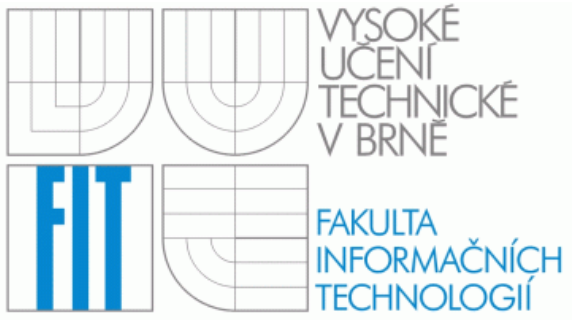


Síťové aplikace a správa sítí

Seminární práce – zadání č. 6

Komunikace v systému DNS

Jan Kupčík
(xkupci00)

14. listopadu 2004

Obsah

1. Úvod

Všechny aplikace, které zajišťují komunikaci mezi počítači používají k identifikaci komunikujících uzlů IP adresu. Pro člověka jako uživatele jsou však IP adresy těžko zapamatovatelné, a proto byl vymyšlen systém, který umožňuje pro IP adresu zavést jméno síťového rozhraní (počítače), přesněji řečeno doménové jméno. Toto doménové jméno lze pak používat téměř všude, kde je možno použít IP adresu. Výjimkou, kdy se musí používat IP adresa, je identifikace samotného jmenného serveru.

Vazba mezi doménovým jménem a IP adresou je definována v DNS databázi. DNS (domain name system) je celosvětově distribuovaná databáze, jednotlivé části této databáze jsou umístěny na tzv. jmenných serverech. Informace jsou zde uloženy v textových souborech ve tvaru zdrojových vět (resource records – RR vět).

Jestliže tedy nějaká aplikace (na aplikační resp. prezentační úrovni) potřebuje komunikovat s aplikací na vzdáleném počítači a od uživatele obdrží doménové jméno této vzdálené stanice, je nutné zjistit její IP adresu. K tomu obvykle bývá určena komponenta systému zvaná resolver. Nejedná se o konkrétní program, nýbrž jde o s soustavu knihovních funkcí, které se sestavují s aplikačními programy požadujícími tyto služby (např. ftp, WWW prohlížeč, …).

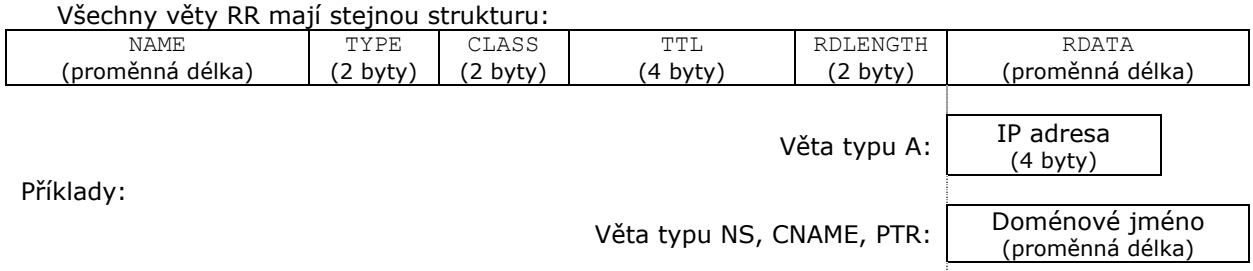
Následující text se zabývá protokolem DNS, který je použit při přenosu RR vět klientovi a knihovními funkcemi v operačním systému FreeBSD umožňujícími klást dotazy DNS serverům a dekódovat obsah odpovědí.

2. Věty RR

Informace o doménových jménech a jim příslušejícím IP adresách, stejně tak jako všechny ostatní informace distribuované pomocí DNS, jsou uloženy v paměti DNS serverů ve tvaru RR vět. Jmenný server naplňuje svou paměť několika způsoby. Autoritativní data načte ze souborů na disku (o tato data se stará administrátor), nebo je získá pomocí požadavku Přenos zóny z paměti jiného serveru. Neautoritativní data získává postupně z paměti jiných serverů v průběhu vyřizování jednotlivých dotazů.

V případě, že DNS klient (resolver) potřebuje získat informace, pak požaduje po jmenném serveru příslušné věty RR. Klient může např. chtít po serveru věty typu A, které obsahují IP adresy pro dané doménové jméno.

Všechny věty RR mají stejnou strukturu:



Pole `NAME` obsahuje doménové jméno. `TYPE` a `CLASS` uchovávají typ a třídu věty (neboli typ sítě), jejich možné hodnoty jsou uvedeny níže v tabulkách. `TTL` je údaj v sekundách, určuje dobu platnosti RR věty, tedy jak dlouho může být odpověď udržována v cache jako platná. `RDLENGTH` obsahuje délku části `RDATA`. V poslední části `RDATA` je uložena pravá strana zdrojové věty.

Typy RR vět (bez experimentálních a zastaralých), hodnoty jsou vyjádřeny desítkově:

Označení věty (anglický název) **Hodnota** **Popis dat v poli `RDATA`**
A (A host address) 1 32-bitová IP adresa odpovídající doménovému jménu.
CNAME (Cannonical name for an alias) 5 Doménové jméno specifikující synonymum k jménu uvedeném v poli `NAME`.
HINFO (Host information) 13 2 pole: `CPU` – řetězec identifikující typ procesoru, `OS` – řetězec identifikující operační systém na počítači `NAME`.
KEY (Security key) 25 Veřejný klíč zóny používaný pro podepisování při autorizaci.
MX (Mail exchange) 15 2 pole: `preference` – 16-bitové číslo bez znaménka; čím nižší číslo, tím vyšší počet mail požadavků je směrováno na tento server.

