Zavod za elektorniku, mikroelektroniku, računalne i inteligentne sustave
Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu

Stjepan Groš

MREŽE RAČUNALA

Pripreme za laboratorijske vježbe

Zagreb, 2003.

Promjene na dokumentu

Verzija	Datum	Promjene
1.00	14. 11. 2003.	Inicijalna verzija.
1.01	02. 12. 2003.	Ispravke pogrešaka i dodaci manjih pojašnjenja.
1.02	11. 12. 2003.	Manje ispravke formatiranja.
2	25. 12. 2008.	Specijalno i limitirano Božićno izdanje.

DON'T PANIC

Sadržaj

DON'T PANIC

Uvod

Laboratorijske vježbe iz predmeta *Mreže računala* služe kao praktična nadopuna predavanjima i auditornim vježbama, a na njima se stječu iskustva i produbljuju znanja iz područja umrežavanja računala. Laboratorijske vježbe sastoje se od dvije grupe zadataka. Prvu grupu čine programerski zadaci, a drugu grupu čine zadaci konfiguriranja odgovarajućih konfiguracija.

Dolazak na laboratorijske vježbe je obavezan, osim ako to nije eksplicitno drugačije navedeno. Laboratorijske vježbe predaju se asistentu u unaprijed definiranim terminima. Kako bi se izbjegle moguće zabune za svaku vježbu biti će posebno naglašeno da li ih je potrebno raditi u laboratoriju ili se mogu raditi i doma. Bez obzira na mogućnosti obavljanja labosa doma rezervirani termini za rad u laboratorijima (PCLAB1, PCLAB2, A109 i A110) na raspolaganju su studentima te se u njima mogu dolaziti obavljati laboratorijske vježbe. U tjednima u koje pada praznik na dan kada se održava labos iz Mreža računala službeno nema laboratorijskih vježbi, no studenti imaju pravo doći u ostalim terminima u tom tjednu i raditi laboratorije. Za svaku laboratorijsku vježbu određuju se termini predaje i broj bodova koje se ostvaruje. Bez obzira na broj bodova koje je moguće ostvariti laboratorijske vježbe se **moraju** predati i to je nužni uvijet za pravo na potpis!

Dodatak ovoj pripremi čini CD na kojemu se nalazi sva potrebna programska podrška za obavljanje vježbi. To uključuje i VMWare program, ali je za taj program potrebno zatražiti licencu putem Web sučelja na Internetu. Detalji se nalaze opisani u dodatku B. Distribucija tog CD-a će biti riješena posebno i sve potrebne informacije će se nalaziti na Web stranicama predmeta.

Sadržaj CD-a

- VMWare (Windows NT/2000/XP i Linux verzije) sa korisničkim uputama
- Sva virtualna PC računala korištena tijekom labosa
- man stranice u PDF obliku
- Dodatna dokumentacija za sve mrežne naredbe (ip, lartc)
- tftp implementacije za 5. vježbu
- Priprema za laboratorijske vježbe u PDF obliku

Laboratorijske vježbe rade se na Linux operacijskom sustavu koji je kompatibilan sa Unix operacijskim sustavom. Razlog odabira tog OS-a leži u činjenici da je daleko opremljeniji mrežnom podrškom od mnogih drugih raspoloživih operacijskih sustava, a naročito s obzirom na Windows porodicu. Cjelokupne upute pisane su specifično za taj operacijski sustav te su vjerojatna odstupanja u slučaju korištenja drugih Unix operacijskih sustava. Radna okolina sastoji se od VMWare programa opisanog u dodatku. Unutar tog programa izvršava se odgovarajuće konfiguriran Linux operacijski sustav, a potom se na tom Linux operacijskom sustavu obavljaju vježbe.

U pripremi se objašnjava način korištenja pojedinih funkcija i pomoćnih programa operacijskog sustava. Kada te funkcije ili programi imaju uputstvo koje je moguće vidjeti na samom računalu tada se koristi sljedeća notacija `socket(2)`, tj. navodi se ime funkcije (programa), a u zagradi je navedena grupa uputstava kojoj ta funkcija (program) pripada. Uputu za tako navedenu funkciju (program) moguće je dobiti korištenjem programa `man(1)`, tj. izvršavanjem sljedeće naredbe:

```
$ man 2 socket
```

Određeni programi imaju uputstva i u tzv. info stranicama. To se u ovim uputama ne naglašava posebno, ali je to uglavnom naglašeno unutar same man stranice.

Radna okolina laboratorijskih vježbi sastoji se od već spomenutog VMWare programa sa instaliranim Linux operacijskim sustavom i odgovarajućom programskom podrškom. Određene labose (konkretno 5. i 6. laboratorijska vježba) moguće je načiniti i pod Windows operacijskim sustavom korištenjem Cygwin programske podrške.

U izradi ove pripreme svesrdnu pomoć, u vidu primjedbi i sugestija, pružili su Leonardo Jelenković i Marko Čupić.

Sve primjedbe na laboratorijske vježbe i ovu pripremu su dobrodošle i mogu se uputiti osobno ili na mail adresu `sgros@zemris.fer.hr`.

DON'T PANIC

Vježba I – Komunikacija upotrebom podatkovnog sloja

1. Uvod

Računalna mreža pruža usluge aplikacijama koje te usluge koriste pozivajući funkcije operacijskog sustava. Cilj ove i nekoliko narednih laboratorijskih vježbi je upoznavanje sa načinom korištenja tih usluga prilikom izrade korisničkih programa. Za početak, u ovoj laboratorijskoj vježbi koriste se usluge podatkovnog sloja. Korištenje usluga podatkovnog sloja upotrebljava se isključivo u specijalnim aplikacijama, dok velika većina aplikacija koristi usluge transportnog ili viših slojeva. Transportni sloj tema je idućih laboratorijskih vježbi.

Kako bi programer mogao koristiti usluge koje mu pruža mreža, potrebno je koristiti API koji nudi OS preko biblioteka isporučenih sa mrežnom komponentom OS-a. Postoji mnoštvo različitih API-ja na različitim razinama apkstrakcije, kao npr. XTI/TLI, STREAMS, RPC, CORBA, itd. No dominantan i temeljni API je tzv. **socket API** koji se je prvi puta pojavio 1983. godine prilikom izdavanja 4.2BSD Unix operacijskog sustava. U sljedećem poglavlju taj API je detaljnije opisan sa naglaskom na dio potreban za obavljanje ove laboratorijske vježbe.

U današnje vrijeme mreže su toliko integrirane u računarstvo da se dobar dio implementacije mrežnih protokola nalazi u samoj jezgri operacijskog sustava, između ostalog i zbog efikasnosti. Te implementacije su dosta komplicirane no ipak je zanimljivo pogledati kako je to izvedeno u stvarnosti, tj. u operacijskom sustavu koji se koristi i u komercijalnim, a ne samo hobističkim ili znanstvenim projektima. To je moguće zahvaljujući dostupnosti, tj. otvorenosti koda *BSD i Linux operacijskih sustava. Iako ti svi operacijski sustavi pripadaju Unix porodici, implementacije su potpuno različite zbog povijesnih razloga i različitih licenci.

2. Zadatak

U ovoj laboratorijskoj vježbi potrebno je načiniti program koji omogućava sljedeću funkcionalnost:

1. slanje ARP upita na mrežu i primanje odgovora.
2. osluškivanje mreže i prikaz svih viđenih ARP paketa, i
3. poslužitelj koji sadrži bazu IP adresa i Ethernet adresa i odgovara na sve upite koji se nalaze u njegovoj bazi.

ARP, skraćeno od *Address Resolution Protocol*, je način na koji se mrežne adrese (treći sloj ISO/OSI modela) pretvaraju u sklopovske adrese (adrese drugog sloja ISO/OSI mrežnog modela). U ovoj laboratorijskoj vježbi pretvaranje se obavlja isključivo između IPv4 adresa i Ethernet adresa.

Za slanje upita program se poziva iz komandne linije na sljedeći način:

```
# ./arp <sucelje> query <trazena ip adresa> <ip adresa sucelja>
```

Prvi parametar je sučelje na koje se šalje upit, te sa kojega se primaju paketi. Drugi parametar je IP adresa čiju Ethernet adresu želimo saznati. Treći parametar je IP adresa sučelja kroz koje se šalje upit (prvi parametar). Tijekom izvršavanja program mora ispisati upit koji je poslao, te što je primio kao odgovor. Nakon što program pošalje upit čeka određeno vrijeme i ako ne primi odgovor ponovo šalje upit na mrežu. Nakon određenog broja pokušaja, ako nije dobio odgovor, odustaje s pretpostavkom da ta mrežna adresa ne postoji.

Kako bi se osluškivala mreža treba se koristiti sljedeći način poziva programa:

```
# ./arp <sucelje> listen
```

Nakon toga program se izvršava dok god se ne prekine pritiskom na tipke Ctrl-C. Zadanu kombinaciju tipki presreće operacijski sustav i potom zatvara pristupne točke i prekida izvršavanje programa pa nije potrebno u programu eksplicitno presretati tu kombinaciju tipki. Za svaki primljeni upit program ispisuje liniju sljedećeg oblika:

```
who-has 192.168.1.2 tell 192.168.1.1 at 01:02:03:04:05:06
```

Prva IP adresa je adresa koja se traži, potom dolaze IP i Ethernet adresa pošiljatelja kako su zapisana u ARP paketu. Za svaki primljeni odgovor na neki upit program ispisuje liniju sljedećeg oblika:

```
192.168.1.2 is-at 03:04:05:06:07:08
```

Konačno, kako bi se pokrenuo poslužitelj koji prima upite i odgovara na njih u slučaju da se IP adresa nalazi u njegovoj bazi koristi se sljedeći oblik naredbe:

```
# ./arp <sucelje> server
```

Isključivi parametar u ovom slučaju je sučelje sa kojega se primaju upiti i šalju odgovori. U trenutku pokretanja program prvo čita tekstualnu datoteku `arp.db` u kojoj su zapisani parovi IP adresa – Ethernet adresa. Svaki par nalazi se u zasebnoj liniji. Primjer nekoliko parova je:

```
192.168.1.1    00:FA:85:C6:AA:11
192.168.1.2    00:FA:85:C6:AA:12
192.168.1.3    00:FA:85:C6:AA:13
192.168.1.4    00:FA:85:C6:AA:14
```

Maksimalan broj parova nije definiran. Nakon što je pročitana baza program osluškuje mrežu i za svaki primljeni upit provjerava da li se nalazi u bazi. Ako se nalazi tada odgovara na upit, u suprotnom ignorira upit. Svaki upit i odgovor na njega treba prikazati na monitoru. Format ispisa isti je kao i u slučaju osluškivanja ARP prometa. Program se prekida sa Ctrl-C.

U svim slučajevima, prilikom slanja Ethernet okvira, kao izvorišnu adresu (polje SA) potrebno je postaviti Ethernet adresu sučelja preko kojeg se šalje paket.

3. Pristupne točke i API za njihovo korištenje

Svaka aplikacija prije no što može koristiti mrežu mora kreirati pristupnu točku – **socket**. Pristupna točka je entitet koji se kreira na osnovu tri parametra i preko kojega aplikacije koriste usluge računalne mrežne. Sa ta tri parametra definira se operacijskom sustavu vrsta uslugu koju želimo od računalne mreže. U slučaju ove laboratorijske vježbe potreban nam je pristup podatkovnom sloju pri čemu će aplikacija generirati sva potrebna zaglavlja. Drugim riječima, od operacijskog sustava očekujemo da sve što mu proslijedimo za slanje pošalje bez ikakvih modifikacija, dok sve primljene pakete proslijedi aplikaciji u neizmjenjenom obliku. U daljnjem tekstu sva objašnjenja ograničena su na potrebe ove laboratorijske vježbe, dok se detalji mogu pronaći u odgovarajućim uputstvima (man stranice).

DON'T PANIC

Kreiranje pristupne točke obavlja se pozivom funkcije `socket(2)`. Tom funkcijom samo se najavljuje jezgri operacijskog sustava namjera korištenja mrežnih usluga određenog tipa na što kernel odnosno odgovarajuće funkcije u sistemskim bibliotekama kreiraju potrebne strukture za rad sa mrežom. Funkcija kao rezultat vraća deskriptor koji se koristi prilikom poziva drugih mrežnih funkcija slično kao što se deskriptor otvorene datoteke koristi u funkcijama za čitanje i pisanje u datoteku.

Prototip funkcije ima sljedeći oblik:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netpacket/packet.h>
#include <net/ethernet.h>

int socket(int domain, int type, int protocol);
```

dok argumenti imaju sljedeća značenja:

- `domain`

  Ovaj argument određuje koju grupu komunikacijskih usluga želimo koristiti. Vrijednost koja nas zanima za ovu laboratorijsku vježbu je `AF_PACKET` čime naznačujemo da želimo pristupati podatkovnom sloju. Prefix `AF_` potiče od skraćenice **Address Family** no može se koristiti i prefiks `PF_` što dolazi od **Protocol Family**.

- `Type`

  Uz pomoć ovog argumenta dodatno se specificira karakteristika mrežne usluge koju želimo koristiti. U ovoj laboratorijskoj vježbi potrebno direktno pristupiti podatkovnom sloju, i ne želimo da operacijski sustav bilo što mijenja na odlaznim i dolaznim paketima. Kako bi to postigli potrebno je koristiti konstantu `SOCK_RAW`.

- `Protocol`

  Konačno, ovaj argument bira protokol koji se koristi za implementaciju željene vrste komunikacije u danoj grupi komunikacijskih usluga. Budući da najčešće samo jedan protokol implementira određenu vrstu komunikacije to se za ovaj parametar najčešće navodi 0, a to znači da kernel i/ili funkcije u bibliotekama same odaberu odgovarajući protokol na osnovu prva dva parametra. U slučaju podatkovnog sloja ovim parametrom određujemo tip paketa što ih prenosi podatkovni sloj, tj. određujemo vrijednost polja `Type` u Ethernet okviru. Moguće vrijednosti su `ETH_P_IP` za IP protokol, `ETH_P_ARP` za ARP protokol i čitav niz drugih protokola. Za potpuni popis potrebno je pogledati zaglavnu datoteku `linux/if_ether.h`.

  Ovim argumentom određuje se sadržaj polja u okviru koji putuje po lokalnoj mreži. Kako na lokalnoj mreži poredak riječi koje se sastoje od dva okteta može biti različit od poretka na računalu potrebno je koristiti funkciju `htons(3)` kojoj je argument konstanta a rezultat je ispravan mrežni poredak okteta.

Osim već spomenute mrežne usluge koja se definira sa konstantom `SOCK_RAW` , postoji i mrežna usluga definirana konstantom `SOCK_DGRAM`. Razlika je u tome da prilikom primanja paketa OS ukloni Ethernet zaglavlje te ga se posebno isporučuje aplikaciji, dok prilikom slanja aplikacija mora posebno proslijediti adresu primatelja i OS sastavlja Ethernet zaglavlje neposredno prije slanja.

Povratna vrijednost funkcije je deskriptor pristupne točke, tj. pozitivan broj. Ako je došlo do

greške prilikom kreiranja pristupne točke vraća se -1 te je varijabla `errno` postavljena na kod koji dodatno opisuje razlog pojave greške. Funkcija `perror(3)` se može koristiti za dobijanje teksta koji opisuje grešku.

Nakon što je kreirana pristupnu točku tipa `AF_PACKET` možemo vezati na neko sučelje. To se obavlja pozivom funkcije `bind(2)`. Prototip funkcije je sljedeći:

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr,
                  socklen_t addrlen);
```

Drugi argument je adresna struktura tipa `struct sockaddr_ll`, ali se funkciji prosljeđuje kao generički tip `struct sockaddr`. Treći argument sadrži veličinu u oktetima koju zauzima adresna struktura. Generičke adrese i adrese tipa `struct sockaddr_ll` detaljnije su opisane u nastavku teksta. Za funkciju `bind(2)` potrebno je još napomenuti da se koriste samo dva člana strukture `sockaddr_ll` i to `sll_family` i `sll_ifindex`. Prvi je potrebno postaviti na konstantu vrijednost `AF_PACKET`, a drugi na broj sučelja na koji želimo povezati pristupnu točku.

Već nakon kreiranja pristupna točka se može koristiti za slanje i primanje okvira, iako bez poziva funkcije `bind(2)` slanje može biti problematično. Slanje okvira obavlja se pozivom funkcije `sendto(2)`.

Prototip te funkcije ima sljedeći oblik:

```
#include <sys/types.h>
#include <sys/socket.h>

int sendto(int sockfd, const void *msg, size_t len, int flags,
                  const struct sockaddr *to, socklen_t tolen);
```

dok argumenti imaju sljedeća značenja:

- `sockfd`

  Deskriptor koji je dobijen pozivom funcije `socket(2)` nakon uspješnog kreiranja pristupne točke.

- `msg`

  Pokazivač na spremnik u kojemu se nalazi okvir koji je potrebno poslati.

- `len`

  Broj okteta u spremniku.

- `flags`

  Argument uz pomoć kojega se mogu definirati dodatni parametri slanja, no u ovom slučaju kao vrijednost ovog argumenta može se postaviti 0.

