Dokumentace k projektu pro předmět ISA

Programování síťové služby: Jednoduchý LDAP klient

Autor: Jan Vybíral, xvybir05@stud.fit.vutbr.cz

OBSAH

Kapitola 1

Úvod do problematiky

1.1 LDAP

Lightweight Directory Access Protocol (LDAP) je protokol pro ukládání a přístup k datům na adresářovém serveru. Podle tohoto protokolu jsou jednotlivé položky na serveru ukládány formou záznamů a uspořádány do stromové struktury. Je vhodný pro udržování adresářů a práci s informacemi o uživatelích (např. pro vyhledávání adres konkrétních uživatelů v příslušných adresářích, resp. databázích). Protokol zasílá data ve standardu ASN.1 kódovaná pomocí BER (Basic Encoding Rules). Úkolem bylo naprogramovat jednoduchého klienta, který by umožňoval zasílání LDAP dotazů na vyhledání osob v adresáři FIT VUT a následně zpracovával a zobrazoval odpovědi serveru.

1.2 BER kódování

Každý datový element (např. číslo nebo řetězec) je zakódovaný jako identifikátor typu dat, informace o délce těchto dat a pak až samotná data. Tento typ kódování se nazývá TLV kódování (Type-Length-Value):

1	2..2+n	2+n..2+n+length
Type	Length	Value

Type - tato část sestává z osmi bitů, které charakterizují povahu dat v části value:

8	7	6	5	4	3	2	1
Class		P/C	Number				

Class:

Class	Popis	Bit 8	Bit 7
Universal	Nativní typ ASN.1	0	0
Application	Validní pouze pro jednu specifickou aplikaci	0	1
Context-specific	Záleží na kontextu	1	0
Private	Může být definováno v privátních specifikacích	1	1

P/C:

1 = constructed (např. SEQUENCE, obsahuje další TLV prvky)
0 = primitive (např. integer)

Number - specifikuje konkrétní datový typ.

Popis LDAP zpráv: http://tools.ietf.org/html/rfc177

Kapitola 2

Návrh aplikace

2.1 Převod filtru do prefixové podoby

V konfiguračním souboru je podle zadání filtr uveden v infixové notaci, BER kódování ale pracuje s prefixem, proto jsem se tedy rozhodl udělat nejdříve převod. Ten pro jednoduchou nezanořenou logiku specifikovanou v zadání spočívá v posunutí symbolů & a | o jednu pozici doleva a posunutí ! před danou rovnost, takže např. z řetězce a=b&c=!d&e=f vznikne řetězec &a=b&!c=de=f. Kvůli potřebě pozdějšího zpracování tohoto výrazu při zakódovávání, kdy musím vědět, kde začíná a končí každá část filtru, jsem se rozhodl přidat do tohoto výsledku ještě závorky tímto způsobem: (&(a=b)(&(!(c=d))(e=f))).

2.2 Zakódování prefixového filtru

Rozhodl jsem se zakódovávat filtr odzadu dopředu, protože je podle BER potřeba počítat velikosti jednotlivých částí a vkládat je před ně. Celé je to konečný automat se stavy 0 – 4. Počátečním stavem je stav 0. Postupně se prochází filtr znak po znaku a v závislosti na přečteném znaku a aktuálním stavu se provádějí akce a mění stavy, přičemž výsledný zakódovaný řetězec vzniká v proměnné result, ve které je na počátku prázdný řetězec:

Aktuální stav	Aktualni znak z filtru	Další stav	Akce
0	')'	0	
	'*'	3	
	jiný znak	2	result = jiný znak + result
1	'('	1	
	'&'	1	result = AND + result
	'|'	1	result = OR + result
	'!'	1	result = NOT + result
	')'	0	
2	'*'	3	result = FINAL + result
	'='	2	result = STRING + result
	'('	1	result = STRING + result result = EQUALITY MATCH + result
	jiný znak	2	result = jiný znak + result
3	'*'	3	result = ANY + result
	'=' následované '*'	3	result = SEQUENCE + result
	'='	3	result = INITIAL + result result = SEQUENCE + result
	'('	1	result = STRING + result result = SUBSTRINGS + result
	jiný znak	3	result = jiný znak + result

Když se má vložit type-length část (věci napsané velkými písmeny), procházím result od začátku, dokud nenarazím na byte, kterým může začínat další TLV část, a počítám velikost, která se má vložit.

