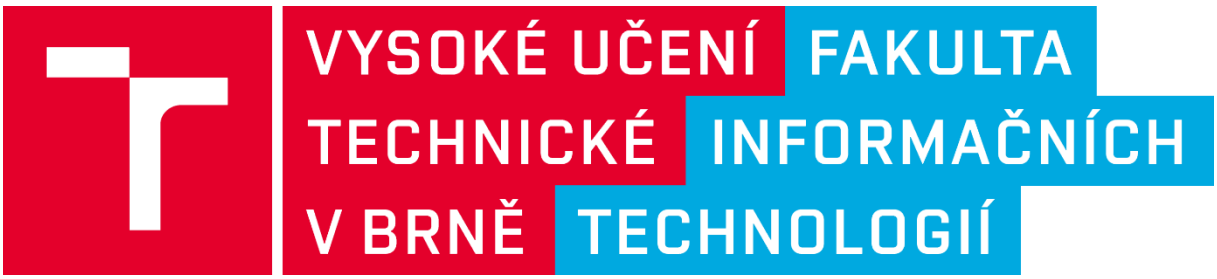
Vysoké učení technické v Brně

Fakulta informačních technologií



Síťové aplikace a správa sítí

2020/2021

Dokumentace projektu

Filtrující DNS resolver

Daniel Pátek (xpatek08)

Brno, 2. listopadu 2020

1. Obsah

1 Úvod

Cílem řešeného projektu je návrh a implementace filtrujícího DNS resolveru. Tento program bude schopný na určeném síťovém portu zachytávat provoz a filtrovat pakety protokolu DNS. Součástí projektu je vytvoření dokumentace.

Program bude podporovat pouze DNS dotazy typu A a bude fungovat na protokolu UDP. Nebude podporovat DNSSEC. Kromě portu, na kterém bude program přijímat dotazy, bude možné specifikovat, na jaký resolver se mají vyfiltrované dotazy odeslat. Dále je pro spuštění programu zapotřebí dodat soubor s filtrovanými doménami. Tento soubor musí být textového typu a musí být lokálně uložen.

Při běhu program nic nevypisuje, pro výpis paketů a běhu programu je nutné program spustit s parametrem `-v`.

2 Uvedení do problematiky

Pro implementaci programu, který se má chovat jako DNS server, je zapotřebí nejprve nastudovat problematiku tohoto protokolu a způsoby, jakým je možné takovýto program vytvořit.

Osobně jsem začal podrobným nastudováním normy tohoto protokolu[1]. Zde jsem se dozvěděl, jak vypadá DNS paket a z čeho se skládá. Jeho hlavička má velikost 8 bajtů a mimo jiné udává, zda se jedná o dotaz nebo odpověď. Také obsahuje tzv. response code, který udává typ odpovědi.

Následně jsem nastudoval, jakým způsobem se implementují síťové aplikace, využívající tzv. sockety[2]. Základem takovéto aplikace je otevřít socket pro příchozí spojení. Následně je vhodné vytvořit nekonečný cyklus pro příjem zpráv a jejich zpracování. To odpovídá chování serveru. V mém projetu využiji sockety na bázi UDP připojení.

3 Implementace

3.1 Základní údaje

Pro psaní programu jsem zvolil programovací jazyk C. Mezi nejdůležitější použité knihovny patří `sys/socket.h[3]` pro práci se sokety a `arpa/nameser.h[4]` pro použití struktury DNS hlavičky. Obě knihovny jsou součástí standardu POSIX a fungují na všech Unix systémech.
Samotný kód se nachází v jednom souboru `dns.c`. Celý program je rozložen na několik hlavních částí, které jsou popsány níže.