- `to`

  Struktura koja sadrži adresu primatelja, a opisana je u daljnjem tekstu. Tu strukturu je potrebno inicijalizirati odgovarajućim vrijednostima prije no što se koristi.

- `tolen`

Veličina prethodne strukture, odnosno `sizeof(to)`.

Funkcija vraća broj poslanih okteta, ili vrijednost -1 u slučaju greške. Ako je došlo do greške tada je varijabla `errno` postavljena na kod koji pobliže određuje uzrok njene pojave.

Primanje paketa obavlja se funkcijom `recvfrom(2)`. Ta funkcija ima sljedeći prototip:

```
#include <sys/types.h>
#include <sys/socket.h>

int recvfrom(int sockfd, void *buf, size_t len, int flags,
                  struct sockaddr *from, socklen_t *fromlen);
```

Povratna vrijednost funkcije je broj primljenih okteta, odnosno -1 u slučaju greške. Funkcija blokira u slučaju da nema pristiglih podataka. Kako je primanje paketa podatkovnog sloja nedjeljiva operacija to će funkcija uvijek vratiti kompletan paket podatkovnog sloja. Argumenti te funkcije analogni su argumentima funkcije `sendto(2)`. U nastavku su opisane samo razlike u odnosu na funkciju `sendto(2)`.

Drugi argument funkcije je pokazivač na spremnik u koji se pohranjuje pristigli okvir, treći argument, `len`, određuje maksimalnu veličinu spremnika. Peti argument, `from`, je pokazivač na strukturu u koju se upisuje adresa sa koje je pristigao paket, a u šesti argument, `fromlen`, se upisuje duljina petog argumenta. U slučaju korištenja petog argumenta, nužno je prije poziva funkcije `recvfrom(2)` inicijalizirati varijablu `fromlen` na raspoloživu količinu prostora u `from` spremniku! Kako se u našem slučaju sve potrebne adrese nalaze u samom okviru to su vrijednosti petog i šestog argumenta nebitni i mogu se postaviti na konstantu NULL. Tijekom izvršavanja programa svaki okvir koji pristigne i ima tip paketa identičan tipu navedenom prilikom poziva funkcije `socket(2)` prvo se proslijeđuje našoj aplikaciji (ili bilo kojoj aplikaciji koja koristi pristupnu točku na podatkovnom sloju sa tom vrstom paketa), a nakon toga proslijeđuje se i aplikaciji kojoj je namijenjena. Ako želimo da nam se proslijeđuju svi okviri, bez obzira na protokol koji se u njima nalazi, tada je prilikom kreiranja spojne točke potrebno koristiti konstantu `ETH_P_ALL` kao treći argument.

Nakon što više nije potrebna pristupna točka zatvara se pozivom funkcije `close(2)`.

3.1.Adresiranje na podatkovnom sloju

Adrese su već spominjane prilikom objašnjavanja načina na koji se koriste funkcije `sendto(2)` i `recvfrom(2)`. Budući da se pomoću pristupnih točaka koriste usluge različitih mrežnih slojeva to se i korištene adrese nužno razlikuju. Kako bi se riješio taj problem funkcije za rad sa pristupnim točkama deklarirane su sa da primaju generičke adrese, tj. `struct sockaddr`. Tijekom izvršavanja programa dinamički se utvrđuje o kojoj vrsti adresa je riječ. Jedan od načina na koji se to obavlja je uz pomoć same adresne strukture. Bez obzira o kojoj vrsti adresa se radi prvo polje je uvijek isto (16-bitni cijeli broj) i on pobliže određuje tip adresne strukture. Na osnovu tog broja uvijek se može znati što se točno nalazi u ostatku strukture.

DON'T PANIC

Na podatkovnom sloju koristi se adresna struktura `struct sockaddr_ll`. Ta struktura, isto kao i vrsta pristupnih točaka korištenih u ovoj laboratorijskoj vježbi, detaljnije je opisana u `raw(7)`. Za njeno korištenje potrebno je uključiti zaglavnu datoteku `linux/if_arp.h`. U strukturu `struct sockaddr_ll` upisuje se odgovarajuća adresa i neki dodatni podaci podatkovnog sloja. Struktura ima sljedeća polja:

- `unsigned short sll_family`

  Ovaj član strukture sadrži konstantu koja određuje o kojoj vrsti adrese se radi. U našem slučaju vrijednost ovog polja se uvijek inicijalizira na vrijednost `AF_PACKET`.

- `unsigned short sll_protocol`

  Podatkovni dio okvira sadrži podatke iz viših mrežnih slojeva. Ovo polje određuje o kojem protokolu je riječ. Npr., ako je tu upisana vrijednost `ETH_P_IP` tada se u Ethernet okviru prenosi paket IP protokola, a ako je upisana vrijednost `ETH_P_ARP` tada se prenosi paket ARP protokola.

  Ovo polje je 16-bitni broj koji mora biti u mrežnom poretku okteta. Kako bi se to postiglo vrijednost polju se ne dodjeljuje direktno već uz pomoć funkcije `htons(3)`. Argument te funkcije je 16 bitni broj čiji okteti su u poretku računala na kojemu se izvršava program *(host byte order*), dok je povratna vrijednost 16 bitni broj čiji okteti su u standardom poretku definiranom za mrežu *(network byte order*).

- `int sll_ifindex`

  Indeks sučelja na koji se primaju ili šalju podaci. Svako mrežno sučelje na računalu ima dodijeljen svoj redni broj. Naredba `ip(8)` prilikom ispisa sučelja za svako sučelje ispisuje njegov indeks. Programski, broj sučelja se može saznati korištenjem funkcije `ioctl(2)`. Prilikom slanja ovo polje se koristi kako bi se operacijskom sustavu reklo kroz koje sučelje je potrebno poslati paket.

- `unsigned char sll_halen`

  U ovom polju zapisan je broj okteta u adresi podatkovnog sloja. Za Ethernet tu je zapisan broj 6 budući da se Ethernet adresa sastoji od 6 okteta.

- `unsigned char sll_addr[8]`

  Adresa podatkovnog sloja. Broj važećih okteta zapisan je u polju `sll_halen`.

U programu se također koriste različiti zapisi IP adrese te su potrebne funkcije koje vrše konverziju IP brojeva iz ASCII zapisa u odgovarajuće strukture i obratno. Tu zadaću obavljaju funkcije `inet_pton(3)` i `inet_ntop(3)` čije deklaracije glase:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);
int inet_ntop(int af, const void *src, char *dst, socklen_t cnt);
```

DON'T PANIC

Obje funkcije su u stanju prevesti više različitih tekstualnih zapisa adresa različitih protokola, te im se kao prvi argument navodi porodica protokola čiju adresu treba očekivati. U našem slučaju radi se o protokolu IPv4 te se kao prvi argument proslijeđuje konstanta `AF_INET`. Funkciji `inet_pton(3)` drugi argument je pokazivač na ASCII zapis IP adrese i konačno, treći argument je pokazivač na polje `sin_addr` strukture `struct sockaddr`.

Funkciji `inet_ntop(3)` drugi argument je pokazivač na odgovarajuću strukturu `struct in_addr`, treći argument je pokazivač na spremnik u koji se smješta konvertirana adresa, a zadnji argument je veličina tog spremnika.

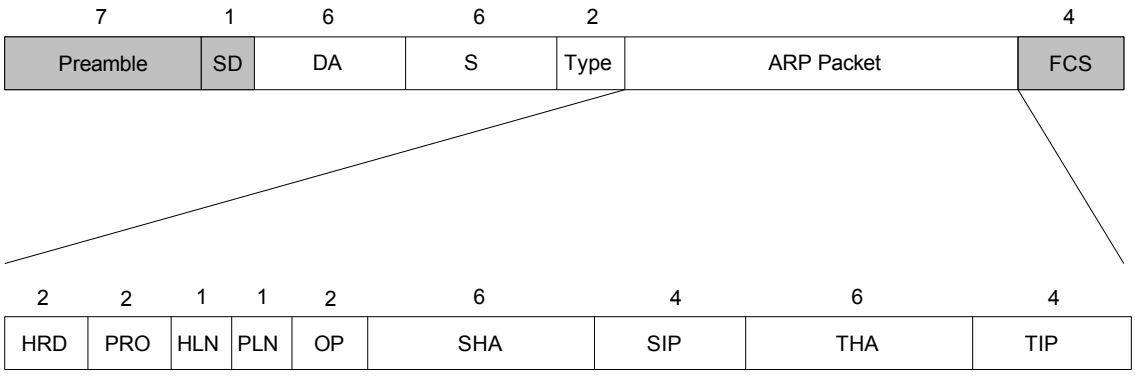
> **Napomena**: Na starijim implementacijama socket API-ja umjesto funkcija `inet_pton(3)` i `inet_ntop(3)` koristile su se funkcije ***`inet_aton(3)`*** i ***`inet_ntoa(3)`***. Te funkcije su zastarjele budući da se mogu koristiti isključivo sa IP adresama vezanim uz verziju 4 IP protokola. Njihov opis ovdje nije dan ali se kao i za sve ostale funkcije može naći u on-line uputstvima.

4. Upute za izradu vježbe

Vježba je prilagođena Unix operacijskom sustavu i u ovom obliku je nije moguće napraviti u nekom drugom okruženju što uključuje i Windows operacijski sustav. Kao radna okolina preporuča se Linux distribucija sa kernelom 2.4 i glibc bibliotekom čija verzija je barem 2.2. Sve moderne Linux distribucije uglavnom zadovoljavaju te uvijete. Vježba zahtijeva root ovlasti te ju nije moguće raditi na računalu pinus.cc.fer.hr. U slučaju da se vježba ipak radi na nekom drugom Unix operacijskom sustavu tada je potrebno pripaziti da postoje dosta velike razlike u korištenju usluga podatkovnog sloja te da pokušaj izrade vježbe na jednom OS-u i demonstracija na nekom drugom mogu biti prilično problematični.

U ovom poglavlju dani su dodatni naputci koji bi trebali olakšati izradu vježbe. Za izradu ove vježbe nužno je poznavanje formata Ethernet okvira, te formata ARP paketa. Ti formati prikazani su na slici 1, a u tablici su dana značenja pojedinih polja. Sjenčana polja kreira mrežna kartica neposredno prije slanja okvira. Iznad svakog polja dana je njegova veličina u oktetima. U zaglavnim datotekama postoje već predefinirane strukture za Ethernet zaglavlje (polja DA, SA i Type) i ARP zaglavlje (polja HRD, PRO, HLN, PLN i OP) te je zgodno koristiti te strukture za definiranje strukture kompletnog paketa. Detalji se mogu vidjeti u datotekama `net/ethernet.h` i `net/if_arp.h`.

Nužno je primjetiti kako polje SHA i SIP uvijek sadrže adresu računala koje šalje zahtijev, a polja THA i TPA adresu računala koje prima zahtijev! Pripaziti također prilikom slanja okvira da neosjenčan dio bude veličine 60 okteta, budući da je minimalna veličina ethernet okvira (bez polja *Preamble* i *SD* 64 okteta). Kako bi se postigla potrebna veličina koristi se popunjavanje oktetima (*byte stuffing*).



Slika 1. Format ARP paketa i pripadajućeg Ethernet okvira

DON'T PANIC

Za izradu ove vježbe pripremljen je virtualni PC koji sadrži sve neophodne komponente za obavljanje programskih vježbi. Na njemu je instaliran RedHat operacijski sustav (verzija 9.0), a mrežna kartica direktno je spojena na mrežnu karticu fizičkog PC računala. Osim razvojnih alata uključene su i referentne stranice koje se mogu pregledavati upotrebom programa `man`. Sljedeće referentne stranice su korisne za ovu vježbu: `socket(2)`, `sendto(2)`, `recvfrom(2)`, `ip(7)`,`packet(7)`,`netdevice(7)`.

Polje	*Vrijednost*	*Značenje*
DA		Ethernet adresa na koju je potrebno ispručiti okvir.
SA		Ethernet adresa sučelja sa kojega je okvir poslan
Type	`ETH_P_ARP`	Tip paketa koji se nalazi u podatkovnom dijelu Ethernet okvira
HRD	`ARPHRD_ETHER`	Tip sklopovske adrese u ARP zahtijevu. U ovom slučaju u pitanju je Ethernet adresa.
PRO	`ETH_P_IP`	Protokol mrežnog sloja čija adresa se pretvara u adresu podatkovnog sloja. U ovom slučaju u pitanju je IP adresa.
HLN	`ETH_ALEN (6)`	Duljina adrese podatkovnog sloja u oktetima
PLN	`4`	Duljina adrese mrežnog sloja u oktetima
OP	`ARPOP_REPLY/ ARPOP_REQUEST`	Polje koje određuje da li ARP paket predstavlja zahtijev ili odgovor
SHA		Ethernet adresa pošiljatelja
SIP		IP adresa pošiljatelja
THA		Ethernet adresa primatelja (u slučaju zahtijeva to je difuzna adresa)
TIP		IP adresa primatelja

Tablica 1. Značenje pojedinih polja na slici 1

Jedan od mogućih redoslijeda za izradu laboratorijske vježbe i način ispitivanja ispravnosti koda je sljedeći:

1. Napraviti osluškivanje mreže. Da li program dobro radi provjeriti upotrebom programa `ping(8)` uz pomoć kojega treba slati pakete računalima na lokalnoj mreži. Pogodno je da računala kojima se pokušava pristupiti naredbom `ping(8)` ne postoje budući da će na taj način stalno biti generirani ARP zahtijevi bez odgovora što olakšava njihovo praćenje. Ako računalo postoji tada nakon jednog ARP zahtijeva računalo pamti odgovor te idući puta ne kada se pokrene naredba `ping` ne šalje ARP zahtijev.

   Obratiti pozornost da je prije početka ispitivanja potrebno aktivirati sučelje u slučaju da ono nije aktivirano. Aktiviranje sučelja opisano je u dodatku koje se odnosi na mrežne naredbe Linux operacijskog sustava.

   Točan način okolina za ispitivanje programa opisan je kasnije u ovom poglavlju.

2. Nakon što je završen modul za osluškivanje mreže pogodno je napraviti modul za postavljanje ARP upita na mrežu. Da li taj modul ispravno šalje pakete na mrežu može se provjeriti korištenjem već završenog modula za osluškivanje mreže ali i uz pomoć već spomenutog `tcpdump` programa.

   Za implementaciju vremenskog ograničenja upotrijebiti funkciju `select(2)`. Primjer kako se ta funkcija koristi dan je u uputstvu koje se dobije naredbom `man`. U danom primjeru koristi se standardni ulaz (opisnik datoteke 0) dok se u ovoj laboratorijskoj vježbi koristi

pristupna točka koju vraća funkcija socket. Shodno tome potrebno je modificirati primjer, tj. na mjesta gdje je ukodirana nula postaviti varijablu u kojoj je upisan broj pristupne točke, a prilikom poziva funkcije `select(2)` umjesto prvog parametra – koji je u primjeru 1 – potrebno je proslijediti vrijednost pristupne točke uvećanu za 1!

3. Konačno, preostalo je napraviti poslužitelj ARP zahtijeva. Za testiranje ovog modula može poslužiti modul za postavljanje upita, tj. postavljanjem upita moramo dobiti adrese navedene u konfiguracijskoj datoteci.

Treba primjetiti kako se dio funkcionalnosti različitih modula preklapa. Tako, primjerice, modul za osluškivanje mreže očekuje da se pojavi upit isto kao što modul za posluživanje čeka na pojavu zahtijeva. Ta preklapanja mogu se iskoristiti kako bi se kod bolje organizirao te načinio efikasnijim i lakšim za održavanje.

Prilikom pozivanja programa specificira se sučelje kroz koje je potrebno slati okvire. Također, traži se da se kao izvorišna adresa (polje SA) postavi ethernet adresa tog sučelja. Kako bi se iz imena sučelja (npr. `eth0`) dobio njegov indeks može se koristiti sljedeći fragment koda:

```
...
#include <sys/ioctl.h>
...

int s;
struct ifreq ifdev;

s = socket(AF_INET, SOCK_DGRAM, 0);
memset(&ifdev, 0, sizeof(struct ifreq));
strncpy(ifdev.ifr_name, imesucelja, IFNAMSIZ);
if (ioctl(s, SIOCGIFINDEX, &ifdev) < 0) {
         /* Potrebno je prijaviti gresku, ne moze se
          * dobiti indeks sucelja (npr. sucelje ne postoji)
          */
}
close (s);
```

Tijekom izvršavanja gornjeg fragmenta C koda pretpostavlja se da je `imesucelja` pokazivač tipa `char` na string koji sadrži ime samog sučelja. Nakon uspješnog izvršavanja u polju `ifdev.ifr_ifindex` nalazi se indeks sučelja.

Kako bi se saznala ethernet adresa sučelja može se koristiti sljedeći fragment koda:

```
...
#include <sys/ioctl.h>
...