`exchange` – doménové jméno mailového serveru.
NS (Authoritative name server) 2 Doménové jméno autoritativního jmenného serveru dané domény.
PTR (Domain name pointer) 12 Doménové jméno, které identifikuje stejného hosta jako záznam v poli `NAME`. Používá se např. u reverzních překladů.
SIG (Security signature) 24 Podpisová věta používaná při autentizaci v zabezpečených DNS.
SOA (Start Of Authority) 6 Určení jmenného serveru, který je autoritativním zdrojem informací pro danou doménu. Věta SOA je vždy právě jedna. Obsahuje 7 polí: 1. jméno počítače, na kterém jsou data uložena (jméno primárního jmenného serveru) 2. poštovní adresa osoby zodpovědné za data (první tečka je ve skutečnosti @) 3. `Serial` – 32-bitové číslo verze databázového souboru 4. `Refresh` – jak často mají sekundární servery ověřovat svá data 5. `Retry` – jestliže sekundární server nemůže kontaktovat primární po uplynutí intervalu `Refresh`, bude to dále zkoušet každých `retry` sekund 6. `Expire` – jestliže se sekundárnímu jmennému serveru nepodaří kontaktovat primární do `expire` sekund, přestane poskytovat informace. Platí `expire` > `refresh` 7. `TTL` – hodnota se vztahuje ke všem záznamům v DB souboru (jako výchozí hodnota) a je poskytována jmenným serverem v každé jeho odpovědi. Určuje, jak dlouho mohou neautoritativní servery udržovat daný záznam ve své paměti cache (0 neumožňuje ukládání do cache) Pole 4-7 jsou 32 bitové-hodnoty vyjadřující dobu v sekundách.
TXT (Text string) 16 Textový řetězec s popisem.
WKS (Well known service description) 11 Popis obvyklých služeb serveru v protokolech TCP a UDP. Obsahuje 3 pole: 1. `Address` – 32-bitová IP adresa. 2. `Protocol` – 8-bitová hodnota portu. 3. `Bit map` – bitové pole proměnné délky zarovnané na 8 bitů.
AAAA (IP6 address) 28 IP6 adresa.

Třídy RR vět (bez experimentálních a zastaralých), hodnoty jsou vyjádřeny desítkově:

Označení	Hodnota	Význam
IN	1	Internet
CH	3	Chaos
HS	4	Hesiod

3. Protokol DNS

Doménová služba je realizována jednoduchým protokolem, který pracuje způsobem dotaz – odpověď. Klientem zde může být komponenta systému – resolver, který posílá DNS serveru dotazy.

Protokol DNS spadá do aplikační vrstvy, přenos svých paketů svěřuje transportním protokolům UDP a TCP. Dotaz i odpověď jsou přenášeny vždy stejným transportním protokolem.

U dotazů na překlad (tj. žádost o RR větu) je dávána přednost protokolu UDP. V případě, že je odpověď DNS serveru delší než 512 B, vloží se do odpovědi pouze část informací nepřesahující tuto velikost a v záhlaví se nastaví bit TC specifikující, že se jedná o neúplnou odpověď. Klient si v tomto případě může vyžádat kompletní odpověď protokolem TCP. U přenosu dat např. mezi primárním a sekundárním jmenným serverem se používá protokol TCP. Komunikace probíhají na portu 53/UDP a 53/TCP.

3.1 Operace protokolu DNS

DNS protokol umožňuje přenášet několik druhů operací.

Operací, kterou využívá resolver, je DNS QUERY. Jedná se o získání RR věty. Touto operací se bude zabývat zbývající text.

S novými rozšířeními protokolu DNS přibývají i další typy: DNS NOTIFY (viz. RFC 1996 – informování slave serverů o změně dat v zóně) a DNS UPDATE (viz. RFC 2136 – dynamické opravy záznamů v DNS databázi).

3.2 Formát paketu DNS QUERY

DNS používá stejný formát paketu pro dotaz i odpověď. Paket se může skládat až z pěti sekcí, avšak vždy musí obsahovat aspoň sekci záhlaví.

HEADER (záhlaví)
QUESTION (dotazy)
ANSWER (odpovědi)
AUTHORITY (autoritativní jmenné servery)
ADDITIONAL (doplňující informace)

3.2.1 Sekce záhlaví

Záhlaví paketu je povinné, je obsaženo v dotazu i odpovědi. Obsahuje položky specifikující, zda se jedná o otázku či odpověď, počty vět v jednotlivých sekcích, atd.

Struktura:

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	bit
ID																
QR	OPCODE				AA	TC	RD	RA	Z			RCODE				
QDCOUNT																
ANCOUNT																
NSCOUNT																
ARCOUNT																

První dva byty záhlaví (ID) obsahují identifikátor zprávy. Tento identifikátor generuje klient a server ji kopíruje do své odpovědi. Používá se k párování dotazu a odpovědi, klient tedy může posílat více dotazů současně, aniž by musel čekat na odpověď.

