

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Sít'ové aplikace a správa sítí – Dokumentace k projektu  
Čtečka novinek ve formátu Atom s podporou TLS

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
1.1	Účel a funkce programu . . . . .	2
1.2	Použití programu . . . . .	2
1.2.1	Kompilace programu . . . . .	2
1.2.2	Spouštěcí parametry . . . . .	3
1.2.3	Soubor s feedy . . . . .	3
<b>2</b>	<b>Zadání projektu a teoretický kontext</b>	<b>4</b>
2.1	Rozbor zadání . . . . .	4
2.2	Protokol HTTP . . . . .	4
2.3	SSL a TLS . . . . .	4
2.3.1	Certifikáty . . . . .	4
2.4	Webové feedy . . . . .	4
<b>3</b>	<b>Implementační detaily</b>	<b>5</b>
3.1	Modularita aplikace . . . . .	5
3.2	Použité knihovny . . . . .	6
3.3	Zpracování parametrů . . . . .	6
3.4	Zpracování URL adresy . . . . .	6
3.5	Zpracování souboru feedfile . . . . .	7
3.6	HTTP komunikace . . . . .	7
3.6.1	Inicializace . . . . .	7
3.6.2	Navázání připojení . . . . .	7
3.6.3	Totožnost serveru . . . . .	8
3.6.4	Tvorba HTTP požadavku . . . . .	8
3.6.5	Načítání odpovědi . . . . .	8
3.6.6	Kontrola HTTP kódu odpovědi . . . . .	9
3.6.7	Deinicializace . . . . .	9
3.7	Zpracování XML struktury a výpis výsledků . . . . .	9
3.7.1	Co modul hledá a ukládá . . . . .	9
3.7.2	Hledané informace podle specifikace RSS či Atom . . . . .	10
3.7.3	Pomocné funkce pro procházení struktury . . . . .	11
3.8	Pomocný modul Util . . . . .	11
3.9	Automatické testy . . . . .	12
<b>4</b>	<b>Závěr</b>	<b>13</b>
4.1	Možná vylepšení . . . . .	13

# Kapitola 1 Úvod

Tato dokumentace popisuje funkcionalitu a implementaci programu FeedReader, což je čtečka feedů s novinkami ve formátu RSS nebo Atom.

## 1.1 Účel a funkce programu

Tento program napsaný v jazyce C slouží ke stažení a čtení XML souborů s feedy zapsanými ve formátu RSS nebo Atom. Program podporuje komunikaci s nezabezpečenými servery i servery zabezpečenými pomocí SSL či TLS. Program umožňuje předání explicitně zadaných certifikátů pro ověření totožnosti vzdáleného serveru.

## 1.2 Použití programu

Aplikaci je možné nastavit pomocí parametrů, které jsou platné pro dané spuštění. Aplikace umožňuje stáhnout a zpracovat jeden konkrétní feed, případně více feedů zadaných ve speciálním souboru.

Obecný vzor pro použití aplikace feedreader:

```
feedreader <URL | -f <feedfile> [-c <certfile>] [-C <certaddr>] [-T] [-a] [-u] [-h]
```

Ukázka spuštění:

```
./feedreader http://www.nejakyweb.cz/rss.xml -c certifikat.crt -T -u
```

V případě nesprávného použití parametrů se zobrazí zpráva s upozorněním a vysvětlením chyby. Parametry si lze kdykoliv připomenout spuštěním programu pouze s parametrem `-h`.

### 1.2.1 Kompilace programu

Program je možné zkompileovat pomocí přiloženého souboru Makefile příkazem `make`. Ke kompilaci se používá překladač GCC se zdrojovými kódy napsanými v jazyce C podle standardu C99.

Testy k aplikaci lze spustit příkazem `make test`. Vyčistit veškeré překladem vytvořené soubory lze příkazem `make fullclean`.

### 1.2.2 Spouštěcí parametry

Níže jsou uvedeny parametry spuštění a jejich význam či případná omezení:

Parametr	Název	Význam	Omezení
URL	Adresa feedu	Představuje jednu adresu feedu pro zpracování	Lze zadat pouze jednu adresu URL nebo jeden soubor feedfile. Adresa musí začínat <code>http://</code> nebo <code>https://</code> . U adres je možné uvést explicitní port za hostitelem uvozený dvojtečkou.
-f feedfile	Soubor s adresami feedů	Umožňuje zadat cestu k textovému souboru, který obsahuje více adres URL s feedy.	Lze zadat pouze jednu adresu URL nebo jeden soubor feedfile. Podrobnosti o souboru feedfile naleznete v části 1.2.3.
-c certfile	Soubor s certifikáty	Umožňuje zadat cestu k souboru s certifikáty, vůči kterým se má ověřit totožnost serveru.	Lze zadat maximálně 10 souborů s certifikáty. Pro zadání více souborů je nutné použít parametr <code>-c</code> opakovaně.
-C certaddr	Cesta k adresáři s certifikáty	Umožňuje zadat cestu k adresáři s certifikáty, vůči kterým se má ověřit totožnost serveru.	Lze zadat maximálně 10 adresářů s certifikáty. Pro zadání více adresářů je nutné použít parametr <code>-C</code> opakovaně. Certifikáty musí mít správný název (viz pomůcka <code>c_rehash</code> knihovny OpenSSL).
-T	Zahrnout informace o čase položek	Ve výpisu položek feedu se zobrazí časová informace o tvorbě nebo aktualizaci dané položky.	Pokud není datum u položky uvedeno, nezobrazí se
-a	Zahrnout informace o autorech položek	Ve výpisu položek feedu se zobrazí informace o autorech dané položky (jméno či e-mailová adresa)	Pokud není autor u položky uveden, nezobrazí se
-u	Zahrnout přiložené adresy u položek	Ve výpisu položek feedu se zobrazí asociovaná adresa dané položky. Adresa obvykle směřuje k úplnému obsahu.	Pokud není adresa u položky uvedena, nezobrazí se
-h	Zobrazit nápovědu k programu	Nápověda obsahuje stručný popis parametrů a jejich použití.	Tento parametr musí být uveden samostatně a jako jediný

