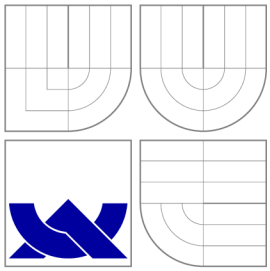
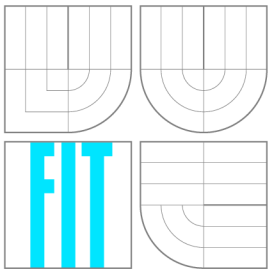


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZŠÍŘENÍ IMPLEMENTACE PROTOKOLU FTP V KNIHOVNĚ LIBCURL

EXTEND OF FTP IMPLEMENTATION IN LIBCURL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PAVEL RAISKUP
AUTHOR

VEDOUCÍ PRÁCE — Ing. KAMIL DUDKA
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá rozšířením implementace protokolu FTP v multiprotokolové knihovně libcurl. Toto rozšíření se týká možnosti stahování sady souborů z FTP, které jsou dány vzorkem obsahujícím zástupné znaky přímo v modifikované FTP URL adrese. V práci je kladen důraz na použitelnost rozšíření a také efektivitu implementace, především při komunikaci s FTP serverem. Vzhledem k tomu, že je knihovna široce využívána stávajícími aplikacemi a jinými knihovnami, je důležité udržet zpětnou kompatibilitu knihovny. Stejně jako knihovna libcurl je i rozšíření psáno v jazyce C.

Abstract

This work elaborates on the design and implementation of an extension to the multiprotocol network library known as libcurl. The extension focuses on using the FTP protocol as found in libcurl, making it possible to download bundles of files that are specified by a matching pattern extracted directly from a URL. In addition to being readily available to and easily understood by libcurl users, the extension aims to allow efficient communication with FTP servers. Taking into account that libcurl is widely used by applications and other libraries, the main design criterion of this project was to maintain backward compatibility with libcurl. Like libcurl, this extension is also written in the C programming language.

Klíčová slova

FTP, knihovna, jazyk C, curl, libcurl, síťové protokoly

Keywords

FTP, library, C language, curl libcurl, network protocols

Citace

Pavel Raiskup: Rozšíření implementace protokolu FTP
v knihovně libcurl, bakalářská práce, Brno, FIT VUT v Brně, 2010

Rozšíření implementace protokolu FTP v knihovně libcurl

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Kamila Dudky. Veškeré literární prameny, ze kterých bylo při práci čerpáno, jsou uvedeny.

.....
Pavel Raiskup
17. května 2010

Poděkování

Děkuji Danielovi Stenbergovi za jeho profesionální a věcný přístup při konzultaci problémů spojených s řešením této práce. Dále děkuji Kamilu Dudkovi za cenné rady, připomínky a metodické vedení práce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá rozšířením knihovny libcurl o možnost jednoduše použitelného stahování výčtu souborů ze složky FTP serveru. Tento výčet souborů bude zadán knihovně pomocí jedné modifikované FTP URL adresy, která bude obsahovat zástupné znaky, jako to známe například z linuxových terminálů při práci se soubory i ze systému Windows při vyhledávání souborů v souborovém systému.

Například, pokud bude uživatel knihovny chtít stáhnout všechny soubory, které končí příponou `txt`, z kořenového adresáře FTP serveru na doméně `example.com`, pak bude odpovídající formát URL adresy `ftp://example.com/*.txt`.

Podobný způsob práce s knihovnou nebyl doteď v knihovně implementován a současná implementace FTP protokolu v knihovně není připravena na práci s více soubory pomocí jedné URL adresy. Doposud se vždy s knihovnou pracovalo tak, že pro každý požadovaný soubor bylo nutné přenastavovat URL adresu vzdáleného souboru a spouštět znovu přenos souboru – viz. kapitola popisující rozhraní knihovny 2.3.

Součástí práce je také dokumentace vzniklých úprav v aplikačním rozhraní (*API – Application Programming Interface*), tj. úprava manuálových stránek knihovny. Rovněž byla rozšířena sada automatických testů, kterými knihovna disponuje, o několik testů pokrývajících nový kód v knihovně.

Při úpravách je důležité zachovat princip aplikačního rozhraní a všechny změny musí dodržovat dosavadní platformovou nezávislost knihovny. Knihovna je napsaná v jazyce C, proto musí být i rozšíření v tomto jazyce, aby bylo možné knihovnu využívat i na platformách, které neumožňují překlad vyšších programovacích jazyků.

Kapitola 3 obsahuje studii různých knihoven pro práci s FTP a jejich přístupu k řešení dílčích podproblémů, které jsou potřeba pro implementaci rozšíření v libcurl. V kapitole 3 jsou uvedeny jednotlivé kroky návrhu rozšíření, které jsem od výsledné funkčnosti očekával, a jak si funkčnost představovala komunita vývojářů. Kapitola 5 potom shrnuje způsob implementace rozšíření a drobné rozdíly mezi implementací a návrhem.

Kapitola 2

Základní pojmy

V této kapitole shrnu několik základních pojmů, okrajově se zmíním o historii projektu curl a seznámím čtenáře s podobnými projekty pro práci se síťovým protokolem FTP.

2.1 Protokol FTP

Protokol FTP je síťový protokol, pracující na aplikační vrstvě, kde je k přenosu dat využíváno spolehlivého protokolu TCP (*Transmission control protocol*). Protokol byl definován v RFC 959 [7], a dalších rozšíření se doposud dočkal v RFC 2228, RFC 2640, RFC 2773, RFC 3659 [8] a RFC 5797.

Na rozdíl od HTTP[1] (*HyperText Tranfer Protocol*) protokolu, kde se využívá jedno TCP spojení pro pro zadávání požadavků i přenos dat, FTP používá dva kanály:

1. Řídící (kontrolní), kterým klient dává požadavky serveru
2. a datový, kterým jsou následně data přenášena, a to jak od klienta k serveru, tak ze serveru ke klientovi (stahování/odesílání).

Služba FTP má dle RFC 1700[6] přiděleny standardní porty 20 a 21 pro datové a řídící spojení. Existují dva typy FTP spojení – aktivní a pasivní. U aktivního spojení se datový kanál otevírá ze strany serveru, v případě pasivního spojení dochází k otevírání datového kanálu ze strany klienta. Pasivní režim si tedy žádá veřejnou IP adresu na straně klienta. Knihovna libcurl podporuje jak aktivní, tak pasivní spojení.

2.2 Knihovny pro práci s FTP

Práce s protokolem FTP je pro aplikačního programátora poměrně pracná, (je třeba starat se o různá spojení, alokaci a dealokaci zdrojů, atd.), a to je jeden z důvodů vzniku knihoven pro práci s ním. Kapitola 3.1 shrne poznatky o knihovnách, které pracují s FTP protokolem. Některé z uvedených jsou multiprotokolové, podobně jako libcurl. Pro porovnání FTP knihoven napsaných v Javě jsem vybral jen dvě (JFtp a Jakarta Commons Net), které na základě srovnání [10] vyšly jako nejpoužitelnější (vzhledem k licenci, funkčnosti zpracování LIST odpovědi 4.3 a dostupnosti zdrojových kódů).

[1] Konkrétně protokol verze 1.1

2.2.1 Projekt Curl

Z knihovny libcurl, jejímž rozšířením se tato práce zabývá, se během své již relativně dlouhé historie stal mocný nástroj pro práci se síťovými protokoly.

Tato historie se začala odvíjet v roce 1997, v době, kdy Daniel Stenberg, zakladatel projektu curl, začal pracovat na aplikaci pro přepočet měn, která měla být dostupná uživatelům IRC (*Internet Relay Chat*). Protože všechny potřebné informace byly přístupné na internetu, bylo pouze zapotřebí automatizovat jejich získávání. To vedlo Daniela k úpravě již existujícího volně dostupného nástroje httpget – tento nástroj dále upravoval, přidával jiné protokoly a položil tím základy multiprotokolovému nástroji curl [2].

V dnešní době nástroj curl — a tedy i knihovna libcurl, na které je založen — umí pracovat s protokoly TP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS, FILE, IMAP, SMTP, POP3 a RTSP. Tento program, přístupný z příkazové řádky, je dostupný na UNIX systémech, na Windows i na méně známých platformách.

