VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Síťové aplikace a správa sítí – projekt

Čtečka novinek ve formátu Atom a RSS s podporou TLS

29. října 2022

Vojtěch Dvořák (xdvora3o)

Obsah

1 Přehled teorie **3**
- 1.1 Uniform resource locator (URL) . . . 3
- 1.2 Syndikace obsahu prostřednictvím RSS 2.0 a Atom . . . 3
- 1.3 Hypertext transfer protocol (HTTP) . . . 4
- 1.4 Transport layer security (TLS) . . . 4
- 1.5 Shrnutí . . . 4

2 Odevzdané soubory **5**

3 Popis implementace **6**
- 3.1 Poznámka k terminologii . . . 6
- 3.2 Použité technologie . . . 6
- 3.3 Architektura programu . . . 6
- 3.4 Popis fungování programu . . . 7
  - 3.4.1 Zpracování uživatelských vstupů . . . 7
  - 3.4.2 Analýza souboru s adresami . . . 7
  - 3.4.3 Kontrola adres a jejich analýza . . . 7
  - 3.4.4 Získání dat . . . 9
  - 3.4.5 Zpracování získaných dat . . . 9
  - 3.4.6 Výpis hodnot . . . 9
- 3.5 Ošetření chybových a neočekávaných stavů . . . 10
  - 3.5.1 Chybová hlášení a varování . . . 10
  - 3.5.2 Propagace chybových kódů . . . 10

4 Rozšíření a omezení **11**

5 Testování **11**
- 5.1 Testovací skript . . . 11
- 5.2 Testovací server . . . 12
- 5.3 Průběh testování . . . 12

6 Uživatelský manuál **13**
- 6.1 O programu . . . 13
- 6.2 Použití programu . . . 13
- 6.3 Chybové stavy programu . . . 13
- 6.4 Návratové kódy programu . . . 13
- 6.5 Příklady spuštění programu . . . 14
  - 6.5.1 Jednoduché čtení novinek z RSS zdroje . . . 14
  - 6.5.2 Jednoduché čtení novinek z více zdrojů . . . 14

Úvod

Cílem projektu bylo vytvořit program, který bude stahovat data ve formátu RSS 2.0 nebo Atom a následně je bude předkládat uživateli v předepsané formě (viz zadání). Popsané téma mě zaujalo hlavně svým významem pro běžné uživatele. Tato dokumentace shrnuje základní informace nezbytné pro pochopení programového řešení projektu a popisuje jeho implementační detaily. V poslední části je rovněž přítomen uživatelský manuál. Zdroje, které mi byly nápomocny při získávání know-how a vytváření dokumentace jsou uvedeny v sekci reference.

Zdroje důležité pro tvorbu a fungování programového řešení jsou uvedeny v souboru *README*, případně je zřetelně označený úsek kódu, do kterého se promítly. Zatímco této jazyk dokumentace a výstupu je český z důvodu dodržení předepsaného formátu výstupů, soubor *README* a komentáře jsou psány v jazyce anglickém pro zachování konzistence se zbytkem zdrojových kódů. V angličtině jsou také vypisovány chybové hlášky, aby byly jednotné s chybovými zprávami z použitých knihoven, které jsou pouze v angličtině.

Úseky kódů, uživatelských vstupů a specifikací jsou vysázeny pro přehlednost vysázeny v prostředí `verbatim`. Názvy knihoven, souborů a modulů jsou vysázeny *kurzívou*.

1 Přehled teorie

Následující sekce shrnuje nejdůležitější znalosti nutné pro pochopení programového řešení projektu. Ačkoliv by bylo jistě možné uvést více technologií a standardů, které v projektu participují, uvádím zde pouze ty, jež měly přímý vliv na návrh programu, z důvodu udržení rozumného rozsahu dokumentace.

1.1 Uniform resource locator (URL)

Jedná se o podmnožinu identifikátorů URI (uniform resource identifiers), které představují jednotný způsob označování jak fyzických, tak virtuální zdrojů v internetu (např. dokument, služba, číselná hodnota...). URL k této identifikaci přidává ještě informaci o primárním mechanismu přístupu ke zdrojům - tedy, v kontextu sítí jejich umístění (location) [1]. Dále v textu jsou pro lepší srozumitelnost nahrazovány obecnějším pojmem URL adresa nebo pouze adresa. Formát identifikátorů URI vypadá v zobecněné podobě následujícím způsobem (viz [1, str. 16]):

```
scheme ":" hier-part [ "?" query ] [ "#" fragment ]

hier-part   = "//" authority path-abempty
              / path-absolute
              / path-rootless
              / path-empty
```

Pro tento projekt budeme uvažovat, že `scheme` může být nahrazeno za jeden ze řetězců `http://`, `https://` nebo `file://`. Z tohoto důvodu můžeme část `hier-part` zjednodušit na:

```
hier-part   = "//" authority path-abempty
              / path-absolute
```

Tato pravidla jsou totiž specifická pro podporovaná schemata (viz [4, str. 17] a [7]). Znalost formátu adres je důležitá pro jejich analýzu, kterou musíme provést abychom získali z URL adresy jednotlivé části. Ty totiž potřebujeme pro navázání spojení s požadovanými servery a adresování dokumentů s novinkami.

1.2 Syndikace obsahu prostřednictvím RSS 2.0 a Atom

V obou dvou případech se jedná o formáty dokumentů založených na značkovacím jazyku XML. Nejčastějším účelem těchto dokumentů je informování uživatele o novém obsahu například na webových stránkách (zprávy, aktuality, příspěvky).

Uživatelé by nový obsah na internetu mohli samozřejmě odebírat i naivním způsobem – za použití nějakého indexu s odebíranými URL (např. záložek v prohlížeči), který by položku po položce manuálně kontroloval. Díky formátům RSS a Atom se tento proces pro uživatele výrazně zjednodušuje. Uživatelé si pouze udržují seznam adres odkazující na webové služby (respektive dokumenty v těchto formátech). Služby samotné pak rozhodují jaké informace do nich umístí, aby na ně uživatele upozornily. V případě potřeby si je uživatel stáhne pomocí jiné služby (např. specializovaným programem, modulem v prohlížeči nebo jinou webovou službou). Tento způsob publikování obsahu třetí stranou, je označován jako syndikace obsahu [8].