### 1.2.3 Soubor s feedy

Pomocí parametru `-f` je možné předat adresu k textovému souboru s více adresami URL k feedům. Program v takovém případě soubor otevře, načte jednotlivé adresy URL a pokusí se zpracovat všechny zde uvedené adresy.

Očekáván je textový soubor UNIXového typu s adresami uvedenými každá samostatně na novém řádku. Pokud bude na jednom řádku nalezeno více adres oddělených bílými znaky, načtou se všechny adresy.

Do souboru je možné zapsat i komentář (např. s vysvětlivkami obsahu URL adres). Komentář je nutné uvést znakem `#` a platí až do konce daného řádku.

## Kapitola 2 Zadání projektu a teoretický kontext

V této kapitole je stručně rozebráno zadání projektu a poté je velice okrajově uvedena teorie k jednotlivým částem.

### 2.1 Rozbor zadání

Aplikace má být schopna navázat zabezpečené i nezabezpečené připojení se vzdáleným serverem, odeslat mu HTTP dotaz na dokument identifikovaný adresou URL, převzít HTTP odpověď, tu zpracovat, oddělit hlavičku od těla zprávy a tělo zprávy se poté pokusit jako XML soubor naparsovat a vypsát příslušné informace, přičemž se očekává, že daný soubor bude ve formátu RSS nebo Atom.

Mezi další požadovanou funkcionalitu patří zpracování nepovinného souboru feedfile s více adresami URL, možnost vyhodnotit důvěryhodnost vzdáleného serveru pomocí vlastních certifikátů a vše s příslušnými srozumitelnými výstupy pro uživatele. Zobrazované části XML souboru ve výstupu lze specifikovat příslušnými parametry.

Aplikace má být robustní a funkční na serverech merlin.fit.vutbr.cz a eva.fit.vutbr.cz.

### 2.2 Protokol HTTP

Protokol HTTP je jednoduchý textový protokol, kdy klient zasílá serveru požadavek (většinou požadavek na nějaký soubor) a server požadavek zpracuje (vrátí soubor), případně odešle chybu.

Každá HTTP zpráva (požadavek i odpověď) se skládá ze dvou částí - hlavičky a těla, které jsou oddělené prázdným řádkem.

V těle odpovědi je pak požadovaný dokument. Hlavička obsahuje různé parametry a režijní informace. V tomto projektu je důležitý zejména návratový kód (např. 200 = vše v pořádku). V těle zprávy se pak bude nacházet požadovaný XML dokument s feedem.

### 2.3 SSL a TLS

Jedná se o kryptografické protokoly, které umožňují zabezpečenou komunikaci po síti, aniž by bylo možné zachytit význam dat, případně data zfalšovat. K tomu slouží šifrovací klíče, které si mezi sebou obě strany vymění a na základě kterých bude komunikace pokračovat. Součástí je také autentizace komunikujících stran pomocí certifikátů.

V rámci komunikace HTTP je zabezpečení realizováno protokolem HTTPS.

#### 2.3.1 Certifikáty

Během komunikace dochází k výměně certifikátů, které slouží k ověření identity serveru (ale v určitých případech i klienta). Server odešle svůj certifikát a klient jej porovná se svou databází důvěryhodných certifikátů. Certifikát musí být platný a zároveň musí být podepsán známou certifikační autoritou.

### 2.4 Webové feedy

Webové feedy slouží jako prostředek pro usnadnění čtení novinek na webových stránkách v internetové síti. Uživatel se jednoduše přihlásí k odběru feedů a příslušná aplikace kontroluje soubory s feedy a informuje uživatele o případných novinkách. Feedy jsou většinou ve formátu XML, který umožňuje snadné zpracování programy. Mezi nejrozšířenější formáty feedů patří RSS a Atom.

