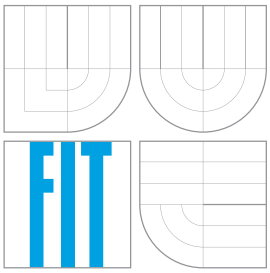
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

PROXY DNS GENERUJÚCE ŠTATISTIKY DOTAZOV

AUTOR PRÁCE BRANISLAV BLAŠKOVIČ

BRNO 2012

Obsah

Kapitola 1

Úvod

Táto dokumentácia je študentský projekt do predmetu ISA - Sieťové aplikácie a správa sietí. Popisuje program dns_stat, ktorý sa správa ako DNS proxy - preposiela prijaté DNS packety od klienta na vzdialený DNS server a odpoveď od serveru preposiela naspäť ku klientovi. Počas tohoto prevodu číta packet a ukladá si data, z ktorých neskôr tvorí štatistiky, keď obdrží signál SIGUSR1. Formát štatistík je možné ovplyvniť argumentami programu. Ako implementačný jazyk som zvolil C++ pre jednoduchšie ukladanie dát štatistík.

Kapitola 2

Popis implementácie

2.1 Základná štruktúra programu

2.1.1 Rozdelenie do tried

Program je rozdelený do celkovo 3 tried. Trieda, ktorá slúži na čítanie packetu sa volá **Packet**. Druhá trieda má názov **Statistic** a slúži na ukladanie dát do pripravených tabuliek, z ktorých sa neskôr generujú štatistiky. Treťou triedou je **DNS_Listener**, ktorá spája funkcionalitu predošlích dvoch tried a zabezpečuje jadro programu - príjem požiadavok od klienta a ich zasielanie DNS serveru. Táto trieda taktiež riadi cyklické priraďovanie zadaných DNS serverov od užívateľa pomocou parametra **-s**.

V hlavnej funkcii **main()** sa vytvorí nová instancia triedy **DNS_Listener**, ktorá obsahuje v sebe instanciu triedy **Statistic**.

2.2 Popis komunikácie cez sockety

Program čaká v nekonečnom cykle, pretože na implementáciu sú použité neblokujúce sockety. Vďaka tomu sa dajú jednoducho nastaviť časy na vypršanie vyčkávania odpovede.

Pre vytvorenie socketu a prijímanie dát sú využité funkcie **udt_init()**, **udt_send()** a **udt_receive()** z predmetu IPK. Funkcie boli mierne poupravené, aby boli začlenené do triedy **DNS_Listener**.

Príjímaný packet sa načíta do bufferu a zasiela sa na spracovanie objektu **Packet** pomocou metódy **set_packet()**. Takto pripravený objekt sa predá objektu **Statistic** metódou **add()**, ktorá je popísaná v sekcii Popis ukladania štatistík.

Packet sa ďalej preposiela DNS serveru, ktorý je zvolený argumentami **-s**. Pri čakaní na odpoveď od DNS serveru program počíta čas čakania, aby v prípade potreby ukončil čakanie a pokračoval ďalej. Ak príde odpoveď, je preposlaná naspäť ku klientovi. Adresu klienta získame zo štruktúry *structsockaddr* a funkcie **recvfrom()**, ktorú stačí predať funkcii **sendto()** pre zaslanie packetu naspäť.

2.2.1 Vypršanie časového limitu čakania na odpoveď

Program si počíta čas, ktorý čaká na odpoveď od DNS serveru. V prípade, že čakacia doba prekročí stanovený limit, čakanie ukončí. Tento limit je možné nastaviť pomocou voliteľného argumentu **-timeout**, ktorým sa dá limit nastaviť v milisekundách. Ak táto hodnota nie je nastavená, program čaká 1 sekundu.

DNS servery, na ktoré sa dlho čaká pre nás ale nie su prijateľné. Preto je možné si spustiť program s viacerími prepínačmi **-s** a tým sa bude používať viacero DNS serverov, medzi ktorými sa cyklicky rotuje. Ak požiadavka na niektorý z nich vyprší v časovom limite a my použijeme prepínač **-remove-broken**, tento server sa vyradí zo zoznamu a už sa viac nepoužije. Posledný DNS server sa zo zoznamu nevyraďuje a program na to upozorňuje aj výpisom na štandardný chybový výstup, že server síce prekročil limit čakania, ale je posledný v zozname a preto ho nemožno odstrániť, pretože by program stratil funkcionalitu.