Význam dalších dvou bytů, kde jsou uloženy řídící bity, popisuje tabulka:

Pole	**Počet bitů**	**Hodnota**
Query / Response	1	0 – zpráva je dotazem, 1 – odpověď
OPCODE	4	Typ požadavku: 0 – standardní otázka (QUERY) 1 – inverzní otázka (IQUERY) 2 – otázka na status (STATUS) 4 – požadavek DNS NOTIFY 5 – požadavek DNS UPDATE
Authoritative Answer	1	0 – odpověď není autoritativní 1 – odpověď je autoritativní
TrunCation	1	1 – odpověď byla zkrácena na 512 bytů. Pokud má klient zájem o celou odpověď, pak musí dotaz zopakovat pomocí protokolu TCP.
Recursion Desired	1	Bit může být nastaven v dotazu a je zkopírován do odpovědi. Je-li jeho hodnota 1, je požadován rekurzivní překlad.
Recursion Available	1	1 – server umožňuje rekurzivní překlad
Z	3	Rezervováno pro budoucí použití, musí být vždy 0.
RCODE	4	Kód odpovědi: 0 – bez chyby 1 – chyba ve formátu dotazu, server jej neumí interpretovat 2 – server neumí odpovědět 3 – jméno z dotazu neexistuje, tuto odpověď mohou vydat pouze autoritativní jmenné servery 4 – server nepodporuje tento typ dotazu 5 – server odmítá odpovědět, např. z bezpečnostních důvodů

Poslední čtyři 16-bitová čísla bez znaménka obsahují počet vět v daných sekcích:

- QDCOUNT – počet vět v sekci dotaz
- ANCOUNT – počet vět v sekci odpověď
- NSCOUNT – počet vět s odkazy na autoritativní jmenné servery
- ARCOUNT – počet vět v sekci doplňující informace

3.2.2 Sekce dotaz

Při zasílání dotazu obsahuje paket pouze hlavičku a sekci s dotazem. V odpovědi je paket doplněn o příslušné sekce, hlavička upravena a otázka zkopírována.

Sekce dotazu obsahuje tři pole:

QNAME (proměnná délka)	QTYPE (2 byty)	QCLASS (2 byty)

Pole QNAME obsahuje doménové jméno.

Protokol DNS nepoužívá pro vyjádření doménových jmen tečkovou notaci. Každá část jména je uvozena bytem obsahujícím délku řetězce, tečky jsou vypuštěny. Na konci je byte s hodnotou 0. Příklad uložení doménového jména www.vutbr.cz:

binární data v paketu [hex]:	03	77	77	77	05	76	75	74	62	72	02	63	7A	00
příslušný ASCII znak:		w	w	w		v	u	t	b	r		c	z	

QTYPE specifikuje typ dotazu, tj. požadovaný typ věty v odpovědi. QCLASS určuje třídu dotazu. Hodnoty těchto polí jsou uvedeny v přehledu RR vět (kapitola 2). Do obou těchto polí lze zapsat hodnotu 255 označující, že jsou požadovány všechny typy vět resp. věty všech tříd.

3.2.3 Sekce odpověď, autoritativní servery a doplňující informace

Pakety odpovědi obsahují obvykle vedle záhlaví a zopakované sekce dotazu ještě tři sekce. V sekci odpovědi je uložena RR věta, která byla dotazem vyžádána, sekce autoritativní jmenné servery obsahuje doménová jména jmenných serverů uvedených ve větách NS. Poslední sekcí jsou doplňkové údaje, kde jsou obvykle uloženy IP adresy autoritativních jmenných serverů.

Věty v těchto sekcích jsou běžné RR věty. Jejich struktura je uvedena v kapitole 2. Všechna doménová jména jsou ve zvláštním tvaru charakteristickém pro DNS paket, viz. kapitola 3.2.2 – popis pole QNAME. Dále se zde může uplatňovat komprese, která je vysvětlena níže.

3.3 Komprese v DNS paketu

Komprese slouží k redukci velikosti DNS paketu. V něm se může stejné doménové jméno nebo jeho část vyskytovat opakovaně. Komprese spočívá v tom, že je toto opakující se jméno uvedeno pouze jednou a každý další výskyt tohoto jména je nahrazen ukazatelem na první výskyt.

Klasický zápis doménového jména je v tečkové notaci: řetězec.řetězec....řetězec. Celé jméno může být dlouhé maximálně 255 znaků, řetězec pak maximálně 63 znaků. V DNS paketu je použit jiný způsob zápisu doménového jména – před každým řetězcem je byte s hodnotou délky tohoto řetězce. Díky zvolené hranici 63 znaků (63_{10} = 0011111111_2) lze snadno identifikovat, co byte, který následuje za řetězcem, představuje:

- 00000000_2 označuje konec doménového jména
- $00xxxxxx_2$ (a číslo je větší než 0) udává délku následujícího řetězce
- $11xxxxxx_2$ znamená odkaz na předcházející výskyt zbývající části jména
- kombinace $01xxxxxx_2$ a $10xxxxxx_2$ jsou určeny pro budoucí využití

Ukazatel je dlouhý 16 bitů. V prvních dvou nejvyšších bitech obsahuje jedničku, v dalších bitech pak pořadové číslo byte od začátku DNS paketu. Pole ID v hlavičce je tedy na offsetu 0, slovo s řídícími bity má offset 2, atd.