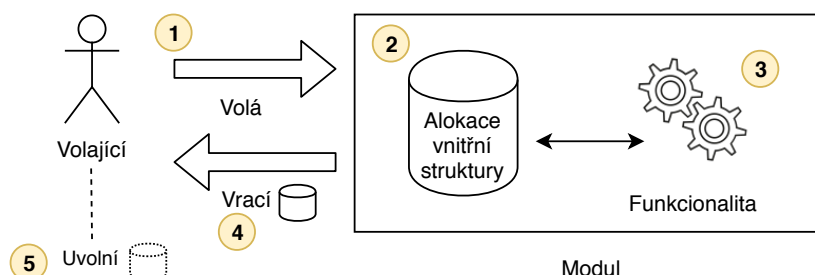
## Kapitola 3 Implementační detaily

Tato kapitola popisuje záležitosti týkající se vývoje, tedy kódu, vnitřní struktury aplikace a podrobnější informace o jednotlivých modulech.

### 3.1 Modularita aplikace

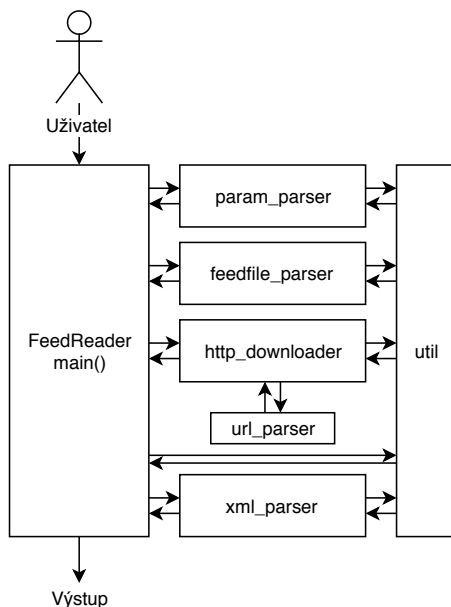
Protože se řešení programu skládá z více podúloh, jsou jednotlivé dílčí úkony děleny do samostatných modulů. Výhodou je jednodušší vývoj a údržba aplikace a možnost otestování jednotlivých funkcí bez závislosti na zbytku programu.

Všechny moduly pracují na stejném principu znázorněném na následujícím obrázku:



Navracená vnitřní struktura typicky obsahuje požadovaná data zpracovaná modulem a také případný chybový kód a zprávu, aby bylo možno posléze informovat uživatele. Všechny moduly poskytují funkce pro správné uvolnění navracené vnitřní struktury.

Návrh celé aplikace vypadá nějak takto:



Ve funkci `main()` se tedy pouze budou volat jednotlivé moduly a nakonec se vypíše výsledky. Veškerá logika je na modulech, které fungují nezávisle.

## 3.2 Použité knihovny

Aplikace využívá dvou knihoven:

- OpenSSL – využití pro nezabezpečenou i zabezpečenou komunikaci, BIO sockety, předávání certifikátů atd... Testovány byly verze 1.0.2k-fips a 1.0.2p-freebsd.
- libxml2 – využití pro parsování staženého XML souboru a následné procházení naparsované struktury. Testovány byly verze 2.9.1 a 2.9.7.

Žádné jiné nestandardní knihovny aplikace nevyužívá. Bylo zvažováno využití knihovny getopt, kvůli rozdílností na jednotlivých platformách (např. v řetězci `optstring` má první znak '-' pokaždé jiný význam) byly parametry nakonec načteny „ručně“, což ve výsledku urychlilo vývoj a bylo jednodušší.

## 3.3 Zpracování parametrů

Ústřední funkcí tohoto modulu je funkce `parse_parameters()`, která přijímá přímo parametry příkazové řádky a jejich počet. Stačí předat přímo `argc` a `argv`.

Samotné zpracování je pak jednoduchý stavový automat, který prochází jednotlivé parametry. Nejdříve se zjistí, o jaký typ vstupu se jedná, k tomu slouží výčtový typ `arg_type`:

- `ARG_TYPE_TEXT` – obyčejný text, jedná se o hodnotu nějakého parametru nebo adresu URL.
- `ARG_TYPE_PARAM_SHORT` – parametr zadaný s jedním písmenem, např. '-x'.
- `ARG_TYPE_PARAM_LONG` – parametr zadaný s jedním písmenem, které je následováno jeho hodnotou bez mezery, např. '-xhodnota'.
- `ARG_TYPE_INVALID` – slouží k ošetření NULL, prázdného řetězce nebo samotné pomlčky.

Po stanovení typu se určí, zda je parametr povinný, zda již nebyl použit, zda nezpůsobí nekompatibilitu parametrů (např. `feedfile` použit spolu s adresou URL) apod.

Pokud dojde k chybě, do návratové struktury se vloží příslušný chybový kód (definovaný v hlavičkovém souboru modulu), alokuje se chybová zpráva a funkce modulu končí.

Během parsování parametrů se jednotlivé načtené parametry a jejich hodnoty předávají do výsledné struktury tak, aby je bylo snadné načíst při jejich následném zpracování. Předávané hodnoty se vždy alokují nově a nejsou závislé na původním poli parametrů `argv`.

