Projekt ISA 2019/20: Dns resolver

autor: xplsek03 (Michal Plšek)

Obsah

Použití programu a odevzdané části

Program se spouští s následujícími volbami:

./dns [-x] [-r] [-6] -s dns_server [-p číslo_portu] adresa

./dns --help

Popis parametrů

--help: nápověda k programu

-x: rezeverzní DNS dotaz, adresa může být zadána pouze jako Ipv4/v6 adresa

-r: rekurzivní dotaz

-6: dotaz typu AAAA místo defualtního A (použití Ipv6)

-s: dns server, zadaný jako doménové jméno nebo IP adresa

-p: volitelné číslo portu, výchozí 53

adresa: dotazované doménové jméno (nebo IP adresa v případě rDNS dotazu)

Odevzdané části

dnsresolver.c, functions.c, unctions.h, Makefile, README, manual.pdf

Součástí programu je také dynamický (!) generátor testů v souboru tests.py. Generuje výsledky testů do složky /tests, a to do souborů:

generated (všechny testy)

passed

failed

Základní informace

Hlavním úkolem dns resolveru je překlad doménových jmen na IP adresy (i zpětně za předpokladu reverzního dns dotazu) a výpis záznamů, které k doménovému jménu náleží. Plně podporované třídy záznamů: IN, CS, HS, CH. Plně podporované typy záznamů: CNAME, A, AAAA, NS, PTR, SOA, TXT, ostatní typy záznamů tiskne jako HEX. V případě jiné třídy záznamu program jako třídu uloží „???“, v případě jiného typu uloží „??“.

Návrh a popis implementace

Buffery pro uložená doménová jména mají v kódu délku řetězce 257 (256 + „\0“) znaků, při procházení různých implementací jsem si všimnul že se často zapomíná na možnost tečky na konci (kořenová doména). Maximánl délka doménového jména je podle RFC 255 znaků, maximální délka labelu (jednotlivá doména = část doménového jména) je 63 znaků.

V případě požadavku na reverzní dotaz (rDNS, možnost x) dojde k ověření, že je zadaný argument ip adresa (IPv4 nebo Ipv6). V případě rDNS požadavku je procházen strom domén addr-in.arpa (ipv4) nebo ip6.arpa, IP adresy jsou revertovány, což je kvůli struktuře stromu (reverzní pořadí umožňuje snadnější správu prostoru doménových jmen). V případě, že se jedná o standartní požadavek, dojde k ověření platnosti doménového jména. Následuje validace protu (0 – 65535, i nula je možná).

Dochází ke kontrole, zda argument dns serveru zadán jako IP adresa nebo jako doménové jméno. Pokud se jedná o IP adresu, musí vyhovovat požadovanému formátu (v případě argumentu -6 musí být Ipv6 a obráceně), jinak program skončí chybou: Pokud je požadován dotaz typu AAAA, předpokládá se, že zařízení podporuje na některém svém síťovém rozhraní Ipv6, pokud ne a později dojde k pokusu o odeslání, je program přerušen systémovou chybou. V případě nezadaného argumentu -6 je odesílán standartní dotaz typu A a předpokládá se podpora Ipv4. V případě, že dns server hodnota je zadána jako doménové jméno, dojde k převodu na IP adresu pomocí systémového volání getaddrinfo. Pokud existují adresy k tomuto jménu, otestuje se k nim postupně připojení a pokud se jej podaří dosáhnout, vrátí se první použitelná adresa příslušné verze. V rámci testování připojení se vytváří socket, který se poté zavře.

Po ověření vstupních argumentů (naparsovaných pomocí knihovny getopt) se vytváří DNS datagram. Velikost DNS datagramu (operujeme na úrovni UDP/TCP, v našem případě jen UDP; takže datagram) může být větší než 512B (RFC 5625). Jako ID v hlavičce je použito id procesu, pro nastavení flagů je místo C bitových

polí použito maskování bitů. Je nastavena hodnota bitu rekurze, pokud je požadována rekurze argumentem -r. Požadované doménové jméno/IP adresa je převedena do formátu dns: posloupnost ve formátu délka labelu, následovaná znaky labelu (včetně kořenové domény, tzn. Ukončení výsledného řetězce znakem 0). Převod se používá kvůli kompresi doménových jmen, v tomhle případě kvůli možnosti více dotazů (a později v odpovědi kvůli poskládání doménového jména do původního tvaru, s použitím offsetu). Pokud je požadováno rDNS, dotaz je vždy typu PTR, za předpokladu argumentu -6 je dotaz nastaven na typ AAAA, jinak na typ A. Třída dotazu je výchozí IN (1).