4. Implementace v operačním systému FreeBSD

Jestliže aplikace pod operačním systémem UNIX potřebuje klást dotazy DNS serverům, může využívat knihovní funkce resolveru, které jsou zde implementovány. Je nutné připojit tyto hlavičkové soubory:

- sys/types.h – datové typy používané ve funkcích resolveru
- netinet/in.h – musí být přilinkován pro bezchybný překlad souboru resolv.h
- arpa/nameser.h – konstanty ns_c_xx, ns_t_xx, datové struktury a funkce ns_xx
- resolv.h – konstanty RES_xx a funkce res_xx
- netdb.h – proměnná h_errno, konstanty HOST_NOT_FOUND, TRY_AGAIN, …

4.1 Funkce pro vytváření a zasílání požadavků

`int res_search(const char *dname, int class, int type, u_char *answer, int anslen)`

Funkce `res_search` spadá do nejvyšší úrovně, volá ji např. `gethostbyname()`. Tato funkce implementuje vytváření a předávání dotazů DNS serveru prostřednictvím volání `res_query()`.

Parametry:

- `dname` – doménové jméno, jehož záznam je požadován. Pokud není název plně kvalifikován, postupně se programově doplňuje dle informací v souboru uvedeném v systémové proměnné `HOSTALIASES`. V takovém případě se volá `res_query()` tak dlouho, dokud není nalezeno platné doménové jméno nebo nejsou vyčerpány všechny možnosti.
- `class` – označení typu sítě (třídy vět), ve které se bude vyhledávat. Konstanty pro toto pole jsou definovány v hlavičkovém souboru `arpa/nameser.h`, standardní hodnota je `ns_c_in`, která označuje Internet.
- `type` – specifikace požadavku (resp. který typ RR věty je očekáván). Konstanty se opět nacházejí v `arpa/nameser.h`. Příklad: pro zjištění věty typu MX je konstanta `ns_t_mx`.
- `answer` – ukazatel na předem alokovaný buffer pro umístění odpovědi. Jeho velikost by měla být minimálně `PACKETSZ` bytů. (definice v `arpa/nameser.h`).
- `anslen` – alokovaná velikost bufferu.

Návratová hodnota je typu `int`, udává velikost odpovědi zkopírované do bufferu. V případě chyby je vrácena -1 a nastavena proměnná `h_errno`.

`int res_query(const char *dname, int class, int type, u_char *answer, int anslen)`

Funkce `res_query` implementuje dotazování na RR záznamy. Pomocí `res_mkquery()` vytvoří požadavek pro DNS server v binární podobě, zašle jej voláním `res_send()` a vyhodnotí, zda přišla odpověď.

Protože funkce `res_search()` a `res_query()` mají stejné parametry, je jejich vysvětlení uvedeno výše. Jediný rozdíl je v tom, že tato funkce nedoplňuje doménová jména, dotazuje se na přesný řetězec uvedený v proměnné `dname`.

Tato funkce vrací stejně jako v předchozím případě velikost příchozí odpovědi, v případě chyby -1.

Poznámka: před voláním této funkce by měla být nastavena proměnná `errno` na hodnotu 0. Při volání funkce `res_search()` toto neplatí, protože ta nuluje proměnnou automaticky.

```
int res_mkquery(int op, const char *dname, int class, int type, const u_char *data,
                int datalen, const u_char *newrr, u_char *buf, int buflen)
```

Funkce `res_mkquery` vytváří binární podobu požadavku pro DNS server; tedy naplní hlavičku, upraví doménová jména do podoby používané v DNS paketu (viz. 3.2.2), pokud je to možné, použije se komprese paketu a doplní zbývající položky v sekci dotazu.

Parametry:

- `op` – konstanta udávající typ požadavku – nejčastěji to bývá konstanta `ns_o_query`, zbývající jsou uvedeny v `arpa/netinet.h`.
- `dname`, `class`, `type` – viz. `res_query()`
- `data` – buffer s daty pro inverzní dotazy. V případě `ns_o_query` se předává NULL.
- `datalen` – velikost bufferu v parametru `data`. Jestliže `data` je NULL, předává se 0.
- `newrr` – ukazatel na buffer určený pro dynamickou aktualizaci dat mezi DNS servery. Pokud požadavek není právě aktualizace záznamů, předává se NULL.
- `buf` – ukazatel na buffer, do něhož se zkopíruje vytvořený paket. Buffer by měl být alokován na minimální velikost `PACKETSZ`.
- `buflen` – velikost bufferu předávaného v parametru `buf`.

Jestliže dojde při vykonávání této funkce k chybě, vrátí -1, jinak je vrácena velikost vytvořeného paketu.

`int res_send(const u_char *msg, int msglen, u_char *answer, int anslen)`

Funkce `res_send` zašle požadavek prostřednictvím paketu UDP nebo TCP (záleží na nastavení ve struktuře `_res`).

Parametry:

- `msg` – ukazatel na buffer obsahující požadavek v binární podobě
- `msglen` – velikost bufferu msg
- `answer` – ukazatel na buffer, do kterého se zapíše odpověď
- `anslen` – velikost bufferu `answer`

Návratová hodnota udává skutečnou velikost odpovědi, v případě chyby hodnotu -1. Hodnota -1 a proměnná `errno` nastavena na `ECONNREFUSED` označuje, že jmenný server nebyl nalezen.

`int res_init(void)`

Funkce `res_init` načte konfiguraci ze souboru `resolv.conf` a inicializuje strukturu `_res`. Jestliže se po volání některé z předchozích funkcí zjistí, že doposud nebyla načtena konfigurace, je nejprve spuštěna `res_init()`. Samozřejmě je možné tuto funkci volat explicitně, např. v části inicializace aplikace.

Návratová hodnota je vždy 0, pokud v průběhu načítání konfigurace dojde k chybě, je ignorována.