Nástroj curl i knihovna libcurl jsou vyvíjeny jako otevřený kód (*Open Source*) pod licencí MIT a při vývoji je tedy nutné veškeré změny konzultovat s komunitou jejich vývojářů. Kompletní zdrojové kódy jsou dostupné na stránkách [12].

Pro práci s protokolem FTP se využívá pouze URL. Práce na úrovni protokolu není možná. Knihovně tedy řekneme o stažení souboru `ftp://example.com/photo.jpg` a ona se o vše transparentně postará. Pro uživatele knihovny je práce s protokolem FTP v principu úplně stejná, jako práce s jinými protokoly.

2.3 Rozhraní knihovny libcurl

Knihovna umožňuje konzistentní práci, ať jde o kterýkoliv z implementovaných internetových protokolů. Při práci s knihovnou je nutné[2]:

1. Inicializovat knihovnu a datové struktury `curl_easy_init()`,
2. zadat knihovně veškerá nastavení `curl_easy_setopt()` a
3. spustit přenos `curl_easy_perform()`.

Názvy funkcí jsou samovysvětlující, přesto bych ještě podtrhl důležitost funkce `curl_easy_setopt()`. Funkce je obecně definována na tři a více parametrů, ale pravidlem je, že mívá právě tři parametry. První je ukazatelem na strukturu `CURL`, druhým je hodnota z výčtu možností nastavení, a třetím je hodnota, kterou chci nastavit. Například tedy, pokud budu chtít nastavit URL adresu, ze které chci stahovat data, zavolám:

```
curl_easy_setopt(curlptr, CURLOPT_URL, "http://example.com/")
```

Jednoduchý program napsaný v jazyce C za pomoci knihovny libcurl může vypadat například jako ukázka 2.1. V seriózním programu by samozřejmě měly být kontrolovány návratové hodnoty všech curl_easy volání.

To byl základní přístup k práci s knihovnou libcurl. V tomto případě se vždy v jednom okamžiku (v jednom procesu) pracuje s jedním přenosem. Pokud chceme v jednom procesu stahovat více souborů najednou, je možné využít „multi“ rozhraní[3].

[2] Jedná se tzv. „easy“ rozhraní, existuje také „multi“ rozhraní, zmiňuji se o něm kousek dále, ale zájemce odkazuji na manuálové stránky projektu.

[3] Pro souběžné stahování více souborů je samozřejmě možné využít více procesů nebo více vláken. Při použití vláken je třeba dbát na to, aby nebyla jedna curl struktura sdílena ve všech vláknech.

```
/**
 * Program stáhne obsah kořenového adresáře serveru debian.org
 * a vytiskne jej na standardní výstup
 */
#include <curl/curl.h>

int main()
{
  curl_global_init();
  CURL *curl = curl_easy_init();
  curl_easy_setopt(curl, CURLOPT_URL, "ftp://example.com/");
  curl_easy_perform(curl); // provede přenos obsahu koř. adresáře
  curl_easy_cleanup(curl);
  curl_global_cleanup();
}
```

Obrázek 2.1: Příklad využití knihovny

Multi rozhraní je nadstavbou nad easy rozhraním. Pracujeme s ním tak, že si jednoduše vytvoříme více `CURL` struktur pomocí `curl_easy_init()`, a tyto ukazatele předáme do multi struktury `CURLM` pomocí `curl_multi_add_handle()`. Za pomoci funkce `select(2)` a `fd_set` struktur poté v případě možnosti přenosu pomocí některé `CURL` struktury voláme funkci `curl_multi_perform()`. Podrobněji na oficiálních webových stránkách projektu curl [12] a manuálových stránkách knihovny.

2.4 Vývoj libcurl

Knihovna libcurl je otevřený software. Na vývoji se podílelo a podílí několik desítek vývojářů z celého světa. Pro komunikaci mezi vývojáři se využívá běžných mailing listů, což slouží jako komunikační kanál a zároveň jako perzistentní veřejný záznam rad, návrhů a úprav knihovny.

Pro okamžitou pomoc je možné využít IRC kanálu na `freenode.net`. Podle zkušeností s dosavadním průběhem komunikace na IRC, slouží tento komunikační kanál spíše pro rady pro uživatele knihovny, ale i vývojáři zde můžou řešit (a občas řeší) problémy spojené s úpravami knihovny.

Pro vývoj rozsáhlého projektu, jako je libcurl se využívá verzovacích systémů. Já měl při práci na knihovně zrovna štěstí na změnu verzovacího systému. Po dvanácti letech se 20. března 2010 přešlo z CVS (*Concurrent Version System*) na git [4].

[4] Zajímavost: V překladu z britského slangu (hanlivě) „nemanželský“, viz. historie gitu.

Kapitola 3

Analýza současných řešení

Nepodařilo se mi najít žádnou aplikaci, ani knihovnu, která by v současné době umožňovala podobnou funkčnost, kterou má za cíl tato práce. Co se ale týká běžné práce s protokolem FTP, viz. 3, existuje velké množství knihoven a aplikací. V této kapitole proberu jejich výhody a nevýhody.

Zmíním se o několika možnostech porovnání zadaného vzorku se skutečnou adresou (tedy o implementacích funkcí, které uvažují využití zástupných znaků) a o problematice získávání informací o obsahu adresáře FTP serveru.

3.1 Knihovny pro práci s FTP

3.1.1 FtpLib

Knihovna je implementována v jazyce C a použitelná na unixových systémech, Windows a VMS (*Virtual Memory System*).

Knihovna ftplib obsahuje sadu rutin, která implementuje protokol FTP. Umožňuje otevřít vzdálený soubor „za letu“ pro zápis a čtení. Umožňuje samozřejmě práci v aktivním i pasivním režimu.

Práce s knihovnou je snadná, rutiny jsou srozumitelně dokumentovány. Co se týká práce s adresářem, tak není implementována žádná možnost zpracování obsahu adresáře. Uživatel nemá tedy možnost zjistit informace o souborech v adresáři jinak, než si říct o seznam souborů (odpovědí je plain-text odpověď podobná výstupu unixového příkazu `ls -l`) a následně si tuto odpověď ručně v aplikaci zpracovat.

Implementace protokolu je realizována takovým způsobem, že práce se vzdálenými soubory připomíná práci se soubory na lokálním souborovém systému. Soubor je první nutno otevřít (pro zápis nebo čtení) a následně z něj číst, nebo do něj zapisovat. Po skončení zápisu nebo čtení ze souboru, se musí soubor podobně jako při práci s lokálními soubory uzavřít.

3.1.2 Jakarta Commons Net

Jakarta Commons Net [4] je knihovna, která implementuje internetové protokoly, podobně jako libcurl. Původně byla tato knihovna vyvíjena jako komerční produkt a v roce 1998 byl zdrojový kód darován sdružení Apache Software Foundation. Zdrojový kód byl poté převeden pod otevřenou Apache licenci.

Implementačním jazykem je tentokrát Java, použití knihovny proto s sebou nese výhody objektového návrhu. Například právě pro protokol FTP existují dvě základní třídy — FTP a FTPClient — kde třída FTPClient dědí z třídy FTP. Se třídou FTP se pracuje podobně jako s čistým protokolem, obsahuje totiž metody odpovídající příkazům protokolu (pro `CWD` existuje metoda `cd()`, pro `RETR` metoda `get()`, atd.). Naopak, třída FTPClient již obsahuje vyšší míru abstrakce, kdy si stačí říct o konkrétní úkol (například stáhnout soubor v určitém adresáři, podobně jako u libcurl) a knihovna se o to poté sama automaticky postará. Tedy provede seznam potřebných FTP příkazů.

Z hlediska bakalářské práce je velice zajímavé, že knihovna má v sobě implementováno i metodu, která umí zpracovat LIST odpověď serveru (*listFiles*). Tato metoda vrací pole objektů, obsahující informace o názvu, velikosti souboru, uživateli, skupině a ostatní informace, které odpověď na příkaz LIST obsahuje.

Jakarta Commons Net v současné době implementuje protokoly FTP/FTPS, NNTP, SMTP, POP3, Telnet, TFTP, Finger, Whois, rexec/rcmd/rlogin, Time (rdate) a Daytime, Echo, Discard a NTP/SNTP.