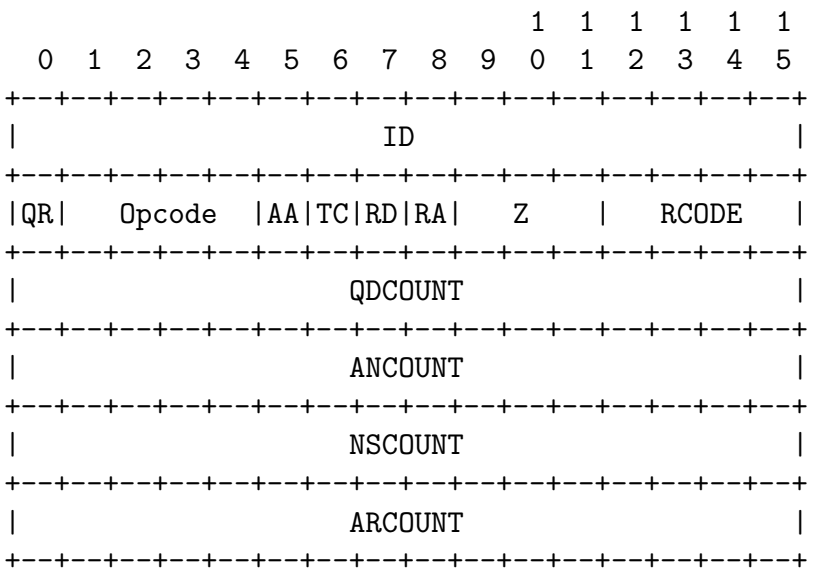
2.3 Popis čítania packetu

2.3.1 Hlavička packetu

Prijatý packet sa číta postupne po oktetoch, ako sa uvádza v kapitole 4.1.1 RFC 1035[2]. Jeho hlavička sa ukladá do datového typu **TPacket_data**, ktorý obsahuje štruktúru so šiestimi hodnotami typu **uint16_t**.

Do týchto hodnôt sa postupne uchovávajú hodnoty z hlavičky packetu. Pre použitie v našom programe ale nie sút tieto hodnoty podstatné a program si ich ukladá len pre úplnosť alebo prípadné rozšírenie.

Hlavička vyzerá nasledovne:

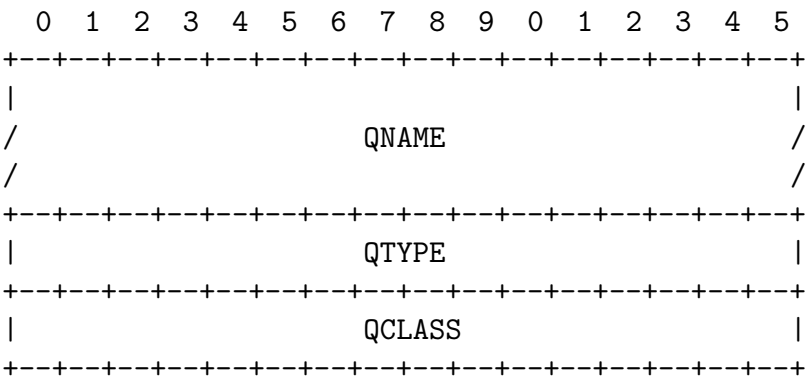


2.3.2 Doménové meno

Čítanie doménového mena prebieha v cykle. Každá časť domény je určená najprv počtom znakov, ktoré budú nasledovať. To znamená, že sa najprv prečíta číslo, na základe ktorého vieme, koľko znakov má nasledovať. Táto postupnosť je ukončená číslom 0.

Po doménovom mene nasleduje typ požiadavky, označený ako **qtype**. Tento typ nie je určený ako textový literál, ale má pridelené číslo. O prevod číselného typu do textovej podoby sa stará funkcia **get_type()**, ktorá páruje čísla s názvami podľa tabuľky [1].

Štruktúra časti packetu s doménovým menom a typom vyzerá nasledovne:



Podla RFC [2] môže byť v tejto sekcii viacero doménových mien. Počas testovania sa mi ale nepodarilo túto situáciu nasimulovať, kedže ani nástroj *dig* neposiela viacero požiadavok v jednom packete. Problematiku som konzultoval s vývojárom komponenty *bind*, ktorý taktiež nevedel o prípade, kedy by takýto packet mohol vzniknúť. Preto po načítaní typu $QTYPE$ program končí s čítaním.

2.4 Popis ukladania štatistík

Program počas svôjho behu priebežne ukladá štatistiky do predpripravených tabuliek. Na ukladanie všetkých hodnôt sa využíva datový typ **map**.