Modul pro zpracování parametrů nepodporuje tzv. „dlouhé parametry“ uvozené dvěma pomlčkami.

## 3.4 Zpracování URL adresy

Modul pro zpracování URL adresy má jediný cíl – převzít textový řetězec, zkontrolovat přítomnost protokolu na začátku a rozkouskovat adresu na hostitele, port a zbytek. V případě, že není port explicitně uveden, použije se výchozí port. Modul aktuálně podporuje protokoly HTTP (port 80) a HTTPS (port 443).

Tento modul je implementován velmi jednoduchým stavovým automatem, který čte řetězec po znacích a přepíná své fáze čtení. Modul nijak hluboce neověřuje korektnost předané adresy URL. V případě, že se nepodaří adresu rozkouskovat, dojde k chybě. V rámci celého programu je řešení korektnosti adresy URL ponecháno v rukou serveru, na který se požadavek odešle.

### 3.5 Zpracování souboru feedfile

Také modul pro zpracování souboru feedfile je implementován jako jednoduchý čtecí stavový automat.

Nejdříve se otevře daný soubor (je-li zadán, lze totiž předat i řetězec s textem) a následně se načítají jednotlivé adresy s feedy. Modul neví, že se jedná o adresy – nekontroluje tedy ani jejich správnost. Zkrátka veškeré texty jsou alokovány do paměti a předány ve výsledné struktuře.

V případě, že se právě nenačítá nějaká adresa - a narazí se na znak '#', automat se přepne do režimu čtení komentáře a až do konce řádku nebo souboru nic nenačítá. Podmínka, že se v danou chvíli nesmí načítat nějaká adresa, je přítomna proto, že v adrese URL se může objevovat znak kotvy, proto aby to bylo považováno za komentář, musí být adresa a komentář odděleny alespoň jedním bílým znakem.

V rámci robustnosti aplikace jsou vráceny veškeré texty i na jednom řádku oddělené alespoň jedním bílým znakem. To znamená, že se na jednom řádku může nacházet i více adres.

### 3.6 HTTP komunikace

Úkolem modulu `http_downloader` je navázat připojení se vzdáleným serverem, provést kontrolu certifikátů (v případě zabezpečeného připojení), zaslat požadavek na cílový soubor podle zadané URL adresy a následně převzít odpověď a uložit ji do cílové struktury. K tomu byly využity BIO sockety knihovny OpenSSL.

#### 3.6.1 Inicializace

Modul nejdříve zpracuje pomocí modulu s URL parserem předanou adresu URL. Tím získá potřebné údaje, jako například hostitele a port, ke kterému se má připojit.

Následně se provádí nezbytná inicializace knihovny OpenSSL funkcemi `SSL_load_error_strings()` a `SSL_library_init()`.

#### 3.6.2 Navázání připojení

Poté podle toho, zda se jedná o HTTP nebo HTTPS protokol se způsob navázání spojení se serverem liší:

- **Nezabezpečené připojení** – V případě nezabezpečeného připojení je situace jednoduchá – vytvoří se BIO socket pomocí funkce `BIO_new_connect()`, které se předá hostitel a port. Pokud se socket podařilo vytvořit, modul se pokusí o spojení se vzdáleným serverem (funkce `BIO_do_connect()`).
- **Zabezpečené připojení** – V případě zabezpečeného připojení je situace malinko složitější, ale většinu práce za nás odvádí knihovna OpenSSL.

Nejdříve je nutné vytvořit kontext zabezpečení (SSL\_CTX) pomocí funkce `SSL_CTX_new()`. Této funkci je nutné předat metodu, která závisí na tom, zda plní aplikace roli serveru, klienta nebo obojího. V tomto případě se používá metoda `SSLv23_client_method()`.

Následně se načtou případné uživatelem zadané certifikáty. Knihovna OpenSSL obsahuje funkci `SSL_CTX_load_verify_locations()`, která umí načítat certifikáty jak ze souboru, tak z adresáře.

Náš modul obsahuje pomocné funkce `provide_certfile()` a `provide_certdir()`, které slouží právě k předání certifikátů a které musí být volány ještě před samotným „spuštěním“ modulu. Modul si certifikáty uloží a nyní je i použije. Pokud nebyly certifikáty zadány, použijí se výchozí systémové certifikáty (funkce `SSL_CTX_set_default_verify_paths()`).

Pomocí připraveného kontextu je nyní konečně možné připravit BIO socket. Pro zabezpečený socket se používá funkce `BIO_new_ssl_connect()`. Dalším krokem je nastavení SSL tak, aby knihovna OpenSSL prováděla úvodní komunikaci (handshake) na pozadí kdykoliv si o to server zažádá[1]. Nejdříve je potřeba získat speciální SSL objekt pomocí funkce `BIO_get_ssl()` a ten následně předat modifikační funkci `SSL_set_mode()`, které se předá režim `SSL_MODE_AUTO_RETRY`. Nakonec je potřeba



BIO socketu nastavit hostitele a port (funkce `BIO_set_conn_hostname()`). Pokud vše uspělo, modul se pokusí o spojení se vzdáleným serverem (funkce `BIO_do_connect()`).