3.2 Funkce `main()`

Hlavní část programu, funkce `main()`, má za úkol zpracování argumentů programu, otevření socketů pro příchozí a odchozí spojení a následné spuštění serveru. S pomocí funkce signal() nastavuje, jak má program správně skončit v případě přerušení systémem nebo uživatelem.
Pro zpracování argumentů využívám funkci `getopt()[5]`. Součástí tohoto zpracování argumentů je i funkce `printHelp()`, která má za úkol vytisknout informace o používání programu.
O otevření socketů se starají funkce `socket()` a `bind()[7]`. Následně, s použitím `poll()[8]`, spustím nekonečný cyklus reprezentující server. Tuto funkci používám, abych dokázal zachytit všechna spojení.

```
//first have to check both sockets for connection
if (poll(fds, 2, -1) == -1) {
    fprintf(stderr, "Unable to poll descriptors. Poll: %s\n", strerror(errno));
    clear();
}
```

Obrázek 1 - Použití funkce poll() v nekonečném cyklu

Dál v tomto cyklu se podle typu paketu rozhodne, zda se jedná o dotaz či odpověď, a spustí se samotná kontrola paketu.
Pokud se jedná o dotaz a program nezjistí žádnou chybu, uloží se adresa a port odesílatele a dotaz se přepošle na skutečný DNS server pomocí funkce `sendto()`.

Když program objeví v dotazu chybu, přetvoří tento paket na chybovou odpověď, kterou odešle tazateli. Pokud se jedná o odpověď, zkontroluje se a pošle správnému tazateli zpět.

3.3 Funkce `getDnsRequestData()`

Tato funkce se stará o získání jména tázaného serveru, typu a třídy z obdrženého DNS paketu. Je zajímavá především algoritmem, jenž projde část paketu, kde se nachází jméno serveru. Podle standardu je jméno rozděleno do částí podle toho, kde se v něm nachází tečky. Na tomto místě je číslo, které udává, kolik znaků se nachází v následující části. Celé jméno je zakončeno tečkou.

```
//first we have to skip header
char *ptr = buffer + sizeof(HEADER);
int sum = ptr[0];
int offset = 0, counter = 1, counterDst = 0;
// get name from the dns packet (according to RFC 1035)
while(sum) {
    if(offset) url[counterDst++] = '.';
    while (counter <= offset + sum) {
        url[counterDst++] = ptr[counter++];
    }
    sum = ptr[counter];
    offset = counter++;
}
url[counterDst] = '\0';
```

Obrázek 2 - algoritmus procházení jména v DNS paketu

3.4 Funkce `isBlacklisted()`

Tato funkce je velmi jednoduchá a efektivní. Má za úkol zjistit, zda je dotazovaná doména v seznamu filtrovaných domén. Jako parametr přijme jméno domény a následně s pomocí cyklu `for()` začne procházet celý blacklist, dokud na tuto doménu nenarazí. Důležité je, že funkce zachytí i poddomény těch zakázaných. To je umožněno pomocí knihovní funkce `strstr()`, která najde výskyt jednoho řetězce v druhém. Takto například odfiltruji dotaz na `fit.vutbr.cz` i v případě, že mám ve filtrovaných doménách pouze `vutbr.cz`.

```c
int isBlacklisted(char *name) {
    //search the list
    for(unsigned long i = 0; i < blacklist->size; i++) {
        if (strstr(name, blacklist->r[i])) {
            // found a match
            return 1;
        }
    }
    return 0;
}
```

Obrázek 3 - Funkce isBlacklisted()

3.5 Funkce `getDnsFilter()`

Důležitou součástí programu je seznam filtrovaných domén. Tato funkce načte soubor a domény v něm uloží do proměnné v programu. Je důležité zmínit, že funkce bude načítat do té doby, než mu fukce `realloc()[9]` nepovolí rozšířit paměť. Také je důležité, že existuje limit na délku jednoho doménového serveru, konkrétně se jedná o 512 bajtů.
Samotná funkce spravuje dynamické pole v rámci struktury `blacklist` a načítá do něj postupně všechny řádky (doménová jména) ze zadaného souboru. Samozřejmě ignoruje řádky, které začínají znakem „#“.

3.6 Funkce `clear()`

Poslední z nejdůležitějších součástí programu je funkce `clear()`. Jedná se o pomocnou funkci, která se spustí, kdykoliv je program ukončen. Její spuštění i při „násilném ukončení“ (`sigterm`) zajišťuje již zmíněná funkce `signal()`. Tato funkce hlavně uvolňuje alokovanou paměť a zavírá sockety.