3.1.3 SourceForge JFtp

Implementačním jazykem knihovny JFtp [5] je opět Java. Podle srovnání [10] vychází asi jako nejpoužitelnější knihovna pro práci s FTP (zdarma nebo open source). Podle mého osobního názoru, je knihovna obtížněji použitelná a méně srozumitelná. Alespoň co se týká prvního použití. To do jisté míry souvisí s tím, že je knihovna primárně určena pro stejnojmenný Java grafický nástroj a při návrhu knihovny se příliš nepočítalo s jejím využitím mimo tento nástroj.

JFtp knihovna umožňuje sama o sobě také zpracování LIST odpovědi. Malou nevýhodou může být absence zpracování časové známky souboru, což Jakarta Commons Net zvládá.

3.1.4 Shrnutí funkčnosti knihoven

FtpLib

Knihovna neumožňuje automaticky zpracování LIST odpovědi, ale poměrně pohodlně umožňuje využívat protokol FTP. Veškeré zpracování a následné stažení souborů, které by vyhovovaly vzorku se zástupnými znaky si musí aplikační programátor udělat sám. Vlastnosti knihovny:

- Velikost knihovny jen 14 KiB [1],
- jednoduchost využití,
- nemožnost přímého zaslání příkazu serveru,
- implementováno v jazyce C.

Jakarta Commons Net

Java knihovna Jakarta Commons Net obsahuje třídu FTPClient, která je na tom ve využití FTP protokolu nejlépe. U této třídy je možnost práce s protokolem stejná jako v FTPLib, je zde možnost pracovat přímo na úrovni protokolu s textovými příkazy, ale také navíc jsou zde metody, které zvyšují míru abstrakce nad protokolem. Tohle poznáme například při přenosu souboru nebo při získávání seznamu souborů ze serveru, protože knihovna umí například velmi dobře zpracovat i odpověď serveru na LIST požadavek.

[1] Měřena statická knihovna distribuce Debian při instalaci z repozitáře.

- Implementováno v jazyce Java,
- multiprotokolová knihovna,
- objektový přístup a snadná použitelnost,
- umožňuje získat seznam souborů ve vzdáleném FTP adresáři (objektově).

JFtp
Další knihovna s otevřeným kódem pro Javu, která je funkčně podobná té předchozí. Použití je o dost krkolomnější — obsahuje mnoho tříd, se kterými je nutné se naučit — neumí (na rozdíl od Jakarta Commons Net) vyčíst časovou značku z LIST odpovědi.

- Implementováno v jazyce Java,
- objektové řešení, které ale není příliš intuitivní,
- umožňuje získat seznam souborů z FTP adresáře,
- v současné době se dále nevyvíjí.

libcurl
Po rozšíření, kterému se věnuje moje bakalářská knihovna bude podporovat jednoduše použitelné zpracování LIST odpovědi, přes nízkoúrovňový jazyk C. Umožňuje snadnou plně automatickou práci s protokolem pomocí URL adres.

- Implementační jazyk je jazyk C,
- platformově nezávislá,
- multiprotokolová knihovna,
- konzistentní práce se všemi protokoly pomocí URL,
- stále vyvíjený živý projekt,
- existují navázání do jiných jazyků jako je PHP, haskell, aj.

Další knihovny je možné vyčíst z tabulky 3.1. Srovnání rychlosti lze najít v článku [11].

3.2 Implementace využití zástupných znaků

Mechanismus použití zástupných znaků je podobný využití regulárních výrazů, které jsou implementovány i v jazyce C například v `regex.h` nebo `pcre.h`. Složitost implementace je však proti regulárním výrazům o dost menší, což je dáno menšími možnostmi při porovnávání vzorku s řetězcem. To však může být považováno za výhodu, pokud půjde o náročnost využití syntaxe uživatelem. V základu existuje několik klasických typů zástupných možností:

1. Znakem **?** pro zastoupení právě jednoho libovolného znaku,
2. znakem ***** pro zastoupení libovolného počtu libovolných znaků a
3. výčtem znaků v hranatých závorkách [**abc**], [**a-z**] či [**!abc**], pro zastoupení znaku, který je, či v příkladu s vykřičníkem není, jedním ze znaků v tomto výčtu.

Informace o současných implementacích:

Název projektu	Poznámky	Implementační jazyk
JScape iNet Factory	– komerční projekt – podporuje paralelní přenos	Java
FTP Client Class	– pouze pro Windows	C++
IBM FTP Bean Suite	– absence JavaDoc	Java
SourceForge JFtp	– absence Javadoc – podporuje rekurzivní přenos stromu	Java
SUN JDK	– balík není schválen v JDK6	Java
jvFTP (Bea Petrovicová)	– český autor – neumí příkaz RETR (navázání)	Java
FTPLib	– nezpracovává LIST – neobjektový přístup – jednoduchost použití	C
libcurl	– funguje na mnoha platformách – jednoduché použití – práce s FTP pouze přes URL	C

Tabulka 3.1: Srovnání knihoven pro práci s FTP.

fnmatch(3)

POSIX[3] funkce `fnmatch(3)` implementuje všechny zmiňované možnosti syntaxe, navíc s dalšími rozšířeními. Tuto funkci využívá spousta aplikací, které se používají z terminálu, protože její využití může uživateli terminálu významně ušetřit práci.

Využití této funkce by tedy pro zamýšlené rozšíření bylo ideální. Jak se ale zmíním ještě v kapitole 4, rozhodlo se, že se tahle funkce pro potřeby knihovny nevyužije (respektive se nevyužije její již hotové implementace).

PathMatchSpec()

Funkce pracuje podobně, jako `fnmatch(3)`, ale neimplementuje notaci hranatých závorek. Další nevýhodou funkce je, že její využití je závislé na Windows API knihovně `Shlwapi.dll` — její využití je v libcurl zcela vyloučeno. Pro Unicode a ANSI znaky existují ekvivalenty `PathMatchSpecW()` a `PathMatchSpecA()`.

Příklady aplikací s využitím wildcard matchingu

Využití známe nejen ze situací z unixových terminálů, když chceme například vymazat všechny soubory s příponou .txt ze složky příkazem `rm *.txt`. Podobnou funkčnost umožňuje i vyhledávání na Windows, kdy chceme například najít všechny soubory, jejichž název obsahuje určitý textový řetězec.

Další zajímavé místo využití je systém DNS. Pokud chceme, aby DNS server vracel konkrétní IP adresu pro doménové jméno, které končí `example.com`, můžeme do konfiguračního souboru zadat řádek:

```
*.example.com. 3600 IN A 10 127.0.0.1.
```

Využití zástupných znaků do jisté míry může podporovat i samotný FTP server. O této vlastnosti některých serverů se zmíním v kapitole 4.4. Příklady využití zástupných znaků pro program `rm` (mazání souboru z terminálu) můžete vidět v tabulce 3.2.

Příkaz	**Popis**	**Odpovídající soubory**
`rm *.txt`	smazat soubory s příponou .txt	`textfile.txt` `otherfile.txt`
`rm ??.*`	smazat soubory s názvem o dvou písmenech a libovolné koncovce	`11.dat` `U2.mp3`
`rm ph[0-9].jpg`	smaže soubory začínající řetězcem `ph`, následovaným libovolně dlouhou sadou čísel a končícím koncovkou `jpg`	`ph1.jpg` `ph2.jpg`

Tabulka 3.2: Příklady využití zástupných znaků.

Kapitola 4

Návrh řešení

V této kapitole popíšu úvodní návrh rozšíření knihovny a shrnu prvotní fázi komunikace s komunitou vývojářů libcurl, kdy jsme návrh diskutovali.

Zadání práce — a tedy předmět rozšíření knihovny — byly inspirovány několika požadavky uživatelů knihovny na stahování ze složky na FTP serveru pomocí zástupných znaků. To mělo mělo za následek vznik nového „TODO“ na stránkách projektu.