Připojení se vzdáleným serverem nemusí z režijních důvodů být v danou chvíli dostupné, knihovna OpenSSL ovšem dokáže určit, kdy je vhodné se o spojení pokusit znovu. K tomu slouží funkce `BIO_should_retry()`. Pokud vrací funkce `true`, připojení se zkouší neustále v cyklu. Pokud se připojení nenaváže a funkce začne vracet `false`, znamená to neúspěšné připojení se serverem.

**Poznámka k timeoutu:** Může nastat situace, kdy od serveru nepřichází vůbec žádná odpověď. Jedná se zejména o případy, kdy se knihovna snaží navázat připojení se serverem, kde není daný port otevřen a komunikace probíhá na portu jiném. V takovém případě aplikace čeká na timeout, jehož délka je řízena operačním systémem a nelze aplikačně změnit. Tato situace by se dala vyřešit například tak, že se operace čekání dá na samostatné vlákno a pokud se připojení nezdaří do určité doby, vlákno se ukončí.

### 3.6.3 Totožnost serveru

Pokud se připojení podaří, pak je potřeba ověřit totožnost serveru pomocí certifikátů. K tomu slouží funkce `SSL_get_verify_result()`. Pokud není hodnota rovna `X509_V_OK`, server není důvěryhodný a připojení bude ukončeno.

### 3.6.4 Tvorba HTTP požadavku

Pomocí funkce `BIO_write()` je do socketu zadán tento požadavek:

```
GET <soubor> HTTP/1.0
Host: <host>
User-Agent: Feedreader-xsipos03
Accept: application/xml
Accept-Charset: UTF-8,*
Cache-Control: private, no-store, max-age=0
Connection: close
```

Verze protokolu HTTP 1.0 je zvolena proto, že je jednodušší pro zpracování, neobsahuje například přenos odpovědi prostřednictvím chunků (Chunked transfer encoding). Podle specifikace platí, že pokud klient odešle na server požadavek HTTP nižší verze, pak server odpoví v hlavičce nejvyšší verzí, kterou podporuje, zpráva ovšem musí obsahovat pouze náležitosti týkající se verze, o kterou klient zažádal [3].

Hlavička `Cache-Control` je pro server spíše doporučeným postupem. Server ji ale nemusí dodržet. Klient v tomto případě oznamuje, že si žádný obsah neudrzuje pro opakované použití. Po odeslání stejného požadavku vícekrát by měl server tedy obsah zpětně i vícekrát zaslat. Testováním se ukázalo, že servery se podle této hlavičky ne vždy řídí.

### 3.6.5 Načítání odpovědi

Jakmile je odesílání požadavku dokončeno, načítá se odpověď serveru. K tomu slouží funkce `BIO_read()`. Program si udržuje buffer fixní velikosti, do kterého postupně ukládá celou odpověď. Ta může přicházet po částech (typicky 4-16 kB). Buffer se postupně dynamicky rozšiřuje o konstantní hodnotu tak, aby se do něj zpráva vešla celá.

Po dokončení načítání se jako poslední znak bufferu nastaví nulový znak, který není součástí HTTP odpovědi, nejednalo by se tedy o platný řetězec jazyka C.

### 3.6.6 Kontrola HTTP kódu odpovědi

Před předáním načtené odpovědi do výsledné struktury se nejdříve zkontroluje HTTP kód odpovědi, který je vždy předáván jako DRUHÉ slovo hlavičky. Pokud tento kód není roven 200 (OK), pak něco neproběhlo v pořádku a odpověď neobsahuje požadovaný dokument. V takovém případě se příslušná chybová zpráva s vysvětlením situace předá do výsledné struktury pro pozdější předání uživateli.

### 3.6.7 Deinicializace

Po úspěšné komunikaci je nakonec vrácena načtená odpověď – tedy HTTP hlavička a následně i tělo zprávy (požadovaný dokument). Nakonec je zavolána pomocná funkce `free_openss()`, která provede deinicializaci knihovny OpenSSL a uvolnění paměti. V této funkci se volají různé funkce knihovny, které by měly využitou paměť uvolnit.

**Poznámka:** Knihovna OpenSSL na serverech `merlin.fit.vutbr.cz` a `eva.fit.vutbr.cz` nenabízí jednoduchý způsob, jakým paměť uvolnit. Na serveru Eva navíc některé uvolňovací procedury způsobují chybu SIGSEGV. Výsledkem je, že na serveru Merlin sice dojde k úplnému uvolnění paměti, ale také ke spouště chybám čtení a uvolňování (alespoň podle nástroje `valgrind`), a na serveru Eva je část paměti ztracena.

## 3.7 Zpracování XML struktury a výpis výsledků

Modul pro zpracování XML struktury nejdříve předá XML soubor knihovně Libxml2, která vytvoří v paměti strukturu, kterou lze pomocí funkcí snadno procházet a číst (případně upravovat) její obsah.