Často se zkratka RSS (Realy simple syndication) nesprávně používá jako označení pro všechny typy formátů určených k syndikaci obsahu. Ve skutečnosti jde všek pouze o jednu rodinu těchto formátů, z nichž nejaktuálnějším je právě RSS 2.0 [3]. Pro jednoduchost se v této dokumentaci často uvádí pouze zkratka RSS bez označení verze, kterou je v těchto případech implicitně myšlena verze 2.0. Alternativou k RSS je Atom., jenž je opět dialektem XML. Formální specifikaci tohoto formátu je možné nalézt zde [9].

Jako u jiných dialektů XML, tak i u RSS 2.0 a formátu Atom, se data strukturují pomocí značek, tzv. tagů. Kromě textu určeného k přímému zobrazení zde tedy nacházíme velké množství metainformací, které datům obsaženým v dokumentu dávají sémantický význam nebo určují formátování. Tudíž například jinak obyčejný text ohraničený značkami `<title></title>` a zanořený do prvku `<entry>` dostává význam názvu příspěvku. Které struktury nás v dokumentech zajímají nepřímo uvádí zadání v podobě informací, jež mají být uživateli vypsány.

Ačkoliv je lze teoreticky přenášet skrze různé aplikační protokoly, v rámci toho projektu budeme uvažovat pouze cestu skrze protokol HTTP(S), případně čtením ze souboru.

1.3 Hypertext transfer protocol (HTTP)

Jde o aplikační protokol určený ke komunikaci mezi klientem (např. webový prohlížeč) a serverem. Komunikace začíná na straně klienta, který vyšle požadavek serveru. Požadavek obsahuje metodu (co serveru požaduje), cestu, verzi protokolu, hlavičky s dodatečnými informacemi, případně zprávu. Server klientovy odpovídá prostřednictvím HTTP odpovědi, která se sestává ze hlaviček a zprávy. Zpráva může obsahovat data v různých formátech.

Formát zprávy může být specifikován v hlavičce `Content-Type`, což klientovi umožní vybrat vhodný způsob vizuální reprezentace přijatých dat. HTTP servery typicky naslouchají na portu 80 (viz iana.org) a jako transportní protokol obvykle využívá TCP, ale jako u jiných technologií, i zde existují alternativy. Příkladem je například transportní protokol QUIC využívaný HTTP/3, který je postaven nad UDP ([6]). Protože však není nutné z pohledu funkcionality programového řešení ani z pohledu zadání využívat nejnovější verze HTTP, omezíme se na používání verzi 1.0. Ta by měla být s následujícími verzemi zpětně kompatibiln za předpokladu, že klientský program podporuje hlavičku `Host` [5]. Využívání novějších verzí by naopak vedlo ke značným komplikacím při zpracování odpovědí (např. kvůli pipeliningu na straně serveru).

V kontextu tohoto projektu bude HTTP protokol důležitý pro přenos Atom/RSS dokumentů, a proto jeho podpora programem bude jistě zásadní.

1.4 Transport layer security (TLS)

Protože se s rozšířením Internetu začaly objevovat další a další hrozby, bylo nutné vytvořit nová zabezpečená řešení. Jedním z přístupů bylo přidání dodatečné bezpečnostní vrstvy mezi vrstvu transportní a aplikační. Funkci této bezpečností vrstvy zajišťoval zprvu SSL protokol, jež umožnil vytvořit spojení, která se z pohledu aplikace chovala nejen jako spolehlivé[1], ale také zabezpečené kanály pro přenos dat. Nástupcem SSL je TLS protokol. Oním zabezpečením rozumíme následující vlastnosti (viz [10]):

- integrita – data nemohou být během přenosu libovolně modifikována aniž by to nebylo možné zjistit
- důvěrnost – data je možné vidět v nezašifrované podobě pouze na koncových stanicích komunikace
- autentizace – minimálně server musí vždy prokázat svoji identitu

Protokol HTTP obohacený o tuto bezpečností vrstvu je označován jako HTTPS a server podporující tento protokol obvykle naslouchá na portu s číslem 443[2]. Kompletní fungování TLS je možné prozkoumat například zde [10]. Pro tento projekt bude nejdůležitější znalost úlohy digitálních certifikátů v tomto protokolu.

Nezbytnou úlohu mají v TLS principy asymetrické kryptografie, v níž je nezbytná výměna veřejných klíčů. Certifikáty jsou důkazem, že veřejný klíč, který klient při ustavení TLS spojení obdržel patří skutečně serveru, s nímž chce komunikovat. Digitální certifikát totiž asociuje veřejný klíč a informace o jeho držiteli [2, str. 9].

Pravost certifikátu zaručuje podpis certifikační autority (zkráceně CA). Aby nebylo nutné. Klientská aplikace však musí být schopna ověřit také podpis této certifikační autority, jež může být pro klienta zcela neznámá. To zajišťuje certifikát vydaný jinou certifikační autoritou. Vzniká tak hierarchie certifikačních autorit, jež umožňuje, aby si klient udržoval certifikáty pro relativně malé množství CA na základě, kterých může již samostatně ověřit obdržených poskytnutých certifikátů. Ty jsou obvykle uloženy ve formátu X509 [2].

Zadání požaduje, aby výchozí cesty k těmto certifikátům, které jsou definované v systému klienta, mohl uživatel prostřednictvím argumentů programu měnit.

1.5 Shrnutí

Znalost pojmů v této kapitole umožňuje správně provést klíčové akce, které tvoří nejpodstatnější části programového řešení projektu, a sice:

- adresování dokumentu v RSS/Atom formátu (URL)
- stažení dokumentu dokumentu ze zdroje v Internetu (HTTP a TLS)
- syntaktickou analýzu stažených dokumentů (RSS 2.0 a Atom)

Jejich vhodnou implementací a doplněním o další pomocné procedury získáme výsledný program, který by měl splňovat všechny požadované standardy.