4 Použití programu

Program se spouští v příkazové řádce.
Nejprve je zapotřebí program přeložit. Toho docílíme příkazem `make`.
Následně můžeme program spustit. Buď pomocí příkazu `make run`, kdy se program spustí s předdefinovanými parametry (`./dns -p 5300 -s 8.8.8.8 -f tests/big_filter`), nebo pomocí `./dns` a vlastních parametrů.
Parametry jsou zejména `-s` (server ve tvaru doménového jména nebo Ipv4, na který se budou odesílat dotazy), `-f` (soubor se seznamem filtrovaných domén) a volitelný parametr `-p` (číslo portu, na kterém má server poslouchat; výchozí hodnota je 53). Volitelně můžeme použít ještě `-v` pro výpis informací o běhu programu a `-h` pro výpis pomocného textu.
Testování programu lze spustit příkazem `make test`. Pro odstranění dříve přeloženého programu můžeme použít `make clean`.

5 Příklad výstupu programu (parametr -v)

```
xpatek08@merlin: ~/ISA/ISA-project-2020$ ./dns -v -s 8.8.8.8 -f big_filter -p 5300
[-v] Verbose mode turned on.
[-s] Server ip selection: 8.8.8.8
[-f] Filter file name selection: big_filter
[-p] Port selection: 5300
127.0.0.1#45563         -->     8.8.8.8#53              query:  google.com
127.0.0.1#45563         <--     8.8.8.8#53              answer: google.com
127.0.0.1#49324         -->     8.8.8.8#53              query:  seznam.cz
127.0.0.1#49324         <--     8.8.8.8#53              answer: seznam.cz
127.0.0.1#35816         -->     127.0.0.1#35816         blacklisted:    cj.com
127.0.0.1#40223         -->     8.8.8.8#53              query:  vutbr.com
127.0.0.1#40223         <--     8.8.8.8#53              answer: vutbr.com
127.0.0.1#49362         -->     127.0.0.1#49362         blacklisted:    moat.com
127.0.0.1#56297         -->     8.8.8.8#53              query:  vutbr.cz
127.0.0.1#56297         <--     8.8.8.8#53              answer: vutbr.cz
127.0.0.1#46036         -->     127.0.0.1#46036         blacklisted:    cj.com
127.0.0.1#50110         -->     127.0.0.1#50110         blacklisted:    cj.com
```

Obrázek 4 - Výstup programu při použití parametru -v

6 Chybové výstupy programu

Chybové výstupy programu lze rozdělit do dvou skupin.
První skupinou jsou chybové hlášky vypisované do výstupu `stderr`. Zde patří zejména chyba při otevírání souboru, chyba při alokaci paměti, chyba při špatně zadaném jménu serveru nebo špatně zadaného jiného parametru. Dále se jedná o chyby při vytváření socketu nebo při práci s pakety (chyba odesílání nebo příjmu). Chyba zde bude vypsána i v případě příjmu DNS dotazu špatného nebo nepodporovaného formátu.
Druhou skupinou jsou odesílané pakety s některým z chybových návratových kódů. Může se jednat o 1 při chybném formátu paketu, `4` při dotazu na neimplementovanou funkcionalitu a `5` v případě zablokování překladu pomocí filtru.

7 Testování programu

Testování probíhalo dvěma způsoby.

První způsob spočíval v testování při implementaci, kdy jsem chování a výstup svého programu porovnával s jiným programem, konkrétně s programem `wireshark[6]`. Pozoroval jsem, odkud a kam jednotlivé DNS dotazy putují a jakým způsobem na ně reaguje můj program. Také jsem sledoval jejich obsah, zejména typ dotazu a kód odpovědi.