2.3 Dekódování odpovědi

Postupuji ve zprávě od začátku do konce. Je to konečný automat s 5 stavy:

0. Jsme na SEQUENCE (0x30)
1. Jsme na messageID (0x02)
2. Jsme na APPLICATION 4 (0x64)
3. Jsme na string (0x04)
4. Jsme na APPLICATION 5 (0x65)

Počátečním stavem je stav 0. V každém stavu se provede nějaká akce (někdy na základě podmínky) spočívající buď v zapsání části searchResponse do výsledné proměnné nebo v posunutí se v searchResponse (přeskočení nějaké její části, buď TL (type-length část) nebo celého jednoho elementu TLV)):

Aktuální stav	Podmínka	Akce	Další stav
0		Přeskočit TL sequence	
	Jsme na integer		1
	Jsme na string		3
1		Přeskočit TL messageID	
	Jsme na APPLICATION 1		4
	Jsme na APPLICATION 4		2
	Jsme na APPLICATION 5		4
2		Výsledek += odřádkování Přeskočit TL APPLICATION 4 Přeskočit objectName	0
3		Přeskočit TL string Výsledek += attributeType Výsledek += '=' Přeskočit TL SET OF Přeskočit TL string Výsledek += attributeValue Výsledek += '&'	0
4		Přeskočit TL APPLICATION 5 Přeskočit ENUMERATED Kontrola resultCode Přeskočit TL string Kontrola MatchedDN Přeskočit TL string Kontrola errorMessage	Tady dojdeme na konec searchResponse.

Výsledek má potom tvar např. :

cn=Vybíral Jan&uid=xvybir05&
cn=Vybíral Petr&uid=xvybir07&
cn=Vybíral Radim&uid=xvybir06&

Kapitola 3

Implementace

3.1 Zpracování konfiguračního souboru

Provádi se ve funkci `int config(char filename[], string & host, int & port, string & base, int & depth, string & filter, string & result)`. Program předpokládá, že konfigurační soubor bude vždy v přesně tom tvaru, jak je uvedeno v zadání. Takže se prochází jen každý druhý řádek, data se interpretují podle toho, co má na daném řádku podle specifikace v zadání být. Načtená data se ukládají do patřičných proměnných pro jednotlivé parametry v tom tvaru, jak jsou napsány v konfiguračním souboru.

3.2 Uživatelské vstupy

K načítání vstupů slouží funkce `string input(string filter)`, která nalezne ve filtru z konfiguračního souboru první výskyt podřetězce %s a následně se zeptá uživatele na vstup, kterým nahradí tento podřetězec. Toto se provádí, dokud ve filtru zůstává nějaké %s.

3.3 Generování zpráv

Celý searchRequest se vytváří ve funkci `string createrequest(int id, string base, int depth, string filter, string result)`, která ho vytváří odzadu kvůli nutnosti počítat velikosti prvků a vkládat je před ně. Tato funkce v části, kde vkládá filtr, volá funkci `string encode(string filter)`, která nejdříve převede filtr do prefixu zavoláním funkce `string toprefix(string filter)`, a ho zakóduje. Podobným způsobem se vytváří i unbindRequest ve funkci `string create_unbind(int id)`. BindRequest se negeneruje, protože může být pokaždé stejný, je řešen jako konstanta.

3.4 Kódování délek jednotlivých elementů v BER

Je třeba řešit možnost, že údaj délky bude tak vysoké číslo, že bude muset být rozděleno na více bytů. K tomu slouží především funkce `string divide_number(int number)`, která z celočíselné reprezentace udělá 1–n bytovou posloupnost, která bude v LDAP zprávě. Tuto funkci využívá funkce `void divide_length(string & message, int value)`, která její výsledek spolu s jeho délkou vloží do generované zprávy. Naopak je třeba u přijaté LDAP zprávy informaci o délce správně spojit a převést do celočíselné reprezentace. K tomu zase slouží funkce `int getlength(strng divided_length)`.

3.5 Komunikace se serverem

Komunikace probíhá prostřednictvím BSD socketů. Odesílání dat na server provádí funkce `int send(int s, string message)`. Funkce `int receive(int s, string & response)` přijme jednu TCP zprávu. Z funkce main se volá, dokud nebyl přijat konec searchResponse.

