Dokumentácia k projektu

Síťové aplikace a správa sítí

Varianta: DNS resolver

17.11.2019 Adrián Tulušák (xtulus00)

Obsah

1. Úvod

Cieľom projektu bolo vytvoriť program DNS, ktorý bude vedieť zasielať dotazy na DNS servery a v čitateľnej podobe bude vypisovať prijaté odpovede na štandardný výstup. Program využíva komunikáciu len prostredníctvom UDP. Program je implementovaný v jazyku C++.

2. Návrh a implementácia

Aplikácia je rozdelená do viacerých tried a funkcií, v ktorých je implementované chovanie jednotlivých častí. Prototypy funkcií, štruktúry, použité knižnice a definície konštánt sú uvedené v hlavičkovom súbore dns.h. Ostatné časti programu a hlavne implementácia tried a funkcií sú v súbore dns.cpp.

2.1. Spracovanie vstupných argumentov

Po spustení aplikácie sa vytvorí inštancia triedy *Arguments*, ktorá obsahuje informácie o tom, ktorý z vstupných argumentov bol použitý a prípadne s akými hodnotami. Následne je v tejto inštancií volaná metóda *handleArguments()*, ktorá prečíta všetky argumenty zadané pri spustení a ak je niektorý z možných argumentov rozoznaný, jeho odpovedajúcej premennej sa priradí *true*, čo značí, že už bol úspešne prečítaný a nemôže byť použitý znovu. Všetky argumenty majú zadaný presný formát, v ktorom môžu byť použité okrem adresy, ktorú žiadame preložiť. Ak žiaden z argumentov nevyhovuje, pripíše sa hodnota argumentu do tejto adresy. Ak je v nesprávnom formáte, to sa vyhodnotí až neskôr alebo to vyhodnotí DNS server ako neznámu adresu. Následne prebieha kontrola, či boli použité povinné parametre. V prípade nevyhovovania ktoréhokoľvek z argumentov je volaná funkcia *err()*, ktorá ukončí program s príslušným návratovým kódom (popísané sú v *dns.h*).

2.2. Vytvorenie Query

Po úspešnom spracovaní vstupných argumentov nasleduje vytvorenie DNS Query. Program vytvorí dve inštancie triedy *bufferClass*, ktorá je tvorená bufferom charov o dĺžke 512, kvôli obmedzeniu dĺžky UDP paketu je teda táto dĺžka viac než dostačujúca. (vychádzajúc z Síte a komunikace – Petr Matoušek, Kapitola 3.4). Táto trieda tiež obsahuje pointer na koniec (aktuálnu pozíciu) v bufferi.

2.2.1. Header

Nasleduje volanie funkcie *sendQuery()*. Táto funkcia zostrojí a odošle DNS paket. Vytvorením inštancie triedy *Header* sa už v konštruktore tejto triedy vygeneruje takmer celá hlavička a následne sa metódou *RFlag()* nastaví flag požadovania rekurzie v závislosti od zadaných parametrov. Hlavička sa zapisuje rovno do bufferu, ktorý bude odosielaný.

2.2.2. Question

Po vytvorení hlavičky sa vytvorí zvyšok Query – teda Question. Slúži na to trieda *Question*. Tá si v konštruktore nastaví flagy požiadaviek (reverzná query, ipv6 namiesto ipv4) a hodnoty DNS serveru a adresy na preloženie, ktoré sú získané zo vstupných argumentov. Ak sa jedná o štandardnú požiadavku (nie reverznú), je volaná metóda *addAddressToBuffer()*. Funkcia *addAddressToBuffer()* rozdelí adresu na subdoménu (rozdelí v podľa bodky) a potom jednotlivé subdomény pripisuje do bufferu bo formáte danom štandardom RFC1035 (dĺžka reťazca, ktorý bude nasledovať a za ním samotný reťazec – kompresia používaná v DNS v tejto časti nie je použitá, nakoľko v požiadavke to pokladám za zbytočné)

V prípade reverznej požiadavky je v konštruktore triedy *Question* volaná metóda *reverseQuery()*. Táto metóda je rozdelená na dve časti: na spracovanie IPv4 adresy a IPv6 adresy. Pri spustení programu nie je ničím dané, ktorá z adries bude zadaná (pri použití argumentu „-r“ sa použitie/nepoužitie argumentu „-6“ ignoruje – zo zadania mi nebolo celko zrejmé, ako sa v takejto situácií má program zachovať, preto som sa rozhodol pre takéto použitie), a preto na začiatku tejto metódy prebehne kontrola prostredníctvom RegEx-ov, či je použitá IPv4 alebo IPv6 adresa. Ak ani jednému RegEx-u zadaná adresa nevyhovuje, program sa ukončí s príslušnou chybou návratovou hodnotou (6 – chybná zadaná adresa).