4.1 Více souborů na jeden požadavek

Nápad integrace využívání zástupných znaků pro stahování z FTP serveru s sebou přinesl jednu nepříjemnost — stávající rozhraní knihovny totiž nebylo pro takový přístup připraveno. Dosavadní přístup k užívání knihovny byl následující (příklad podobného využití knihovny jsem uvedl na příkladu 2.1):

1. Inicializovat knihovnu,
2. zadat URL adresu jednoho cíle, kterého se bude následující přenos týkat,
3. zadat nastavení pro následující přenos (například pokud jde o stahování, tak kam se má soubor stáhnout na lokální disk)
4. spustit přenos,
5. pokud chceme stahovat další soubor, skočit zpět na bod 2.
6. a uklidit prostor, který potřebovala knihovna.

Knihovna doposud nepodporovala stažení více souborů při zadání jedné URL adresy. Hlavní problém rozšíření nastává v bodu 3. Z pohledu knihovny jsou přenášená data pro jednu URL chápána jako dále nedělitelná, v případě FTP protokolu odpovídají obsahu jednoho souboru nebo výpisu adresáře. Pokud se tedy tento přenos bude skládat z více souborů, je potřeba o tomto informovat aplikačního programátora. Ten potom může rozhodnout, jak s jednotlivými částmi přenosu naloží. Proto bylo už v době návrhu jasné, že bude potřeba rozšířit také aplikační rozhraní knihovny, aby tyto informace poskytovalo.

4.2 Rozšíření rozhraní knihovny

4.2.1 Callback funkce

V počátku návrhu se řešili dvě možnosti rozšíření rozhraní:

1. Navrhnout jednu callback[1] funkci — přidat do rozhraní jednu konstantu CURLOPT[2] odpovídající ukazateli na funkci — a tato funkce bude volána před zahájením stahování každého individuálního souboru, a nebo

2. vytvořit dvě callback funkce (tj. dvě nová nastavení), jedna z nich bude volána před zahájením stahování, druhá bude volána po stažení konkrétního souboru.

Pokud bude uživatel chtít pro každou část přenosu, která odpovídá jednomu vzdálenému souboru, vytvářet právě jeden soubor v lokálním souborovém systému, potom bude nutné každý z nich otevírat a zavírat. A bylo by vhodné, kdyby se tak dělo uvnitř těchto callback funkcí. Obecně by tohle šlo řešit i první zmiňovanou možností (tj. jedna callback funkce), ale využití pro uživatele by bylo nepohodlné. Minimálně jednou za celý průběh stahování by bylo nutné volat pro jeden soubor tuto funkci dvakrát (pokud první volání otevírá první soubor, pak callback ještě nemá co uzavírat). Bylo by tedy nutné na konci zavolat callback funkci, která již bude pouze uzavírat poslední soubor.

Tato nepříjemnost by byla natolik nepohodlná, že jsem se rozhodl pro možnost druhou. V prvotním návrhu tedy vznikl požadavek pro implementaci dvou nových nastavení — `CURLOPT_STARTUNIT` a `CURLOPT_ENDUNIT` — což jsou ukazatele na funkce, jejichž prototypy mají jeden parametr, který bude ukazovat na název vzdáleného souboru:

```
/**
 * Prototyp funkce, která bude volána před zahájením stahování
 * každého odpovídajícího souboru
 */
typedef long (*curl_startunit_callback)(const char *filename, void *ptr);

/**
 * Prototyp funkce, která bude volána po skončení stahování každého
 * odpovídajícího souboru.
 */
typedef long (*curl_endunit_callback)(const char *filename, void *ptr);
```

Kde parametr `filename` znamená název vzdáleného souboru (ang. remote file) a parametr `ptr` je ukazatel na libovolná data, která si uživatel přeje callback funkcím předávat (například ukazatel na strukturu, která bude nést informace o průběhu stahování, počtu stažených souborů, atd.).

Prototyp a název těchto funkcí se během několikáté iterace úprav změnil (jak lze vidět v příkladu užití rozšíření), ale princip práce s callback funkcemi zůstal stejný.

[1] Výraz *callback* užívám pro funkci, která není definována v knihovně, ale definuje ji uživatel knihovny a následně předá knihovně její adresu, aby ji mohla zpětně zavolat.

[2] CURLOPT je předpona konstant, které se používají v rozhraní knihovny (konkrétně ve funkci `curl_easy_setopt`) pro různá nastavení chování knihovny, viz. kapitola 2.3.

4.2.2 Povolení zástupných znaků v URL adrese

Podle RFC1738 [1] je možné, aby v URL adrese existovaly i znaky, které mají v rozšíření sloužit jako znaky zástupné (tj. v době návrhu zamýšlené znaky '?', '*', '[' a ']'). Aby bylo možné provést porovnání i se vzorkem, který obsahuje znak z toho výčtu i jako speciální znak, i jako normální, je potřeba zavést do knihovny možnost předznamenání znaků, které zruší význam speciálního znaku. Zvykem bývá, že k tomuto předznamenání slouží zpětné lomítko.

Už při návrhu tedy bylo jasné, že se bude muset toto rozšíření explicitně zapínat, protože samostatná existence speciálního znaku v URL adrese nemůže spouštět „wildcard" stahování. V souborovém systému klidně může existovat soubor, který se jmenuje přesně `*.txt`. Pro povolení této funkce jsme se s vývojáři dohodli na novém nastavení `CURLOPT_WILDCARDMATCH`, které může nabývat hodnot `0L`/`1L` (long int) pro vypnuto/zapnuto.

4.3 Jak získat seznam souborů

Další otázkou bylo, jakým způsobem zjistit obsah vzdáleného adresáře. V protokolu FTP k tomu slouží několik vestavěných příkazů. Některé byly přidány do protokolu později, jsou tedy na serverech implementovány jen zřídka, některé z hlediska potřeb rozšíření naopak neposkytují dostatek informací.

NLST

Odpověď serveru na příkaz NLST je z hlediska zpracování v nejjednodušším formátu. Odpověď má následující formát:

```
filename_1\r\n
filename_2\r\n
...
filename_N\r\n
```

Pokud by tedy šlo o jednoduchost zpracování a také jeho jednoznačnost, byl by toto nejvhodnější příkaz. V některých případech je však potřeba kromě samotného názvu souboru znát také nějaká metadata — velikost souboru, typ souboru (složka, běžný soubor, zařízení, symbolický odkaz) a časovou známku (timestamp) souboru.

LIST

Příkaz LIST již dává (obvykle) uživateli dostatečný soubor informací. V nejběžnějším případě vypadá formát odpovědi souboru následovně:

```
TS|OP|<->|UŽIVATEL|SKUPINA|<->|Č1|<->|Č2|<->|Č3|<->|NÁZEV_SOU_1|\r\n
TS|OP|<->|UŽIVATEL|SKUPINA|<->|Č1|<->|Č2|<->|Č3|<->|NÁZEV_SOU_2|\r\n
...
TS|OP|<->|UŽIVATEL|SKUPINA|<->|Č1|<->|Č2|<->|Č3|<->|NÁZEV_SOU_N|\r\n
```

Kde `TS` znamená „typ souboru", `OP` znamená „oprávnění" a `Č1 Č2 Č3` znamená čas poslední modifikace souboru. Značka `<->` znamená oddělovač ve formě mezer a značka `|` znázorňuje logické oddělení položek. Jedná se o formát odpovědi, jakou dává program `/bin/ls` při spuštění s parametrem `-l`.

Problémem odpovědi na tento typ dotazu je, že žádný standard přesně neříká, jaké informace má odpověď obsahovat — a servery se podle toho taky v praxi chovají. Staré FTP servery běžící na MS-DOS či Windows FTP Service poskytují formát nekompatibilní jak se zmiňovaným formátem, tak s Mac formátem nebo s OS/2 formátem a opačně (více viz. kapitola 5).

MLSD/MLST

Příkazy MLSD a MLST byly přidány do protokolu FTP v RFC 3659 [8]. Vznikly na základě požadavku na strojové zpracování odpovědi klientem, čili důvodem byl stejný problém, s jakým jsem se potýkal. Z pohledu na formát odpovědi se jedná přesně ten typ odpovědi, který jsem původně hledal.

Problémem těchto dvou příkazů zůstává jejich podpora na FTP serverech — ta je v dnešní době stále minimální[3].