Struktura se do tohoto modulu předává jako textový řetězec, proto bylo pro její naparsování využito funkce `xmlReadMemory()`, která vrací ukazatel na `xmlDoc`, což je již zpracovaná struktura, která se předává v následných funkcích pro procházení.

V rámci zpracování struktury se také předávají příznaky:

```
XML_PARSE_NOERROR | XML_PARSE_NOWARNING
```

Tyto příznaky vypínají jakýkoliv výstup knihovny Libxml2, protože aplikace si chyby řeší sama. Pro aplikaci je pouze důležité, zda byl XML vstup validní či nikoliv. Pokud ne, uživatel se dozví, že server nezaslal validní soubor a proto nemohl být zpracován.

### 3.7.1 Co modul hledá a ukládá

Úkolem modulu je vytáhnout z XML dokumentu požadované informace o feedu. Jedná se zejména o:

- Název feedu
- Názvy položek
- Datum změny/tvorby položek
- Autory položek
- URL adresy přiřazené k položkám

Protože modul netuší, které informace uživatel vlastně vyžaduje (na základě předaných parametrů), vyhledá se všechna užitečná data a ta se uloží do výsledné struktury. V hlavní části programu (funkce `main()`) se pak vypíše to, co uživatel zadal. K dispozici budou všechny nalezené informace. Vypíší se pouze ty požadované.

### 3.7.2 Hledané informace podle specifikace RSS či Atom

V rámci robustnosti aplikace neočekává, že budou veškeré využitě tagy v XML dokumentu odpovídat specifikacím. Aplikace umožňuje i jejich záměnu. Často se totiž vyskytují „hybridní“ feedy, které například míchají tagy RSS 1.0 a RSS 2.0. Proto byl modul vyvinut tak, aby vyhledával veškeré dostupné informace nezávisle na úvodní deklaraci verze nebo typu feedu.

**Poznámka k namespace:** Kanály novinek počínaje RSS 1.0 mohou definovat tzv. jmenné prostory (namespace)[2]. Ty jsou definovány v úvodu a poté se jednotlivé tagy používají s předponou oddělenou dvojtečkou. Knihovna Libxml2 tuto předponu vynechá a čte tagy bez ní. Problém nastává ve chvíli, kdy autor feedu v úvodu definici namespace opomene. Pak je tag identifikován celým názvem včetně oné dvojtečky. Proto při vyhledávání informací programem jsou vždy prohledávány obě varianty tagů – s předponou i bez ní.

#### 3.7.2.1 Název feedu

Název celého feedu je vyhledáván v těchto umístěních:

- KOŘEN -> <channel> -> <title>
- KOŘEN -> <title>

Pokud není název feedu nalezen, uvede se *feed bez názvu*.

#### 3.7.2.2 Položka

Jednotlivé položky jsou vyhledávány jako řetězec sousledných položek na stejné úrovni, a to takto:

- KOŘEN -> <channel> -> <item>
- KOŘEN -> <item>
- KOŘEN -> <entry>

Pro každou položku se do výsledné struktury modulu alokuje podstruktura, do které se poté ukládají nalezené informace. Jednotlivá data položek jsou pak vždy podřazenými tagy dané položky.

#### 3.7.2.3 Název položky

Název položky je pro všechny feedy uveden na jednotném místě:

- POLOŽKA -> <title>

Pokud není název položky nalezen, uvede se *záznam bez názvu*.

#### 3.7.2.4 Datum položky

Modul vyhledává datum tvorby či poslední aktualizace položky. Poslední aktualizace má přednost:

- POLOŽKA -> <dc:date>
- POLOŽKA -> <date>
- POLOŽKA -> <pubDate>
- POLOŽKA -> <updated>

### 3.7.2.5 Autor položky

U autora položky může být k dispozici více informací. Obvykle jméno i e-mail. Upřednostňováno je jméno autora. Pokud se u položky nachází autorů více, vypsan je ten první.

- POLOŽKA -> <dc:creator>
- POLOŽKA -> <creator>
- POLOŽKA -> <author>

E-mail a jméno autora ve feedu Atom se pak obvykle vyskytuje v těchto podřazených tazích[4]:

- AUTOR -> <name>
- AUTOR -> <email>

### 3.7.2.6 URL adresa položky

URL adresa položek je obvykle součástí atributu href:

- POLOŽKA -> <atom:link> -> atribut href
- POLOŽKA -> <link> -> atribut href
- POLOŽKA -> <guid isPermaLink=true>

Všimněte si, že v tagu <guid> se adresa URL nachází pouze tehdy, pokud je zároveň její atribut isPermaLink nastaven na true. To znamená, že daná URL adresa by měla fungovat i jako jednoznačný identifikátor[5].

### 3.7.3 Pomocné funkce pro procházení struktury

Aby byla aplikace přehlednější, byly vytvořeny pomocné funkce, které usnadňují procházení výsledné struktury a hledání příslušných dat.

Funkce `findXmlNodeByPath()`, která umožňuje zadat posloupnost textových řetězců, které představují strukturu zanoření hledaného tagu. Struktura se poté projde a vrátí se požadovaný tag. Pokud se nepodaří tag nalézt, vrátí se NULL.