[1] Jedním z požadavků TLS a SSL vrstev je použití spolehlivého transportního protokolu

[2] `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt`

2 Odevzdané soubory

V archivu, který byl vložen do informačního systému, byly odevzdány tyto soubory:

- *manual.pdf* – dokumentace projektu
- *README* – stručné informace o projektu (jeho knihovní závislosti, příklady spuštění apod.)
- *tests* – adresář s testovacími případy
- *tests_serverside* – adresář pro testovací server s XML dokumenty a PHP skripty
- *feedreadertest.sh* – testovací skript (viz kapitola Testovací skript)
- *Makefile* – definice cílů pro program *make*
- *feedreader.c, feedreader.h* – zdrojový kód a hlavičkový soubor hlavního řídícího modulu programu (obsahuje funkci `main`)
- *common.c, common.h* – modul s nízkoúrovňovými funkcemi zajišťujícími alokaci paměti a práci s některými datovými strukturami
- *cli.c, cli.h* – modul zajišťující komunikaci s uživatelem (vyjma výpisu novinek)
- *url.c, url.h* – modul zajišťující zpracování a validaci adres URL
- *http.c, http.h* – modul obsahující funkce zodpovědné za HTTP(S) komunikaci, včetně analýzy a zpracování odpovědí od serverů
- *feed.c, feed.h* – modul zodpovědný za analýzu dokumentů XML v Atom a RSS formátu

3 Popis implementace

Hlavním cílem implementace programu bylo vytvořit co nejrobustnější program, který bude vhodně reagovat na neočekávané stavy. To bylo ostatně i jedním z požadavků zadání projektu. K tomu mi výrazně pomohla modulární architektura, jež se pozitivně projevila i v případech, kdy bylo nutné některé části programu upravovat.

3.1 Poznámka k terminologii

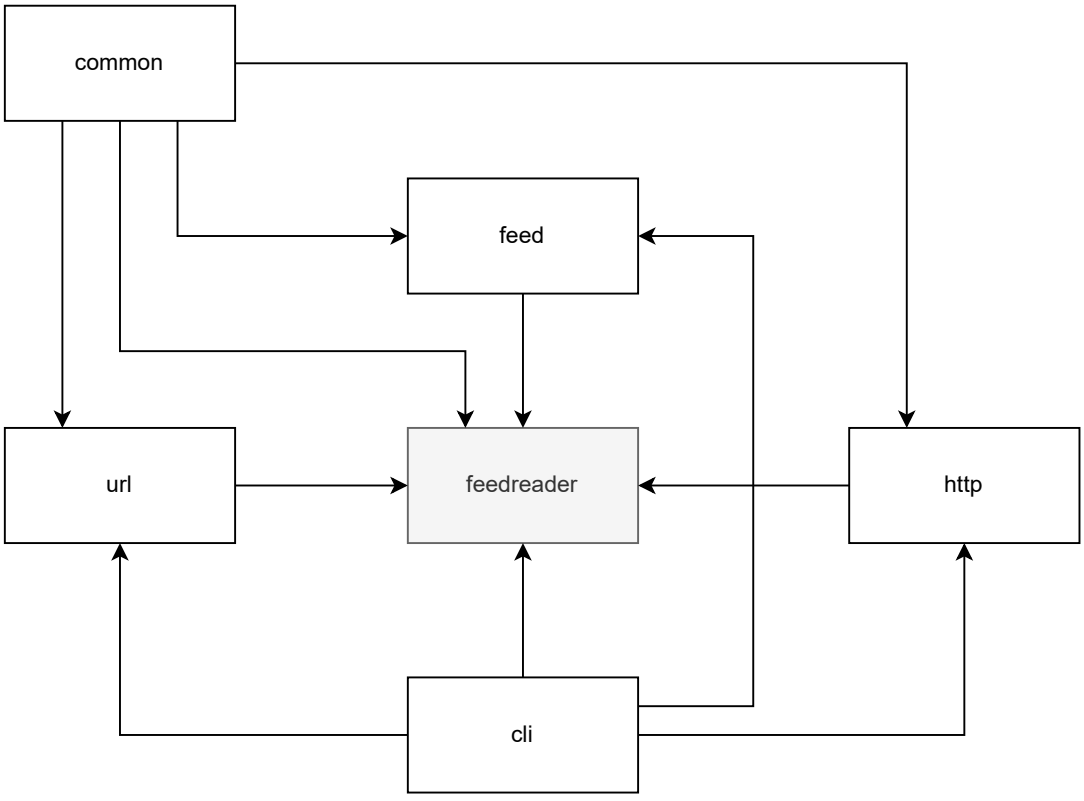
V kapitole je často používáno slovo proces pro označení části programu, která je zodpovědná za provedení určitých dílčích úkonů. Nejedná se tedy o proces v kontextu operačních systémů.

3.2 Použité technologie

Pro implementaci byl zvolen jazyk C. Kromě standardních knihoven jazyka C (jejich kompletní seznam je k nalezení v README) jsou použity doporučené knihovny ze zadání - *OpenSSL* pro navázání zabezpečeného spojení a *libxml2* pro syntaktickou analýzu dokumentů ve formátu XML (tedy v tomto případě konkrétně Atom a RSS 2.0). Potřebné know-how a standardy jsou popsány v 1.

3.3 Architektura programu

Program *feedreader* je členěn do modulů, které spolu komunikují prostřednictvím rozhraních definovaných v hlavičkových souborech. Tato architektura mi umožnila snadno měnit moduly programu bez nutnosti zásahu zásahu do jiných (za dodržení stejného rozhraní). Architekturu programu a vzájemné závislosti jednotlivých znázorňuje následující schéma:



Obrázek 1: Architektura programového řešení. Obdélníky představují moduly. Hrany znamenají závislost modulů. Modul, u kterého hrana začíná, poskytuje funkcionalitu modulu na konci hrany.

3.4 Popis fungování programu

3.4.1 Zpracování uživatelských vstupů

Modul `feedreader` je hlavní řídící jednotkou. Zde, konkrétně ve funkci main, získává program řízení po jeho spuštění a také končí provádění a rodičovskému procesu návratový kód. Jak je již obvyklé u programů v jazyce C, funkce main dostává dva parametry – pole s argumenty programu a počet položek v tomto poli. Tento formát je argumentů programu je nevhodný pro další postup. Z tohoto důvodu jsou nejdříve argumenty zpracovány volání funkce `parse_opts` z modulu `cli`.

Implementace této funkce záměrně nepoužívá knihovní funkci `get_opt` a jí podobné, aby bylo možné zadávat přepínače programu i ve tvaru např. -Tua. Funkce vrací chybový kód jako návratovou hodnotu a pomocí výstupního parametru vrací také speciálně navrženou datovou strukturu `settings_t`.

Tato datová struktura obsahuje sadu proměnných, jež buď to nabývají pouze hodnot `true`/`false` – v případě přepínačů bez parametrů nebo mohou obsahovat ukazatel na argument použitého přepínače (příp. hodnotu `NULL` přepínač použit vůbec nebyl). Navíc je zde přítomen také ukazatel na řetěz, který se mezi argumenty vyskytoval bez přepínače – jde o URL adresu zdroje (viz Uživatelský manuál). Struktura `settings_t` je abstrakcí argumentů programu, kterou je možné snadno předat dalším funkcím prostřednictvím příslušného parametru a můžeme z také ní získat požadovaná data, kdykoliv je to zapotřebí.