Po diskusi na mailing listu jsme se nakonec rozhodli pro zpracovávání nejsložitější varianty, tedy pro příkaz list. I přesto, že se nikdy nedosáhne stoprocentní funkčnosti konečného automatu, který bude sloužit jako nástroj pro toto zpracování.

Omezení délky seznamu souborů

Během návrhu taky vyvstal nápad, že by bylo dobré minimalizovat přenos mezi serverem a klientem již při příkazu LIST. Obecně může být velikost seznamu (tj. velikost odpovědi v bajtech) větší, než je velikost součtu obsahů daných souborů a větší část síťového přenosu bude představovat právě odpověď na příkaz LIST.

Možnost omezení délky odpovědi serveru bývá zpravidla skutečně implementována. Jedná se však o vedlejší efekt toho, jakým způsobem je příkaz LIST řešen. Klient má možnost omezit velikost odpovědi pomocí parametru příkazu LIST, který také může obsahovat zástupné znaky. Pokud se příkazu zadá parametr `*.txt`, odpověď serveru se omezí na výčet souborů, které odpovídají tomuto vzorku. Tato funkčnost je dána tím, že spousta unixových FTP serverů využívá pro generování odpovědi na LIST příkaz vestavěného programu `ls` v kombinaci se shellem, a tedy přidané parametry LIST příkazu předává přímo shellu. Pro zajímavost lze uvést, že lze obvykle (pokud server běží na systému založeném na Unixu) zadat i parametry **`-a -h`**, kde tyto parametry znamenají „všechny soubory“ (včetně „.“ a „..“) a „čitelné člověku“ (human readable).

4.4 Způsob rozhodování o odpovídajícím souboru

V této podkapitole se budu věnovat možnostem porovnání vzorku obsahujícím zástupné znaky s konkrétním názvem souboru. Problematika rozhodnutí, zda název souboru odpovídá či neodpovídá předloze.

Rozhodl jsem se vyhnout obvyklým knihovnám pro práci s regulárními výrazy. Tyto knihovny obvykle nejsou dostupné na méně obvyklých architekturách, na kterých ale knihovna libcurl funguje. Navíc je zvykem pro práci se jmény souborů používat hvězdičkovou notaci v tom smyslu, že hvězdička znamená 0 až n libovoných znaků, nikoliv jako 0 až n násobek předchozího znaku, což je zvykem u regulárních výrazů. Proto se uvažovalo o již zmiňované funkci `fnmatch(3)`. Existují implementace této funkce šířené pod svobodnými licencemi (např. LGPL), jsou napsané efektivně a na unixových systémech je tato funkce obvykle

[3] To je dáno tím, že RFC bylo vydáno v roce 2007 — tj. v době tvorby práce před třemi lety.

dostupná. Na rozdíl od knihoven pro práci s regulárními výrazy není tato funkce (alespoň v GNU implementaci) založena na převodu regulárního výrazu na konečný automat.

Problémem u této funkce je standard POSIXv2 (Portable Operating System Interface). Ne všechny architektury, umožňující běh libcurl jsou standardu POSIX. To vede k tomu, že nelze fnmatch použít implicitně jako rozhodovací funkci — celá knihovna by se stala nepoužitelnou na systémech, které fnmatch nepodporují.

Byla by sice možnost upravit konfigurační skript a pomocí podmíněného překladu vyloučit funkčnost z architektur, které `fnmatch()` neobsahují. To by ale byla škoda, protože by veškerá funkčnost rozšíření (i platformě nezávislá) zůstala nevyužita. Bohužel, žádnou jinou implementovanou funkci, která by se funkci `fnmatch()` podobala, byla licenčně přijatelná a byla pro libcurl vhodnější, jsem nenalezl.

Proto jsem se rozhodl dát možnost uživateli knihovny tuto rozhodovací funkci měnit pomocí další callback funkce. Nové nastavení `CURLOPT_FNMATCH` bude typu ukazatel na funkci, a prototypem této funkci bude typ:

```
typedef int (*curl_fnmatch_callback)(const char *pattern,
                                     const char *string);
```

Dále jsem se rozhodl pro implementaci implicitní funkce `Curl_fnmatch`, která bude využita v případě, že uživatel nenastaví svoji vlastní porovnávací funkci. Tímto krokem se zajistí konzistentní chování na všech podporovaných platformách a zároveň bude v případě potřeby bude možnost využít i jiných typů regulárních výrazů, což už ale případně zůstane na starost aplikačnímu programátorovi.

Kapitola 5

Implementace

Pro implementaci rozšíření jsem využil jazyka C. Automatické konfigurační skripty projektu libcurl generují `Makefile`, se jehož pomocí `make` zajistí automatický překlad a sestavení knihovny libcurl. Zdrojové kódy projektu jsou psány v normě ISO89. Pro automatické testování existuje v projektu sada speciálních nástrojů, které umožňují jednoduše provést testy funkčnosti bez nutnosti práce se sítí. Tyto nástroje jsou napsané v jazyce Perl[9].

5.1 Zásah do principu práce knihovny

Jak jsem se již zmiňoval v předchozích kapitolách, dosud nebylo možné na základě zadání jedné URL přenášet více nezávislých souborů. Vždy bylo nutné inicializovat přenos individuálního souboru a po skončení přenosu se zbavit veškerých alokovaných zdrojů. Původně se s něčím takovým v knihovně vůbec nepočítalo, takže úprava rozhraní knihovny nebyla zcela triviální.

Princip práce knihovny se skládá z několika částí a dá se rozdělit z hlediska abstrakce do několika úrovní:

Veřejné (Public) API
: které má k dispozici uživatel, a které obsahuje sadu funkcí pracující nad CURL ukazatelem. Typ CURL je void ukazatel na interní datové struktury. Tím, že se jedná o void ukazatel, je zamezena dereference tohoto ukazatele a je tedy uživateli knihovny znesnadněno zasahování do interních datových struktur knihovny. Veřejné API tedy slouží k tomu, aby se nastavila interní zapouzdřená data knihovny a mohla být poté taktéž pomocí API spuštěna požadovaná operace.

Funkce nejvyšší úrovně
: Tyto funkce rozdělují práci knihovny do několika částí:

- `Curl_pretransfer`, kde probíhá inicializace potřebných částí knihovny. Dochází zde například k inicializaci interního „Progress baru“ (ukazatel průběhu přenosu), inicializaci SSL struktur, ošetření vstupních cookies či session[1] proměnných.
- `connect_host`, v této funkci dochází k připojení ke vzdálenému serveru na transportní vrstvě. Vytváří se tedy sokety, které jsou navázány na správné porty poža-

[1] Jedná se o proměnné protokolu HTTP, které jsou přenášeny v hlavičkách protokolu a slouží obvykle aplikačnímu programátorovi k implementaci relační vrstvy nad protokoly TCP/IP.

dovaného protokolu. Dochází zde k DNS překladu adres a ustanovení komunikace v samotném protokolu (například přihlášení uživatele).

- `Curl_do` je funkce, ve které je zažádán server o přenos. Jedná se tedy obvykle o komunikaci na úrovni aplikačního protokolu, kdy se klient i server dostanou do situace před samotným přenosem a následně je dohodnutý přenos proveden (příklad u FTP protokolu může být takový, že se provede sada FTP příkazů — přihlášení, změna pracovní složky na serveru, zjištění velikosti souboru, požádá se server o přenos — a poté se přejde do přenosové fáze, kdy je otevřen datový kanál FTP a teprve v tuto chvíli je aplikační programátor informován o stavu přenosu).
- `Curl_done` funkce se poté stará o režii nad inicializovaným přenosem po skončení (přerušení) přenosu. Stará se o Progress bar (ukazatel průběhu přenosu), ošetřuje uváznutí serveru, aj. Důležité je, že nedochází k odpojení od vzdáleného serveru proto, aby bylo možné případně znovu využít alokovaných zdrojů tohoto připojení (pokud ovšem nedojde k chybě) opětovným voláním `curl_easy_perform()`.

Všechny ze zmiňovaných funkcí jsou volány, ať už knihovna pracuje s jakýmkoliv aplikačním protokolem implementovaným v knihovně. Jsou tedy jakousi abstrakcí nad nižšími funkcemi konkrétních protokolů (obvykle implementovány jako callback funkce se stejným prototypem). Ty jsou odsud, pokud pro daný protokol existují, volány.