Poznámka k bitovému maskování: k odstranění paddingu se posílá zpráva compileru: v souboru functions.h:

`#pragma pack(push, 1)`

`#pragma pack(pop)`

Pokud by se přidal padding, struktura hlavičky DNS by místo stávajících 10B měla 12B a buď by tím utrpělo parsování nebo přehlednost kódu (použití `sizeof(HEADER)-2`).

Dochází k vytvoření socketu, podle zadaných argumentů buď Ipv4 nebo Ipv6. V obou případech je nastaven timeout socketu na 3 vteřiny, pokud je nedojde k přijetí odpovědi, je dotaz zaslán ještě dvakrát s čekáním 0.25s.

Drobná poznámka k funkcím ntohs() a ntohl(), používaným při parsování odpovědí. Tyto funkce převádí hodnoty – short / long z formátu používaného síťovými zařízeními. Protokol IP používá podle RFC 1700 pořadí bajtů big-endian, jedná se o převod pořadí bajtů ze sítě na pořadí bajtů u klienta (v drtivé většině little-endian, relativně k zadání projektu).

V případě přijetí odpovědi je ověřeno, že je id v hlavičce stejné jako id odchozího datagramu a jestli je AA bit nastaven jako odpověď. Pokud je nastaven některý z bitů Opcode (i v případě jiné než oficiální kombinace bitů, tzn chyby číslo 1-5), je program ukončen.

Pokud je argumentem -r vyžadována rekurzivní odpověď (rekurzivní zjišťování záznamu, pokud ho zadaný dns server tedy podporuje, probíhá tak, že dns server postupně vyhledává záznam a pokud ho nemá v cache, oslovuje další dns servery, než se dostane k požadované odpovědi. V případě kořenových serverů je dobré vědět, že vždy fungují iterativně, aby nedocházelo k jejich přetěžování. Iterativní dotazování – tedy nerekurzivní – probíhá tak, že dns server, který nemá povolenou/implementovanou rekurzi vrátí adresu dns serveru, který obsahuje záznam o doméně - zpravidla kořenový server. Prostě se jedná o rekurzivní dotaz, který se zastaví na kroku jedna a další dotazování v případě zájmu uživatele/uživatelského programu nechá na resolveru, který data odeslal).

Poznámka k zadání: v zadání je uvedeno, že resolver si musí poradit se záznamy typu CNAME. Otázkou je, jestli je to myšleno tak, že má resolver v případě nerekurzivní odpovědi umět vytisknout CNAME záznam, nebo tak, že v případě nerekurzivního postupu sám postupovat rekuzivně. Protože je logické (a účelem resolveru), aby se dns resolver snažil dostat k finální odpovědi, tj k záznamům které obsahuje doménové jméno, jež je uvedeno v záznamu CMAKE požadované domény, bokem jsem implementoval verzi resolveru, který postupně odesílá DNS datagramy tak dlouho, až se dostane k cílovému záznamu. Tahle verze není součástí odevzdávaného archivu, protože složitost implementace byla asi dvojnásobná a nebyl čas na ni napsat testy. Otázkou totiž je, jestli argument -r (dokumentace: požadována rekurze (RD = 1), jinak bez rekurze) znamená, jestli požadujeme rekurzi po dns serveru nebo po dns resolveru. Pokud bychom totiž požadovali rekurzi po dns serveru a on by ji nepodporoval, úkolem dobrého dns resolveru by pořád bylo poskytnout co nejlepší - nejpřesnější - rekurzivní odpověď. V zadání je zmíněno, že nejasnosti ohledně implementace máme vyřešit podle sebe a popsat v dokumentaci. Vzhledem k tomu, že resolver neposkytuje výsledky dál (do prohlížeče nebo k nějakému dalšímu zpracování), ale tiskne je na výstup, zadání chápu tak, že argument -r znamená nastavení bitu RD requestu na server a tím se s rekurzí končí (dojde pouze k vytisknutí příslušné odpovědi). V případě, že by tohle byl úmyslný chyták a byla vyžadována rekurze po resolveru, tak jsem ochoten poskytnout verzi zdrojového kódu, která toto chování implementuje[1].

Vypíše se údaj o tom, jestli byla odpověď zkrácena (TC bit v hlavičce odpovědi) a jestli je autoritativní (zadaný dns server z hlediska k doménovému jménu v požadavku).

Je vypsán obsah DNS requestu: pro případ nějaké manipulace s DNS requestem na serveru je zkontrolována otázka: jestli je počet otázek v hlavičce DNS datagramu =1, jestli je třída otázky IN a jestli je typ otázky stejný jako typ v dns requestu, který jsme odesílali. Poté dojde k vytisknutí sekce odpovědí.