`extern int h_errno`

Tato proměnná uchovává kód chyby poslední operace. Může nabývat těchto hodnot:

- `HOST_NOT_FOUND` – požadované doménové jméno nebylo nalezeno
- `TRY_AGAIN` – DNS server nenalezen nebo odpověděl `SERVFAIL`
- `NO_RECOVERY` – při vytváření odpovědi došlo k chybě, chybné doménové jméno
- `NO_DATA` – doménové jméno bylo nalezeno, avšak požadovaná věta nikoli
- `NETDB_INTERNAL` – interní chyba resolveru

4.2 Struktura _res

Každá z výše uvedených funkcí (začínající `res_`) čte z této struktury nastavení. Změnou hodnot v této struktuře lze dosáhnout odlišného chování funkcí. Definice je umístěna v souboru `resolv.h`.

Proměnná `retry` udává, kolikrát bude funkce `res_send()` při neobdržení odpovědi ve stanoveném časovém limitu posílat paket s požadavkem. Interval mezi jednotlivými pokusy se nastavuje v `retrans`.

Ve struktuře se nachází 32-bitová proměnná `options`. Nastavením (čtením) jednotlivých bitů tohoto dvojslova lze dosáhnout požadovaného chování výše uvedených funkcí. Přehled bitů, které jsou implementovány (číslo v závorce udává nastavení bitu po inicializaci, 0 – funkce deaktivována):

- `RES_INIT` (0) – je-li nastaven, byla volána funkce `res_init()`.
- `RES_DEBUG` (0) – výpis ladících informací při zpracovávání funkcí resolveru.
- `RES_USEVC` (0) – je-li hodnota bitu rovna 0, jsou používány UDP pakety, v opačném případě TCP pakety (spolehlivý přenos, avšak vyšší režie).
- `RES_STAYOPEN` (0) – jestliže jsou využívány pro přenos TCP pakety, pak hodnota 1 tohoto bitu zaručuje, ze po komunikaci zůstane spojení otevřeno.
- `RES_IGNTC` (0) – ignorování bitu Truncation – v případě, že se odpověď nevešla do UDP paketu (a je nastaven bit TR v hlavičce paketu) je navázáno spojení pomocí TCP za předpokladu, že je tento bit roven 0. V opačném případě je bit TR ignorován, zpráva zůstane neúplná.
- `RES_RECURSE` (1) – nastavením bitu na 0 je zrušeno rekurzivní dotazování (bit RD v hlavičce paketu odpovídá hodnotě tohoto bitu)
- `RES_DEFNAMES` (1) – automatické přidávání předem nastavené domény k doménovému jménu, jestliže je jméno neúplné
- `RES_DNSRCH` (1) – automatické přidávání záznamů ze seznamu předem definovaných domén k jménu, které nekončí tečkou
- `RES_INSECURE1` (0) – ignorování odpovědí ze serveru, které nebyly vyžádány
- `RES_INSECURE2` (0) – ignorování odpovědí, kde sekce dotazu neodpovídá původnímu požadavku
- `RES_NOALIASES` (0) – jestliže se nastaví tento bit na 1, je zakázáno používání synonym (aliasů) definovaných v souboru určeným proměnnou prostředí `HOSTALIASES`.

4.3 Funkce pro dekódování odpovědi DNS serveru

`int ns_init_parse(const u_char *msg, int msglen, ns_msg *handle)`

Funkci `ns_init_parse` je nutno zavolat před použitím ostatních funkcí s prefixem `ns_`, slouží k naplnění datové struktury, kterou pak využívají ostatní rutiny.

Parametry:

- `msg` – ukazatel na buffer s odpovědí DNS serveru
- `msglen` – velikost odpovědi
- `handle` – ukazatel na datovou strukturu `ns_msg`, kterou tato funkce naplní daty z odpovědi.

Návratová hodnota je v případě úspěchu 0, jinak -1.

Poznámka: u následujících funkcí se vždy vyskytuje parametr typu `ns_msg` – předává se do něj struktura, která byla inicializována prostřednictvím funkce `ns_init_parse()`.

```
const u_char *ns_msg_base(ns_msg handle)
const u_char *ns_msg_end(ns_msg handle)
int ns_msg_size(ns_msg handle)
```

Tyto funkce vracejí ukazatel na začátek, ukazatel na konec a délku odpovědi DNS serveru.

```
u_int16_t ns_msg_id(ns_msg handle)
```

Funkce `ns_msg_id` vrací identifikační číslo z hlavičky paketu.

```
u_int16_t ns_msg_get_flag(ns_msg handle, ns_flag flag)
```

Funkce `ns_msg_get_flag` vrací údaj položky z paketu dle hodnoty v proměnné `flag`. Význam vrácené hodnoty je uveden v kapitole 3.2.1.

Parametr `flag` může nabývat jednu hodnotu z těchto konstant:

- `ns_f_qr` – dotaz / odpověď
- `ns_f_opcode` – typ otázky
- `ns_f_aa` – autoritativní odpověď
- `ns_f_tc` – odpověď zkrácena
- `ns_f_rd` – klient požaduje rekurzivní překlad
- `ns_f_ra` – server umožňuje rekurzivní překlad
- `ns_f_rcode` – návratový kód odpovědi

```
u_int16_t ns_msg_count(ns_msg handle, ns_sect section)
```

Funkce `ns_msg_count` vrací počet záznamů v sekci určené parametrem `section`. Údaj odpovídá hodnotě jednoho z následujících polí v hlavičce: QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT.