V neposlední řadě je důležité říct, že knihovna pracuje natolik efektivně, že nejen že se klient neodpojuje od serveru při znovu použití CURL ukazatele, ale ani zbytečně nemění stavové informace protokolu (přihlášení uživatele, aktuální pracovní složka, aj.). S touto vlastností je potřeba při vývoji knihovny počítat a neprogramovat rozšíření, které by například zbytečně komunikovalo se serverem.

Funkce protokolu

Funkce protokolu se již věnují komunikaci se serverem. V knihovně je obecně snaha o minimalizaci funkčnosti implementovaných v těchto funkcích. Cílem je, aby tato funkčnost byla co nejvíce přesunuta do vyšší vrstvy (a bylo tedy možné tento kód sdílet všemi protokoly), což ovšem není vždy možné. Zde se jedná například o funkce, které slouží pro přechod mezi složkami (např. FTP) serveru, způsob požádání o stažení souboru či implementace protokolové autentizace.

Funkce pro abstrakci komunikace nad sokety

Tyto funkce jsou opět využívány vyšší vrstvou v této hierarchii. Jsou využívány pro odesílání shluků dat přes sokety, je zde možnost ověření velikosti přenesených dat a je zde abstrakce nad specifickými typy soketové komunikace (například „ping-pong“ mechanismus pro protokoly, ve kterých dochází na jednom kanálu k obousměrné komunikaci typu „dotaz-odpověď“ vícekrát po sobě. Tento přístup je vhodný právě pro FTP či SMTP).

To, že libcurl inteligentně využívá alokované zdroje daného protokolu mělo za následek již zmiňované komplikace s návrhem. Bylo nutné velmi dobře porozumět internímu grafu volání funkcí, abych dovedl určit nejlepší místo k vložení patřičného kódu a práce s protokolem zůstala efektivní (nebo, jak se později ještě zmíním, se dokonce efektivita práce s protokolem nepatrně zvýšila).

V podstatě existovaly z počátku dvě možnosti jak upravit knihovnu pro potřeby rozšíření.

- Buďto změnit princip práce s knihovnou prostřednictvím API — tj. změnit princip použití funkce `curl_easy_perform`, která by byla oproti stávajícímu stavu volána vícekrát (dle počtu odpovídajících souborů, nyní je volána pouze jednou pro jeden soubor) a nebo

- změnit princip práce knihovny vevnitř, a tedy nezměnit způsob práce s knihovnou. Tento princip s sebou však nese komplikace z toho důvodu, že nebude možné jednoduše ošetřovat chybové stavy přenosů jednotlivých souborů. Bude tedy proveden přenos dat, který se skládá z obsahů daných souborů. Ošetření chybových stavů je možné nad celým přenosem, nikoliv nad dílčími částmi.

Nakonec, z důvodu nepříjemností, které by uživateli knihovny vznikly změnou principu práce s ní, jsem se rozhodl pro druhou možnost. Z hlediska implementace šlo o snad komplikovanější způsob, zároveň mi byl na mailing listech doporučen způsob první, ale implementační komplikace, které tím vznikly, se ani z počátku nejevily tak nepřípustně.

Výsledkem implementace tedy bylo, že vznikl uvnitř knihovny nekonečný cyklus, který provolává všechny funkce nejvyšší úrovně do té doby, dokud má co stahovat. Všechny tyto funkce jsou uzpůsobeny pro stahování jednoho souboru, a tedy vlastně plně využívám prostředků knihovny sekvenčně. Jinak řečeno se jedná o konečný automat, který v první fázi stahuje odpověď serveru na LIST příkaz (a zároveň jej zpracovává a rozhoduje o tom, jestli daný název souboru odpovídá předloze), v další fázi probíhá stahování odpovídajících souborů a v poslední fázi dochází k dealokaci obsazených zdrojů. Stejně, jako jednotlivý přenos souboru ze sekvence odpovídajících souborů, je i LIST odpověď brána jako samostatný přenos.

Trik tedy spočívá v tom, že:

- Je potřeba šikovně přesměrovat obsah přenosu prvního souboru (jedná se o LIST odpověď) do nově implementované interní callback funkce,

- ve které je jeho soubor ve stejném čase zpracován,

- poté sekvenčně stáhnout odpovídající soubory

- a nakonec se zbavit pomocných datových struktur, pro které je nežádoucí, aby zůstaly dále v paměti.

Tento postup byl potřeba navrhnout tak, aby nedocházelo v žádném případě ke zbytečné protokolové komunikaci (procházet strukturou serveru od kořene až do cílové složky znova pro každý soubor, či se dokonce znovu přihlašovat). Protože LIST odpověď poskytuje informaci o velikosti vzdáleného souboru, nabídla se možnost upravit knihovní konečný automat pro běžnou práci s FTP tak, aby se na velikost souboru nedotazoval pokaždé znova, a tedy se práce s protokolem ještě více zrychlila.

Výsledná FTP komunikace při využití mého rozšíření knihovny je vidět v příkladu 5.1. Jak je z tohoto příkladu patrné, žádný z provedených příkazů není proveden zbytečně a má tedy svůj smysl [2].

[2]Komunikace se samozřejmě v praxi může lišit, jedná se o příklad. V tomto případě je možné se k serveru připojit bez autentizace, daná složka existuje a je neprázdná.

```
USER anonymous          // autentifikace uživatele
PASS ftp@example.com
PWD
CWD path                // přechod do patřičné složky
CWD to
CWD files
EPSV                    // požadavek na pasivní přenos (seznamu souborů)
TYPE A                  // bude proveden ASCII přenos
LIST                    // zde dojde ke stahování LIST odpovědi
EPSV                    // požadavek na další pasivní přenos (první soubor)
TYPE I                  // provedený přenost budou binární data
RETR filename01         // příkaz pro přenesení souboru
...
EPSV                    // dle počtu souborů se tyto příkazy opakují
TYPE I
RETR filenameN
...
QUIT                    // požadavek na ukončení řídícího spojení
```

Obrázek 5.1: Záznam příkazů, které obdrží FTP server při využití rozšíření.

5.2 Zpracování odpovědi na LIST příkaz

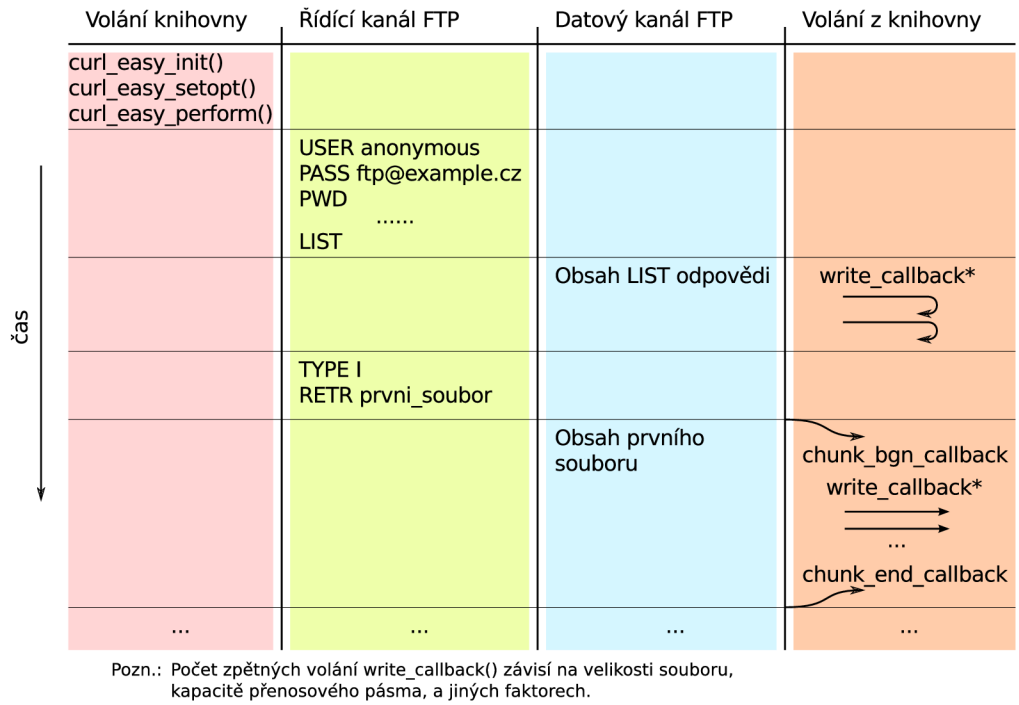
Zpracování LIST odpovědi je nepříjemná záležitost kvůli neexistující specifikaci pro formát této odpovědi. Díky tomuto existuje napříč spektrem serverů více (méně či více častých) formátů.