Funkce `getNotNullXmlNode()`, v této funkci se jako parametry očekávají volání předchozí funkce. Jedná se o funkci, která slouží k zpřehlednění v případě, kdy se nějaká informace prohledává ve více různých strukturách. Cílem je najít tag alespoň v některé z nich. Tuto funkci by šlo nahradit také klasickým větvením.

Funkce `findNextXmlNodeOnSameLevel()`, která vyhledá tag se zadaným názvem v úrovni zadaného kořenového tagu. Tato funkce se vyhledává pro hledání dalších položek feedů, které se zpravidla nacházejí ve struktuře na stejné úrovni.

## 3.8 Pomocný modul Util

Tento pomocný modul obsahuje funkce a konstanty, které sémanticky nespádaly do žádného z jiných modulů. Jednalo se o tyto položky:

- Konstanty s nastavením, např. maximální počet certifikátů, velikost nárůstu bufferu pro ukládání dat nebo možnost pro zapnutí ladění.
- Makro pro převod čísel na řetězce. Převodem se myslí obalení uvozovkami, aby šlo daný řetězec použít v konkatenci.

- Funkce `malloc_string()`, která umožňuje alokovat paměť a uložit do ní konkatenci řetězců zadaných jako parametry.
- Makro pro tisk ladicích výpisů umožňující podobné formátování jako funkce `printf()`.

### 3.9 Automatické testy

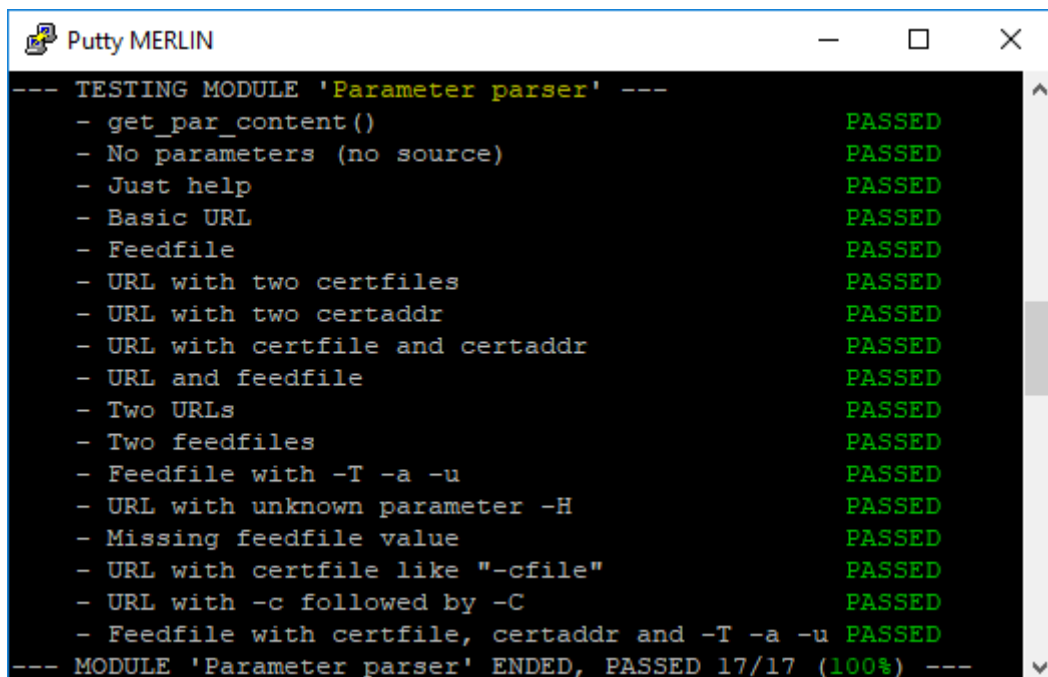
Jelikož byl program napsán jako sada modulů, které jsou vzájemně nezávislé, není problém tyto moduly jednotlivě otestovat. K tomu byla vytvořena testovací sada nazvaná `test_suite`. Tato sada poskytuje rozhraní pro testování funkcionality programů v jazyce C.

Postup pro otestování programu pomocí této testovací sady:

1. Testovací program musí nejdříve zavolat funkci `begin_test_session`, pomocí které se testovací sada inicializuje a je připravena k otestování volaného programu.
2. Následně je potřeba zavolat funkci `test_module` a předat jí ukazatel na funkci, která se má v rámci testovaného modulu spustit. Typicky se jedná o funkci, která v sobě obsahuje veškeré testy pro daný modul.
3. Daná funkce tedy volá jednotlivé testy prostřednictvím funkce `run_test`, která obsahuje rovněž ukazatel na funkci s daným testem.
4. Daná funkce s testem potom obsahuje porovnávací funkce `assert`, které jsou rovněž dodávány v rámci testovací sady a umožňují porovnání různých primitivních datových typů. Navíc umožňuje i tvořit vlastní porovnávací výrazy.
5. Po zavolání všech assertů v rámci daného testu, po dokončení všech testů v rámci modulu a po otestování všech modulů v rámci programu se pak zavolá funkce `end_test_session`, která testování ukončí a vypíše výsledky.