Uvedená struktura bude dále v textu označována jako nastavení (programu). Další fází je kontrola platnosti nastavení. Je zjištěno, zda není použita neplatná kombinace přepínačů nebo jestli uživatel nepoužil přepínač `-h` nebo jeho delší verzi `--help`. Při volbě tohoto přepínače je na standardní výstup vypsána nápověda a program je ukončen aniž by cokoliv dalšího zpracovával. Neplatná kombinace přepínačů zahrnuje v případě tohoto programu pouze jediný případ, a sice použití přepínače -f spolu se specifikováním URL. Pokud uživatel tuto kombinaci použije je vypsána chybová hláška podle mechanismu popsaného v kapitole Ošetření chybových a neočekávaných stavů a program je ukončen.

V ostatních případech nastává příprava seznamu URL adres, ze kterých se požaduje stažení novinek. Uživatel tento seznam může specifikovat dvěma způsoby – buď to uvedením jedné adresy coby argumentu programu bez přepínače nebo textovým souborem v předepsaném formátu, který může obsahovat libovolný počet adres. Cesta k tomuto souboru je uvedena za přepínačem `-f` při spouštění programu v příkazové řádce.

3.4.2 Analýza souboru s adresami

Soubor s adresami je zapotřebí samozřejmě z analyzovat a jeho obsah přenést do datových struktur vhodných pro další provádění programu. Je proto čten znak po znaku a ukládán do dynamicky alokovaného pole, reprezentováno datovým typem `string_t` z modulu `common`. Čtení souboru se poté řídí následujícími pravidly (v souladu se zadáním projektu):

- Každý neprázdný řádek obsahuje jednu URL adresu. - Bílé znaky jsou při čtení ignorovány (v platných URL adresách se v nezakódované podobě nemohou nacházet podle RFC..., nedochází tak ke ztrátě žádné informace). - Pokud je prvním platným (nebílím) znakem na řádku znak #, je celý řádku považován za komentář a je ignorován (pokud se znak # nachází na jiné než na první pozici v řádku je uložen jako součást adresy).

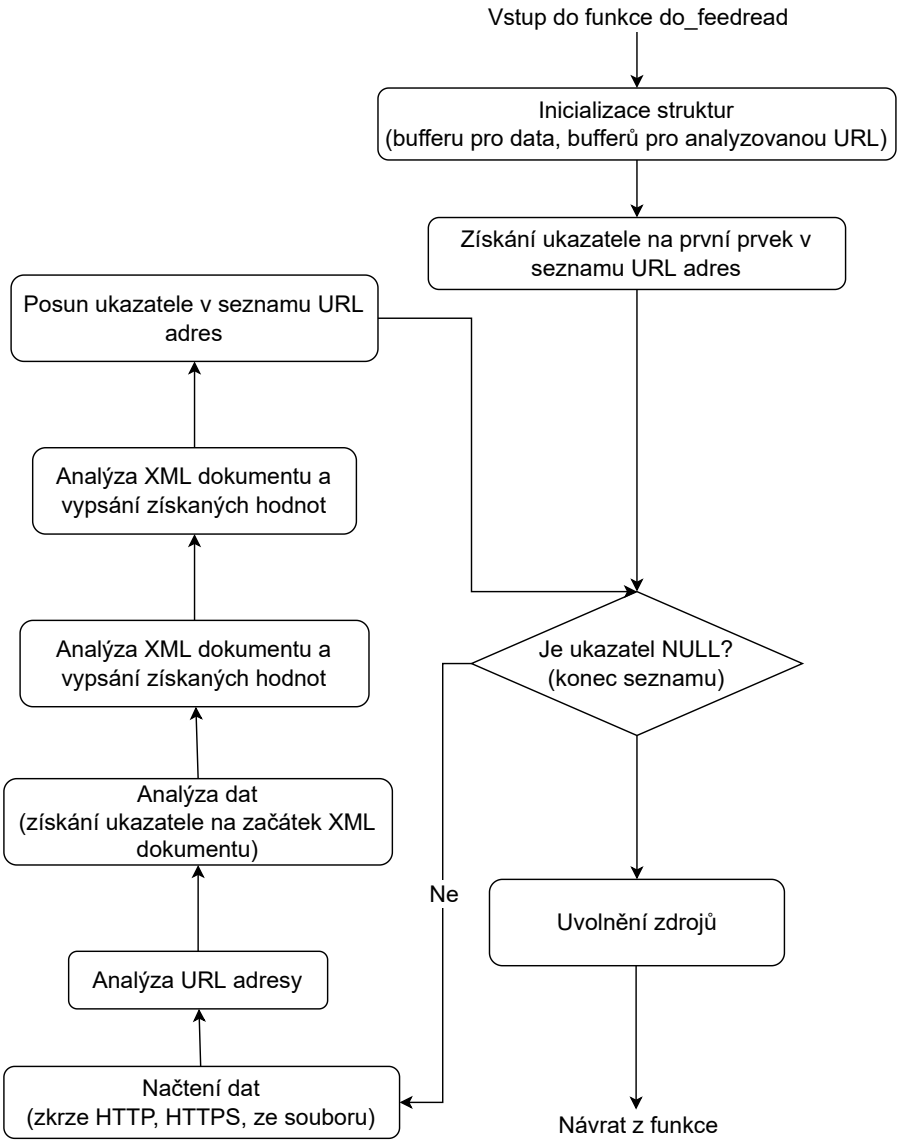
Proces analýzy zajišťuje funkce `parse_feedfile`. Platnost uvedených adres v souboru není při čtení nijak kontrolována a jednoduše se z těchto, nyní už dynamicky alokovaných řetězců v paměti, vytvoření jednosměrný spojový seznam (struktura `list_t` z modulu `common`. Tato struktura je následně propagována dalším funkcím programu v rámci jejich argumentů. Není v rámci souboru nalezen žádný platný řetěz znaků, který by mohl být do seznamu uložen, je tato skutečnost oznámena uživateli prostřednictvím varování na standardní chybový výstup.

Pokud se uživatel rozhodne specifikovat zdrojovou adresu přímo, je proces jednodušší.Vytvoří vždy pouze jednoprvkový spojový seznam. Přestože by zdánlivě tvorba seznamu v tomto případě nebyla nutná, tímto způsobem je zajištěno jednotné rozhraní pro zbytek programu. Další fáze jeho zpracování totiž pouze sekvenčně prochází takto získaný seznam a nad každým řetězcem v tomto seznamu provádí níže popsané procedury. Tato hlavní smyčka programu se nachází ve funkci `do_feedread` (viz 2).