Pro implementaci rozšíření jsem se rozhodl implementovat podporu pro dva nejčastější formáty, a to pro unixový `/bin/ls` formát a pro DOS formát. I přesto, že jich existuje dle [10] více, nepodařilo se mi na jiný formát narazit. Přesto jsem v průběhu implementace s možností rozšíření počítal a implementoval zpracování tak, aby bylo snadné dopsat kdykoliv podporu pro jiné formáty.

Pro implementaci zpracování odpovědi jsem zvolil celkem zajímavé místo. Z hlediska přístupu k práci uvnitř knihovny by bylo nejčistším řešením, kdyby proběhlo zpracování uvnitř funkce `Curl_do` (tedy ve fázi domlouvání klienta se serverem, co se vlastně bude přenášet). Tohle by bylo ale velmi pracné a neefektivní, protože FTP posílá LIST odpověď po datovém kanálu, nikoliv po řídícím. Navíc by došlo do jisté míry k duplicitě kódu v knihovně. Proto jsem využil callback funkci (nastavitelnou pomocí `CURLOPT_WRITEDATA`), kterou má aplikační programátor obvykle k dispozici pro zpracování obsahu přenosu souboru ze serveru.

Napsal jsem tedy funkci, jejíž prototyp odpovídá typu `curl_write_callback` a její pozici v paměti předávám knihovně tak, jako by to udělal aplikační programátor. Obsah této funkce zpracovává odpověď při příchozích datech ze serveru za běhu, a knihovna si „myslí“, že data zpracovává aplikace. Ve skutečnosti se však vytváří v paměti struktura typu obousměrně vázaný seznam, nesoucí (již zpracované) informace o obsahu adresáře, lépe řečeno o souborech, které jsou daném adresáři a zároveň odpovídají předepsanému vzoru.

Implementace samotného zpracování informací je implementována jako konečný stavový automat se dvěma úrovněmi stavů. První úroveň symbolizuje položku z poskytovaného



Obrázek 5.2: Systém volání funkcí v libcurl

výčtu informací o daném souboru a druhá úroveň je pro přesnější zpracování dané položky. Dále se tento konečný automat při inicializaci rozhodne, o jaký typ formátu odpovědi jde (DOS/UNIX).

Drobná nevýhoda tohoto přístupu je v tom, že od prototypu `curl_write_callback` se neočekává, že by měl být volán pouze jednou pro každý přenos. Je volán právě tehdy, pokud je ze soketu co číst — tedy je volán během přenosu souboru vícekrát a vždy může nést různý počet znaků. A nikdy se neví, zda je tato funkce volána poprvé, či naposledy. Takže konečný stavový automat si své stavové informace musí ukládat napříč každým z těchto jednotlivých volání a vystavět si obousměrně vázaný seznam s informacemi postupně. Pro bližší snazší pochopení může posloužit obrázek 5.2. Zpětné šipky symbolizují volání callback funkce typu `curl_write_callback`, která je definována uvnitř knihovny.

Jak už jsem psal, využil jsem obousměrně vázaného seznamu a to seznamu, který je implementován v libcurl. Jedná se o generický seznam, který může nést jakékoliv informace. Pro nesené informace jsem navrhl generickou strukturu `curl_fileinfo`, která může nést veškeré informace o daných souborech. Mezi těmito informacemi bude vždy název souboru, jeho velikost a typ souboru (složka, soubor, symbolický odkaz, atd.). Dále však můžou existovat další nepovinné informace, jako jsou práva, časová známka, název vlastníka souboru a jiné. Protože spousta z těchto informací bude uložena v paměti jako řetězce, zavedl jsem kvůli efektivitě práce s pamětí uvnitř této struktury jeden hlavní alokovaný blok paměti, do kterého budou uloženy všechny tyto informace a na pozice v této paměti budou ukazovat ukazatele reprezentující dané hodnoty. Alokace této struktury s sebou nese poté jen dvě systémová volání `malloc` (alokace celé struktury a alokace zmíněného paměťového bloku).

5.3 Funkce fnmatch

Ač se využití funkce `fnmatch` (implementuje mechanismus využití zástupných znaků) nejevilo jako sebemenší problém, nenalezl jsem žádnou vhodnou implementaci, která by se dala využít. Jednak dosud neexistuje port pro tuto funkci pro Windows (a pravděpodobně ani pro jiné POSIX nekompatibilní systémy), jednak ani na různých distribucích unixu není zaručené její stejné chování.

Rozhodl jsem se pro vlastní implementaci alternativy funkce fnmatch. Pro shrnutí uvedu, že mě k tomu vedly následující důvody:

- Neexistující multiplatformní implementace fnmatch, ani jí podobné funkce,
- existující implementace mají napříč systémy odlišné chování,
- vyřazení užití funkce pomocí konfiguračního skriptu pro vytvoření překladového skriptu by vedlo tedy k úplně odlišnému chování různě přeložených knihoven,
- úplně vyřazení knihovny z implementace by vedlo k zbytečné spotřebě paměti (zmiňovaný obousměrně vázaný seznam bude tím menší, čím striktnější pravidlo bude pomocí vzorku aplikačním programátorem specifikováno)
- a v neposlední řadě v průběhu komunikace s komunitou vyvstaly názory, že by byla existence interní funkce byla vhodná.

Vznikla tedy funkce `Curl_fnmatch`, která svými parametry, ani návratovými hodnotami nesplňuje předpis POSIX, ale dává knihovně jednotné chování nezávisle na operačním systému, pod kterým běží.

Mechanismus porovnávání probíhá staticky pomocí konečného automatu, kde jednotlivé stavy jsou dány zadaným vzorkem. Každý znak symbolizuje jeden stav konečného automatu (lépe řečeno, pozice uvnitř zadaného vzorku, na kterou ukazuje ukazatel symbolizující stav konečného automatu — ten se postupně posouvá až na konec vzorku).

Vzhledem k různorodosti cílových architektur knihovny nebylo vhodné implementovat tuto funkčnost jako dynamicky generovaný interpretovaný kód (jako je tomu u regulárních výrazů). Byl by to jednak vzhledem k potřebné síle odpovídajícího jazyka dost silný prostředek, a především, každá zbytečná alokace na slabších architekturách může nepříznivě ovlivnit běh celé knihovny, respektive celé aplikace.

5.4 Manuálové stránky

Součástí implementace bylo i patřičné dokumentování rozšíření rozhraní knihovny pro její uživatele. Knihovna pro tyto dokumentační účely používá formát klasických manuálových stránek (`roff formát`). Odsud jsou poté generovány dokumentační stránky v HTML formátu, zveřejněné na stránkách projektu[3].

Vzhledem ke vzniku šesti nových voleb v rozhraní `curl_easy` vznikly odpovídající položky v dokumentaci. Další dokumentační část se týkala nově vzniklých (chybových) návratových kódů knihovny.

[3] http://curl.haxx.se/libcurl/c/

5.5 Testy

Při implementaci jsem si psal vlastní jednoduché programy, které testovaly funkčnost mého rozšíření. Dokonce jsem vytvořil dvě parametrizovatelné aplikace, které jsem poslal na mailing list. Tento přístup však nestačil pro efektivní testování rozšíření ani pro přijetí kódu komunitou.

Bylo nutné vytvořit sadu testů pro automatický testovací systém projektu. Tyto testy mají speciální formát, podobný XML. Takto specifikovaný test pracuje s curl nástrojem (*curl tool*) přeloženým proti knihovně libcurl testované verze. Každý blok daného testovacího souboru má svoji sémantiku — jsou zde bloky, které srovnávají výstup curl nástroje s obsahem daného bloku, blok, který umožňuje definovat povolený výčet návratových kódu tohoto programu a je zde dokonce možnost vytvořit vlastní mini-aplikaci pro specifické účely, pro které nelze curl nástroj využít, a podstrčit jej pro užití v tomto testu.

