovat parametry zadané uživatelem. Klient obsahuje vlastní argument parser, jelikož použití `getopt` by bylo příliš komplikované kvůli tomu, že argument `server` není uvozen žádnou značkou. V situaci, kdy uživatel zadá nesprávné parametry (či vůbec nezadá povinné), mu program vypíše popis chyby a nápovědu pro použití. Pokud uživatel zadá složku, která neexistuje, klient se ji pokusí vytvořit (při chybě informuje uživatele a skončí). Je to proto, že na základě mých zkušeností je pro uživatele příjemnější, když se program o vytvoření složky pokusí, než kdyby vyhodil chybu. Po úspěšné kontrole, zda jsou všechny parametry v pořádku (povinné argumenty jsou zadány a úspěšně se načetly přihlašovací údaje z příslušného souboru), se klient snaží vytvořit spojení se zadaným serverem a autorizovat uživatele dle zadaných údajů. Pokud neuspěje, uživatel je informován na standardním chybovém výstupu. Pokud uspěje, přejde klient k samotné funkčnosti - posílání zpráv serveru a ukládání odpovědí. Celá funkčnost klienta je obalena v konstrukci `try - catch`, což zajistí směrování všech chyb na jedno místo, vypsání nápovědy pro uživatele a korektní ukončení programu.

3.2 Šifrované spojení

K navázání zabezpečeného `SSL/TLS` spojení byla použita knihovna `openssl`. Jejím použitím jsem se inspiroval tímto článkem, který byl uveden v zadání projektu, což mi dost ulehčilo implementaci, jelikož článek přímo říká, co přesně se má použít k navázání spojení.

K zabezpečenému spojení je třeba porovnat certifikát serveru s lokálním certifikátem, což řeší funkce z knihovny `openssl`. Ověřovat se může buď přímo soubor zadán parametrem `-c`, nebo specifikováním složky s certifikáty (parametr `-C`).

Pokud uživatel specifikuje parametrem `-c` určitý certifikát, který ale není platný, klient se bude snažit najít platný certifikát ve specifikované (či nespecifikované, a tedy klientem nastavené) složce s certifikáty.

Může se zde vyskytnout problém při použití přepínače `-C` způsobený faktem, že funkce pro zpracování certifikátu nepracuje s `.pem` soubory, ale se speciálními symlinky obsahující hash certifikátu, může být proto nutné použít funkci `c_rehash` k vytvoření těchto symlinků. Další problém může být s přístupem do složky `\etc\ssl\certs\` kvůli nedostatečnému oprávnění. V takovém případě je nutné nastavit dostačující oprávnění příkazem `chmod`, nebo nastavit složku obsahující certifikáty někam, kam má aplikace přístup.

Při ladění jsem se dozvěděl, že knihovna `openssl` tvoří malé konstantní memory leaky, bohužel s tímto faktem nelze nic moc dělat.

3.3 Čtení ze serveru

Po úspěšném nastavení komunikace serverem klient čte a zapisuje data pomocí funkce `bio_read()`, resp. `bio_write()` (touto částí jsem se inspiroval článkem, který je uveden v doporučené literatuře k projektu). Při vývoji jsem narazil na problém se čtením ze serveru. Občas se stalo, že funkce `bio_read` stále čekala na proud dat ze serveru, ten ale již neměl co zaslat. Je to kvůli použití `blocking` socketů. Při použití `non-blocking` varianty klient ale nepřečetl vůbec nic. Protokol `IMAP` má pevnou strukturu a číslování instrukcí, nastala tedy otázka, zda nečíst buď do té doby, než se v odpovědi objeví ukončovací řádek odpovědi (číslo instrukce a výsledek dotazu). Druhá možnost je číst velikost obsahu zprávy, která se objeví v každé hlavičce.