3.6 Dekódování odpovědí serveru

Rozkódování přijaté searchResponse se provádí ve funkci `string decode(string response, bool & response_end, bool & ldap_error)`. Přeskakování zajišťují funkce `int skip_length(string message, int i, int & length)`, která přeskočí i z type na první znak value a přitom vrátí samotnou délku v parametru, a `int skip_tlv(string message, int i)`, která dělá to samé s celým TLV záznamem. Pokud bylo naraženo na APPLICATION 5 (0x65), byla přijata celá odpověď. Pokud ne, přijde ještě další část, což funkce indikuje nastavením response_end na false, aby funkce main věděla, že má čekat na další zprávu od serveru.

3.7 Tisknutí výstupu

Provádí se ve funkci `void print_results(string decoded, string attributes)`. Decoded je výstup funkce decode. Ten je ve tvaru podle 2.3. Attributes je řetězec obsahující mezerami oddělené atributy z konfiguračního souboru, jejichž hodnoty se mají v tomto pořadí vypsat na výstup. Pro každý řádek v decoded se provede: postupně pro všechny atributy v attributes se tento atribut vyhledá na řádku a následující část za = až po &, tedy jeho hodnota, se přidá do výsledného výstupu. Následně se vloží pro lepší přehlednost oddělovač | nebo odřádkování v případě, že jde o poslední atribut. Pokud decoded obsahuje prázdný řetězec, vypíše se informace o tom, že nebylo nic nalezeno. Pokud attributes obsahuje prázdný řezězec, vypíše se celá vrácená odpověď včetně jmen všech atributů.

3.8 Ukončení programu

Korektní ukončení programu je řešeno odchycením signálu SIGINT (ctrl+c), při kterém se zavolá funkce `void finish(int s)`, ve které dojde k odeslání unbindRequestu, zavření socketu a ukončení aplikace. Kvůli nutnosti manipulovat v této funkci se socketem i id zprávy, přičemž může mít jen jeden parametr typu int, je proměnná id globální.

3.9 Funkce main

Pokud nebyl program spuštěn s jedním parametrem, vypíše se chyba a program se ukončí. Dále se naváže spojení se serverem, pošle se bindRequest a přijme bindResponse. Dále pokud filtr v konfiguračním souboru obsahuje nějaký podřetězec %s, uživatel je požádán o jeho zadání a to tolikrát, kolik takových podřetězců tam je. Pokud tam není žádný, uživatele se na nic neptám. Následně se odešle searchRequest a přijímají se odpovědi, dokud nebyla přijata konečná část bindResponse (APPLICATION 5). Server totiž může poslat odpověď odděleně v několika úsecích. Po přijetí celé odpovědi se tato vytiskne. Pokud ve filtru je %s, je uživatel znovu dotazován na vstup, dokud nezadá ctrl+c, kdy se program ukončí. Pokud filtr nevyžaduje žádný vstup, program se ukončí po prvním běhu.

Kapitola 4

Návod na použití:

1. Přeložte program příkazem make v adresáři, kde je zdrojový soubor myldapsearch.cpp a soubor Makefile.
2. Program následně spustíte příkazem $./myldapsearch <název konfiguračního souboru>
3. Pokud filtr v konfiguračním souboru obsahuje volitelné podřetězce, budete dotázáni na jejich postupné zadání jeden po druhém. Zadávejte je v tom pořadí, v jakém jsou ve filtru. Po zadání posledního z nich se objeví výsledky, případně chybové hlášení (stává se, pokud odpovídajících záznamů je příliš mnoho, server potom vrátí jen prvních pár a chybu.
   V takovém případě se částečný výsledek nevypisuje, program pouze zahlásí chybu a doporučí zúžit vyhledávací kritéria.) Poté budete znovu automaticky vyzváni k zadání dalších údajů pro další vyhledávání a takhle pořád dokola, dokud nestisknete ctrl+c, načež se program korektně ukončí.
4. Pokud filtr v konfiguračním souboru neobsahuje volitelné podřetězce, vypíše se výsledek vyhledávání (příp. chyba) a program se okamžitě sám ukončí.
5. Pro správnou funkčnost s českými znaky a jejich korektní zobrazování musí mít konfigurační soubor a konzole nastaveno kódování UTF-8.