3.4.3 Kontrola adres a jejich analýza

Jelikož URL adresy jsou příliš komplexní pro knihovní funkce zajišťující síťovou komunikaci, které vyžadují uvedení pouze doménového jména nebo IP adresy požadovaného serveru (a čísla portu služby), je nutné provést rozbor URL adres. Cílem tohoto rozboru není jen oddělit informace obsažené v adrese, ale také pokud možno co nejlépe zkontrolovat její platnost. Tento úkol je jednou ze zodpovědností modulu *url*.

Funkce `parse_url` exportovaná tímto modulem prostřednictvím aparátu standardní knihovny *regex.h* prochází adresu a vyhledává její části podle zjednodušeného formátu URL adresy uvedeného v Teoretické části. Vyhledávání jednotlivých sekcí URL je založeno na použití regulárních výrazů, jež jsou sestaveny na základě popisu formátu těchto sekcí



Obrázek 2: Algoritmus funkce `do_feedread`, která obsahuje smyčku pro zpracování všech uživatelem specifikovaných adres.

ve standardu [1]. Oním zjednodušením formátu je například přijímání pouze některých schémat – konkrétně `file://`, `http://` a `https://`, protože jiné způsoby získání dat než čtení ze souboru a protokol HTTP(S) program nepodporuje.

Programové řešení má ambice být robustní a uživatelsky přívětivé. Z tohoto důvodu některé chybějící části adresy nevyústí rovnou v ukončení programu (resp. v ukončení zpracování daného zdroje). Ošetřeno je chybějící schéma, za které je dosazeno výchozí schéma https:// (uživatel je obeznámen s touto situací prostřednictvím varování). Dále je chybějící cesta v URL adrese doplněna výchozí hodnotou /. Program se také vypořádá s autentizační částí URL, která se dnes již nepoužívá. V tomto případě je tato část ignorována a uživateli je vypsáno varování (podle doporučení v [4]).

Při použití schématu `file://`, je striktně vyžadován správný formát URL adresy typické pro toto schéma. V případě že chybí nějaká z povinných částí, kterou nebylo možné nahradit nebo je v adrese přítomna nějaká neidentifikovatelná sekce (svým formátem neodpovídá očekávané části adresy), je vypsáno chybové hlášení a zpracování příslušné adresy je ukončeno.

Program provádí také tzv. percent encoding. Jde o způsob zakódování znaků, které nejsou standardem pro URL povoleny nebo mají v daném kontextu jiný význam. Uživatel tak může v rámci cesty v URL použít např. znaky národních abeced a program se postará o jejich konverzi, aby byla následně vytvořená HTTP zpráva korektní. Omezením však je, aby všechny uživatelské vstupy byly kódovány pomocí UTF-8 [1, str. 20]. Konverzi kódování vstupů na UTF-8 totiž program automaticky neprovádí.

3.4.4 Získání dat

Data je možné získat vícero způsoby v závislosti na použitém schématu. Nejběžnější je použití schématu `https://`, resp. `http://` a data tak získat skrze modul http, jenž se stará o komunikaci HTTP servery. Alternativou je použití schematu `file://`, které data čte ze souboru daného absolutní cestou* a tím tento modul obchází, což je výhodné zejména pro účely testování (můžeme se v některých testech zaměřit pouze na analýzu uživatelských vstupů a XML dokumentu).

Modul HTTP na základně schématu provede rozhodnutí, zda navázat zabezpečené spojení nebo nezabezpečené spojení. V obou dvou případech je logika komunikace stejná, liší se pouze proces počátečního spojení. U HTTP se pouze nastaví doménové jméno serveru a portové číslo, na kterém HTTP server naslouchá. Obě dvě informace jsou dostupné díky analýze v předchozím kroku. Následuje pokus o ustavení spojení skrze funkci z knihovny *OpenSSl* `BIO_do_connect`.

U HTTPS je postup složitější. Před ustavením spojení dojde k nastavení souboru, resp. složky, s certifikáty, jež se mají použít k ověření identity serveru. V případě, že při spuštění programu ani jedna z cest zadána nebyla, jsou použity výchozí cesty knihovny *OpenSSL*, případně cesty definované v proměnném prostředí systému. Možné je také specifikovat jako složku s certifikáty (parametr `-C`), tak i soubor s certifikáty (`-c`). V tomto případě je hledán certifikát v souboru a poté ve složce. Protože použitá knihovní funkce `SSL_CTX_load_verify_locations` vyhledává certifikáty ve složce prostřednictvím hashování jména držitele certifikátu, může být nutné nad složkou spustit nástroj *c_rehash*[3], který ve složce vytvoří správně pojmenované symbolické odkazy.

Následuje pokus o ustavení spojení a poté ověření certifikátu. Po těchto procedurách je z pohledu dalšího zpracování již vytvořen spolehlivý kanál pro přenos dat. Program jakožto klient zahajuje komunikaci požadavkem ve tvaru:

Server mu odpoví HTTP odpovědí, která je v cyklu nahrávána do bufferu, dokud není uložena celá (buffer je dynamicky rozšířen na potřebnou velikost). Následně je provedena analýza odpovědi – je zkontrolován formát (zda jsou v odpovědi přítomny hlavičky a zda jsou končeny sekvencí `\r\n\r\n`), kód HTTP odpovědi, je nalezen začátek dat obsažených ve zprávě a z hlaviček jsou extrahovány některé další důležité informace. Těmi je alternativní URL (hlavička `Location:`) pro účely přesměrování a MIME typ obsažené zprávy (`Content-Type:`).

Podporované hodnoty hlavičky `Content-Type` jsou definovány v souboru *http.h*. Pokud je nalezen nepodporovaný typ, je zpracování URL ukončeno a důvod je vypsán uživateli. Tímto způsobem jsou ošetřeny situace, kdy adresa neodkazuje a XML dokument ale např. na HTML stránku. V případě že je kód zprávy ve tvaru `3xx` je jedná se o přesměrování.

Z hlavičky `Location:` jsou získány data o nové destinaci a ta jsou použita pro vytvoření nové položky ve spojovém seznamu URL adres (nachází se hned za právě zpracovávanou položkou). V případech, kdy je kód odpovědi jiný než `2xx`, je zpracování dané odpovědi ukončeno a uživatel je informován chybovým hlášením.