Obě možnosti mají svá úskalí v tom, že pokud by byly tyto informace obsažené v textu e-mailu, čtení by skončilo. Já si vybral první možnost - klient tedy čte, dokud `read` nevrátí nulový počet přenesených dat nebo dokud se v odpovědi neobjeví konec instrukce (`aX OK`, `aX BAD` nebo `aX NO`, kde `X` je číslo instrukce). Číslo instrukce jsem zvolil dostatečně bizarní na to, aby se neobjevilo v těle e-mailu, což by znamenalo nesprávnou detekci ukončení čtení.

3.4 Stažení zpráv

Po korektní autentizaci a výběru e-mailové schránky klient přejde do cyklu, kde načítá e-maily. Zvolené přepínače ovlivní `FETCH` příkaz pro stažení e-mailů tak, aby vyhovoval funkčnosti. Z `RFC5322` jsem se dočetl, že povinné položky hlavičky jsou `Date` a `From`, ostatní jsou volitelné. Klient ale každopádně stáhne všechny hlavičky příznakem `BODY[HEADER]`. Tělo e-mailu je specifikováno flagem `BODY.PEEK[TEXT]` (to zajistí stažení všech částí e-mailu i s přílohou v textové formě). Může se stát, že se stáhne text e-mailu ve více formách (například `text/plain` a `text/html`), je poté na uživateli, aby si ze zprávy vzal, co potřebuje.

Při testovaní se vyskytla anomálie se serverem `imap.gmail.com`, a to, že když klient specifikoval ve `FETCH` tělo e-mailu pomocí `BODY[]`, server odpověď rozdělil do dvou částí - na hlavičku a tělo, což se ale neshodovalo s mým způsobem zpracování zpráv. Bohužel jsem se nedozvěděl, proč se toto na `imap.gmail.com` děje, při použití `BODY.PEEK[TEXT]` se odpověď vrátila korektně. Rozdíl v `BODY.PEEK[TEXT]` a `BODY[TEXT]` je ten, že `BODY.PEEK` nutně nenastavuje novým e-mailům flag `\Seen`, ale v kombinaci s `BODY[HEADER]` se e-mail označí jako přečtený tak jako tak.

Výsledek instrukce FETCH se ukládá do jednoho velkého bufferu, který je následně zpracováván. Vznikl tím ale problém se stažením příliš velkého množství dat - v jedné chvíli už se buffer nemohl dále rozšířit a program skončil s chybou bad_alloc. Tento problém je vyřešen tak, že po každých cca 50 MB přenesených dat klient čtení zastaví a funkce pro ukládání e-mailů uloží e-maily z dosavadního bufferu (tím je zároveň z bufferu smaže a uvolní tak místo).

Stažení nových zpráv funguje obdobně, pouze je nutný jeden požadavek na server navíc (SEARCH UNSEEN), jež vrátí seznam ID zpráv, které zatím nebyly orazítkovány flagem \Seen. Tento seznam dále klient zpracuje a zakomponuje do FETCH požadavku. Stažení pouze hlaviček je implementováno tak, že klient ve FETCH požadavku nepožádá o BODY.PEEK[TEXT].

3.5 Ukládání zpráv

Zde přišlo na řadu klasické zpracování textu. Jak jsem již zmínil, C++ disponuje nástroji, které tuto práci hodně ulehčují. Ukládání e-mailů je vyřešeno funkcí string::find, kdy klient hledá v cyklu začátek každé zprávy - ten je definován standardizovaným řádkem IMAPu, který obsahuje UID zprávy (to funkce najde a na základě UID pojmenuje uložený e-mail) a flagy. Po hlavičce následuje informace o velikosti zprávy. Tu klient získá a na základě toho načte tuto délku z bufferu. Tím je získaná jedna zpráva, která je navíc hned smazána z bufferu a cyklus pokračuje od začátku.

Po cca 50 MB přenesených dat se čtení pozastaví a přichází na řadu ukládání. Pokud je funkce zavolána v takovém stavu, je třeba navíc ověřit, zda je zpracovávaný e-mail kompletní (tak, že funkce vyhledá začátek další zprávy). Pokud jej nenajde, ukončí se a čtení ze serveru pokračuje. Po skončení čtení ze serveru (server buď nemá co poslat, nebo je detekován konec IMAP instrukce) se klient odhlásí zprávou LOGOUT.