2.4.1 Pridávanie záznamov do štatistík

DNS_Listener volá funkciu **add()** nad objektom **Statistic**. V argumentoch funkcie predáva informáciu o požiadavku od klienta. Funkcia **add()** ukladá údaje do tabuliek na základe argumentov programu. Pokiaľ používateľ nevyžaduje niektoré typy štatistík, tak sa neukladaju, aby sa šetrila pamäť. Pokiaľ sa nezadá žiadny argument ovplyvňujúci štatistiku, tak sa uchováva len počet dotazov na náš server.

2.4.2 Výpis štatistík

Ked program obdrží signál **SIGUSR1** zavolá funkciu **print_log()**. Táto funkcia ďalej volá funkcie na výpis zaznamenaných štatistík.

Pre ukladanie program využíva tabuľky v tvaroch $map < string, int >$ a $map < time\_t, map < string, int >>$, podľa toho, o aký typ štatistiky sa jedná.

Týmto spôsobom vznikajú redundantné údaje pri zvolení širších štatistík. Redundantnosť by sa dala riešiť pomocou trocha komplikovanejšej tabuľky, ktorá by redundantné data nemala. Pri tomto type by bolo ale nemožné alebo veľmi ťažké meniť údaje, ktoré chceme uložiť, pretože by tabuľka stratila celistvosť. Výhodu by mala len vtedy, ak by sme požadovali všetky druhy štatistík. Preto som zvolil variantu viacerých tabuliek.

Štatistiky podľa typu, zdroja a dotazu (-type, -source, -destination)

Údaje program uchováva v tabuľke tvaru $map < string, int >$, kde indexuje typom a číslo vyjadruje počet takýchto požiadavok. O výpis sa stará funkcia **print_map()**, ktorá pomocou iterátora prechádza postupne tabuľku a vypisuje dáta.

Štatistiky za poslednú hodinu -hour

V prípade, že používateľ vyžaduje štatistiky za poslednú hodinu, môže k tomu využiť prepínač **-hour**. Pri vytváraní štatistík sa začne využívať aj čas požiadavku. Aby sme vedeli určiť, ktoré záznamy vypísať, je potrebné pri spustení programu si uložiť aktuálny čas . Potom môžeme časy požiadavkov s týmto časom porovnávať a filtrovať len tie položky, ktoré nie su staršie ako jedna hodina.

O samotný výpis na štandardný výstup sa stará funkcia **print_map_last_hour()**, ktorá ako vstupný argument dostáva mapu v tvare $map < time\_t, map < string, int >>$, v ktorej sa dá indexovať časom a zároveň aj typom, zdrojovou alebo cieľovou adresou (podľa typu štatistiky). Ako výsledok dostaneme počet požiadavok o danom čase.

Priemerné hodnoty za hodinu -average

Ak sa použije prepínač **-average**, tak sa k bežným štatistikám pridá aj štatistika s priemernými hodnotami za hodinu. K tomuto výpočtu znova využijeme čas spustenia programu, ktorý bol spomenutý v sekcii Štatistiky za poslednú hodinu -hour. Na výpočet priemeru sa používa nasledujúci vzorec:

$$average = \text{number\_of\_requests} * 3600.0/(time\_now - time\_start)$$

Zo vzorca je vidieť, že čas požiadavku už nie je potrebný. Preto je tento typ štatistík rýchlejší a pamäťovo menej náročný, ako štatistiky za poslednú hodinu, kde bolo potrebné ochovávať viac údajov.

Histogram -histogram

Pri použití prepínača $-histogram$ sa k štatistikám pridáva aj grafické znázornenie priemerného počtu požiadavok pre jednotlivé hodiny dňa. Výsledok je vypísaný na štandardný výstup ako tabuľka, ktorá na riadkoch obsahuje hodiny dňa, ktorým je priradený percentuálny podiel požiadavok v tej danej hodine a zároveň aj grafické

znázornenie pre lepšiu predstavivosť. Výsledok môže vyzerať ako napríklad na obrázku nižšie (pre ilustráciu je vyobrazená len časť histogramu).

```
|---------------------------|
| Histogram
| 12:00 | .......... | ~   3%
| 13:00 | +......... | ~  13%
| 14:00 | .......... | ~   6%
| 15:00 | +++++..... | ~  51%
| 16:00 | .......... | ~   0%
|---------------------------|
```

O výpis histogramu sa stará funkcia **print_map_histogram()**. Štatistiky pre histogram sa uchovávajú v poli čísel o veľkosti 24 prvkov (hodiny dňa). V poli sa indexuje aktuálnou hodinou a hodnota predstavuje počet požiadavok. Aby sme získali priemerny počet požiadavok vzhľadom na dni v percentách, je na výpočet použitý vzorec.

$$percent = (log[hour] * 100.0 / number\_of\_requests)$$

2.5 Obsluha signálov

Program reaguje na niekoľko signálov. Signály obsluhuje funkcia **signal_catch()**, ktorá obsahuje *switch* pre rozdielne reakcie na signál, ktorého kód obdrží ako argument.