3.4.5 Zpracování získaných dat

Odkaz na začátek XML dat v HTTP odpovědi (příp. odkaz na začátek bufferu s obsahem souboru) je propagován do modulu feed, který zajišťuje rozbor RSS a Atom dokumentů spolu s formátovaným výpisem informací v nich obsažených. I přes možné chyby v těchto dokumentech se analyzátor snaží načíst pokud možno jeho celý obsah. V nastavení knihovny *libxml2*, která analýzu zajišťuje, jsou použity možnosti, jež tuto toleranci chyb zajišťují. Tolerance chyb je nicméně možná pouze do určité míry (např. chybějící značka `<channel>` v RSS dokumentu není během analýzy tolerován).

Zda bude pro analýzu použit analyzátor pro Atom nebo RSS určuje očekávaný typ dokumentu získaný v hlavičce zprávy a název kořenového elementu v XML.

Hlavními kritérii, podle kterých jsou požadované informace v dokumentech vyhledány, jsou název a úroveň zanoření struktur (ta jsou určena na základě specifikace formátů). Pokud jsou nalezeny redundantní značky s hledanými informacemi je vždy uložena poslední z nich. Jedinou výjimkou je značka `<link>` ve formátu Atom. V tomto případě má přednost značka, která má atribut `rel` nastavený na hodnotu `alternate` nebo tento atribut nemá (implicitně je `rel` nastaveno na `alternate`)[4].

Data o zdroji a jednotlivých příspěvcích jsou ukládána do struktury `feed_doc_t`, což je opět jednosměrně vázaný seznam. Bez ohledu na nastavení programu jsou pro jednoduchost sbírána všechna data (jména autorů, asociované adresy i čas poslední aktualizace) a konkrétní uživatelské nastavení (přepínače) pouze určují, zda se ve výsledném výpisu zobrazí.

3.4.6 Výpis hodnot

Získaná data jsou poté vypsána ve formátu definovaného zadáním. Nejdříve je vypsán název zdroje. Pokud chybí, je nahrazen značkou `<neznamy zdroj>`. Pokud by tato značka vypsána nebyla, mohl by si uživatel snadno opticky spojit

[3] Viz dokumentace OpenSSL `https://www.openssl.org/docs/manmaster/man3/SSL_CTX_load_verify_locations.html`

[4] Toto opatření bylo zavedeno kvůli uživatelské přívětivosti – jistě by nebylo vhodné primárně zobrazovat link např. na přílohu novinky (`enclosure`) místo adresy na novinku samotnou

seznamy ze dvou zdrojů. Poté je sekvenčně procházen spojový seznam a jsou vypisována data obsažená v jeho položkách, které představují jednotlivé příspěvky v dokumentu s novinkami (u RSS odpovídá struktuře `<item>...</item>` a u formátu Atom `<entry>...</entry>`).

Jestliže chybí název příspěvku je nahrazen značkou `<nepojmenovano>`. Výpisy dalších informací záleží pak už na konkrétním uživatelském nastavení (použitých přepínačích) a na tom, zda je bylo možné najít. Vždy jsou však vypisovány v pořadí Autor, URL a Aktualizace.

Pokud nebylo možné nějakou informaci najít nebo nebyl její výpis aktivovaná, vůbec se pod daným názvem příspěvku nezobrazí. Při použití alespoň jednoho z přepínačů `-T`, `-u` nebo `-a` je za každým příspěvkem vypsán prázdný řádek (dle požadavků zadání). V případě, že byly adresy specifikovány souborem, jsou výpisy z jednotlivých zdrojů také zakončeny prázdným řádkem. Při použití výše uvedených přepínačů a souboru s adresami je tedy výpis zakončen dvěma prázdnými řádky.

3.5 Ošetření chybových a neočekávaných stavů

3.5.1 Chybová hlášení a varování

Výpisy na standardní chybový výstup lze rozdělit na varování a chybová hlášení (a případně také ladící výpisy, které by však měly být v odevzdané verzi deaktivovány). Informace obsažené v obou typech výpisů jsou v angličtině, aby byly konzistentní s chybovými hlášeními knihovních funkcí, která jsou v některých případech použita pro detailní specifikaci chyb (viz Úvod). Týká se to například situace, kdy není možné ověřit důvěryhodnost certifikátu. O výpisy chybových hlášení se starají výhradně funkce v modulu *cli*.

Účelem varování je tomto programu informovat uživatele o provádění některých akcí, které přímo nemusí vyplývat ze sémantiky fungování programu nebo ze zadání. Například, v případě chybějícího schématu URL, je doplněno výchozí schéma (`https://`) a vypsáno varování, aby uživatel nebyl příliš překvapen. Dalším příkladem je varování oznamující přesměrování. Varování je možné vypnout triviálním zakomentování definice makra `CLI_WARNINGS`.

Chybová hlášení oznamují uživateli událost, která neočekávaně ukončila jednu z fází provádění programu. Jsou rozdělená do tříd podle typu chyby z nichž každou označuje určitý chybový kód (viz 3.5.2). Každé chybové hlášení má formát:

```
feedreader: <třída chyby>: <konkrétní chybové hlášení>
```

Funkce pro výpis varování a chybových hlášení téměř vždy volány funkcemi, kde daná událost vzniká. Výjimku tvoří nízkoúrovňové funkce z modulu *common*, u nichž je tato zodpovědnost delegována na funkci volající. Funkce volající se tedy už o výpis starat nemusí a věnuje se pouze zpracování chyby nebo její propagaci.

3.5.2 Propagace chybových kódů

Při bližším zkoumání signatury funkcí si je možné všimnout, že většina z nich vrací hodnotu typu `int`. Jedná se o chybový kód, který označuje třídu dané chyby. Tímto způsobem se informace o chybě šíří napříč programem.

Součástí hlavičkového souboru modulu *cli*. Je výčtový typ, který obsahuje makra pro všechny povolené chybové kódy. Tyto kódy se používají jak interně, pro komunikaci jednotlivých funkcí mezi sebou, tak i pro návratové hodnoty celého programu. Speciálním chybovým kódem je `SUCCESS` (s hodnotou 0), který signalizuje, že provádění funkce proběhlo úspěšně. Význam dalších chybových kódů je k nalezení v uživatelském manuálu.