IPv4 adresa sa, ako je uvedené v RFC1035, rozdelí po bajtoch (hodnoty oddelené bodkou), obráti sa a pripíše sa k nej reťazec „in-addr.arpa.“. Presne tento postup je vykonávaný v metóde *reverseQuery()*. V cykle sa adresa rozdelí podľa bodky a vytvorí sa vector z týchto 4 adries. Nad týmto vectorom sa zavolá funkcia *reverse()*. Vector sa prepíše do stringu, jednotlivé časti vectoru sa oddelia bodkou a pripíše sa reťazec „in-addr.arpa.“ (v tomto programe sa vždy na konci domény píše bodka – keď pri vstupe nie je použitá, dopíše sa tam). Následne sa string už len zapíše do bufferu metódou *addAddressToBuffer()*, teda rovnako ako pri zadaní doménového mena.

IPv6 adresa sa spracuje obdobne ako IPv4. Rozdielom je to, že sa nerozdeľuje podľa bodky, ale podľa dvojbodky a tiež skutočnosť, že IPv6 adresa môže byť skrátená. Tu sa vytvoria dva vectory, jeden pre časť pred skrátením a druhý pre časť po skrátení. Jednotlivé bajty sa potom rovnakým spôsobom ukladajú do týchto dvoch vectorov. Následne sa doplnia nuly medzi tieto dva vectory tak, aby mala adresa dĺžku 128 bitov a vytvorí sa jeden kompletný vector. Toto sa deje v metóde *fillIPv6WithZeroes()*, ktorá vracia kompletný vector. Potom už je postup rovnaký ako pri IPv4. Adresa sa prevráti, pripíše sa „ip6.arpa.“ a zavolá sa funkcia *addAddressTobuffer()*.

Na konci sa do bufferu pripíše 0, čo symbolizuje bodku na konci doménového mena a značí to koniec časti QName.

Po vyplnení QName sa už vyplní len QType a QClass pomocou pripravených metód *addShort()*, ktoré pripíšu 2 bajty do bufferu.

2.3. Odosielanie Query a prijatie Answer

Po naplnení bufferu sa buffer odošle prostredníctvom UDP paketu funkciou *sendto()*. Následne sa odpoveď z funkcie *recvfrom()* priradí do bufferu pre prijatú správu.

2.4. Spracovanie odpovede

Ukončením funkcie *sendQuery()* sa aplikácia vracia do *main()*, kde pokračuje volaním funkcie *parseAnswer()*. Táto funkcia skontroluje ID v hlavičke DNS paketu, či sa zhoduje s odoslaným ID. Následne skontroluje aj zvyšok hlavičky, najmä flagy. Kontrola flagov je implementovaná bitovými maskami vo funkcií *checkRcvdHeader()*. Táto funkcia vracia objekt triedy *DNSHeaderParams* naplnený údajmi vyčítanými z hlavičky. Po kontrole sa posunie pointer na aktuálnu pozíciu v bufferi a program začína čítať ostatné sekcie. Čítanie všetkých 4 sekcií prebieha veľmi podobne. Najprv sa prečíta počet položiek v časti a následne sa v cykle číta daný počet položiek funkciou *readAnswer()*. Tieto počty sú už uložené v atribútoch triedy *DNSHeaderParams*.

Funkcia *readAnswer()* najprv zavolá metódu triedy *bufferClass - readAddress()*, ktorá z bufferu prečíta adresu. Táto funkcia spracováva aj kompresiu DNS paketov cez odkazy na už použitú časť adresy. Čítanie tejto kompresie je riešené rekurzívnym volaním metódy *readAddress()* ale vždy s iným offsetom. Po prečítaní adresy sa prečítajú z bufferu 2 bajty s hodnodou typu odpovede, ďalej 2 bajty s typom triedy odpovede, 2 bajty s hodnotou TTL, 2 bajty dĺžky odpovede a následne dáta odpovede o dĺžke uvedenej v predošlej položke. Dáta sa čítajú v metóde *readRdata()*. Metódu tvorí hlavný *switch-case*, ktorý rozlišuje spracovanie odpovede podľa typu. Program dokáže rozpoznať typy A, NS, MD, MF, CNAME, SOA, MB, MG, MR, NULL, WKS, PTR, HINFO, MINFO, MX, TXT a AAAA. Dáta vyhodnocuje u typov A, AAAA, CNAME, PTR. U ostatných v časti „dáta“ vypíše „unknown“.