Základným signálom je $SIGUSR1$, na ktorý program reaguje tak, že zavolá metódu **print_log()** nad objektom **Statistic**, ktorý je ako atribút objektu **DNS_Listener**.

Ku korektnému ukončeniu aplikácie príde pri obdržaní signálu $SIGINT$ alebo $SIGTERM$. Funkcia **signal_catch()** na to reaguje tak, že pomocou príkazu *delete server* vyvolá konštruktor triedy **DNS_Listener**, ktorý uvoľní pamäť, ktorá sa počas behu obsadila funkciou **malloc()** a program ukončí korektne s kódom 0.

V prípade, že by aplikácia chcela pristúpiť do pamäte, ku ktorej prístup nemá, zachytáva aj signál $SIGSEGV$ a snaží sa program korektne ukončíť s výpisom chybovej hlášky na štandardný chybový výstup a kódom 1.

Kapitola 3

Kritické časti

Program obsahuje niekoľko kritických častí, na ktoré je treba si dávať pozor a ktoré môžu byť jeho nedostatkom.

3.1 Jedno vlákno

Celá aplikácia beží v jedinom vlákne a ani sa nedelí na subprocesy. Program som vystavil záťažovým testom, keď mi bežala aj 3 dni nonstop na servery, ktorý som si nastavil ako primárny DNS server. Ani pri náročnom prechádzaní internetu som nespozoroval problémy s rýchlosťou. V prípade správne nastavenej hodnoty pre vypršanie požiadavku a dobrom zozname DNS serverov je rozdiel oproti klasickému DNS serveru skoro nebadateľný.

Udržaním programu v jednom vlákne je zdrojový kód trocha prehladnejší a jednoduchší na implementáciu aj veľkosť výsledného súboru.

3.2 Neblokujúce sockety

Pre komunikáciu som sa rozhodol pre neblokujúce sockety aj z dôvodov, ktoré sú popísané v sekcii Popis komunikácie cez sockety. V prípade využitia blokujúcich socketov by som sa vyhol zbytočne veľkému vyťažovaniu procesora. Toto vyťaženie som ale znížil pridaním funkcie **usleep()**, ktorá čaká približne 10 milisekund v každom cykle. Tým som záťaž procesora znížil z 94% na hodnoty okolo 0.5% (hodnoty mám z pozorovania priebehu počas plnej záťaže).

3.3 Chybný packet

Program počíta s tým, že mu príde korektný DNS packet. Jeho hlavičku ukladá do pripravenej štruktúry a následne pokračuje v jeho čítaní. Ak mu príde nekorektný packet, aplikácia sa môže správať nekorektne. Taktiež nepočíta s viac ako jedným doménovým menom v packete. Táto situácia je podľa RFC platná, ale z dôvodov uvedených v sekcii Doménové meno som sa ju rozhodol ignorovať.

Kapitola 4

Záver

Tento projekt mi dal veľmi veľa nových znalostí z oblasti komunikácie cez sockety a taktiež náhľad do fungovania DNS serverov. Čítanie binárnych dát bol pre mňa taktiež novinkov. Program by šiel implementovať aj bez tried ale som rád, že som sa rozhodol ich využiť, pretože som sa tým naučil používať aspoň základy objektovo orientovaného programovania v jazyku C++.

Na projekte som pracoval s radosťou a mám s ním aj ďalšie plány po odovzdaní - rád by som pridal logovanie do súbora a načítanie už existujúcich štatistík, aby sa dalo v ich zaznamenávaní pokračovať v prípade pádu aplikácie (odpojenie elektrického prúdu alebo iné zapríčinenie).

Literatura

[1] List of DNS record types [online].
http://en.wikipedia.org/wiki/List_of_DNS_record_types.

[2] Mockapetris, P.: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION [online]. http://www.ietf.org/rfc/rfc1035.txt.