Parametr `section` může nabývat jednu hodnotu z těchto konstant:

- `ns_f_qd` – dotazy
- `ns_f_an` – odpovědi
- `ns_f_ns` – autoritativní jmenné servery
- `ns_f_ar` – doplňující informace

```
int ns_parserr(ns_msg *handle, ns_sect section, int rrnum, ns_rr *rr)
```

Funkce `ns_parserr` extrahuje RR větu z odpovědi. O kterou se jedná určují parametry `section` a `rrnum`.

Parametry:

- `handle` – ukazatel na strukturu inicializovanou pomocí `ns_init_parse()`
- `section` – parametr je vysvětlen u popisu funkce `ns_msg_count()`
- `rrnum` – číslo RR věty, která je požadována, věty se číslují od 0. Pro zjištění počtu vět se používá `ns_msg_count()`.
- `rr` – ukazatel na datovou strukturu, která bude naplněna údaji požadované RR věty.

Návratová hodnota je 0, v případě chyby -1.

```
char *ns_rr_name(ns_rr rr)
u_int16_t ns_rr_type(ns_rr rr)
u_int16_t ns_rr_class(ns_rr rr)
u_int32_t ns_rr_ttl(ns_rr rr)
u_int16_t ns_rr_rdlen(ns_rr rr)
const u_char *ns_rr_rdata(ns_rr rr)
```

Tyto funkce vracejí příslušný záznam z věty, popis je uveden v kapitole 2.

Jediný parametr je zde typu `ns_rr`, do něhož se předává proměnná inicializovaná funkcí `ns_parserr()`.

```
int ns_name_compress(const char *exp_dn, u_char *comp_dn, size_t length,
                     const u_char **dnptrs, const u_char **lastdnptr)
```

Funkce `ns_name_compress` komprimuje doménové jméno. Standardně se nevolá přímo, protože je volána funkcí `res_mkquery()`.

Parametry:

- `exp_dn` – plně kvalifikované doménové jméno, které se bude komprimovat
- `comp_dn` – ukazatel na buffer, do kterého se uloží zkomprimované doménové jméno
- `length` – velikost bufferu předávaného do `comp_dn`
- `dnptrs` – pole ukazatelů na předchozí zkomprimovaná doménová jména. `dnptrs[0]` obsahuje ukazatel na začátek zprávy (DNS paketu), poslední ukazatel je NULL. Před prvním použitím této funkce je tedy potřeba zapsat adresu začátku zprávy do `dnptrs[0]`, NULL uložit do `dnptrs[1]`. Při volání funkce se již tyto adresy automaticky mění.

- `lastdnptr` – adresa posledního prvku pole `dnptrs`.

Návratová hodnota udává délku zkomprimovaného doménového jména, v případě chyby -1.

```
int ns_name_uncompress(const u_char *msg, const u_char *eomorig,
                       const u_char *comp_dn, char *exp_dn, size_t length)
```

Funkce `ns_name_uncompress` je opakem předchozí funkce, tedy převede zkomprimované doménové jméno na klasický řetězec ukončený znakem \0. Její použití je převážně určeno na upravení doménového jména, které vrací např. `ns_rr_rdata()` do úplné čitelné podoby.

Parametry:

- `msg` – ukazatel na začátek zprávy (paketu)
- `eomorig` – adresa prvního bytu za zprávou
- `comp_dn` – řetězec se zkomprimovaným doménovým jménem
- `exp_dn` – ukazatel na buffer, do kterého se doménové jméno „rozbalí". Tento buffer by měl být minimálně `MAXDNAME` bytů dlouhý.
- `length` – velikost bufferu v `exp_dn`

Návratová hodnota udává velikost zkomprimovaného doménového jména, nikoli kompletního (rozšířeného) doménového jména. V případě chyby vrací -1. Důvodem, proč tato funkce vrací počet bytů zkomprimovaného jména je to, že při postupném procházení údajů z části RDATA je potřeba znát, kolik tento řetězec zabírá místa, aby jej bylo možné přeskočit a přejít na další.

```
u_int ns_get16(const u_char *cp)
void  ns_put16(u_int s, u_char *cp)
```

DNS pakety obsahují položky, které jsou typu `unsigned short` (16 bitů) – např. type, class…. Pomocí těchto dvou funkcí lze z adresy udané parametrem `cp` hodnotu přečíst (`ns_get16()`) nebo uložit (`ns_put16()`).

```
u_long ns_get32(const u_char *cp)
void   ns_put32(u_long l, u_char *cp)
```

Stejně jako v předchozím případě se tyto funkce používají na čtení / zápis 32-bitových hodnot (např. položka TTL) z RR věty.

5. Příklad odpovědi DNS serveru

DNS serveru byl zaslán požadavek pomocí tohoto příkazu:

```
res_search("vutbr.cz", ns_c_in, ns_t_mx, buf, PACKETSZ);
```

Příchozí odpověď:

```
00000000  65 4c 85 80 00 01 00 01  00 02 00 03 05 76 75 74  |eL...........vut|
00000010  62 72 02 63 7a 00 00 0f  00 01 c0 0c 00 0f 00 01  |br.cz.....À.....|
00000020  00 01 51 80 00 09 00 05  04 6d 61 69 6c c0 0c c0  |..Q......mailÀ.À|
00000030  0c 00 02 00 01 00 01 51  80 00 0c 02 6e 73 03 63  |.......Q....ns.c|
00000040  65 73 03 6e 65 74 00 c0  0c 00 02 00 01 00 01 51  |es.net.À.......Q|
00000050  80 00 0c 05 72 68 69 6e  6f 03 63 69 73 c0 0c c0  |....rhino.cisÀ.À|
00000060  28 00 01 00 01 00 00 02  58 00 04 93 e5 03 34 c0  |(.......X...å.4À|
00000070  3b 00 01 00 01 00 01 d7  2f 00 04 c3 71 90 e9 c0  |;......×/..Ãq.éÀ|
00000080  53 00 01 00 01 00 00 7d  c4 00 04 93 e5 03 0a     |S......}Ä...å..|
0000008f
```