Další testování jsem již prováděl po napsaní testovacího skriptu `tests.sh`. Tento skript spustí DNS server na portu 5300 a následně pomocí příkazu `dig` testuje, zda server funguje správně. Tento typ testování využívá seznam vyloučených domén v souboru `tests/filter`, seznam správných domén v souboru `tests/good_domains` a seznam domén, jež mají být testovány jako filtrované, `tests/bad_domains`. Skript tyto domény jednu po druhé projde a otestuje, zda chování programu odpovídá tomu, co je od něj vyžadováno.
Skript je možné spustit pomocí příkazu `make test`.

```
xpatek08@merlin: ~/ISA/ISA-project-2020$ make test
gcc -Wall -Wextra -Werror -g -o dns dns.c
sh tests/tests.sh
RUNNING SERVER "./dns -s 8.8.8.8 -f tests/filter -p 5300"

________________________________________________________________________
                      TESTING GOOD DOMAINS
________________________________________________________________________

RUNNING "dig -p 5300 @127.0.0.1 +short google.com"
TEST PASSED - received 216.58.201.110
RUNNING "dig -p 5300 @127.0.0.1 +short seznam.cz"
TEST PASSED - received 77.75.75.176
RUNNING "dig -p 5300 @127.0.0.1 +short guta.fit.vutbr.cz"
TEST PASSED - received 147.229.9.11
RUNNING "dig -p 5300 @127.0.0.1 +short dns.google.com"
TEST PASSED - received 8.8.4.4
RUNNING "dig -p 5300 @127.0.0.1 +short centrum.cz"
TEST PASSED - received 46.255.231.106

________________________________________________________________________
                      TESTING BAD DOMAINS
________________________________________________________________________

RUNNING "dig -p 5300 @127.0.0.1 +short adblade.org"
TEST PASSED - no address received
RUNNING "dig -p 5300 @127.0.0.1 +short adblockanalytics.com"
TEST PASSED - no address received
RUNNING "dig -p 5300 @127.0.0.1 +short adbooth.net"
TEST PASSED - no address received
RUNNING "dig -p 5300 @127.0.0.1 +short adbot.com"
TEST PASSED - no address received
RUNNING "dig -p 5300 @127.0.0.1 +short adbrite.com"
TEST PASSED - no address received
```

Obrázek 5 - příklad výstupu testování programu

8 Závěr

Projekt byl pro mě velmi přínosný. Naučil jsem se, jakým způsobem je možné programovat síťové aplikace. Také jsem se dozvěděl, jakým způsobem funguje protokol DNS a vím, jak se s ním pracuje.
Své znalosti jsem si rozšířil také v problematice jiných síťových protokolů, zejména UDP.
Zjistil jsem, co všechno obsahuje hlavička DNS paketu a jakým způsobem se z ní dá strojově vyčíst doménové jméno.
Dokázal jsem dodržet všechny podmínky ze zadání projektu. Program je přeložitelný, funkční a otestovaný. Podporuje komunikaci prostřednictvím protokolu Ipv4 a UDP a DNS dotazy typu A.

Před termínem odevzdání se mezi studenty začala objevovat informace, že tento program musí podporovat komunikaci pomocí protokolu Ipv6. Vzhledem k tomu, že jsem tuto informaci z kontextu zadání neobdržel, můj program pracuje pouze na protokolu Ipv4. Po pravdě by mi ani nedávalo smysl implementovat komunikaci na protokolu Ipv6 a zároveň podporovat pouze dotazy typu A.

Projekt se mi líbil a jeho implementace mě bavila. Bylo poměrně složité nastudovat veškerou teorii, než jsem se dostal k psaní kódu. Samotná implementace mi ale šla velmi dobře.

9 Zdroje

[1] Domain Implementation and Specification [online]. 1987 [cit. 2020-11-14]. Dostupné z: https://tools.ietf.org/html/rfc1035

[2] Socket Programming in C/C++ [online]. 2019 [cit. 2020-11-14]. Dostupné z: https://www.geeksforgeeks.org/socket-programming-cc/

[3] Sys/socket.h - Internet Protocol family [online]. 1997 [cit. 2020-11-15]. Dostupné z: https://pubs.opengroup.org/onlinepubs/007908799/xns/syssocket.h.html

[4] Include/arpa/nameser.h [online]. 1993 [cit. 2020-11-15]. Dostupné z: https://code.woboq.org/gcc/include/arpa/nameser.h.html

[5] Getopt(3) — Linux manual page [online]. [cit. 2020-11-15]. Dostupné z: https://man7.org/linux/man-pages/man3/getopt.3.html

[6] Wireshark [online]. 2020 [cit. 2020-11-15]. Dostupné z: https://www.wireshark.org/

[7] Bind(2) — Linux manual page [online]. [cit. 2020-11-15]. Dostupné z: https://man7.org/linux/man-pages/man2/bind.2.html

[8] Poll(2) — Linux manual page [online]. [cit. 2020-11-15]. Dostupné z: https://man7.org/linux/man-pages/man2/poll.2.html

[9] C library function - realloc() [online]. 2020 [cit. 2020-11-15]. Dostupné z: https://www.tutorialspoint.com/c_standard_library/c_function_realloc.htm