Testovací systém se skládá z několika skriptů psaných v jazyce Perl[9] a ze zásuvných modulů psaných v jazyce C, které jsou z těchto skriptů využity[4].

Výsledkem práce na testech byl vznik šesti nových testů, které kombinují následující účely:

- Otestovat funkčnost vestavěné porovnávací funkce `Curl_fnmatch`.
- Provést test na funkčnost zpracování UNIX či DOS LIST odpovědi.
- Otestovat funkčnost callback funkcí, které oznamují, shluk dat sady souborů přechází z obsahu jednoho souboru na druhý.
- Test *multi* rozhraní, které funguje v neblokujícím režimu.
- Test na využití funkce `curl_easy_duphandle`, která slouží ke kompletnímu překopírování `CURL` struktury a musí zachovat totožné chování duplikované struktury (objektu)
- Otestovat možnost vyměnit porovnávací funkci `Curl_fnmatch` za jinou přímo v aplikaci.

[4] Pro zajímavost uvedu, že libcurl podporuje tzv. „torture“ testy, které simulují selhání práce knihovny z důvodu nedostatku paměti — toto je implementováno pomocí předefinování funkcí pro práci s dynamickou pamětí (`malloc`, `free`, apod.)

Kapitola 6

Výsledky a zhodnocení

V této kapitole bude následovat shrnutí dosažených výsledků a přínos rozšíření knihovny libcurl. Zmíním se také o dalších možnostech vývoje a využití navržených principů.

6.1 Funkčnost rozšíření a přínos

Z hlediska zadání projektu byly splněny všechny požadavky na funkčnost, společně s ohledem na efektivitu knihovny je i protokolová komunikace minimalizována (proti stavu, kdy by si aplikační programátor zpracovával LIST odpověď serveru sám). Snažil jsem se minimalizovat počty alokací paměti kvůli architekturám s méně výkonnými alokátory paměti. Přitom se mi povedlo nezměnit původní chování knihovny a tím udržet zpětnou kompatibilitu. Vznikla tedy aplikačnímu programátorovi, který pracuje s knihovnou libcurl, možnost efektivního stažení několika souborů s využitím zástupných znaků. Pro využití mého rozšíření je potřeba vynaložit minimálního úsilí pro studium rozhraní a naopak odpadá potřeba implementace nízkoúrovňové komunikace se serverem a vlastního zpracování ASCII LIST odpovědi serveru. Implementované rozšíření lze využít čistě pro efektivní zpracování odpovědi bez toho, aby byl kterýkoliv ze souborů stažen — k tomu slouží právě možnost přeskočení souboru (v `CURLOPT_CHUNK_BGN_FUNCTION`).

Rozšíření bylo testováno na `proftpd`, `NcFTPd` a `vsftpd` serverech běžících na linuxových distribucích, `Microsoft FTP service`, `filezilla`, Cerberus FTP server a CesarFTP běžících pod Windows.

Zjednodušený příklad využití rozšíření knihovny je uveden v příloze A, úplný příklad společně s původní verzí knihovny a záplatou aplikující moje rozšíření naleznete na přiloženém kompaktním disku.

6.2 Zahrnutí rozšíření do zdrojových kódů projektu

Po několika několika iteracích úprav a komunikaci s komunitou, především hlavním vývojářem Danielem Stenbergem, bylo rozšíření 13. května 2010 přijato[1] do vývojové větve knihovny s tím, že v další vydané verzi libcurl (7.21.0, která vyjde v červnu 2010) bude již rozšíření obsaženo. Po zahrnutí rozšíření do zdrojových kódů knihovny byly během důkladného testování (jak automatickými testovacími systémy, tak vývojáři samotnými) nalezeny drobné nedostatky, na jejichž odstranění nyní pracuji.

[1] http://github.com/bagder/curl/commit/0825cd80a62c21725fb3615f1fdd3aa6cc5f0f34

6.3 Další možnost rozšíření

Návrh rozšíření byl proveden takovým způsobem, aby bylo možné kdykoliv doprogramovat podporu pro jiný typ FTP serveru, a především, aby bylo možné využít nově navržené rozhraní pro jiné protokoly či jiné účely. V současné době se (mimo FTP) vyvíjí podpora pro protokol IMAP, kdy pomocí jedné URL adresy bude zadán výčet zpráv. Jde o jiný systém využití zástupných znaků, ale v principu se jedná o velmi podobný princip a mohlo by být využito mnou navrženého rozhraní.

Jedno z prvních využití rozšíření bude pravděpodobně vytvoření nového přepínače v curl nástroji, který přepne knihovnu libcurl do režimu akceptujícího zástupné znaky a vypíše se obsah odpovídajících souborů na standardní výstup.

Literatura

[1] RFC 1738 -Uniform Resource Locators (URL). [online], Naposledy navštíveno v březnu 2010.
URL http://tools.ietf.org/html/rfc1738

[2] cURL HISTORY. [online], Naposledy navštíveno v dubnu 2010.
URL http://curl.haxx.se/docs/history.html

[3] fnmatch - match a filename or a pathname. [online], Naposledy navštíveno v dubnu 2010.
URL
http://www.opengroup.org/onlinepubs/000095399/functions/fnmatch.html

[4] Jakarta Commons Net. Naposledy navštíveno v dubnu 2010.
URL http://commons.apache.org/net/index.html

[5] JFtp Homepage. [online], Naposledy navštíveno v dubnu 2010.
URL http://j-ftp.sourceforge.net/

[6] RFC1700 - Assigned Numbers. [online], Naposledy navštíveno v dubnu 2010.
URL http://www.faqs.org/rfcs/rfc1700.html

[7] RFC959 - File Transfer Protocol. [online], Naposledy navštíveno v dubnu 2010.
URL http://tools.ietf.org/html/rfc959

[8] FRC 3659 - Extensions to FTP. [online], Naposledy navštíveno v květnu 2010.
URL http://tools.ietf.org/html/rfc3659

[9] The Perl Programming Language. [online], Naposledy navštíveno v květnu 2010.
URL www.perl.org

[10] Norguet, J.-P.: Java FTP client libraries reviewed. [online], 2003.
URL http://www.javaworld.com/javaworld/jw-04-2003/jw-0404-ftp.html

[11] Norguet, J.-P.: Java FTP libraries benchmarked. [online], 2003.
URL http://www.javaworld.com/javaworld/jw-03-2006/jw-0306-ftp.html

[12] Stenberg, D.: Oficiální stránky projektu cURL. [online], Naposledy navštíveno v květnu 2010.
URL http://curl.haxx.se/

Příloha A

Ukázka práce s rozšířením

```
#include <curl/curl.h>
#include <stdio.h>

struct chunk_data { int islast; };

int chunk_begin(struct curl_fileinfo *finfo, struct chunk_data *data,
                int remains) {
  ... práce v callback funkci ...
  return CURL_CHUNK_BGN_FUNC_OK;
}

int chunk_end(struct chunk_data *data) {
  ... práce v callback funkci ...
  return CURL_CHUNK_END_FUNC_OK;
}

int main() {
  ... inicializace ...
  /* nastavení adresy obsahující zástupné znaky */
  curl_easy_setopt(handle, CURLOPT_URL, "ftp://debian.org/debian-archive/*");
  /* zapnout podporu pro zástupné znaky */
  curl_easy_setopt(handle, CURLOPT_WILDCARDMATCH, 1L);
  /* nastavím funkce, které budou volány před a po stažení dílčího souboru */
  curl_easy_setopt(handle, CURLOPT_CHUNK_BGN_FUNCTION, chunk_begin);
  curl_easy_setopt(handle, CURLOPT_CHUNK_END_FUNCTION, chunk_end);
  /* nastavím uživatelský ukazatel, který bude předáván v CHUNK callback
     funkcích */
  curl_easy_setopt(handle, CURLOPT_CHUNK_DATA, &ch_data);
  /* spustit přenos */
  ret = curl_easy_perform(handle);
  ... dealokace, ošetření návratového kódu ...
}
```