Adresa	Blok dat	Význam		
Záhlaví				
00 – 01	65 4C	identifikátor zprávy = 654C		
02 – 03	85 80	8580 =	100001011000000	
		QR	1	zpráva je odpovědí
		OPCODE	0000	byla položena standardní otázka
		AA	1	odpověď je z autoritativního záznamu
		TC	0	odpověď se vešla do paketu UDP
		RD	1	klient požadoval rekurzivní překlad
		RA	1	server umožňuje rekurzivní překlad
		Z	000	bity rezervované pro budoucí použití
		RCODE	0000	k chybě nedošlo
04 – 05	00 01	QDCOUNT = 1, byl zaslán 1 dotaz		
06 – 07	00 01	ANCOUNT = 1, byla přijata 1 odpověď		
08 – 09	00 02	NSCOUNT = 2, existují 2 autoritativní jmenné servery		
0A – 0B	00 03	ARCOUNT = 3, v odpovědi jsou 3 doplňující informace		

Dotazy		
0C – 15	05 76 75 74 62 72 02 63 7A 00	doménové jméno vutbr.cz, kde je před každým řetězcem uvedena jeho délka, na konci je hodnota 00
16 – 17	00 0F	QTYPE = 15 - požadavek na získání věty MX
18 – 19	00 01	QCLASS = 1 - jedná se o třídu Internet
Odpovědi		
1A – 1B	C0 0C	doménové jméno, díky použití komprese je zde pouze odkaz na text začínající na adrese 0x0C, tedy vutbr.cz
1C – 1F	00 0F 00 01	QTYPE = 15 (záznam MX), QCLASS = 1, viz. adresa 0x16
20 – 23	00 01 51 80	TTL = 86400 s
24 – 25	00 09	RDLENGTH = 9 bytů
26 – 27	00 05	část pole RDATA - protože jde o MX záznam, je zde uložena priorita s hodnotou 5
28 – 2E	04 6D 61 69 6C C0 0C	komprimované doménové jméno mail.vutbr.cz
Autoritativní DNS servery		
2F – 46	C0 0C 00 02 00 01 00 01 51 80 00 0C 02 6E 73 03 63 65 73 03 6E 65 74 00	první autoritativní jmenný server (QTYPE = 2, NS záznam), zbývající údaje viz adr. 0x1A doménové jméno není komprimováno: ns.ces.net
47 – 5E	C0 0C 00 02 00 01 00 01 51 80 00 0C 05 72 68 69 6E 6F 03 63 69 73 C0 0C	druhý autoritativní jmenný server doménové jméno komprimováno: rhino.cis.vutbr.cz
Doplňující informace		
5F – 6E	C0 28 00 01 00 01 00 00 02 58 00 04 93 E5 03 34	doménové jméno: mail.vutbr.cz QTYPE = 1, tedy záznam A, QCLASS = 1 TTL = 600 s IP adresa mail serveru je 147.229.3.52
6F – 7E	C0 3B 00 01 00 01 00 01 D7 2F 00 04 C3 71 90 E9	IP adresa ns.ces.net je 195.113.144.233 TTL = 120623 s
7F – 8E	C0 53 00 01 00 01 00 00 7D C4 00 04 93 E5 03 0A	IP adresa rhino.cis.vutbr.cz je 147.229.3.10 TTL = 32196 s

6. Použitá literatura

RFC 1035 – Domain names – implementation and specification, November 1987. Dokument dostupný na URL http://www.faqs.org/rfcs/rfc1035.html (14. listopadu 2004).

Dostálek, L., Kabelová, A.: Velký průvodce protokoly TCP/IP a systémem DNS, 3. aktualizované a rozšířené vydání, Computer Press, 2002

Kállay, F., Peniak, P.: Počítačové sítě LAN/MAN/WAN a jejich aplikace, Grada Publishing, a.s., 2003

Liu, C., Albitz, P.: DNS and BIND, Third Edition, O'Reilly, September 1998. Dokument dostupný na URL http://www.unix.org.ua/orelly/networking/dnsbind/index.htm (14. listopadu 2004).

7. Příloha – ukázkový program

Ukázkový program realizuje dotazování na záznamy typu A, MX, PTR. Je napsán v jazyce C, překlad úspěšně proběhl pod systémem FreeBSD.
Program vypisuje stejné informace jako aplikace `host` bez parametrů. Jediným rozdílem je to, že lze vložit více parametrů. Příklad: `test_app www.google.com vutbr.cz 147.229.8.12`

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <arpa/inet.h>
#include <resolv.h>
#include <netdb.h>