Odpovědi v DNS reply datagramu jsou různých typů: základní odpovědi, autoritativní odpovědi a přídavné odpovědi. Jejich počet je uveden v hlavičce reply DNS datagramu, podle tohoto údaje lze všechny odpovědi naparsovat. Jména v DNS reply datagramu, která mají proměnlivou délku, program parsuje tak, aby vrátil zároveň i délku doménového jména a tím věděl, o kolik bajtů se má v paketu při parsování dalších údajů posunout. Při parsování řetězců jmen program postupuje, dokud nenarazí na znak 0 (kořenová doména). V případě že narazí na odkaz – ten pozná podle nastavených dovu horních bytů konkrétního bajtu, přeskočí na pozici, kterou tento offset určí (od začátku datagramu). Kořenová doména určí, že řetězec skončil, a dojde k přeložení uloženého řetězce v dns formátu na čitelný formát doménového jména. Struktury odpovědí i otázky jsou dostatečně popsány v komentářích kódu, mají konstantní velikost. Do

1 Zajímavost: tohle perverzní uvažování ve vás zanechá libovolná právnická fakulta, i když jste tam třeba jen chvíli.

struktur DNS záznamů jsem nezahrnul položku Rdata (data DNS záznamu, která následuje po struktuře záznamu RR, viz zdrojový kód). Parsování těchto dat záznamu záleží na typu záznamu. U záznamu A a AAAA se kontroluje délka dat Rdata – data záznamu, pokud vyhovuje (u Ipv4 4B, u Ipv6 16 oktetů), dochází ke speciálnímu parsování. V ostatních případech se data parsují opět funkcí, kteoru se parsuje zbytek proměnlivě dlouhých řetězců. U nepodporovaných formátů (seznam podporovaných výše) se Rdata vypíšou ve formátu HEX.

Testování a dynamické generování testů

Testovací skript je vytvořen v python3. Skript dynamicky vytváří testy z dat, která jsou zadána na začátku skriptu a pomocí knihovny itertools vytvoří všechny dostupné kombinace dat, která byla zadána.

Protože testování všech kombinací může trvat delší dobu (na druhou stranu je pak k dispozici o tom rozsáhlejší výsledek testu), skript všechna zadaná data všech druhů náhodně promíchá, promíchá i pořadí druhů zadaných dat a teprve poté generuje všechny dostupné kombinace testů. Je to proto, že když budu chtít program průběžně odlaďovat, spustím za těchto okolností skript a po provedení několika testů už můžu vesele ladit (tím že každé testování začne jinou kombinací je v případě zabugovaného programu velká šance, že se v testu vyskytne chyba hned po provedení několika jednotlivých testů). Takhle se mi podařilo najít několik chyb za pár sekund, než abych čekal půl hodiny na kompletní report. Kompletní test stojí za to provést až v případě, že se testování několikrát takto „vykliká“, zkrátila se tím průměrná čekací doba na plné výsledky testu[2].

Příklad generování (více komentářů ve zdrojovém kódu): možnost zadání argumentu -x je 0/1, argumentu -r 0/1. Pořadí hodnot argumentů náhodně zvoleno jako -x: 1,0, -r 0,1. Pozice argumentů náhodně zvolena jako -r, -x, seznam argumentů v náhodném pořadí předán do itertools.product(). Výstupní kombinace tedy budou: (-x,-r) => (0,1), (0,0), (1,1), (1,0).

Správnost výstupů zajišťují vestavěné podmínky, které určí očekávaný výsledek testu. Skript původně podporoval kontrolní spuštění programu dig a podrobnou validaci, jestli výstup dns resolveru částečně odpovídá výstupům digu, ale kvůli častým nekonzistnostem digu a resolveru byla tato část validace odstraněna (také proto, že od určité chvíle ve vývoji se nesahá do funkcí pro výpis hodnot záznamů a validace byla tím pádem zbytečná).

2 Je poněkud paradoxní, že ušetřená doba byla investována do psaní tohoto celkem složitého skriptu (na pohled moc složitý není, ale ladění chyb v něm stálo za to).

Literatura

- oficiální studijní podpora k předmětu ISA
- RFC 1034
- RFC 1035
- RFC 3596
- https://www2.cs.duke.edu/courses/fall16/compsci356/DNS/DNS-primer.pdf
- https://gist.github.com/fffaraz/9d9170b57791c28ccda9255b48315168
  (offset a převod doménového jména do dns formátu)