Vyhodnocovanie dát v typoch A a AAAA je implementované samostatne v jednoduchých cykloch o dĺžke danej verzie IP adresy. Dáta z typu CNAME a PTR využívajú metódu *readAddress()*.

Následne sa tieto spracované údaje z každej prečítanej odpovede vypíšu na StdOut.

Po úspešnom vypísaní všetkých odpovedí program vracia hodnotu 0 a ukončí sa.

3. Použitie

3.1. Preklad

Preklad prebieha pomocou súboru CMake. Spustenie prekladu je realizované programom GNU Make použitím príkazu *make*, ktorý vytvorí spustiteľný súbor ***dns***. Príkazom *make clean* je možné odstrániť vytvorené spustiteľné súbory.

3.2. Návod na použitie

Po preložení programu je možné spustiť vytvorený súbor *dns* a to nasledovným spôsobom:

./dns [-r] [-x] [-6] -s server [-p port] adresa

kde:

[ ] znamená voliteľný parameter

-r: požadovaná rekurzia

-x: reverzná požiadavka – pri použití tohto parametra je potrebné zadať adresu vo formáte IP adresy (IPv4 alebo IPv6); tento parameter má prednosť pred „-6“

-6: požiadavka typu AAAA (namiesto implicitného A) – ak je použitý parameter „-x“, tento parameter nemá žiaden význam

-s server: IP adresa alebo doménové meno serveru, kam bude požiadavka odosielaná

-p port: číslo portu, na ktorý bude požiadavka odosielaná

adresa: adresa, ktorá sa žiada preložiť

4. Testovanie

Za účelom testovania som si vytvoril skript v jazyku php, ktorý kontroloval správnosť fongovania programu *dns* a výsledky vypisoval do HTML súboru s názvom *DNS_TEST_LOG.html*. Spustenie testovacieho skriptu je možné príkazom *make test*. Skript spustí všetky súbory s príponou *.src* v priečinku *tests/* v aktuálnom adresári. Súbory s príponou *.src* obsahujú parametre pre spúšťanie programu *dns*. K nim rovnomenné s príponou *.rc* obsahujú očakávanú návratovú hodnotu programu. Ak takýto súbor neexistuje, automaticky sa vytvorí a implicitne sa očakáva návratová hodnota 0.

5. Záver

Myslím si, že by moje riešenie malo spĺňať všetky požiadavky zadania. Pri testovaní som odhalil zopár malých chýb, ktoré boli odstránené.

Pri riešení projektu najviac času zabralo študovanie potrebnej literatúry, predovšetkým už spomínaná kniha Síte a komunikace – Petr Matoušek, RFC1035 a RFC3596. Dôležité bolo pochopenie fungovania DNS serverov a spôsob komunikácie DNS serveru a klienta. Veľmi zdĺhavé bolo vytváranie DNS paketov, predovšetkým hlavičky, kde zapísanie každého bitu vyžadovalo veľa pozornosti a času na dôkladné preskúmanie možností a prípadov, v ktorých nadobúdajú tieto bity konkrétne hodnoty. V časti kontrolovania hlavičky maskovaním je v zdrojovom súbore aj moja pomôcka, ktorú som si vytvoril na základe schémy hlavičky v RFC1035, do ktorej som si označil, ktoré bity sú premenlivé, ktoré nie a aké môžu mať hodnoty.

Zvyšok implementácie už nebol tak zložitý, tak náročný na pozornosť, ani náchylný na chybovosť. Bol síce pomerne rozsiahly, ale implementácia už bola zaujímavejšia, keď nebolo nutné každému bitu venovať 10 minút.

6. Literatúra

[1] Matoušek, P. 2014. Síťové aplikace a jejich architektura. Brno:Akademické nakladatelství, VUTIUM, 2014. 396s. ISBN 9788021437661

[2] Mockapetris, P.V. 1987. Domain names – Implementation and specification. [online] https://tools.ietf.org/html/rfc1035

[3] Thomson, S. 2003. DNS Extansion to support IP Version 6. [online] https://tools.ietf.org/html/rfc3596

[4] Hnidek, J. 2014. Getaddrinfo example. [online] https://gist.github.com/jirihnidek/bf7a2363e480491da72301b228b35d5d

[5] Amidts, 2018. UDP Server-Client implementation in C. [online] https://www.geeksforgeeks.org/udp-server-client-implementation-c/