Pokud dojde v rámci některé funkce k jakékoliv chybě, je chybový kód propagován do funkce, která provedla volání. Ta propaguje kód do své volající funkce a tak dále. Řetězec končí buďto předáním kódu do funkce `main` (ta program ukončí s daným návratovým kódem) nebo předáním kódu do funkce `do_feedread`, která v cyklu zpracovává spojový seznam URL adres. V tomto případě je kód uložen do spojového seznamu k příslušné adrese, kde čeká na zpracování zbylých adres (v souladu se zadáním).

Poté je řízení předáno opět funkci `main`, jež `do_feedread` volá, a ta průchodem seznamu nalezne první nenulový návratový kód, který vrátí jako návratovou hodnotu celého programu. Nastane-li tedy při provádění programu více chybových stavů, které nezpůsobí okamžité ukončení programu, je navrácen kód označující první z těchto stavů.

4 Rozšíření a omezení

Jak je již možné vypozorovat z popisu výše, implementována také funkcionalita, o které se zadání nezmiňuje nebo přímo z něj přímo nevyplývá. Tato rozšíření jsou následující:

- podpora schematu `file://` (implementováno spíše pro testovací účely)
- podpora přesměrování
- podpora percent encoding (pokud jsou data na vstupu v kódování UTF-8)

Omezením jsou závislosti na uvedených knihovnách – *OpenSSL*, *libxml2* a standardních knihoven jazyka C, které jsou potřebné pro překlad. Za omezení by také mohla být považována nutnost použití nástroje *c_rehash* nad složkou s certifikáty (viz Získání dat) nebo nutnost kódování UTF-8, pokud cesty v poskytnutých URL adresách nejsou korektní (dle standardu [1]). Kromě těchto detailů však program žádné omezení nemá.

5 Testování

5.1 Testovací skript

Pro testování funkcionality programu byl vytvořen skript pro interpret bash s názvem *feedreadertest.sh*. Skript rekurzivně prochází adresář *tests*, kde se mohou nacházet buď to přímo testovací případy nebo další adresáře s testovacími případy – testovací sady.

Testovací případ je reprezentován adresářem s textovým souborem *test*, v němž je na na prvním řádku popis testovacího případu (v tomto případě je uvozený znakem `#`) a na druhém seznam argumentů programu pro daný testovací případ. Druhou možností je specifikovat argumenty hned na prvním řádku textového souboru (v tomto případě není první řádek uvozen `#`).

Dále je v rámci testovacího případu možné specifikovat očekávanou návratovou hodnotu programu (v souboru *ret*) a očekávaný výstup programu (*out*). Skript vytváří dočasné soubory pro uchování standardního výstupu (*out.tmp*) a standardního chybového výstupu programu (*err.tmp*). Standardní výstup je porovnán s referenčním výstupem pomocí program diff. Volitelně je také možné při spouštění skriptu použít tyto přepínače:

- `-m` program se během testování spouští pod nástrojem valgrind (musí být v daném systému nainstalován)
- `-v` program uchovává dočasné soubory (např. pro účely ladění)
- `-t nazev_testu` bude proveden pouze testovací případ, reprezentovaný adresářem s názvem *nazev_testu*

Název programu, dočasných souborů apod. je možné změnit modifikací definic proměnných na začátku zdrojového kódu skriptu. Skript produkuje výstup ve formátu:

```
Sat Oct 29 13:22:02 CEST 2022
Running tests of feedreader:
tests/*:
tests/offline/*:
001:    [ PASSED ]      Program without any argument
002:    [ PASSED ]      Specified both feedfile and URL
003:    [ PASSED ]      Empty feedfile
...
025:    [ FAILED ]      Regular Atom file with other info, HTTPS
Why:    Different outputs! (use -v to preserve out.tmp file)
...
032:    [ PASSED ]      Redirection test (needs MAX redir num at least 2)
033:    [ PASSED ]      Redir test with rel url (needs MAX_REDIR_NUM at least 3)
Summary: 33/33 tests passed
```

Na prvním řádku je vždy datum a čas spuštění testů. Následuje výpis výsledků jednotlivých testovacích případů. Vždy je vypsán aktuální adresář (např. `tests/*`) a za ním následují výsledky testovacích případů obsaženým v tomto adresáři. Výsledek je vypsán na jednom řádku, který začíná názvem testovacího případu, následuje výsledek testu (passed/failed) a název testu. V případě, že byl test neúspěšný za řádkem s výsledkem následuje řádek s důvodem neúspěchu.

Výpis končí řádkem, na němž je vypsán počet testů, jež byly úspěšně provedeny, a celkový počet všech provedených testů.

5.2 Testovací server

Jelikož se volně dostupné zdroje RSS a Atom novinek ze své podstaty v čase mění, bylo potřeba pro systematické testování zajistit konstantní vstup pro program ze strany serveru. K tomuto účelu mi posloužil můj studentský WWW server dostupný na `www.stud.fit.vutbr.cz/~xdvora3o/ISA`, jenž podporuje jak HTTP, tak HTTPS spojení (na standardním portech). Testovací skript samozřejmě umožňuje testování i na reálných serverech poskytujících novinky, ale u těchto případů lze spolehlivě kontrolovat pouze návratový kód programu a případné úniky paměti.

5.3 Průběh testování

Program byl otestován celkem na třech systémech: Ubuntu 20.04.5 (v rámci WSL), FreeBSD 13.1 (eva.fit.vutbr.cz) a CentOS Linux 7 (merlin.fit.vutbr.cz).

Testovací případy jsou dle svého charakteru rozděleny do adresářů:

- *offline* - testování nevalidních uživatelských vstupů, zpracování souboru s adresami URL a analýzy souborů v RSS/Atom formátu za obejití modulu *http* pomocí schématu `file://`
- *online* - systematické integrační testování všech modulů programu většinou prostřednictvím testovacího serveru (včetně testování přesměrování pomocí PHP skriptu)
- *real* - testování na reálných zdrojích novinek dostupných v Internetu

V době odevzdání všechny testy na všech výše uvedených strojích **procházejí** (včetně testů se nástrojem valgrind). Testovací skript byl ve všech případech spouštěn příkazem:

```
./feedreadertest.sh && ./feedreadertest.sh -m
```

Ukázky testovacích případů z kategorie *real* jsou uvedeny v rámci 6.5.

6 Uživatelský manuál

6.1 O programu

Program *feedreader* umožňuje přečíst novinky ve formátu RSS 2.0 a Atom ze zdroje (zdrojů) specifikovaných URL adresou (adresami). Novinky s příslušným názvem zdroje vypisuje formátovaně na standardní výstup.

