Dokumentace k projektu Čtečka novinek ve formátu Atom s podporou SSL/TLS

ISA - Síťové aplikace a správa sítí

Jan Jileček

Obsah

1. Popis programu

Program arfeed je určen pro stahování informací ze zdrojů ve formátu Atom. Program po stažení požadovaných informací tyto vypíše na standardní výstup. Je možné přidat doplňující parametry, pro určení informací, které chceme vypsat. Program používá Unixové sokety, standardní knihovny jazyka C a C++, knihovnu OpenSSL a libxml2.

2. Návrh aplikace

Program je tvořen z více tříd, mezi které se řadí:

- Arguments – třída zpracovávající argumenty
- IDownload – rozhraní pro HTTP a HTTPS
- HTTP – třída pro http komunikaci, dědí od IDownload
- HTTPS - třída pro ssl komunikaci, dědí od IDownload a HTTP
- Gadgets – třída s různými pomocnými funkcemi
- xmlViewer – třída pro práci s xml obsahem a výpisem Atom feedu
- ISAException – výjimka, dědí od std::runtime_error, vlastní výjimková třída

3. Popis implementace

Main

Nejprve zpracuji argumenty za pomoci třídy Arguments a její funkce parseArgs. Tato funkce používá pro získávání argumentů funkci getopt. Postupně takto do členských proměnných třídy Arguments, zastupované objektem args, načtu všechny požadované argumenty, mezi které se řadí například URL adresa, argumenty pro ovládání výpisu a načítání certifikátů.

Na základě použití nebo nepoužití souboru s url adresami volám funkci DL víckrát, resp. jednou pro každou adresu v souboru, nebo jednou pro jedinou zadanou adresu. Soubor a adresu nelze kombinovat. Stejně tak nelze kombinovat složku s certifikáty a soubor s certifikáty, rozvedu dále.

Běh funkcí řídím pomocí výjimek – když je výjimka zachycena, vypíši na chybový výstup předanou hlášku a ukončím program s chybou. Při použití feedfile jdu na další adresu, hláška se pouze vypíše. Je tu jedna výjimka, a to „redirection complete“. Popíšu také níže.

Pokud při zpracování URL zjistím port 80, pokračuji vytvořením instance třídy HTTP. Jinak pokračuji vytvořením třídy HTTPS a u obou pak invokuji funkci download.

HTTP download

V konstruktoru nastavím potřebné proměnné. První funkce, kterou volám je initiateConnection. Ta má za úkol vytvoření soketu, DNS přiřazení k IP, naplnění potřebných struktur a následné připojení. Může skončit třemi různými výjimkami.

Pokud připojení proběhne v pořádku, pokračuji zasláním požadavku na server. Ve funkci makeHeaders vytvořím řetězec hlaviček, které následně posílám. Ve funkci send_request posílám následující hlavičky

```
"GET " << path  << " HTTP/1.1\r\nHost: " << server << "\r\nUser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:40.0) Gecko/20100101 Firefox/40.0\r\n" << "Accept: */*\r\n\r\n";
```

Následně čekám na odpověď, resp. na příchozí hlavičky, funkce getHeaders. Postupně načítám a zjišťuji z nich tyto informace:

1. Jedná se o validní HTTP odpověď? Pokud ne, testuji ještě na kód 301 a podle toho vytvářím další instance třídy HTTP/HTTPS, tentokrát ale s o jedna menším limitem přesměrování, pro zamezení nekonečné přesměrovací smyčce. Defaultní hodnota je 10. Jinak pokračuji.
2. Jde o chunked nebo je dána pevná velikost? Tedy „transfer-encoding: chunked“ nebo „content-length“.
3. Při kódu 301 ještě zjišťuji Location, tzn. adresu, na kterou budu přesměrován.

Tato funkce také vyhazuje výjimky.

Následně se přesunuji k funkci get_content, která na základě dříve získaných parametrů provádí stahování obsahu, buď chunked, nebo s danou velikostí. Po získání obsahu vracím řetězec obsahující stažený obsah.

Obsah dále ukládám do souboru „output.xml“ a vracím se z funkce.

HTTPS download

V konstruktoru třídy HTTPS vytvářím potřebné proměnné pro inicializaci a práci s certifikáty a ssl komunikací, vše za pomocí funkcí knihovny OpenSSL. Podle toho, zda je použit soubor s certifikátem nebo složka s certifikáty, volám funkci SSL_CTX_load_verify_locations s rozdílnými parametry. Pokud není specifikováno ani jedno ze zmíněných, používám složku /etc/ssl/certs/. Následně inicializuji cert store a certifikáty ověřuji ještě jednou, důkladněji a tentokrát je přímo načítám do proměnné ctx, která je následně použita v kombinaci s ssl v později vytvořeném ssl připojení.