Přišla na řadu otázka, zda neimplementovat vícevláknový přístup (čtení a ukládání zpráv by probíhalo současně). Zkoušel jsem tuto vlastnost implementovat, bohužel se mi to kvůli větší komplexnosti a časové tísni nepovedlo dotáhnout do konce, aplikace je tedy pouze jednovláknová.

4 Použití aplikace

Aplikaci je nutno spouštět s určitými parametry - některé jsou volitelné, některé povinné. Při nezadání čti špatném zadání se uživateli zobrazí na standardní chybový výstup chyba, kterou uživatel při zadávání udělal, a na standardní výstup nápověda pro použití. Klienta je nutno spustit minimálně s těmito parametry:

[server] - specifikuje název serveru.

-a [auth_file] - specifikuje soubor s přihlašovacími údaji.

-o [output] - specifikuje složku, do které se mají ukládat zprávy.

Další parametry jsou nepovinné, a jsou to:

-p - specifikuje číslo portu. Při nezadání se určí na základe toho, zda je zapnuta šifrovaná komunikace (port je nastaven na 993) či nešifrovaná (port 143).

-b [mailbox] - specifikuje schránku, ze které se mají stáhnout e-maily. Při nezadání tohoto parametru se použije schránka INBOX.

-T - zapíná šifrované spojení.

-c [cert] - specifikuje název souboru s certifikátem. Pokud není zadán, klient vyhledá certifikát ve složce zadané parametrem -C.

`-C [path]` - specifikuje cestu k souboru s certifikátem. Při nezadání se použije složka `\etc\ssl\certs`.

`-n` - zapíná stažení pouze nových zpráv.

`-h` - zapíná stažení pouze hlaviček zpráv.

Je třeba zachovat správný formát některých parametrů. Port musí být číslo a soubor s přihlašovací údaji musí být v tomto formátu:

```
username: [username]
password: [password]
```

Parametry se dají kombinovat. Příklady spuštění tedy mohou být takové:

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

Aplikace se bude snažit připojit na port 143 k serveru `eva.fit.vutbr.cz` (není zapnuto zašifrované spojení), přihlašovací údaje se získají ze souboru `cred` a zprávy ze schránky `Sent` se uloží do složky `maildir`.

Další příklad použití může být takový:

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

V takovém případě se zapne šifrované spojení, klient se pokusí ověřit certifikát `cert.pem` (v případě neplatného certifikátu se bude hledat ve složce cert, v případě neexistujícího souboru `cert.pem` aplikace skončí s chybou) a stáhnou se pouze hlavičky nových zpráv ze schránky `INBOX` serveru `eva.fit.vutbr.cz:993` (port se nastaví na základě zapnutí šifrovaného spojení) do složky `maildir`.

```
./imapcl imap.gmail.com -o output -a cred -T
```

Klient se připojí k serveru imap.gmail.com, certifikát se vyhledá ve složce /etc/ssl/certs.

5 Literatura

Při vývoji klienta jsem čerpal informace z těchto zdrojů:

- **https://tools.ietf.org/html/rfc3501** - `RFC3501` popisuje celý protokol `IMAP`, jehož nastudování je nutné pro komunikaci se serverem. Je třeba znát formát instrukcí, příkazy jako `LOGIN`, `SELECT`, `SEARCH` či `FETCH`. Je třeba také znát flagy, kterými lze specifikovat, co přesně má server vrátit.

- **https://tools.ietf.org/html/rfc5322** - `RFC5322` popisuje standardní formát e-mailu. Tato znalost je nezbytná k produkci správného výstupu.

- **http://www.ibm.com/developerworks/library/l-openssl/** - tutoriál k použití knihovny `openssl`, díky kterému se čtenář rychle zorientuje v použití této knihovny.

- **http://stackoverflow.com** - klasika všech programátorů. Při téměř jakémkoliv problému dokáže tento web programátora správně nasměrovat.