6.2 Použití programu

Za předpokladu, že se nacházíme v adresáři s přeloženým programem, můžeme ho spustit příkazem ve formátu:

```
./feedreader <-f feedfile | URL>  [OPTIONS]...
```

Povinný je jeden z parametrů `URL` nebo `-f feedfile`. `URL` je URL adresa se schématem `http://`, `https://` nebo `file://`. `feedfile` je cesta k textovému souboru se specifikací URL adres. Na každém řádku tohoto souboru je jedna URL adresa. Prázdné řádky a řádky, kde je prvním z nebílých znaků `#`, jsou ignorovány.

`OPTIONS` představuje následující množinu volitelných parametrů programů:

- `-h`, `--help` vypíše nápovědu a ukončí provádění programu
- `-T` V závěrečném výpisu novinek vypíše čas poslední aktualizace každé novinky (je-li uveden)
- `-u` V závěrečném výpisu novinek vypíše asociovanou URL adresu (je-li uveden)
- `-a` V závěrečném výpisu novinek se zobrazí jméno nebo email autora (je-li uveden)
- `-c certfile` Nastavení cesty k certifikátu, který se má použít pro autentizaci serveru
- `-C certaddr` Nastavení cesty k adresáři s certifikáty, které se mají použít pro autentizaci serveru[5]

6.3 Chybové stavy programu

Program oznamuje chybové a neočekávané stavy chybovými hlášeními a varováními na standardní chybový výstup. Případnou chybu navíc signalizuje příslušným kódem.

6.4 Návratové kódy programu

`0`	Úspěch (provádění programu proběhlo bez chyb)
`1`	Chyba použití programu (chybějící argument(y), neplataná kombinace argumentů apod.)
`2`	Chyba čtení ze souboru
`3`	Chyba v důsledku vložení neplatné URL adresy (např. neplatné schéma)
`4`	Chyba spojení se serverem (zadána neexistující URL adresa, bez připojení k internetu apod.)
`5`	Chyba během komunikace se serverem
`6`	Chyba cesty k souboru (byla zadána neplatná cesta)
`7`	Problém s ověřením certifikátu serveru (např. server se prokázal nedůvěryhodným certifikátem apod.)
`8`	Problém na úrovni HTTP komunikace (např. neplatná odpověď od serveru)
`9`	Problém na úrovni XML dokumentu (např. chybějící kořenový element, špatná verze RSS)
`10`	Interní chyba programu (např. chyba alokace paměti)

Při výskytu více než jedné chyby během provádění programu, je navrácen kód první z nich.

[5] V tomto případě může být nutné nejdříve nad složkou spustit nástroj *c_rehash* pro vznik správně pojmenovaných symbolických odkazů na jednotlivé certifikáty

6.5 Příklady spuštění programu

Tato sekce ukazuje několik příkladových spuštění programu spolu s uvedením jeho výstupu. **Při replikování těchto příkladů se bude pravděpodobně výstup lišit** (kvůli aktuálním novinkám poskytnutých zdrojem). Tři tečky ve výpisu programu značí vynechání obsahu z důvodu úspory místa v dokumentaci.

6.5.1 Jednoduché čtení novinek z RSS zdroje

Příkaz:

```
./feedreader "http://rss.cnn.com/rss/cnn_topstories.rss"
```

Výstup:

```
*** CNN.com - RSS Channel - HP Hero ***
There are moments in history where entire armies suddenly stop fighting...
Zelensky says Russia's mobilized troops are poorly trained and equipped
Russia says it repelled drone attack on Crimea
Ukraine dreads 'darkest winter' as Russia takes aim at power grid
...
Liz Cheney on state of the Republican Party and Donald Trump
Samsung names billionaire scion Jay Y. Lee as chairman
```

6.5.2 Jednoduché čtení novinek z více zdrojů

Příkaz:

```
./feedreader -f "feedfile"
```

Obsah souboru `feedfile`:

```
#Atom source
https://xkcd.com/atom.xml
#RSS sources
https://feeds.acast.com/public/shows/5ea17537-f11f-4532-8202-294d976b9d5c
https://rss.art19.com/apology-line
```

Výstup:

```
*** xkcd.com ***
Encryption
Cool S
Fermat's First Theorem
Bubble Universes

*** Unraveled ***
Introducing Hillsong: A Megachurch Shattered
S5 Ep.5: The Profiler's Dilemma
...
S1 Ep. 2: Fail to the Chief
S1 Ep. 1: David and Goliath
Introducing Unraveled: Long Island Serial Killer

*** The Apology Line ***
Wondery Presents: The Generation Why Podcast
Wondery Presents - The Rewatcher: Buffy The Vampire Slayer
Introducing: The Apology Line
```

Reference

[1] Berners-Lee, T.; Fielding, R. T.; Masinter, L. M.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Leden 2005, doi:10.17487/RFC3986.
URL https://www.rfc-editor.org/info/rfc3986

[2] Boeyen, S.; Santesson, S.; Polk, T.; aj.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, Květen 2008, doi:10.17487/RFC5280.
URL https://www.rfc-editor.org/info/rfc5280

[3] Cadenhead; et al: RSS 2.0 specification (current). Březen 2009.
URL https://www.rssboard.org/rss-specification

[4] Fielding, R. T.; Nottingham, M.; Reschke, J.: HTTP Semantics. RFC 9110, Červen 2022, doi:10.17487/RFC9110.
URL https://www.rfc-editor.org/info/rfc9110

[5] Fielding, R. T.; Nottingham, M.; Reschke, J.: HTTP/1.1. RFC 9112, Červen 2022, doi:10.17487/RFC9112.
URL https://www.rfc-editor.org/info/rfc9112

[6] Iyengar, J.; Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, Květen 2021, doi:10.17487/RFC9000.
URL https://www.rfc-editor.org/info/rfc9000

[7] Kerwin, M.: The "file"URI Scheme. RFC 8089, Únor 2017, doi:10.17487/RFC8089.
URL https://www.rfc-editor.org/info/rfc8089

[8] Nica, I.: Content syndication: What it is a how to et aldo it successfully. Září 2019.
URL https://blog.hubspot.com/marketing/how-to-syndicate-content

[9] Nottingham, M.; Sayre, R.: The Atom Syndication Format. RFC 4287, Prosinec 2005, doi:10.17487/RFC4287.
URL https://www.rfc-editor.org/info/rfc4287

[10] Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Srpen 2018, doi:10.17487/RFC8446.
URL https://www.rfc-editor.org/info/rfc8446