Vytváření spojení je provedeno využitím dědičnosti, totiž zavoláním funkce initiateConnection z třídy HTTP, od které ve třídě HTTPS dědím. Ta vytvoří vše potřebné a já mohu hned pracovat. Ve skutečnosti třída HTTPS používá většinu funkcí ze třídy HTTP, znovu implementuji pouze některé funkce, které vyžadovaly odlišný přístup.

Ve funkci initiateConnection vytvářím nové ssl připojení, při chybě ještě otestuji, zda nešlo chybu s certifikáty a v závislosti na tom vypíšu odpovídající chyby a ukončím provádění.

Hlavičky posílám také funkcí send_request, která je virtuální. Pro HTTP je řešena pomocí funkce send a pro HTTPS pomocí funkce SSL_write, vždy se zavolá odpovídající verze.

HTTP vs HTTPS

Zasílání a přijímání je řešeno virtuálními funkcemi, funkce pro stahování je pro obě třídy totožná, místo SSL_read/recv se volá funkce receive. Tato funkce je volána na základě toho, ze které třídy je právě požadována. Pro HTTPS se provede SSL_read, a pro HTTP se provede recv.

V obou funkcích následně čtu přijatá data a získám hlavičky. Již popsaným způsobem je zpracuji. Následuje také již popsané stahování obsahu a jeho uložení do souboru.

Zpracování XML

Ke zpracování XML používám knihovnu libxml2, v konstuktoru třídy xmlViewer načtu funkcí xmlReadFile soubor „output.xml“. Používám přepínače pro potlačení výpisu varování a chyb. Vyhazuji výjimku v případě neúspěšného načtení. V mojí funkci loadTree používám xmlDocGetRootElement pro načtení kořenového elementu a následně v cyklech zpracovávám XML strom.

Pro všechny děti kořenového uzlu testuji jejich název, a na základě toho se zařizuji. Na nejvyšší úrovni rozlišuji pouze titulek a záznam, v případě titulku vypisuji na standardní výstup a v případě záznamu volám funkci getEntry, která záznam zpracuje na nižší úrovni. Z odkazu na objekt třídy Arguments zpracovávám větve stromu – uzly - a postupně vypisuji na výstup.

Za podrobnější popis stojí implementace zobrazování názvu jako prvního, jelikož v Atom feedu mohou být informace zpřeházeny. Mám vytvořen vektor řetězců, a v případě, že se aktuální uzel jmenuje „title“, vkládám na začátek vektoru, jinak přidávám uzel na konec vektoru.

Pokud některý z uzlů aktivuje výpis řádku navíc (např. v případě více zvolených parametrů nebo čtení z feedfile), tak do vektoru přidávám i prázdný řetězec, vyšší vrstva se pak postará o přidání nového znaku konce řádku za každý prvek vektoru, tzn. i za prázdný řetězec, efektivně vytvářející prázdný řádek.

4. Základní použití a informace o programu

Základní kostra pro použití programu je

arfeed http://tools.ietf.org/agenda/atom | -f feedfile [-c certfile] [-C certaddr] [-l] [-T] [-a] [-u]

Nyní něco k argumentům:

- f feedfile, soubor s jednotlivými URL adresami na jednotlivých řádcích. Podpora komentářů, řádky začínající na #
- c certfile, soubor SSL certifikátu
- C cesta ke složce s certifikáty
- l vypíše pouze informace o nejnovějším záznamu
- T vypíše časový údaj záznamu
- a vypíše autora záznamu
- u vypíše URL záznamu

5. Detaily

Při použití jakéhokoliv přepínače upravujícího výpis XML mají být podle zadání odděleny záznamy jedním řádkem. Navíc, při použití přepínače f a použití feedfile mají být zdroje odděleny dalším řádkem. Při této kombinaci tedy vzniknou na konci každého bloku dva řádky.

6. Použité zdroje

Pro implementaci URL parseru (gadgets.h, řádek 12) byl použit kus kódu ze Stack Overflow. Vytvořil ho uživatel jménem wilhelmtell. Adresa příspěvku:
http://stackoverflow.com/a/2616217

Ve třídě HTTP jsem použil kus mého vlastního kódu z projektu do předmětu IPK, datadownloader.cpp, řádky 21 – 27.

Při tvorbě SSL třídy jsem používal dokumentaci:
http://h71000.www7.hp.com/doc/83final/ba554_90007/ch04s03.html

Další zdroje:
https://tools.ietf.org/html/rfc4287