#define TRUE 1
#define FALSE 0

#define ERROR(msg)          \           // Makro pro výpis chybového hlášení a ukončení běhu programu
  {                         \
    fprintf(stderr, msg);   \
    fputc('\n', stderr);    \
    return FALSE;           \
  }

int resolve(const char *szAddress);     // Funkční prototypy
int runDnsQuery(const char *szAddress, int nType, int bNoDataError);

int resolve(const char *szAddress) {    // Je-li szAddress IP adresa v textovém formátu, zjistí se
  in_addr_t addr4;                      // odpovídající doménové jméno, je-li szAddress doménové
  register int i;                       // jméno, zjišťují se informace z RR vět typu A a MX.
  int ipAddr[4] = {0, 0, 0, 0};
  char buf[MAXDNAME];
  int nRet;

  if ((addr4 = inet_network(szAddress)) != -1) {
    for (i = 0; addr4; ) {              // szAddress je IP adresa, uloží se do pole ipAddr
      ipAddr[i++] = addr4 & 0xFF;
      addr4 >>= 8;
    }
    sprintf(buf, "%u.%u.%u.%u.in-addr.arpa",
            ipAddr[i % 4],     ipAddr[(i+1) % 4],
            ipAddr[(i+2) % 4], ipAddr[(i+3) % 4]);
                                        // Zjistí se věta typu PTR
    nRet = runDnsQuery(buf, ns_t_ptr, TRUE);
  }
  else {                        // szAddress je pravděpodobně doménové jméno, zjišťují se věty A a MX
    if ((nRet = runDnsQuery(szAddress, ns_t_a, FALSE)))
      nRet = runDnsQuery(szAddress, ns_t_mx, FALSE);
  }

  return nRet;
}

int runDnsQuery(const char *szAddress, int nType, int bNoDataError) {
  u_char pResAnswer[PACKETSZ];          // Funkce provede dotazování DNS serveru a výpis zjištěných
  int nResAnswerLen;                    // informací.
  ns_msg hMsg;                          // szAddress – doménové jméno
  ns_rr rr;                             // nType – typ požadované RR věty (konstanta ns_t_xx)
  int i, j;                             // bNoDataError – je-li TRUE, hlásí se chyba a funkce skončí,
  char szUncompressed[MAXDNAME];        // jestliže funkce resolveru vrátí chybu NO_DATA. V opačném
  const u_char *p;                      // případě se toto hlášení resolveru ignoruje.

  nResAnswerLen = res_search(szAddress, ns_c_in, nType,   // Zavolá se funkce resolveru pro získání
                             pResAnswer, PACKETSZ);       // požadované RR věty.
  if (nResAnswerLen < 0) {              // Identifikace případné chyby.
    switch (h_errno) {
      case HOST_NOT_FOUND:
        ERROR("Host not found.")
      case TRY_AGAIN:
        ERROR("DNS not found or server returned SERVFAIL")
```

```
      case NO_RECOVERY:
        ERROR("DNS error")
      case NO_DATA:
        if (bNoDataError)
          ERROR("DNS query failed - name exists, but no data record of requested type")
        else
          return TRUE;
      default:
        ERROR("Internal error")
    }
  }
  if (ns_initparse(pResAnswer, nResAnswerLen, &hMsg))   // Naplnění struktury ns_msg.
    ERROR("Internal error - ns_initparse() failed");

  for (i = 0; i < ns_msg_count(hMsg, ns_s_an); i++) {
    if (ns_parserr(&hMsg, ns_s_an, i, &rr) < 0)
      ERROR("Internal error - ns_parserr() failed")

    switch (ns_rr_type(rr)) {    // Zjistí se typ věty a vypíší se informace na konzoli.
      case ns_t_cname:
        if (nType == ns_t_mx)    // Jestliže jedna z vět vrácených při dotazu na větu MX je CNAME,
          break;                 // ignorujeme ji, protože věta CNAME již byla zpracována ve volání
                                 // s požadavkem na větu A. Pokud by ignorována nebyla, byly by
                                 // na výstupu dva stejné řádky.
        if (ns_name_uncompress(ns_msg_base(hMsg), ns_msg_end(hMsg),
                               ns_rr_rdata(rr), szUncompressed, MAXDNAME) < 0)
          ERROR("Internal error - ns_name_uncompress() failed");
        printf("%s is a nickname for %s\n", ns_rr_name(rr), szUncompressed);
        break;

      case ns_t_a:
        printf("%s has address ", ns_rr_name(rr));
        p = ns_rr_rdata(rr);
        for (j = 0; j < ns_rr_rdlen(rr); j++) {
          if (j > 0)             // Převedení IP adresy v binární podobě do textové.
            putchar('.');
          printf("%d", (u_char) (*p++));
        }
        putchar('\n');
        break;

      case ns_t_mx:
        p = ns_rr_rdata(rr);     // 2 byty priority, pak následuje doménové jméno.
        if (ns_name_uncompress(ns_msg_base(hMsg), ns_msg_end(hMsg),
                               p + INT16SZ, szUncompressed, MAXDNAME) < 0)
          ERROR("Internal error - ns_name_uncompress() failed");
        printf("%s mail is handled (pri=%d) by %s\n", ns_rr_name(rr), ns_get16(p), szUncompressed);
        break;

      case ns_t_ptr:
        if (ns_name_uncompress(ns_msg_base(hMsg), ns_msg_end(hMsg),
                               ns_rr_rdata(rr), szUncompressed, MAXDNAME) < 0)
          ERROR("Internal error - ns_name_uncompress() failed");
        printf("%s domain name pointer %s\n", ns_rr_name(rr), szUncompressed);
        break;

    }
  }
  return TRUE;
}

// Vstupní bod programu.
int main(int argc, char *argv[]) {
  int i;

  if (argc == 1) {
    printf("Usage: app domain_name|IP_address {domain_name|IP_address}\n"
           "Example: app google.com 147.229.10.14\n");
    return 1;
  }
  for (i = 1; i < argc; i++) {  // Každý parametr programu zkusíme převést.
    if (i > 1)
      printf("\n");
    resolve(argv[i]);
  }
  return 0;
}
```