Testovací sada byla využita hlavně z toho důvodu, že bylo poměrně neobratné testovat funkčnost a nefunkčnost částí aplikace v rámci dvou různých serverů. Změna v rámci jednoho serveru často ovlivnila funkcionality na druhém a naopak. Testovací sada navíc obsahuje přehledné výpisy, které usnadňují hledání chyb.

Ukázka testování jednoho z modulů:



```

Putty MERLIN
--- TESTING MODULE 'Parameter parser' ---
- get_par_content() PASSED
- No parameters (no source) PASSED
- Just help PASSED
- Basic URL PASSED
- Feedfile PASSED
- URL with two certfiles PASSED
- URL with two certaddr PASSED
- URL with certfile and certaddr PASSED
- URL and feedfile PASSED
- Two URLs PASSED
- Two feedfiles PASSED
- Feedfile with -T -a -u PASSED
- URL with unknown parameter -H PASSED
- Missing feedfile value PASSED
- URL with certfile like "-cfile" PASSED
- URL with -c followed by -C PASSED
- Feedfile with certfile, certaddr and -T -a -u PASSED
--- MODULE 'Parameter parser' ENDED, PASSED 17/17 (100%) ---

```

## Kapitola 4 Závěr

Cílem projektu bylo implementovat čtečku feedů ve formátů Atom a RSS. Projekt se skládal z podproblémů, které byly řešeny pomocí nezávislých modulů. Zadání se podařilo splnit a program byl otestován na operačních systémech CentOS Linux 7 a FreeBSD 11.2. Program byl implementován v jazyce C a přeložen překladačem GCC se standardem C99.

### 4.1 Možná vylepšení

- Na serveru `eva.fit.vutbr.cz` bylo zjištěno, že aplikace způsobuje únik paměti (Memory Leak). Důvodem je chyba v použité knihovně OpenSSL verze 1.0.2p-freebsd, kdy po zavolání uvolňovací funkce `bio_free_all()` na zmíněném serveru dojde k chybě Segmentation Fault. Z tohoto důvodu nebyla tato uvolňovací funkce použita, čímž nedojde k řádnému uvolnění využité paměti. Řešením by mohlo být použití novější verze této knihovny, případně využít knihovnu jinou.
- Aplikace momentálně neumí extrahovat obsah přijaté HTTP hlavičky, která obsahuje množství užitečných informací - například v případě přesměrování informací o tom, kam byl požadovaný objekt přesunut. Jelikož to zadání ovšem nevyžadovalo, nebyla tato funkce pro jednoduchost implementována. Dalším možným využitím by mohlo být informování o prázdné odpovědi (`content-length: 0`), případně hlavičku `content-length` využít také pro okamžité nastavení velikosti bufferu pro uložení těla přijaté zprávy (odpovědi). Momentálně se při zaplnění bufferu jeho velikost zvýší o předem nastavenou konstantní hodnotu.
- V případě více adres URL ke zpracování by mohla aplikace provádět stahování souborů souběžně, například za pomoci vláken (threads). Bylo by ovšem nutné pečlivě implementovat také nutnou režii – správnou synchronizaci a zachování původního pořadí vyžádaných feedů. Výhodou by byl rychlejší výstup pro uživatele, nevýhodou by pak bylo vyšší využití paměti, jelikož by se musely všechny soubory nejdříve stáhnout a zpracovat a až poté vypsát v příslušném pořadí. Nyní probíhá zpracování a vypisování sériově (postupně). Před zpracováním dalšího feedu je předchozí z paměti uvolněn.

## Literatura

- [1] BALLARD, K. *Secure programming with the OpenSSL API* [online]. Poslední změna 16. srpna 2018 [cit. 19. listopadu 2018]. Dostupné na: <<https://developer.ibm.com/tutorials/1-openssl/>>.
- [2] BEGED DOV, G., BRICKLEY, D., DORNFEST, R. et al. *RDF Site Summary (RSS) 1.0* [online]. Poslední změna 9. června 2008 [cit. 19. listopadu 2018]. Dostupné na: <<http://web.resource.org/rss/1.0/>>.
- [3] MOGUL, J. C., FIELDING, R., GETTYS, J. et al. *Use and Interpretation of HTTP Version Numbers* [Internet Requests for Comments]. [b.m.]: RFC Editor, May 1997. 6 s. RFC, 2145. Dostupné na: <<https://tools.ietf.org/html/rfc2145>>.
- [4] NOTTINGHAM, M. a SAYRE, R. *The Atom Syndication Format* [Internet Requests for Comments]. [b.m.]: RFC Editor, December 2005. 43 s. RFC, 4287. Dostupné na: <<https://tools.ietf.org/html/rfc4287>>.
- [5] RSS ADVISORY BOARD. *RSS 2.0 Specification* [online]. Poslední změna 30. března 2009 [cit. 19. listopadu 2018]. Dostupné na: <<http://www.rssboard.org/rss-specification>>.