char ethaddr[6];

if (ioctl(s, SIOCGIFHWADDR, &ifdev) < 0) {
         /* Greska */
}
memcpy (ethaddr, ifdev.ifr_hwaddr.sa_data, ETH_ALEN);
```

Nakon izvršavanja navedenog fragmenta Ethernet adresa upisana je u varijabli `ethaddr`. Prije korištenja gornjeg fragmenta nužno je upisati ime sučelja kako je to učinjeno sa funkcijom `strncpy(3)` u prethodnom fragmentu koda.

DON'T PANIC

4.1.Okolina za ispitivanje programa

Ispitna okolina ovisi o tome da li se program izrađuje doma ili u fakultetskom laboratoriju.

U slučaju razvoja i ispitivanja u fakultetskom laboratoriju mrežna kartica računala na kojemu se razvija program i obavlja ispitivanje direktno je spojena na mrežnu karticu fizičkog računala, a prema tome i na lokalnu mrežu u fakultetskom laboratoriju. Na toj mreži stalno se javljaju arp upiti što je moguće provjeriti korištenjem programa `tcpdump(8)` kako je to opisano u dodatku. Štoviše, ako nakon pokretanja vlastitog programa ništa nije primljeno dulje vrijeme može se pokrenuti spomenuti program kako bi se provjerilo da li ima arp prometa na mreži.

Želi li se na toj mreži generirati ARP zahtijev može se pod Windows operacijskim sustavom otvoriti komandni prompt, te potom u tom promptu napraviti `ping(8)` na neko računalo koje pripada mreži 161.53.73.0/24. Pogodno je pronaći računalo koje nije aktivno jer će se u tom slučaju stalno generirati ARP zahtijevi.

U slučaju razvoja i ispitivanja programa na računalu koje nije povezano na lokalnu mrežu, odnosno nalazi se na lokalnoj mreži na kojoj su ARP paketi rijetki može se složiti jednostavna virtualna mreža. Slika 2 prikazuje jednu moguću konfiguraciju te mreže.

Slika 2. Okolina za ispitivanje programa za slanje i primanje ARP zahtijeva

Potrebno je osim razvojne okoline raspakirati i jedan usmjernik te povezati razvojnu okolinu i usmjernik preko virtualne ethernet mreže. Potom se na usmjerniku pokrene naredba `ping` na adresu 192.168.0.20 (ili neku drugu nepostojeću IP adresu na toj virtualnoj mreži). Nakon pokretanja naredbe `ping` stalno će na virtualnoj mreži biti generirani ARP zahtijevi koji bi se potom trebali vidjeti sa izrađenim programom.

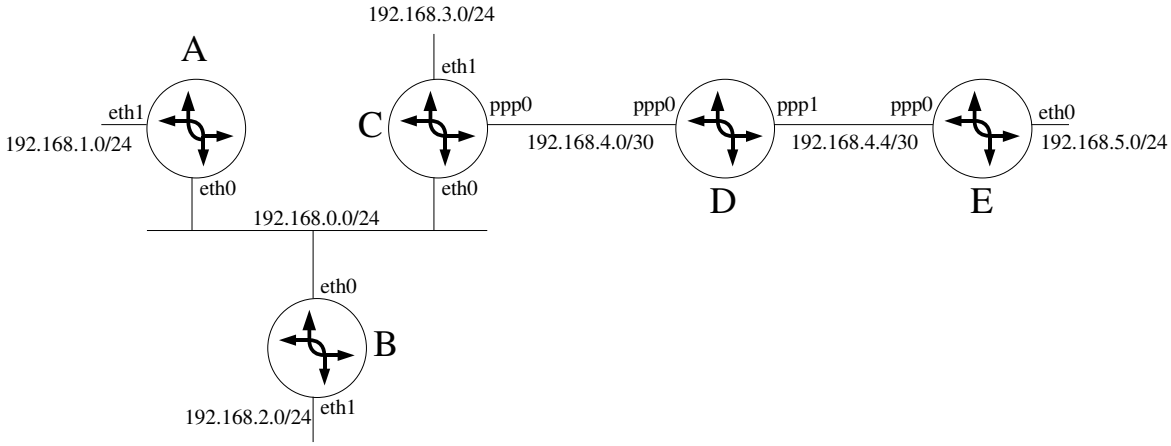
Vježba II – Statičko i dinamičko usmjeravanje

1. Uvod

Cilj ove laboratorijske vježbe je upoznavanje sa principima usmjeravanja na Internetu. U tom smislu tijekom ove vježbe prvo je za zadanu konfiguraciju usmjernika potrebno složiti statičko usmjeravanje, a potom se konfigurira automatsko namještanje svih tablica usmjeravanja. Tijekom vježbe koriste se razni alati za postavljanje parametara usmjeravanja i pronalaženje grešaka u usmjeravanju.

2. Zadatak

Na slici 3 prikazana je konfiguracija 5 usmjernika međusobno spojenih Ethernetom i serijskim vezama. Na slici su označena pojedina sučelja, te su dani rasponi IP adresa koje pripadaju svakoj pojedinoj mreži. U ovoj laboratorijskoj vježbi potrebno je kreirati tu konfiguraciju upotrebom VMWare programa i uzorkom usmjernika, kako je to opisano u uvodnom dijelu priprema za laboratorijske vježbe i u nastavku ove pripreme.



Slika 3. Konfiguracija usmjernika za 2. laboratorijsku vježbu

U prvom dijelu laboratorijske vježbe potrebno je složiti statičke tablice usmjeravanja na svim usmjernicima kako bi se postigla potpuna povezanost računalne mreže. Nakon toga, u drugom dijelu laboratorijske vježbe, potrebno je očistiti tablice usmjeravanja te upotrebom Zebra programa, konfigurirati RIP algoritam usmjeravanja koji će automatski podešavati tablice usmjeravanja.

U oba slučaja potrebno je zaustaviti pojedina sučelja te promatrati kako se mijenjaju tablice usmjeravanja u statičkom i dinamičkom slučaju.

DON'T PANIC

3. Usmjeravanje na Internetu

Internet čini niz računalnih mreža međusobno spojenih usmjernicima. Karakteristika svakog usmjernika je da posjeduje barem dva sučelja – za razliku od računala koja uobičajno imaju samo jedno sučelje. Zadatak usmjernika je da svaki paket koji prime proslijedi (*forward*) na odgovarajući izlaz kako bi taj paket pristigao na odredište zapisano u njegovu zaglavlju. Bitna komponenta tog procesa je tablica prosljeđivanja (*forwarding information base*), tj. tablica koja za svako odredište određuje kako do njega doći. Ta tablica sastoji se od niza zapisa – koje se nazivaju *rute* i smještena je u jezgri operacijskog sustava. Svaka ruta sastoji se od odredišta, te od izlaznog sučelja i sljedećeg čvora. Uz te podatke može se nalaziti još mnoštvo drugih, no oni tijekom ovog jednostavnog razmatranja nisu toliko bitni. Prilikom pretraživanja tablice traži se najbolje, tj. najdulje, preklapanje mrežnog dijela odredišta rute sa odredišnom adresom zapisanom u zaglavlju paketa. Primjerice, neka sljedeća tablica predstavlja dio tablice usmjeravanja u nekom usmjerniku:

Odredište	***Izlazno sučelje***	***Slijedeći čvor***
192.168.1.0/24	eth0	192.168.1.9
192.168.1.0/29	ppp0	192.168.1.2

Na sučelje `eth1` dolazi paket sa odredištem 192.168.1.5, nakon konzultiranja tablice prosljeđivanja taj će paket biti proslijeđen na sučelje `ppp0` do čvora sa IP adresom 192.168.1.2. To je zbog toga što ta ruta ima poklapanje od 29 bita, dok prva ruta ima poklapanje od samo 24 bita. Prilikom usporedbe koristi se bolje, tj. dulje, poklapanje.

U svojoj najjednostavnijoj varijanti usmjernici su računala opće namjene sa uobičajenim operacijskim sustavom i eventualno ponešto specijalizirane programske podrške. U složenijim i zahtijevnijim primjenama usmjernik je specijalno računalo sa dodanim sklopovljem koje mu omogućava vrlo brzo obavljanje usmjerničkih funkcija. Usmjernik ima mnoštvo funkcija koje treba obaviti no tijekom ove laboratorijske vježbe interesira nas isključivo njegova temeljna svrha: usmjeravanje paketa. U laboratorijskoj vježbi koriste se virtualni usmjernici s instaliranim Linux operacijskim sustavom, odgovarajućom programskom podrškom i potrebnim brojem Ethernet i serijskih sučelja. Način na koji se virtualno računalo pokreće, konfigurira, i sl. opisan je u dodatku.

Prilikom konfiguriranja usmjernika nude se dvije mogućnosti. Prva je ručno podešavanje svih ruta. Ta opcija je prihvatljiva ako je mreža mala, te joj se topologija dosta rijetko mijenja. Međutim, u slučaju da je mreža velika – sastoji se od mnoštva ruta – te se dosta često mijenja, tada je bolje koristiti automatsku konfiguraciju usmjernika pomoću *algoritama usmjeravanja*. Algoritmi usmjeravanja ne obavljaju proces proslijeđivanja već je njihova isključiva funkcija razmjena podataka o mreži te izgradnja tablice proslijeđivanja na osnovu tih podataka. Algoritmi usmjeravanja sve primljene podatke o stanju mreže čuvaju u tablici usmjeravanja (*routing information base*) na osnovu koje potom izgrađuju tablicu proslijeđivanja. Tablica usmjeravanja nalazi se izvan jezgre operacijskog sutava. Postoji mnoštvo različitih algoritama usmjeravanja no u ovoj laboratorijskoj vježbi koristi se najjednostavniji – RIP protokol. U današnje vrijeme Internetom dominira OSPF koji ima daleko bolje karakteristike, no nešto je složeniji za konfiguriranje te se zbog toga ne radi na ovim vježbama.

4. Upute za izvršenje laboratorijske vježbe

Prije praktičnog dijela potrebno je napraviti pripremu. Na slici mreže potrebno je označiti kojim virtualnim mrežama pripada koja Ethernet mreža te koje IP adrese će biti dane sučeljima.

Također, potrebno je isplanirati tablice usmjeravanja na svakom usmjerniku.

Nakon pripreme potrebno je kreirati zadanu konfiguraciju, tj. raspakirati potreban broj virtualnih PC-ja te svakog od njih otvoriti u VMWare-u i dati mu odgovarajuće ime. Nakon toga treba uključivati jedan po jedan usmjernik i napraviti osnovno podešavanje njegovih sučelja: davanje IP adresa sučeljima i aktivacija sučelja. To podešavanje treba omogućiti komunikaciju između susjeda. Prilikom tog osnovnog podešavanja koristiti program `ping` za testiranje i pronalaženje grešaka.

Nakon što su sva sučelja inicijalizirana, te su im dodijeljene adrese potrebno je podesiti tablice usmjeravanja. Prilikom sređivanja tablica usmjeravanja ponovo se može koristiti program `ping(8)`, no vrlo koristan je i program `traceroute(8)`. Oba programa opisana su u dodatku.

Nužno je zapamtiti kako Linux kernel inicijalno ne usmjerava pakete. To je mjera predostrožnosti budući da bi u suprotnom moglo doći do zlouporaba. Kako bi se omogućilo usmjeravanje unutar jezgre operacijskog sustava potrebno je izvršiti sljedeću naredbu:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Razmak prije znaka manje je **obavezan** jer u suprotnom ta naredba ima sasvim drugo značenje!

4.1.Dinamičko usmjeravanje

Nakon statičkog usmjeravanja potrebno je u istoj mreži složiti i dinamičko umjeravanje. Prilikom podešavanja dinamičkog usmjeravanja nužno je pokrenuti serijska sučelja na način opisan kod statičke konfiguracije i također je nužno omogućiti usmjeravanje upisivanjem konstante 1 u datoteku `ip_forward`, no više nije potrebno naredbom `ip(8)` podešavati IP adrese. To se sada obavlja kroz konfiguracijske datoteke programa za dinamičko usmjeravanje, tj. podešavanje tablica usmjeravanja.

Za dinamičko podešavanje tablica usmjeravanja koristi se skup programa pod nazivom Zebra, odnosno Quagga. Taj skup programa sastoji se od glavne komponente – programa nazvanog zebra – koji komunicira sa jezgrom operacijskog sustava, te od po jednog programa za svaki algoritam usmjeravanja. U ovoj laboratorijskoj vježbi koristi se `ripd` program koji implementira RIP algoritam usmjeravanja.

Kako bi bilo moguće koristiti dinamičko kreiranje tablica za usmjeravanje potrebno je prvo kreirati odgovarajuće konfiguracijske datoteke – za ove laboratorijske vježbe to su `zebra.conf` i `ripd.conf`, a potom prvo pokrenuti `zebra` program, a nakon toga `ripd` program. Vrlo je bitno da način pokretanja pojedinih programa ide tim redoslijedom. Za obje konfiguracijske datoteke podrazumijeva se da su smještene u direktoriju `/usr/local/etc` no to se može promijeniti korištenjem odgovarajućih opcija programa. Sintaksa konfiguracijskih datoteka dosta je slična sintaksi koja se upotrebljava za konfiguraciju Cisco umjernika, no ipak ima određenih razlika koje proizlaze iz razlika između operacijskim sustavima te mogućnosti koje nude Zebra s jedne strane i Cisco s druge strane. U nastavku je ukratko opisana sintaksa konfiguracijskih datoteka.

Komentar u konfiguracijskim datotekama započinje sa uskličnikom i proteže se do kraja linije. Svaka naredba zauzima jednu liniju. Prazne linije i komentari se ignoriraju. Naredbe se dijele u grupe ili nivoe. Određene naredbe započinju novi nivo i sve što se navodi unutar novog nivoa pobliže određuje naredbu iz višeg nivoa. Kao primjer može se uzeti postavljanje adrese sučelju:

DON'T PANIC

```
interface eth0
         ip address 192.168.2.1/24
```

Tim dvijema naredbama postavlja se adresa sučelju `eth0`. Naredba `interface` započinje novi nivo unutar kojega idu podnaredbe. Nije nužno uvlačenje prilikom navođenja naredbi budući da nema dvosmislenosti.

U konfiguracijsku datoteku za Zebra program postavljaju se parametri kao što su ime usmjernika (`hostname`), sučelja koja usmjernik posjeduje, lozinke itd. Za ovu laboratorijsku vježbu dovoljno će biti postaviti ime usmjernika i adrese sučelja. Ime usmjernika postavlja se naredbom `hostname`. Primjerice sljedeća naredba postavlja usmjerniku ime `RouterU`.

```
hostname RouterU
```

Imena usmjernika postavljaju se kako je to prikazano u prethodnom primjeru. Za svako sučelje potrebno je navesti po jednu `interface` naredbu sa odgovarajućim podnaredbama.

U konfiguracijsku datoteku za RIP protokol usmjeravanja dovoljno je navesti koja sučelja sudjeluju u razmjeni podataka o mreži. To se može obaviti naredbom `network` čiji parametar je sučelje koje treba uključiti. Naredba `network` je podnaredba naredbe `router`, te ju je tako potrebno i pisati. Primjerice, ako želimo uključiti sučelja `ppp2` i `ppp3` tada se to napravi na sljedeći način:

```
router rip
         network ppp2
         network ppp3
```

Prethodni način konfiguracije je dobar za ovu laboratorijsku vježbu, no u praktičnoj primjeni nužno je paziti na sigurnost, tj. kome vjerovati da su podaci koji pristižu o topologiji mreže ispravni, a ne lažirani!

Kada su kreirane konfiguracijske datoteke potrebno je pokrenuti programe. Na uzorku usmjernika danom za laboratorijske vježbe zebra je iskompajlirana u direktoriju `/root/zebra` pa se zebra program pokreće sljedećom naredbom:

```
/root/zebra/zebra/zebra -d
```

dok se rip program pokreće naredbom

```
/root/zerba/ripd/ripd -d
```

Pokrenuti na taj način programi se odmah odspajaju od terminala (*detach*) te se nastavljaju izvršavati u pozadini. Sve dijagnostičke poruke šalju se u sistemski log koji je na predinstaliranom usmjerniku datoteka `/var/log/messages`.

4.2.Opcionalni dodatak zadatku

Studenti koji žele mogu povezati putem serijskog ili ethernet sučelja usmjernike E i B te potom

iskonfigurirati dinamičko usmjeravanje na zadanoj mreži. Taj dodatak se ne boduje, ali je zanimljiv budući da je sada stvorena petlja u mreži. Algoritam usmjeravanja će eliminirati jednu od veza koja čini petlju. U određenih ispada u mreži ta veza se može reaktivirati kako bi se ponovo osigurala povezanost mreže.

DON'T PANIC

Vježba III – DNS i Web poslužitelji

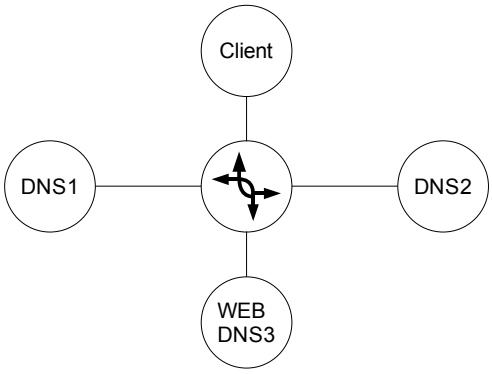
1. Uvod

Internet bi bio beskoristan da se na njemu ne izvršava niz usluga (servisa), primjerice DNS, web, mail i mnoštvo drugih. Te usluge nisu nužne kako bi Internet ispravno funkcionirao, tj. povezivao međusobno udaljena računala, ali su zato nužne kako bi se Internet mogao lakše koristiti i kako bi pri tome bio koristan onima koji ga upotrebljavaju. Cilj ove vježbe je upoznavanje s dva vrlo popularna servisa, DNS i web servisom. Tijekom vježbe potrebno je iskonfigurirati poslužitelje koji će obavljati navedene funkcije, tj. pretvarati simbolička imena u IP adrese i obratno, te posluživati web stranice.

U nastavku teksta pojmovi *servis* i *poslužitelj* koriste se paralelno i oba označavaju program koji se izvršava na nekom računalu i obrađuje pristigle DNS zahtijeve, tj. u ovim laboratorijskim vježbama zahtijeve za konverzijom simboličkih adresa u IP adrese. U pojedinim slučajevima pojam *poslužitelj* označavati će i računalo na kojemu se servis izvršava ali to će tada biti jasno iz konteksta, ili u slučaju da nije jasno iz konteksta, bit će posebno naglašeno.

2. Zadatak

Topologija mreže na kojoj je potrebno realizirati ovu laboratorijsku vježbu prikazana je na slici 4. Cilj laboratorijske vježbe je konfiguriranje DNS poslužitelja s mrežnim parametrima definiranim u tablici 2.



Slika 4. Slika uz zadatak 3. laboratorijske vježbe

DNS poslužitelj	IP adresa	Domena	Ime u bazi	IP adresa
DNS1	192.168.0.2	. (vršna domena)	ns	192.168.0.2
			www	192.168.0.5
DNS2	192.168.1.2	.com	ns	192.168.1.2
			www	192.168.1.5
DNS3	192.168.2.2	amazon.com	gw	192.168.2.1
			ns	192.168.2.2
			www	192.168.2.2
			www2	alias na www
Client	192.168.3.2			

Tablica 2. Parametri mreže prikazane na slici 4

DON'T PANIC

Konfiguriranje povezanosti (usmjeravanja) mreže obaviti korištenjem podrazumijevanih ruta (*default route*). Način dodavanja podrazumijevanih ruta opisan je u dodatku. Nakon što je DNS konfiguriran, te ispravno radi, potrebno je konfigurirati i web poslužitelj. Web poslužitelj mora odgovarati na zahtijeve za stranicama koje se nalaze na adresama www i www2.

Za računalo označeno s *client* koristiti sliku poslužitelja.

3. Konfiguriranje DNS i Web servisa

Način konfiguriranja poslužitelja ovisi o upotrebljenoj implementaciji i specifičnostima upotrebljenog operacijskog sustava. U ovom poglavlju opisan je način na koji se upravlja poslužiteljima na Mandrake distribuciji Linux operacijskog sustava. Ta metoda vrijedi i za RedHat distribuciju, iako je uz određene razlike postupak sličan i na svim ostalim distribucijama. Nakon toga opisan je način konfiguriranja BIND i Apache programa koji su najčešće korištene implementacije za DNS, odnosno Web servise.

3.1.Pokretanje servisa

Pokretanje i zaustavljanje servisa ovisi o operacijskom sustavu i distribuciji na kojemu se servisi trebaju izvršavati. Budući da se u ovim laboratorijskim vježbama koristi Mandrake distribucija, objašnjenja su vezana uz nju. Za pokretanje servisa koristi se naredba `service`. Parametar toj naredbi je servis koji želimo zaustaviti, pokrenuti i slično. Npr., za pokretanje web poslužitelja – čije ime servisa je `httpd` – treba se koristi sljedeća naredba:

```
service httpd start
```

Za zaustavljanje DNS servisa – čije ime je `named` – koristi se sljedeća naredba:

```
service named stop
```

Osim zaustavljanja i pokretanja moguće je upotrebom riječi `restart` zaustaviti servis ukoliko je bio pokrenut te ga ponovo pokrenuti. Prilikom pokretanja servisa, tijekom njegova izvršavanja i tijekom zaustavljanja, sve dijagnostičke poruke šalju se u datoteku `/var/log/messages`. To je tekstualna datoteka čiji se sadržaj može pregledavati bilo kojim alatom koji se inače koristi za uređivanje i pregledavanje tekstualnih datoteka. Ako je potrebno vidjeti samo kraj te datoteke, tada je to moguće koristeći sljedeću naredbu:

```
tail /var/log/messages
```

Naredba `tail(1)` bez ikakvih dodatnih parametara prikazuje zadnjih 10 linija datoteke. U slučaju da je potrebno prikazati primjerice zadnjih dvadeset, poziva se na sljedeći način:

```
tail -20 /var/log/messages
```

3.2.DNS

Najčešće korištena implementacija DNS-a je ISC-ov BIND, trenutno u verziji 9.3.X. Taj se program koristi i u ovoj laboratorijskoj vježbi. BIND prilikom pokretanja čita konfiguracijsku datoteku `/etc/named.conf` u kojoj su definirani svi parametri nužni za rad DNS poslužitelja. Servis koji se pokreće zove se `named`! Konfigurararanje DNS-a sastoji se od sljedećih koraka:

DON'T PANIC

1. Modifikacija datoteke `named.conf`
2. Izrada baze koja služi za prevođenje imena u IP adrese
3. Izrada baze koja služi za prevođenje IP adresa u imena računala

3.2.1. Konfiguriranje DNS usluge

DNS usluge se izvršavaju na točno određenim računalima i obično se uz jedan primarni DNS poslužitelj po domeni izvršava i jedan ili više sekundarnih odnosno pomoćnih poslužitelja na rezervnim poslužiteljima. Na taj se način osigurava dostupnost cjelokupne domene i rasterećenost glavnog poslužitelja. Prikaz potpunog imena računala je u obliku stabla pri čemu se korijen tog stabla – koji se nalazi na desnoj strani imena – crta na vrhu. Upit za određenim računalom uglavnom označava šetnju tim stablom od vrha prema zadanoj domeni, odn. računalu, no taj kompleksan postupak obavljaju sami DNS poslužitelji bez znanja aplikacija koje koriste funkcije za translaciju adresa.

Konkretno, za zadanu konfiguraciju koju je potrebno implementirati, postavljanjem upita na računalu *client* za adresom "www.amazon.com." obavljaju se sljedeća tri upita:

1. Šalje se upit korijenskom poslužitelju u kojemu se traži IP adresa zadanog računala. Budući da korijenski DNS poslužitelj nema podatke o tom računalu, ali ima podatke o računalu koje opslužuje com domenu upućuje računalo client na njega i šalje odgovarajuću IP adresu, koja je u ovoj laboratorijskoj vježbi 192.168.1.2.
2. Ponovo se šalje upit, ovaj puta poslužitelju "com." domene u kojemu se traži IP adresa računala "www.amazon.com.". Rezultat tog upita je IP adresa DNS poslužitelja za domenu "amazon.com." čija IP adresa je 192.168.2.2.
3. Konačno, šalje se upit DNS poslužitelju 192.168.2.2 u kojemu se traži IP adresa računala "www.amazon.com.". Rezultat tog upita je IP adresa 192.168.2.3.

Kao što je već rečeno, datoteka `named.conf` definira parametre poslužitelja i zone koje poslužitelj opslužuje. Parametri poslužitelja definiraju se unutar bloka `options`. U ovoj laboratorijskoj vježbi unutar tog bloka potrebno je samo definirati direktorij unutar kojega se nalaze sve ostale datoteke s dodatnim parametrima konfiguracije. Sintaksa je sljedeća:

```
options {
        directory "<direktorij>";
}
```

Mjesto gdje su smještene dodatne datoteke ovisi o distribuciji Linux-a, odnosno o operacijskom sustavu koji se koristi. U slučaju RedHat distribucije, kao i niza drugih, to je direktorij `/var/named`.

Nakon opcija, definiraju se zone koje poslužitelj opslužuje. Zona se definira sljedećim blokom naredbi:

```
zone <domena> {
        type [ master | slave | hint ];
        file "<datoteka_sa_zapisima>";
}
```

Umjesto `<domena>` potrebno je navesti ime domene čiji poslužitelj je potrebno konfigurirati. Ime

mora biti unutar navodnika i ne smije završavati s točkom. Za svaku zonu koju opslužuje poslužitelj, definira se vrsta poslužitelja koristeći konfiguracijsku direktivu `type`. Mogući parametri te direktive su:

- `master`

  Ova riječ označava da je poslužitelj za navedenu zonu primarni, te da su svi odgovori na upite koje daje autoritativni. U ovoj laboratorijskoj vježbi potrebno je konfigurirati isključivo primarne poslužitelje.

- `slave`

  Ova riječ označava da je za zadanu zonu poslužitelj sekundarni, odnosno pomoćni DNS poslužitelj. Pomoćni poslužitelji prenose podatke o zoni od glavnog poslužitelja, a služe kako bi se cjelokupni sustav učinio otporniji na razne pogreške.

- `hint`

  Ovo je posebna zona čija je domena "`.`". Maksimalno jedna takva zona definirana je u svakom DNS poslužitelju – osim, naravno, u korijenim poslužiteljima. Ta zona u svojoj datoteci sadrži popis glavnih DNS poslužitelja na Internetu, tzv. `root` poslužitelja. Nju poslužitelj koristi prilikom rekurzivnih poziva i potrebna je isključivo ako je poslužitelj na Internetu. Ako poslužitelj nije na Internetu, tj. opslužuje imena unutar Intraneta tada se ta vrsta zone ne navodi u konfiguracijskoj datoteci.

Konačno, direktivom `file` definira se datoteka unutar koje su upisani svi podaci o zoni, tj. sva imena, aliasi, itd. Ako je navedeno ime koje ne počinje znakom `/` tada je to ime relativno u odnosu na direktorij naveden u `options` bloku, u suprotnom je apsolutno ime u datotečnom sustavu.

Format konfiguracijske datoteke ovisi o tipu koji je deklariran direktivom `type`. U slučaju tipa `hint` format je jednostavniji i slijedi sljedeći uzorak:

```
.                       3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A     198.41.0.4
```

U prvoj liniji navedena je domena na koju se odnosi zapis, potom dolazi trajanje zapisa u sekundama, klasa kojoj zapis pripada (IN), tip zapisa i ime DNS poslužitelja za tu domenu. Druga linija definira IP adresu tog poslužitelja, a sastoji se od imena, trajanja zapisa, oznake da se radi o adresi (`A`) i same IP adrese. Dani primjer preuzet je s Interneta i predstavlja važeći zapis, tj. navedeno računalo `A.ROOT-SERVERS.NET.` stvarno je jedno od glavnih DNS poslužitelja Interneta. U slučaju laboratorijske vježbe, kada se radi o laboratorijskim uvijetima, ime nije bitno i može se staviti proizvoljno ime, ali bitna je IP adresa koja mora odgovarati IP adresi glavnog (root) poslužitelja zadanoj u tablici u sklopu definicije zadatka.

Kao što je rečeno, detaljni podaci o zoni kada se radi o tipu `master` nalaze se u datoteci definiranoj direktivom `file`. U toj datoteci svaka linija koja ne počinje od prve kolone mora započeti tabulatorom! Komentari započinju znakom # ili `;` i ignoriraju se do kraja linije.

Datoteka se sastoji od niza zapisa koji se označavaju s RR (od *Resource Record*). Na početku datoteke nalaze se osnovni zapis koji definira neke opće parametre zone i naziva se SOA zapis, ili SOA RR. Taj zapis prikazan je na sljedećem ispisu:

DON'T PANIC

```
$TTL 86400
@       IN      SOA     <ns_server>       <mail_adresa> (
                        2 ;       serial
                        28800 ;   refresh
                        7200 ;    retry
                        604800 ;  expire
                        86400 ;   ttl
                        )

        IN      NS      <ns_server>
```

Prva linija definira podrazumijevano vrijeme u sekundama koliko je dopušteno privremeno pohranjivanje (*caching*) pojedinog zapisa zone i ona ne pripada SOA zapisu, ali BIND od verzije 9 inzistira da se ta direktiva mora pojaviti prva. Za svaki zapis moguće je taj parametar naknadno promijeniti. U ovom slučaju definirano je vrijeme od 24 sata.

Potom dolazi SOA zapis. Prvi je znak @ koji se zamjenjuje imenom zone iz datoteke `/etc/named.conf`. Potom dolazi klasa zapisa, IN, i tip zapisa, SOA. Nakon toga definira se ime poslužitelja domene. To ime mora biti puni naziv i preporučljivo je da završava točkom. Primjer takvog imena je "`ns.amazon.com.`". Ime `ns` je skraćenica od *name server* i može biti proizvoljno, ali je običaj da se uzima to ime, kao što je običaj da web poslužiteljima ime započinje sa www. Kasnije u datoteci mora biti definirana i IP adresa koja odgovara tom imenu.

Nakon imena poslužitelja na koju se odnosi datoteka (*`ns_server`*) dolazi mail adresa osobe zadužene za administraciju DNS poslužitelja. U toj mail adresi potrebno je znak `@` zamijeniti sa točkom. Primjerice, ako je mail adresa `dns-admin@domena.org`, tada je potrebno napisati `dns-admin.domena.org`. U zagradi je naveden serijski broj i vremenski parametri zone. Prilikom promjene zone, tj. dodavanja, uklanjanja ili promjene nekog od imena nužno je promijeniti (inkrementirati) serijski broj kako bi ostali poslužitelji mogli vidjeti da je došlo do promjene u zoni! Zadnja linija ispisa ponovno definira ime glavnog DNS poslužitelja domene. To ime mora kasnije imati zapis koji ga povezuje s konkretnom IP adresom (A zapis)!

Ostali zapisi sadrže podatke o računalima. Postoji više različitih tipova zapisa osim već spomenutog SOA, dok ćemo ovdje spomenuti samo neke češće korištene, a za ilustraciju zapisa koje ćemo opisati koristiti ćemo sljedeći isječak iz konfiguracijske datoteke BIND-a:

```
gandalf  IN      A       161.53.65.11
         IN      AAAA    5f1b:df00:ce3e:e200:0020:0800:2078:e3e3
         IN      MX      5       mailhost.zemris.fer.hr.
         IN      MX      10      mailhost2.zemris.fer.hr.

www      CNAME           gandalf.zemris.fer.hr
```

U prvoj koloni nalazi se ime računala unutar domene (domena je definirana na drugom mjestu te u gornjem isječku nije navedena). Kada ime ne završava točkom na to ime se dodaje ime domene. Potom dolazi klasa zapisa. U ovom slučaju klasa je IN i to je najčešće korištena klasa, dok se ostale koriste u nekim specijalnim situacijama i relativno su rijetke. Nakon klase navodi se tip zapisa. Za klasu IN neki od definiranih tipova zapisa su:

- **A zapis**

  A zapis sadrži IP adresu računala.

- **AAAA zapis**

  AAAA zapis sadrži IPv6 adresu računala. To je stari oblik upisivanja IPv6 adrese. Novi oblik je pomoću A6 zapisa.

- **PTR zapis**

  PTR zapis (koji se ne nalazi u prethodnom primjeru) služi za pretvaranje IP adrese u ime računala.

- **MX zapis**

  MX zapisi definiraju računalo zaduženo za primanje mail-a za određenu domenu ili računalo. U gornjem primjeru primanje mail-a upućeno računalu gandalf obavlja računalo s imenom mailhost.zemris.fer.hr, ili računalo mailhost2.zemris.fer.hr. Brojka neposredno prije imena računala određuje prioritet, što je manji broj to je viši prioritet.

- **CNAME zapis**

  Ovaj zapis predstavlja alias i najčešće se upotrebljava za označavanje ftp i web servera. U gornjem primjeru www.zemris.fer.hr je alias na računalo gandalf.zemris.fer.hr.

Proslijeđivanje upita s poslužitelja za domenu na drugog poslužitelja na poddomenu obavlja se kombiniranom upotrebom NS i A zapisa. Primjerice, da bi poslužitelj com domene proslijedio sve upite za amazon.com domenu poslužitelju ns.amazon.com s IP adresom 192.168.2.2, u datoteku s definicijom com zone potrebno je upisati sljedeće dvije linije:

```
amazon.com.        IN        NS        ns.amazon.com.
ns.amazon.com.     IN        A         192.168.2.2
```

3.2.2. Konfiguriranje računala client

Računalo *client* treba podesiti tako da obavlja funkciju tzv. *caching nameserver*-a. To drugim riječima znači da se na tom računalu (*client*) izvršava DNS poslužitelj, ali to nije glavni poslužitelj ni za jednu domenu, niti je pomoćni poslužitelj nekog glavnog poslužitelja. Takvi poslužitelji u svojoj konfiguracijskoj datoteci `named.conf` sadrže samo zonu "." tipa `hint`. Osim te zone mogu sadržavati i zonu za skup IP adresa 127.0.0.0/8.

Njihova prednost je što upite koje obave privremeno pohranjuju, te u slučaju ponovnog upita za istom adresom mogu odmah odgovoriti, a u određenom broju slučajeva mogu brže odgovoriti jer određen broj upita ne ponavljaju.

Računalu *client*, prema tome, treba podesiti datoteku s korijenskim poslužiteljima (datoteka čije je ime definirano unutar zone "." tipa `hint`) i u nju upisati poslužitelj DNS1 (čija je IP adresa 192.168.0.2). Nadalje, u datoteku `/etc/resolv.conf` treba postaviti da DNS poslužitelj ima adresu 127.0.0.1. Nakon toga, potrebno je pokrenuti sam poslužitelj.

3.2.3. Ispitivanje DNS poslužitelja

Nakon što su modificirane i kreirane sve potrebne konfiguracijske datoteke zone potrebno je provjeriti njihovu sintaksnu ispravnost. Za to se mogu koristiti programi `named-checkconf` i `named-checkzone`. `named-checkconf` ne traži nikakve parametre prilikom poziva i provjerava ispravnost glavne konfiguracijske datoteke DNS poslužitelja (`/etc/named.conf`). `named-checkzone` provjerava ispravnost datoteke koja definira zonu i kao parametre traži ime zone i datoteku u kojoj je zona definirana. Ako oba programa nakon pokretanja ne prijave nikakvu grešku tada se može pokrenuti DNS poslužitelj, u suprotnom potrebno je ispraviti sve

DON'T PANIC

pogreške koje ta dva programa prijave.

Nakon pokretanja DNS poslužitelja preporučljivo je provjeriti što je upisano u sistemski log (`/var/log/messages`) te u slučaju da su prijavljene kakve greške potrebno ih je ispraviti.

U idućem koraku potrebno je ispitati da li DNS poslužitelj ispravno odgovara na upite za imenima unutar zone. Primjerice, ako unutar domene postoji računalo s imenom www, tada je potrebno – korištenjem programa `nslookup`, `host` ili `dig` – pitati poslužitelj za to ime. Ako je poslužitelj dobro konfiguriran tada treba vratiti odgovarajuću IP adresu.

Konačno, da bi proslijeđivanje domena ispravno radilo nužno je da svaki poslužitelj bude "svjestan" kako je on glavni poslužitelj za pojedinu domenu. Primjerice, izvršavanjem sljedećeg upita na DNS poslužitelju za com domenu (DNS2):

```
nslookup -type=ns com.
```

moramo kao odgovor dobiti ime ns.com. Tek potom može se konfigurirati i ispitati proslijeđivanje upita između domena.

3.3.WEB

Kao Web poslužitelj upotrebljava se Apache koji je, kao i BIND, najpopularnija implementacija na Internetu, ali u kategoriji web poslužitelja. Konfiguracija ovog poslužitelja obavlja se uređivanjem datoteke `/etc/httpd/conf/httpd.conf` koju poslužitelj čita prilikom pokretanja ili kada mu se pošalje odgovarajući signal.

Konfiguracijska datoteka podijeljena je u dva dijela. Prvi dio je globalni, dok drugi dio sadrži definiciju *virtualnih poslužitelja*. Postoje dvije vrste virtualnih poslužitelja, od kojih su u ovoj laboratorijskoj vježbi zanimljivi virtualni poslužitelji bazirani na imenu. Ukratko, na jednoj IP adresi moguće je držati više različitih virtualnih poslužitelja. Primjerice, *www.zemris.fer.hr* je virtualni poslužitelj koji se nalazi na istoj IP adresi kao i *sigurnost.zemris.fer.hr*. Kada klijent zatraži neku stranicu s tih poslužitelja on odmah definira i poslužitelj na kojemu se ta stranica nalazi. Virtualni poslužitelji su opcionalni i ako nisu definirani tada Apache koristi ime računala i parametre globalne konfiguracije, tj. na IP adresi je definiran isključivo jedan virtualni poslužitelj.

Virtualni poslužitelji u konfiguracijskoj datoteci koja dolazi s Apache web poslužiteljom definirani su pri kraju. Moguće je da u određenim situacijama budu smješteni u zasebne datoteke te se potom u glavnu konfiguracijsku datoteku uključuju putem `include` direktive. Točan način ovisi o distribuciji. Kako bi se konfigurirali virtualni poslužitelji nužno je prvo definirati IP adresu i pristup na kojemu će se ti virtualni poslužitelji nalaziti. To se obavlja upotrebom direktive `NameVirtualHost`, primjerice:

```
NameVirtualHost 161.53.65.11:8080
```

Ako se ne navede pristup (`:8080`) tada se podrazumijeva pristup 80 koji je rezerviran za web poslužitelje.

Nakon što je definirana IP adresa na kojoj će se nalaziti virtualni poslužitelji potrebno je definirati svaki željeni poslužitelj. Općenito se to postiže na sljedeći način:

```
<VirtualHost 161.53.65.11:8080>
    ServerAdmin www@zemris.fer.hr
    DocumentRoot /home/www/primjer.zemris.fer.hr/htdocs
    ServerName primjer.zemris.fer.hr
    ErrorLog /home/www/primjer.zemris.fer.hr/logs/error_log
    CustomLog /home/www/primjer.zemris.fer.hr/logs/access_log common
</VirtualHost>
```

Gornjim blokom konfiguracijskih datoteka definirani su sljedeći parametri:

- Mail adresa koja će se pojavljivati na stranicama koje Apache web poslužitelj automatski generira je `www@zemris.fer.hr`.
- Sve datoteke koje pripadaju virtualnom poslužitelju smještene su u direktorij "`/home/www/primjer.zemris.fer.hr/htdocs`".
- Ime web poslužitelja putem kojega mu pristupaju klijenti je `primjer.zemris.fer.hr`.
- Datoteka u koju će web poslužitelj upisivati sve greške vezane uz taj poslužitelj je "`/home/www/primjer.zemris.fer.hr/logs/error_log`".
- Datoteka u koju se upisuju svi obrađeni zahtijevi, informativne poruke i slično, dakle sve osim grešaka, je "`/home/www/primjer.zemris.fer.hr/logs/access_log`".

Uz direktive navedene u prethodnom primjeru kada se u web klijentu upiše sljedeći URL:

```
http://primjer.zemris.fer.hr/test/primjer.html
```

poslužitelj će u datotečnom sustavu tražiti sljedeću datoteku:

```
/home/www/primjer.zemris.fer.hr/htdocs/test/primjer.html
```

Ako ne pronađe tu datoteku zapisati će odgovarajuću informaciju u `error_log`, a u suprotnom u `acccess_log`.

Da bi se mogli koristiti virtualni poslužitelji nužno je imati ispravno podešen DNS poslužitelj, tj. DNS mora dati ispravnu IP adresu za traženo ime. Druga mogućnost je da se na lokalnom računalu i poslužitelju u `/etc/hosts` datoteci upiše željeno ime i IP adresa. Tim načinom će virtualni poslužitelj biti dostupan isključivo s računala na kojemu je modificirana `/etc/hosts` datoteka, odnosno s poslužitelja. Ta datoteka ima ekvivalent i na Windows operacijskom sustavu, ali se zove `lmhosts`. Njen točan smještaj ovisi o verziji operacijskog sustava.

Prilikom kreiranja virtualnog poslužitelja preporučljivo je slijediti određenu hijerarhiju direktorija. Recimo, svaki virtualni poslužitelj ima svoj direktorij sa sljedećim poddirektorijima:

- `logs` – direktorij u koji su smještene log datoteke
- `htdocs` – direktorij unutar kojega se nalaze dokumenti i skripte

DON'T PANIC

- `cgi-bin` – direktorij unutar kojega su smještene tzv. CGI skripte koje se u ovim laboratorijskim vježbama ne koriste.

3.3.1. Ispitivanje ispravnosti Web poslužitelja

Nakon što je odgovarajuće modificirana konfiguracijska datoteka web poslužitelja potrebno je provjeriti njenu sintaksnu ispravnost. To obavlja sljedeća naredba:

```
apachectl configtest
```

Svaku pogrešku koju ta naredba detektira u konfiguracijskoj datoteci ispisuje na standardni izlaz. Osim sintaksnih pogrešaka ta naredba je u stanju otkriti i neke semantičke greške. Svaku grešku je potrebno otkloniti prije pokretanja poslužitelja.

U trenutku kada je konfiguracijska datoteka bez sintaksnih pogrešaka može se pokrenuti poslužitelj kako je opisano u uvodnom dijelu ove vježbe. Prilikom pokretanja web poslužitelja sve oktrivene greške upisuju se u globalnu datoteku s pogreškama (globalna direktiva `ErrorLog` u konfiguracijskoj datoteci), a greške vezane uz virtualni poslužitelj u datoteku za greške virtualnog poslužitelja (direktiva `ErrorLog` unutar `VirutalHost` sekcije). Prema tome, nakon pokretanja poslužitelja potrebno je provjeriti da li ima kakvih informacija u tim datotekama.

Zadnja provjera sastoji se od zahtijeva web poslužitelju za stranicama na odgovarajućim poslužiteljima. To je moguće obaviti na dva načina. Prvi način je korištenjem programa `lynx`. `lynx` je tekstualni web preglednik. Primjerice, da bi zatražili index.html datoteku sa poslužitelja www2.amazon.com koristimo:

```
lynx http://www2.amazon.com/index.html
```

Ako se dobije očekivana datoteka tada je web poslužitelj ispravno konfiguriran. U suprotnom, informaciju o mogućim greškama u konfiguraciji potrebno je potražiti u log datotekama za pogreške.

4. Upute za izvršenje laboratorijske vježbe

Preporučeni redoslijed izvršavanja laboratorijske vježbe je sljedeći:

1. Kreirati zadanu mrežnu konfiguraciju u VMWare-u.

2. Podesiti statičko usmjeravanje kako bi se postigla potpuna povezanost mreže.

3. Iskonfigurirati i ispitati DNS poslužitelje. Tijekom konfiguriranja DNS poslužitelja kreće se od korijenskog poslužitelja. Svi ostali DNS poslužitelji imaju informaciju o tom korijenskom DNS-u, o vlastitoj zoni i o DNS poslužiteljima koji se nalaze neposredno ispod njih u hijerarhiji.

   Prilikom definiranja nove zone zgodno je uzeti kao predložak zonu definiranu za localhost domenu, tj. za IP adrese 127.0.0.1/8.

4. Iskonfigurirati i ispitati Web poslužitelj.

Vježba IV – Sigurnost računalnih sustava

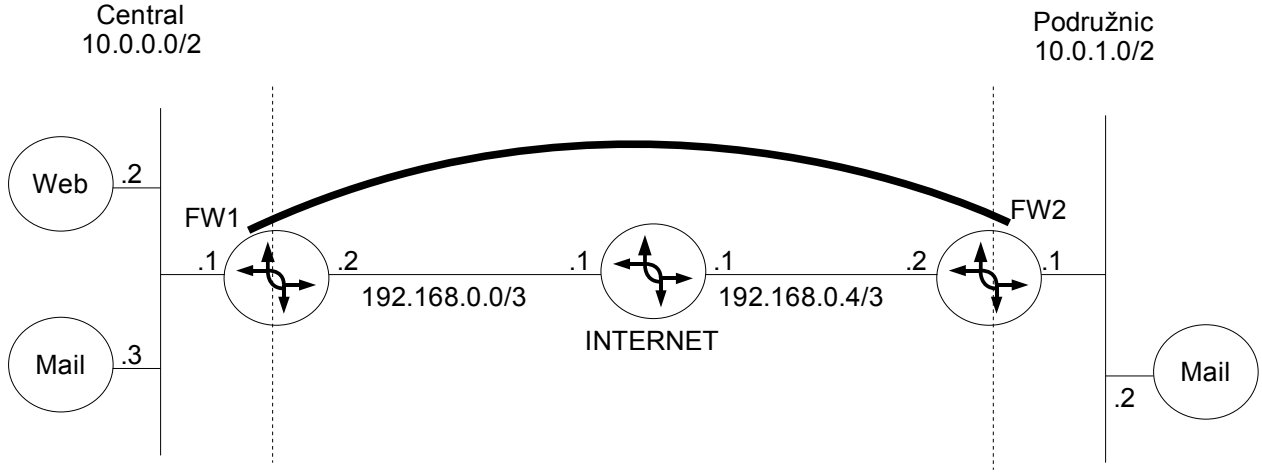
1. Uvod

Sigurnost računalnih sustava kompleksna je i vrlo bitna – i vrlo često ignorirana – tema koja uključuje niz različitih metoda i tehnika. Potpunu sigurnost računalnog sustava i mreže nije moguće postići, no njen sastavni element je sigurnost na svim razinama ISO/OSI mrežnog sustava. Kako pitanje računalne sigurnosti može biti tema barem jednog cijelog predmeta ovdje su samo navedeni neki osnovni elementi. Laboratorijska vježba služi za upoznavanje s *firewall* uređajima i virtualnim privatnim mrežama. Također, tijekom vježbe upotrebljavaju se neki osnovni alati za konfiguriranje.

Temelj sigurnosti neke organizacije čini politika (*policy*) na osnovu koje se ljudi ponašaju, a također se konfiguriraju mrežna i računalna infrastruktura. Politika nije ništa drugo do niz pravila sastavljenih s ciljem očuvanja integriteta svih informacija unutar organizacije.

2. Zadatak

Tijekom laboratorijske vježbe potrebno je načiniti konfiguraciju prikazanu na slici 5. Na toj slici pretpostavlja se da postoji poduzeće koje ima centralu i jednu podružnicu. Centrala i podružnica spojene su putem Interneta virtualnom privatnom vezom, koja je na slici prikazana podebljanom linijom. Internet je modeliran jednim usmjernikom. Na slici su s FW1 i FW2 označeni usmjernici koji povezuju centralu i podružnicu s Internetom, ali ti usmjernici istovremeno imaju i funkciju firewall-a. Osim usmjernika prikazani su i bitni servisi u privatnoj mreži – web i mail poslužitelji. Na slici su označene i IP adrese koje pripadaju pojedinoj mreži, te adrese sučelja. Za ovu laboratorijsku vježbu treba pretpostaviti da su 10.0.0.0/16 privatne IP adrese koje se ne smiju pojaviti na Internetu, dok su sve ostale IP adrese, uključivši i 192.168.0.0/16 važeće Internet adrese. Fiktivno poduzeće, čija mrežna topologija je prikazana na slici, dobilo je za pristup Internetu adrese 192.168.0.0/28.



Slika 5. Mreža za konfiguriranje firewall i VPN-a

Konfiguracija na slici nije potpuna budući da nedostaje primjerice DNS poslužitelj, ali dovoljna je da se prikažu neki osnovni principi slaganja virtualnih mreža i podešavanja firewall uređaja. Tijekom vježbe nije potrebno podešavati Web i mail poslužitelje već se njihov rad emulira. Također, u slučaju nedostatka resursa na računalu domaćinu virtualna računala Web i Mail mogu se nadomjestiti jednim virtualnim računalom.

DON'T PANIC

Cilj laboratorijske vježbe je konfigurirati usmjernike kako bi se postigla povezanost mreže kako je to definirano zadatkom, a potom je potrebno konfigurirati firewall na usmjernicima kako bi se postigao sljedeći niz jednostavnih pravila koja čine politiku poduzeća:

1. S Interneta je moguć pristup isključivo web poslužitelju i mail poslužitelju u centrali.
2. Interni mail poslužitelj može pristupati svim mail poslužiteljima na internetu.
3. Sva ostala računala unutar intraneta nedostupna su s Interneta.
4. Zaposleni mogu na Internetu koristiti isključivo Web.
5. Zaposlenima nije dopušten pristup web poslužiteljima čije adrese započinju sa 192.168.12.0/24.

3. Upute za izradu vježbe

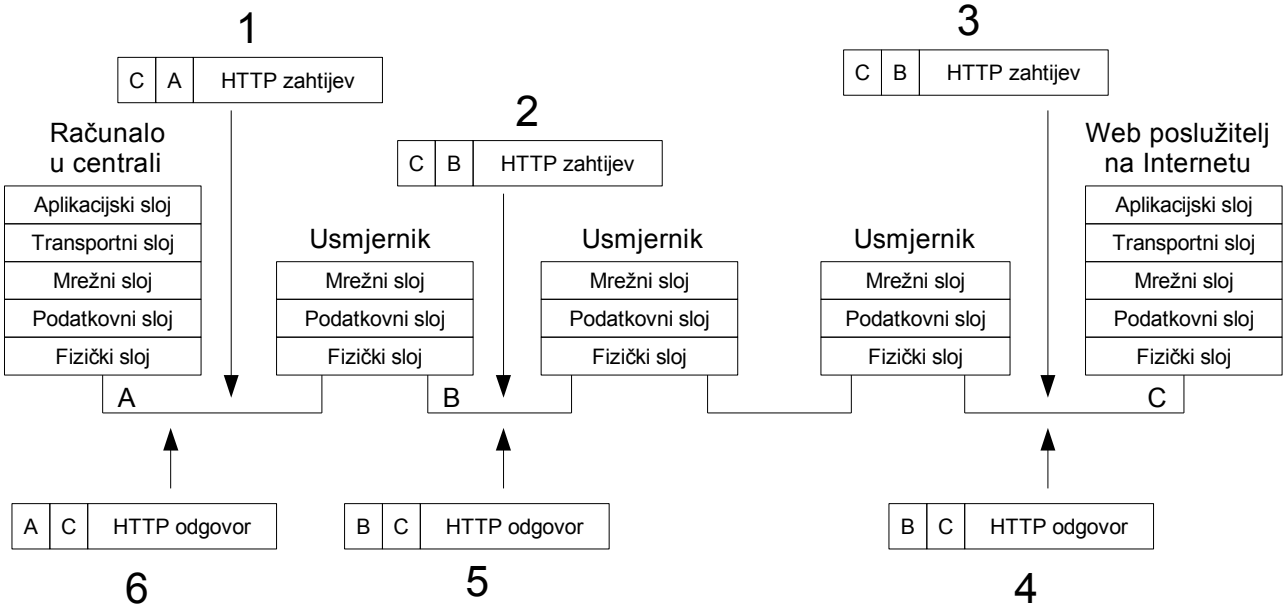
Na slici su već označene sve IP adrese sučelja i mreža. Treba primjetiti kako je poduzeće dobilo na raspolaganje adrese s mrežnim prefiksom 192.168.0.0/28, tj. sveukupno 14 IP adresa, ali da se prilikom korištenja tih adresa one razbijaju na manje raspone. Razlog tome je efikasno korištenje dodjeljenih mrežnih adresa, tj. nepotrebno je koristiti mrežnu masku koja omogućava više IP adresa od potrebnog broja – a na vezama od točke-do-točke (*point-to-point*) dovoljne su dvije IP adrese. Dodatna prednost tog pristupa je u mogućnosti dodavanja još dvije podružnice bez potrebe da se traži novi blok IP adresa.

Tijekom podešavanja prikazane konfiguracije pretpostavit ćemo da su web i mail poslužitelji s Interneta dostupni na IP adresi 192.168.0.1. U nastavku teksta objašnjeni su pojmovi nužni za obavljanje ove vježbe.

Prvi problem koji se javlja prilikom podešavanja konfiguracije prikazane na slici 5 je činjenica kako se unutar centrale može nalaziti do 254 računala i mrežna uređaja koja mogu imati pristup Internetu prema politici poduzeća. No, centrala poduzeća ima dodjeljen blok od 14 IP adresa što bi značilo da samo 14 računala može pristupiti Internetu – i to u nekom vrlo idealnom slučaju. (Prisjetimo se da je za normalno komuniciranje na Internetu nužno imati važeću IP adresu). Jedno moguće rješenje je korištenje aplikacijskog proxy-ja, no u ovoj vježbi koristiti ćemo rješenje na mrežnom sloju, tj. NAT, odnosno njegovu modificiranu varijantu koja koristi dostupne informacije iz viših protokola (*masquerading*). NAT je skraćenica od *Network Address Translation* i označava metodu po kojoj se jedna (ili više) važećih IP adresa koristi za pristup Internetu sa Inraneta na kojemu se koriste privatne IP adrese. Slika 6 će poslužiti za objašnjenje osnovne ideje. Na slici je naznačeno jedno računalo unutar privatne mreže, usmjernik koji povezuje privatnu mrežu s Internetom, potom neki usmjernik unutar samog Interneta i konačno umjernik koji se nalazi na lokalnoj mreži kao i web poslužitelj. Umjesto IP adresa, radi preglednosti, koriste se velika slova abecede, a također nisu označena sva sučelja već samo ona bitna za rad NAT-a u ovom konkretnom slučaju. Slovo A označava privatnu IP adresu koja se ne smije pojaviti na Internetu, dok slova B i C označavaju važeće adrese na Internetu. Radi lakšeg praćenja svi generirani mrežni paketi označeni su brojkom. Sljedeći je redoslijed po kojemu se odvija putovanje paketa od internog računala, do poslužitelja na Internetu i nazad:

1. Računalo kreira IP paket čije odredište je računalo na Internetu, s IP adresom C, a izvorište je računalo s IP adresom A. Taj paket računalo proslijeđuje svom lokalnom usmjerniku.
2. Lokalni usmjernik prihvaća paket, vidi da je upućen na Internet te izvorišnu adresu mijenja svojom – **važećom** – IP adresom i proslijeđuje paket dalje na Internet idućem usmjerniku.
3. Paket stiže do odredišnog računala koje prihvaća HTTP zahtijev i generira odgovor.

4. Odgovor se postavlja u paket čija odredišna adresa je adresa B i proslijeđuje lokalnom usmjerniku na mreži!
5. Nakon što paket proputuje kroz Internet dolazi do sučelja B usmjernika na privatnoj mreži s koje je došao zahtijev.
6. Usmjernik prihvaća paket, mijenja odredišnu adresu u adresu računala na lokalnoj mreži i proslijeđuje ga računalu s kojega je pristigao zahtijev.



Slika 6. Način rada NAT-a

Dakle, osnovna ideja je da kada računalo pošalje paket na Internet ono dolazi do usmjernika koji izvorišnu adresu mijenja svojom IP adresom i proslijeđuje ga na Internet. Kada se paket vraća s Interneta, usmjernik vraća originalnu IP adresu i proslijeđuje paket na internu mrežu. To je princip NAT-a, međutim implementacija je složenija od toga i uključuje mijenjanje korištenih pristupa i vođenje evidencije o svim vezama unutar usmjernika koji obavlja NAT.

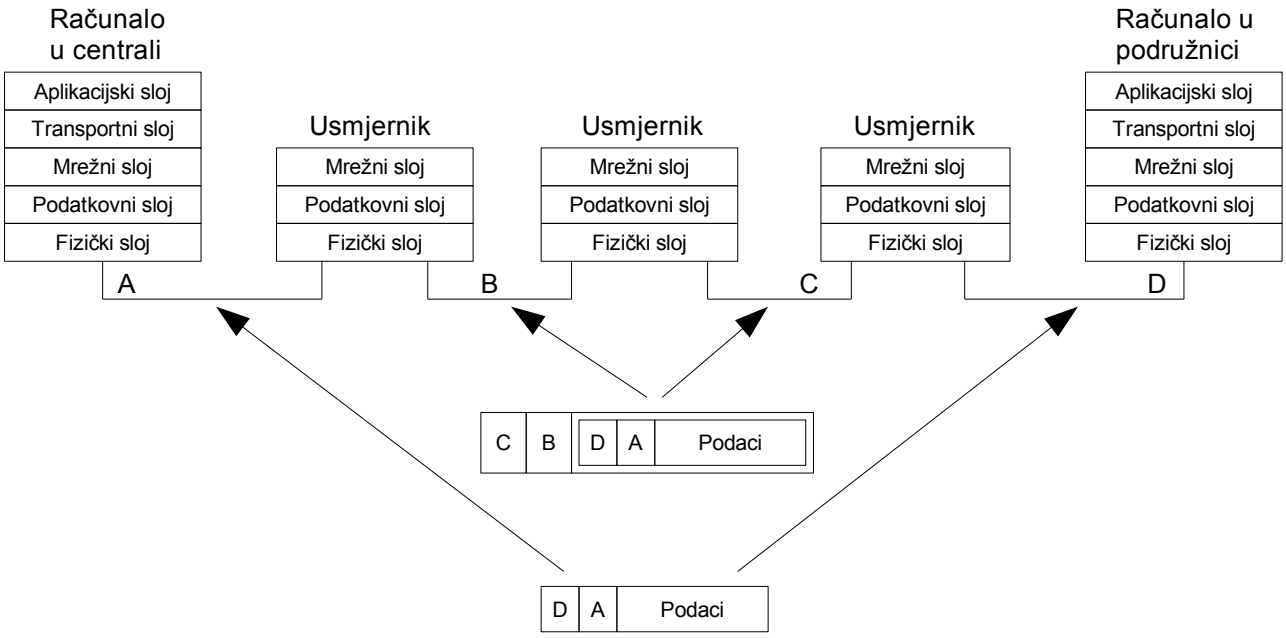
NAT je nastao kao jedan od odgovora na problem alokacije slobodnih IP adresa kojih je bilo sve manje. Ideja je bila da se za više računala dobije jedna IP adresa (ili svega nekoliko) te potom da se više računala koristi tim adresama. U međuvremenu ta se tehnika – iako uvodi neke nove probleme – koristi za rješavanje još nekih problema osim već navedenog.

Drugi problem prilikom podešavanja konfiguracije zadane slikom i tekstom je implementacija firewall uređaja. Iako postoje komercijalni proizvodi, primjerice CheckPoint FireWall, za taj problem postoji rješenje u samom Linux operacijskom sustavu. U taj operacijski sustav ugrađena je vrlo kvalitetna tehnologija koja omogućavaju implementaciju vrlo složenih firewall rješenja. Konkretno, radi se o `netfilter` i na njoj baziranoj `iptables` tehnologiji. `Netfilter` je osnovna infrastruktura na koju se nadograđuje `iptables`. To vrijedi za kernel od verzije 2.4. U starijim verzijama nalaze se neka druga rješenja i ona se neće obrađivati u ovoj laboratorijskoj vježbi.

Treći problem odnosi se na implementaciju VPN-a, tj. privatne mreže koja se za transport koristi javnom Internet infrastrukturom. Opet, kao i kod firewall uređaja postoji više mogućih rješenja koja rade na različitim slojevima ISO/OSI modela. Karakteristika svih njih je da se sav promet koji prolazi javnim dijelom mreže kriptira kako bi se zaštitio integritet i tajnost, te osigurala autentičnost podataka koji se prenose. Pravo rješenje na mrežnom sloju – koji nas interesira u

DON'T PANIC

ovom slučaju bilo bi korištenje IPSec standarda i to u vidu FreeS/WAN implementacije za Linux operacijski sustav. No, kako bi malo pojednostavili stvar u ovoj vježbi neće se koristiti kriptiranje, već samo *tuneliranje*. Poopćeno, *tuneliranje* je pojam koji označava da se u podatkovim jedinicama jednog sloja prenose paketi tog ili nižih slojeva. Principijelno, to je prikazano na slici 7.



Slika 7. Princip tuneliranja

Na slici vidimo izgled mrežnog paketa što ga računalo u centrali, s IP adresom A, šalje računalu u podružnici, s IP adresom D. Radi jednostavnosti IP adrese su označene slovima, te pretpostavljamo da su A i D privatne adrese, a B i C javne adrese. Adrese B i C pripadaju eksternim sučeljima usmjernika koji povezuju poduzeće s Internetom. Očito je da računalo A šalje paket računalu D i normalno ga prosljeđuje usmjerniku na lokalnoj mreži. Usmjernik, konzultirajući svoje tablice, vidi da paket mora poslati na drugu privatnu mrežu preko tunela koji spaja točku B s točkom C. On u tom slučaju kreira novi IP paket za čije odredište stavlja sučelje C usmjernika, a kao izvorište stavlja svoje sučelje B. Korisni teret (*payload*) tog paketa je paket s interne mreže! Nakon što taj paket stigne do sučelja C usmjernika koji pripada podružnici i na kojemu se nalazi druga točka tunela, on vadi nazad IP paket sa odredištem D i prosljeđuje ga na intranet do odredišnog računala D!

4. Upute za izradu vježbe

Na osnovu prethodnog razmatranja i znanja stečenih iz prethodnih laboratorijskih vježbi cijeli postupak izrade vježbe može se podijeliti na sljedeće osnovne korake:

1. Pripremiti sve virtualne PC-je
2. Ostvariti osnovnu povezanost, tj. računala unutar svake interne mreže mogu nesmetano komunicirati međusobno i s lokalnim usmjernikom. Također, usmjernici FW1 i FW2 mogu komunicirati s usmjernikom na Internetu!
3. Potrebno je podesiti osnovna usmjeravanja, tj. FW1 i FW2 moraju imati osposobljenu međusobnu komunikaciju.
4. Kreiranje tunela između usmjernika FW1 i FW2 koji će prenositi pakete internih mreža.

5. Podešavanje firewall-a koji će spriječiti da se izvan internih mreža pristupa internim mrežama, ali će istovremeno omogućiti nesmetanu komunikaciju internih mreža.
6. Podesiti firewall kako bi bila zadovoljena i sva pravila dana u zadatku vježbe. Tj. potrebno je podesiti izlazak s internih mreža na internet.

Od svih navedenih radnji u prethodnim laboratorijskim vježbama nisu obrađivane samo točke 4 i 5 te su te točke dodatno pojašnjene u idućim potpoglavljima.

4.1.Kreiranje tunela

Kreiranje tunela je proces koji se mora obaviti na oba njegova kraja, tj. na oba usmjernika gdje tunel počinje i završava. Postoji više vrsta tunela, tj. načina na koji se jedan IP paket može prenositi unutar nekog drugog paketa. U ovoj laboratorijskoj vježbi koristit ćemo jednostavnu enkaspulaciju IP paketa u IP pakete.

Dakle, na jednoj strani tunela potrebno je kreirati uređaj koji sve što primi upakira u novi IP paket te potom taj novi IP paket proslijedi na internet do druge strane tunela. Kreiranje tunela na jendoj strani obavlja se upotrebom sljedeće naredbe:

```
# ip tunnel add <tunnel_device> mode ipip remote \
        <udaljena_IP_adresa>
```

U toj naredbi potrebno je zamijeniti *tunnel_device* s imenom uređaja koji će obavljati pakiranje IP paketa koje primi te ih potom slati na IP adresu određenu parametrom *udaljena_IP_adresa*. U ovoj laboratorijskoj vježbi udaljena IP adresa je javno sučelje drugog usmjernika. Dakle, ako se naredba `ip(8)` izvršava na FW1, to je javna IP adresa od FW2. Ime sučelja je proizvoljno, uz određena pravila koja se moraju poštivati. Recimo da se tijekom ove laboratorijske vježbe može koristiti ime `tunnel0`.

Nakon izvršavanja naredbe `ip tunnel add` kreiran je uređaj imena *tunnel_device*, tj. `tunnel0`, s kojim dalje postupamo isto kao što postupamo npr. s uređajem `eth0`. To znači da mu je potrebno dodijeliti IP adresu i aktivirati ga. Za IP adresu uređaja koji predstavlja tunel možemo uzeti istu IP adresu kao i interno sučelje usmjernika ili se može odabrati potpuno novi blok adresa za tunel. Ako se odabire IP adresa ista kao i IP adresa privatnog sučelja tada je potrebno za mrežnu masku postaviti vrijednost /32!

Još je preostalo podesiti tablicu proslijeđivanja kako bi jedna strana tunela bila kompletna. Potrebno dodatni rutu uz pomoć koje se privatna mreža na drugoj strani može dostići preko tunela (`dev tunnel0`).

Da bi tunel bio u potpunosti konfiguriran potrebno je na drugoj strani napraviti potpuno analogan niz operacija, s tim da je potrebno postaviti odgovarajuće zamijenjene IP adrese.

Praćenjem prometa na sučeljima usmjernika koji prestavlja Internet može se jasno vidjeti kako kroz njega prolaze paketi koji sadrže u sebi IP pakete. To je moguće zahvaljujući tome što `tcpdump` prepoznaje enkapsulirane pakete te ih ispravno može dekodirati i prikazati.

4.2.Podešavanje firewall-a

Podešavanje firewall-a u Linux operacijskom sustavu obavlja se upotrebom `iptables(8)` programa. Kako bi se zadovoljila pravila navedena u zadatku potrebno je podesiti mrežni sloj usmjernika 1 na sljedeći način:

DON'T PANIC

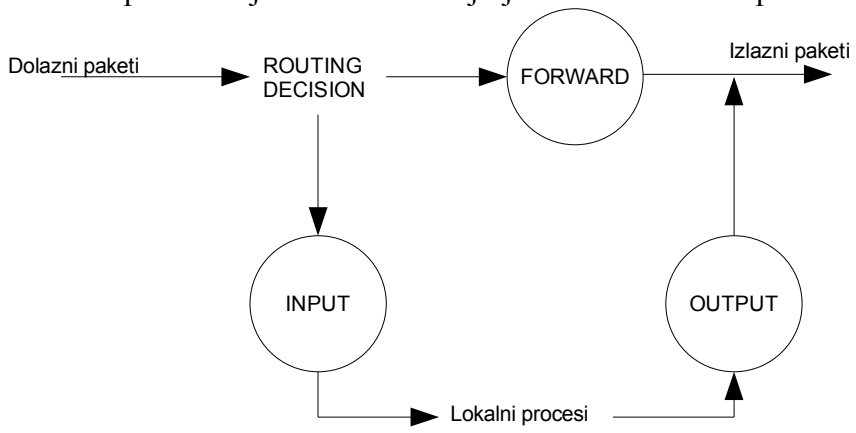
- Svaki paket koji pristiže na javno sučelje i ima postavljenu privatnu adresu u odredišnoj ili izvorišnoj adresi treba odbaciti.
- Na sve HTTP zahtijeve koji dolaze iznutra, a ne pripadaju grupi zabranjenih IP adresa, potrebno je primjeniti NAT i propustiti na Internet.
- Svaki Web zahtijev s interneta potrebno je proslijediti web poslužitelju čija je IP adresa na internoj mreži 10.0.0.2.
- Svaku isporuku mail-a koja dolazi izvana potrebno je proslijediti internom mail poslužitelju čija je IP adresa 10.0.0.3.

Usmjernik 2 potrebno je podesiti na nešto drugačiji način:

- Svaki paket koji pristiže na javno sučelje i ima postavljenu privatnu adresu u odredišnoj ili izvorišnoj adresi treba odbaciti.
- Na sve HTTP zahtijeve koji dolaze iznutra, a ne pripadaju grupi zabranjenih IP adresa, potrebno je primjeniti NAT i propustiti ih na Internet.
- Svaki pokušaj pristupa nekom mail poslužitelju na internetu koji dolazi sa internog mail poslužitelja propustiti van sa odgovarajućom modifikacijom.
- Sve zahtijeve koji dolaze izvana potrebno je blokirati.

4.2.1. Podešavanje Linux firewall-a

Kako bi se moglo obaviti navedena podešavanja potrebno je poznavati način na koji radi `netfilter/iptables` podsustav Linux kernela. Svaki paket koji dolazi sa interneta u računalo prolazi određene podsustave unutar operacijskog sustava. U tim podsustavima određuje se daljnja sudbina paketa, tj. da li se on šalje na neko drugo sučelje van na internet ili se isporučuje lokalnoj aplikaciji. Slična stvar vrijedi i za pakete koje generira neka aplikacija koja se izvršava na lokalnom računalu i koji treba biti poslan na internet. Tijekom obrade paketa u jezgri operacijskog sustava moguće je korištenjem programa `iptables(8)` proslijediti određena pravila jezgri operacijskog sustava uz pomoć kojih se može detaljnije odrediti sudbina paketa.



Slika 8. Pojednostavljen prikaz prolaska paketa kroz jezgru operacijskog sustava

Na slici 8 prikazana je principijelna shema puta što ga prolazi svaki paket u jezgri operacijskog sustava. Na toj slici namjerno su određeni elementi ostavljeni na engleskom jeziku budući da su usko povezani s parametrima `iptables(8)` programa. To su elementi koji se nalaze unutar

DON'T PANIC

kružnica (*Forward*, *Input* i *Output*). Za svaki paket koji pristigne na neko ethernet, serijsko ili bilo koje drugo sučelje (*Dolazni paket*) prvo se određuje da li je taj paket namijenjen nekom lokalnom procesu ili ga je potrebno proslijediti kroz neko drugo sučelje van. Nakon te odluke paket dolazi do tzv. INPUT lanca – ako je namijenjen lokalnom procesu, odnosno do FORWARD lanca ako ga je potrebno poslati nekom drugom računalu/usmjerniku. Konačno, postoji još jedan lanac, OUTPUT, u koji dolaze paketi koje generiraju lokalne aplikacije i šalju preko mreže udaljenim aplikacijama.

INPUT, OUTPUT i FORWARD se nazivaju lancima budući da se u tim točkama svaki paket uspoređuje sa nizom pravila kojima se definira njegova sudbina. Ta pravila čine logički lanac, a potavljaju se upotrebom već spomenutog `iptables(8)` programa. Primjer ispisivanja sadržaja svih navedenih lanaca:

```
[root@localhost]# /sbin/iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[root@localhost]#
```

Upotrebom opcije -L ispisuje se sadržaj svih lanaca. U navedenom primjeru svi lanci su prazni. Za svaki lanac definiran je podrazumijevana akcija. Kada paket pristigne do nekog lanca tada se on uspoređuje slijedno sa svim pravilima. Prvo pravilo koje odgovara po nekim kriterijima se izvršava, a sva ostala se ignoriraju. Ako niti jedno pravilo ne odgovara tada se izvršava navedena podrazumijevana akcija. U prethodnom primjeru podrazumijevana akcija je ACCEPT, tj. paket se propušta dalje. Osim ACCEPT može biti i DROP koji jednostavno uklanja paket kao da nikada nije ni postojao. Primjerice, u jednoj virtualnoj konzoli pokrenite sljedeću naredbu

```
ping 127.0.0.1
```

Ta naredba stalno ispisuje pristigle odgovore. Sada u drugoj konzoli izvršite sljedeću naredbu:

```
iptables -P INPUT DROP
```

Odmah nakon što se izvrši ta naredba ping više ne ispisuje nikakve odgovore. Razlog tome je promjena podrazumijevane akcije za pakete koji dolaze u INPUT lanac. Ono što se dešava je sljedeće:

1. Naredba `ping(8)` šalje paket koji dolazi u OUTPUT lanac. Tu nema nikakvih pravila, a podrazumijevana akcija je ACCEPT prema tome paket prolazi.
2. Sučelje 127.0.0.1 je takvo da svaki paket koji pristigne tom sučelju se vraća nazad kao da je upravo primljeno izvana.
3. Nakon procesa usmjeravanja (*ROUTING DECISION*), pojavljuje u INPUT lancu. U tom lancu nema nikakvih pravila, ali je podrazumijevana akcija DROP te se paket jednostavno odbacuje. Posljedica te akcije je da nije generiran odgovor jer paket nije ni viđen od kasnijeg mrežnog podsustava koji generira odgovor i prema tome `ping(8)` nije dobio odgovor i nije ništa ispisao.

Na staro stanje moguće se je vratiti izvršavanjem ekvivalentne naredbe, tj. potrebno je izvršiti

```
iptables -P INPUT ACCEPT
```

Svaki paket moguće je provjeravati po nizu kriterija. Primjerice, ako se želi blokirati sve pakete koji s lokalnog računala odlaze prema računalu s IP adresom 161.53.65.11, tada je potrebno izvršiti sljedeću naredbu:

```
iptables -A OUTPUT -d 161.53.65.11 -j DROP
```

Opcija -d prima kao argument IP adresu koja se potom uspoređuje s odredištem (*destionation*) svakog paketa i ako odgovara izvršava se akcija definirana opcijom `-j`. U ovom slučaju ta akcija traži da se paket jednostavno odbaci. Ako se sada pogleda sadržaj svih lanaca dobija se sljedeće stanje:

```
[root@localhost]# /sbin/iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             161.53.65.11
[root@localhost]#
```

Kao što se vidi pravilo je dodano u OUTPUT lanac i svaki pokušaj pristupa računalu sa tom IP adresom će biti blokiran. Da bi se pojedino pravilo uklonilo iz lanca prilikom poziva naredbe `iptables` potrebno je samo opciju -A zamijeniti sa opcijom -D, a sve ostalo mora ostati isto. Prema tome, da se ukloni prethodno navedeno pravilo potrebno je izvršiti sljedeću naredbu:

```
iptables -D OUTPUT -d 161.53.65.11 -j DROP
```

Stvari se ponešto razlikuju u slučaju usmjernika. Ako na usmjerniku želimo zabraniti pristup određenom računalu na internetu svim paketima koji pristižu na taj usmjernik tada je prethodno pravilo potrebno dodati u FORWARD lanac, a ne u OUTPUT lanac!

Slično kao što se paket može kontrolirati po odredišnoj adresi može se kontrolirati i po sljedećim kriterijima:

- Izvorišnoj adresi, opcija `-s`.
  Primjer: `-s 192.168.15.0/24`.
- Po protokolu, tcp, udp, icmp, itd sa opcijom `-p`.
  Primjer: `-p tcp`.
- Po odredišnom pristupu u slučaju protokola tcp i udp sa opcijom `--dport`.
  Primjer: `--dport 1024`.
- Na osnovu sučelja s kojega je paket primljen uz pomoć opcije `-i`. Ova opcija može se upotrebljavati isključivo u INPUT ili FORWARD lancu.
  Primjer: `-i eth0`
- Na osnovu izlaznog sučelja uz pomoć opcije -o. Ova opcija se može upotrebljavati isključivo u FORWARD i OUTPUT lancima.
  Primjer: `-o ppp0`

DON'T PANIC

U većini prethodnih opcija moguće je ispred argumenta opciji postaviti znak `!`. Na taj način se invertira značenje. Recimo, ako želimo selektirati pakete koji dolaze na sva sučelja osim eth0, tada bi to pisali na sljedeći način:

```
-o \! eth0
```

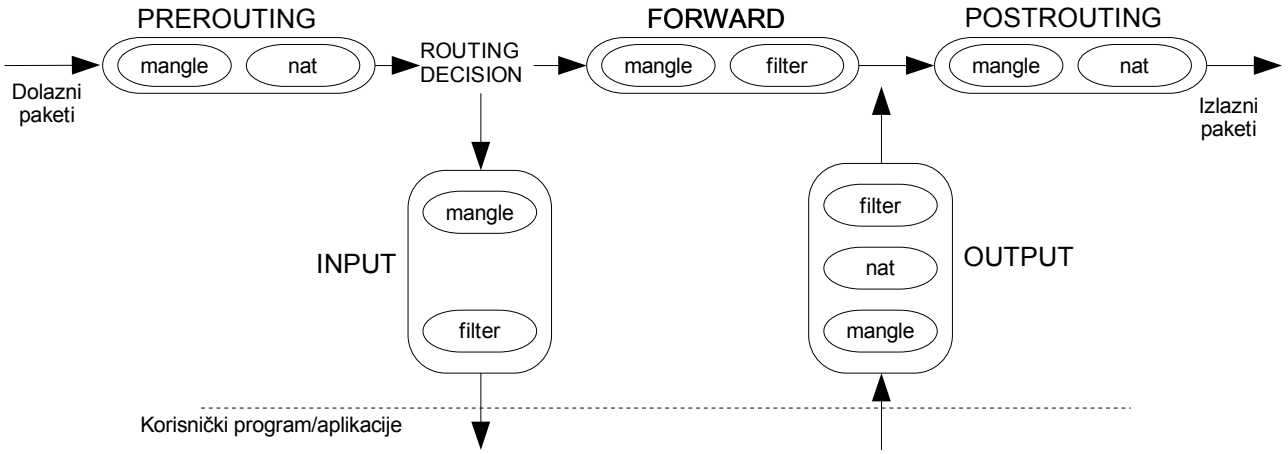
Ispred uskličnika se nalazi obrnuta kosa crta (*backslash*) budući da uskličnik inače ima posebno značenje za ljusku. Ovim putem se to značenje eliminira i uskličnik se prosljeđuje aplikaciji.

U dosadašnjem tekstu spominjane su samo dvije akcije koje se izvršavaju na paketu u slučaju da on odgovara navedenim uvijetima. Prva akcija je DROP i uzrokuje da se paket jednostavno zaboravi, dok je druga ACCEPT i ona označava da se paket treba dalje propustiti. Međutim, osim njih ima još i mnoštvo drugih od kojih su za obavljanje laboratorijskih vježbi potrebna dva. To su DNAT i SNAT.

SNAT se koristi za NAT izvorišne IP adrese i on je potreban kada se iza javne IP adrese usmjernika želi sakriti blok adresa na privatnoj mreži. Konkretno, za ovaj zadatak to znači da adrese 10.0.0.0/24 i 10.0.1.0/24 želimo sakriti kako se ne bi primjetile s interneta. S interneta se treba vidjeti isključivo IP adresa 192.168.0.2 usmjernika FW1 i 192.168.0.6 usmjernika FW2.

DNAT se koristi kada se prepisuje odredišna IP adresa. Konkretno u ovom zadatku, kada neko računalo s interneta treba pristupiti Webu u centrali šalje zahtijev na port 80 računala 192.168.0.2. Na tom računalu se paket prepisuje tako da nova odredišna adresa glasi 10.0.0.2, također na port 80.

Za korištenje tih opcija potrebno je poznavati detaljniji put koji prolaze paketi u jezgri operacijskog sustava. Na slici 8 prikazana je principijelna shema, dok je na slici 9 prikazana nešto detaljnija shema.



Slika 9. Detaljna shema prolaska paketa kroz jezgru Linux operacijskog sustava

Na toj slici vidimo da osim INPUT, OUTPUT i FORWARD lanaca postoje i još dva dodatna. Prvi je PREROUTING i koristi se neposredno prije odluke o usmjeravanju, a drugi je POSTROUTING koji se koristi neposredno prije slanja paketa na izlazno sučelje.

Druga razlika je činjenica da se svaki lanac sastoji od dvije odvojene grupe pravila, dok se OUTPUT lanac sastoji od tri grupe. Te grupe ćemo od sada zvati *tablice*. Redoslijed korištenja pravila iz tih grupa je bitan i odvija se u smjeru strelica. Primjerice, kod OUTPUT lanca prvo se koriste pravila iz tablice *mangle*, potom iz *nat* i na kraju *filter*.

Prilikom poziva naredbe `iptables(8)` lanci se navode kao parametri opcija `-A`, `-D` i `-P` kao

što je to do sada prikazano u primjerima. Tablice se s druge strane biraju opcijom `-t`, osim za opciju `-P` koja ne podržava odabir pojedinih tablica. Recimo, želimo li odabrati tablicu *mangle* lanca POSTROUTING to kažemo na sljedeći način:

```
iptables -A POSTROUTING -t mangle ...
```

Ako se ne navede tablica putem opcije `-t` tada se podrazumijeva tablica *filter*.

Sada se mogu preciznije definirati akcije SNAT i DNAT. DNAT se može koristiti isključivo u tablici *nat* PREROUTING i OUTPUT lanaca, dok se SNAT može korititi isključivo u tablici *nat* POSTROUTING lanca. Obje akcije zahtijevaju dodatni argument. Za SNAT taj argument ima sljedeći oblik:

```
... -j SNAT --to-source <ip adresa> ...
```

Parametar `ip adresa` predstavlja IP adresu koja će se koristiti i za tu IP adresu se u ovoj laboratorijskoj vježbi treba postaviti IP adresa javnog sučelja usmjernika/firewall-a. Za DNAT argument je sljedećeg oblika:

```
... -j DNAT --to-destionation <ip adresa> ...
```

U ovom slučaju IP adresa je adresa računala kojemu želimo preusmjeriti paket.

U dosadašnjem tekstu opisano je kako blokirati odnosno propustiti pojedini paket ovisno o sučelju sa kojega dolazi, na koje odlazi, koji protokol je u pitanju, koje IP adrese su upisane i slično. Međutim, ostaje jedan problem. Problem je kada se sa interne mreže pošalje paket na internet te se s interneta očekuje odgovor! Sva dosadašnja pravila nisu vodila računa o stanju, tj. o povezanosti paketa koji su odaslani i njihovih odgovora. Kao primjer može se uzeti program ping. Pokretanjem tog programa na nekom računalu na internoj mreži dešava se sljedeće:

1. Generirani paket *ICMP Echo Request* pristiže do usmjernika/firewall-a
2. Pravila u FORWARD lancu dopuštaju prolazak paketa na javno sučelje usmjernika.
3. Akcija SNAT u POSTROUTING lancu mijenja izvorišnu IP adresu u IP adresu javnog sučelja.
4. Paket stiže do odredišta gdje se kreira i vraća paket *ICMP Echo Reply*.
5. Taj paket stiže do ulaznog sučelja usmjernika/firewall uređaja.
6. Budući da je politika da se svi paketi koji dolaze izvana odbacuju odgovor se ne prosljeđuje računalu na internoj mreži.

Taj problem rješava se vođenjem evidencije o poslanim paketima i korištenjem SNAT-a, netfilter podsustav to radi implicitno za svaki modificirani paket koji se pošalje. Na taj način kada paket pristigne na ulaz provjerava se da li je taj paket povezan s nekim već odaslanim paketom i ako je tada mu se dodjeljuje status ESTABLISHED – ako pripada nekoj vezi kao što je npr. TCP veza, odnosno RELATED – ako je u vezi sa nekim odaslanim paketom kao što je to slučaju prethodno opisanim *ICMP Echo Request/Reply* paketima. Sada se prethodni niz modificira na sljedeći način:

1. Generirani paket *ICMP Echo Request* pristiže do usmjernika/firewall-a
2. Pravila u FORWARD lancu dopuštaju prolazak paketa na javno sučelje usmjernika.
3. Akcija SNAT u POSTROUTING lancu mijenja izvorišnu IP adresu u IP adresu javnog sučelja.

DON'T PANIC

4. Paket stiže do odredišta gdje se kreira i vraća paket *ICMP Echo Reply*.
5. Taj paket stiže do ulaznog sučelja usmjernika/firewall uređaja.
6. Provjerava se da li je paket odgovor na neki odaslani paket i ako je obavlja se de-SNAT te se paket označava kao RELATED odnosno ESTABLISHED.
7. Svi paketi koji imaju oznaku RELATED, odnosno ESTABLISHED se propuštaju.

Ako prilikom poziva `iptables(8)` programa želimo provjeriti da li paket posjeduje oznake ESTABLISHED i RELATED to se obavlja upotrebom sljedećih opcija:

```
.. -m state --state ESTABLISHED,RELATED ...
```

4.2.2. Ispitivanje ispravnosti postavki

Prilikom korištenja programa `tcpdump(8)` tijekom ispitivanja ispravnosti pravila pripaziti da svi paketi koje `tcpdump(8)` dobije dolaze u INPUT lanac. Prema tome, ako neko pravilo propusti pakete kroz FORWARD lanac, ti se paket neće vidjeti sa `tcpdump(8)` programom sve dok se isto pravilo – ili ekvivalentno – ne doda i u INPUT lanac!

Za ispitivanje ispravnosti konfiguracije može se koristiti sljedeći niz testova:

1. `ping(8)` sa računala unutar jedne interne mreže mora raditi do oba usmjernika/firewall-a, odnosno do svih računala druge interne mreže.
2. Za ispitivanje ispravnog proslijeđivanja na mail odnosno web port koristiti telnet klijent na usmerniku koji predstavlja internet. Prije toga pokrenuti servise *httpd* i *sendmail*. Način pokretanja servisa objašnjen je u uputama za 3. laboratorijsku vježbu.

Vježba V – Bespojna usluga TCP/IP porodice protokola

1. Uvod

Ova vježba služi za upoznavanje sa bespojnom uslugom TCP/IP porodice protokola i karakteristikama te usluge. Tijekom vježbe potrebno je isprogramirati TFTP poslužitelj i klijent na osnovu specifikacije dane u dodatku. TFTP je vrlo jednostavan protokol. Primjer njegove primjene je prijenos datoteka operacijskog sustava prilikom podizanja računala, ili za pohranu parametara konfiguracije na udaljena računala, itd. Protokol je dizajniran tako da njegova implementacija bude što jednostavnija. Zbog toga protokol nema nikakve mehanizme autentikacije, ograničene su mogućnosti autorizacije, nema izlistavanja sadržaja direktorija i drugih mogućnosti koje primjerice ima FTP protokol. Prednosti tog protokola – jednostavnost – istovremeno su i njegovi glavni nedostaci te je prilikom upotrebe tog protokola na Internetu potrebna opreznost kako ne bi došlo do njegove zloupotrebe!

Za izradu ove vježbe potrebno je poznavati način korištenja pristupnih točki za bespojnu uslugu i TFTP protokol. U nastavku su prvo definirani poslužitelj i klijent koje je potrebno isprogramirati, a nakon toga su dane upute za izradu vježbe. U tim uputama ukratko je opisana TCP/IP porodica protokola, nakon toga nešto detaljnije su opisane pristupne točke i to sa naglaskom na dio koji se odnosi na bespojnu uslugu. Na kraju uputa za ovu vježbu nalaze se naputci koji bi trebali olakšati izradu vježbe.

2. Zadatak

Poslužitelj i klijent moraju biti Unix aplikacije koje se pokreću u komandnoj liniji. Obavezni argument poslužitelju je pristup (broj u rasponu od 1024 do 65535) na kojemu poslužitelj čeka zahtijev klijenta. Nije potrebno definirati direktorij iz kojega poslužitelj dohvaća i pohranjuje datoteke, već to može biti radni direktorij u kojemu je poslužitelja pokrenut. Od grešaka koje se mogu pojaviti i koje je potrebno detektirati dovoljne su dvije i to kada tražena datoteka nije pronađena (*File not found*) te kada se ne može pisati u diretkorij (*Access Violation*). Nadalje, mogu se ignorirati razlike u modu prijenosa (*netascii*, *octet*, itd.) te taj mod uvijek postaviti na *octet* i sve podatke tretirati kao prijenos 8-bitnih byte-ova bez ikakve transformacije na podacima koji se prenose.

Argumenti klijentu trebaju biti riječ `get` (za prihvat datoteke sa udaljenog računala na lokalno računalo) ili `put` (za prijenos lokalne datoteke na udaljeno računalo), IP adresa ili ime računala na kojemu se nalazi poslužitelj, pristup na kojemu čeka poslužitelj i na kraju ime datoteke.

Dopušta se izrada dvije varijante poslužitelja/klijenta. U prvoj – jednostavnijoj – varijanti može se pretpostaviti da svi poslani paketi pristižu na odredište bez greške. U drugoj varijanti – složenijoj – potrebno je dodati logiku koja provjerava da li su paketi pristigli na odredište kako je to opisano u samoj specifikaciji.

Primjer pokretanja TFTP poslužitelja:

```
$ tftp_server 32777
```

Primjer poziva TFTP klijenta kako bi se sa poslužitelja dohvatila datoteka `test.c`:

```
$ tftp_client get 127.0.0.1 32777 test.c
```

Kako bi se ta ista datoteka prebacila na poslužitelj koristi se sljedeći oblik:

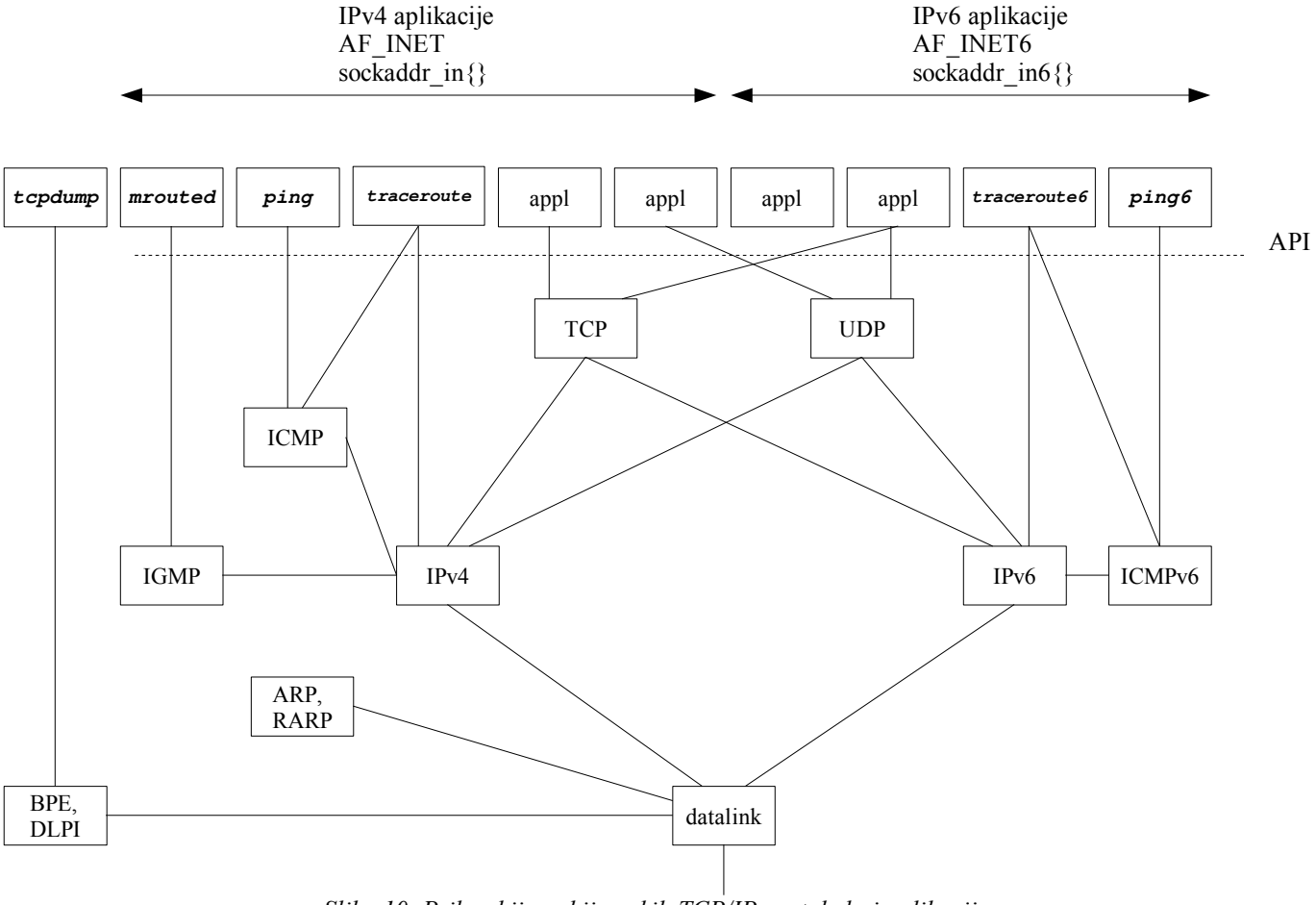
DON'T PANIC

```
$ tftp_client put 127.0.0.1 32777 test.c
```

3. Korištenje bespojne usluge putem pristupnih točaka

3.1.TCP/IP

TCP/IP porodica protokola je dominantan skup protokola koji se danas koristi za povezivanje računala i srodnih uređaja i predstavlja osnovu na kojoj je izgrađen Internet. Iako se u nazivu spominju samo TCP i IP ta porodica uključuje i mnoštvo drugih protokola od kojih je dio prikazan na slici 10. TCP/IP nikada nije nastao, niti je modeliran na osnovu nekog formalnog modela – kao što je to slučaj s OSI stogom – ali se ipak uslojavanje, tj. hijerarhija protokola može se primjetiti iščitavanjem slike odozdo prema gore (ili obrnuto). Na dnu imamo podatkovni sloj koji se nalazi neposredno iznad fizičkog sloja (koji nije prikazan) te s njim direktno komunicira. Iznad njega je mrežni sloj sa IPv4, IPv6 i nekim pomoćnim protokolima. Nakon toga dolaze transportni slojevi TCP i UDP te na kraju imamo aplikacije koje su prikazane na samom vrhu slike. Svi protokoli iz TCP/IP porodice protokola, ali i mnoštvo drugih stvari, definirani su serijom RFC dokumenata koji se besplatno mogu skinuti s Interneta. Mjesta na Internetu gdje su ti dokumenti pohranjeni navedena su na stranicama predmeta. Dio ispod isprekidane crte označene tekstom **API** nalazi se implementiran u jezgri operacijskog sustava te u pomoćnim bibliotekama. Dio iznad isprekidane crte su aplikacijski programi koji su isporučeni sa operacijskim sustavom ili ih je napravio korisnik.



Slika 10. Prikaz hijerarhije nekih TCP/IP protokola i aplikacija

DON'T PANIC

3.2.Način kodiranja poslužitelja i klijenta

Pristupne točke kao pojam već su uvedene tijekom prve laboratorijske vježbe. U ovoj laboratorijskoj vježbi uz pomoć pristupnih točkaka koriste se mrežne usluge prijenosnog sloja, konkretno UDP-a. UDP protokol oslanja se na mrežni sloj, tj. na IP protokol i dodaje vrlo malo funkcionalnosti. Da se podsjetimo, karakteristike UDP protokol su:

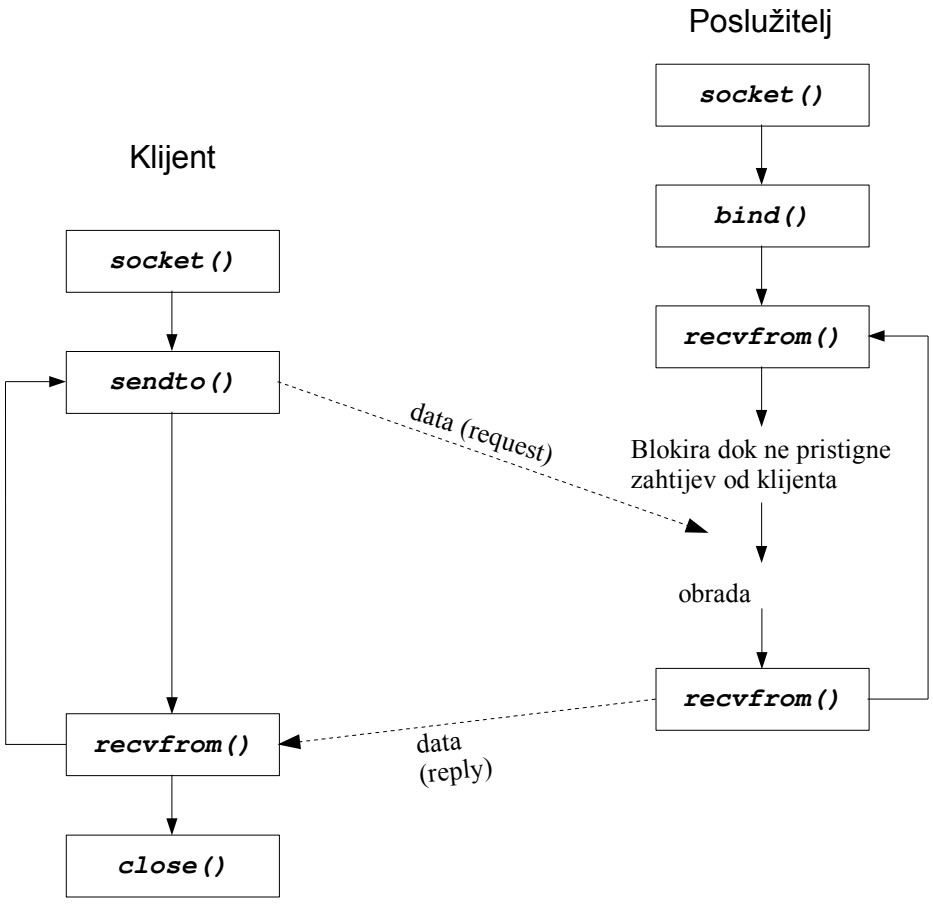
1. Osnovna jedinica prijenosa je *datagram,*
2. veza je bespojna, tj. za svaki datagram mora se navoditi odredišna adresa,
3. ne garantira se isporuka datagrama, ali se može desiti da na odredište pristigne i više kopija jednog datagrama,
4. nije sačuvan redoslijed kojim poruke pristižu,
5. nema kontrole protoka (*flow-control*).

Razlika u odnosu na IP je uvođenje pojma *pristupa* (port). Taj pojam ne predstavlja ništa drugo nego jedan 16-bitni broj u rasponu od 1 do 65535. Taj broj služi za demultipleksiranje dolaznih UDP poruka. Ako dvije aplikacije žele koristiti usluge UDP-a, bez tog broja operacijski sustav ne bi znao kome treba isporučiti dolazni paket te bi bilo nemoguće istovremeno izvršavanje više od jedne aplikacije koja koristi UDP. Ovako, svaka aplikacija definira pristup sa kojim je povezana, a paketi koji pristižu imaju u sebi zapisan pristup kojemu su namijenjeni. Na taj način operacijski sustav točno zna kojoj aplikaciji treba isporučiti pristigle podatke. Prethodno razmatranje vrijedi i za spojnu uslugu, tj. i ona isto koristi pristupe kao što će to biti objašnjeno u idućoj laboratorijskoj vježbi.

Pristupi čiji broj pada u rasponu od 1 do 1023 su rezervirani pristupi i dodjeljivanje tih pristupa pojedinim aplikacijama obavlja posebna organizacija. Recimo, DNS poslužitelju pripada pristup broj 53. Kada želimo poslati upit DNS-u tada ga postavimo u paket koji pošaljemo preko UDP-a na taj pristup. Tamo ga prihvati DNS poslužitelj te potom vraća odgovor. Kako su neke od aplikacija koje čekaju na tim pristupima kritične za ispravno funkcioniranje Interneta, tj. TCP/IP porodice protokola, to je povezivanje na te pristupe zabranjeno neprivilegiranim korisnicima kako ne bi došlo do eventualne zloupotrebe. Naravno da to nije dovoljno za potpunu zaštitu, ali to predstavlja jednu od njenih komponenti.

Kao što je to već rečeno u drugom poglavlju, dominantan API za korištenje mrežnih usluga su tzv. pristupne točke (socket API). Taj API dizajniran je za programski jezik C iako postoje odgovarajuće ekstenzije i prilagodbe drugim programskim jezicima. Sastoji se od zaglavnih datoteka (.h) te biblioteka u kojima se nalazi implementacija dijela funkcionalnosti tog API-ja. Ostatak je implementiran unutar operacijskog sustava. Cijeli sustav je dosta kompleksan i nudi mnoštvo mogućnosti ali u ovoj vježbi ćemo se ograničiti na bespojnu uslugu i funkcionalnosti neophodne za implementaciju vrlo jednostavno poslužitelja i klijenta.

Kao i svaki drugi API tako se i ovaj sastoji od podatkovnih struktura i funkcija koje koriste te podatkovne strukture te obavljaju operacije nad njima. Redoslijed poziva nekih funkcija je za pojedine namjene unaprijed definiran. Na slici 11 prikazan je redoslijed pozivanja funkcija u slučaju kada se želi ostvariti bespojna veza. Neke od funkcija na toj slici već su spominjane u prvoj laboratorijskoj vježbi, no postoje razlike, kako u korištenju već spomenutih funkcija tako i u dodatku nekih novih. Tijekom izrade vježbe potrebno je pretvarati simbolička imena računala u IP adrese. Način na koji se to obavlja opisan je nešto kasnije.



***Slika 11** Funkcije Socket API-ja za UDP poslužitelj/klijent*

Lijeva strana slike odnosi se na poziv funkcija unutar klijenta, dok se desna strana odnosi na poslužitelj. U slijedećem tekstu nešto detaljnije su opisane pojedine funkcije, a zatim i podatkovne strukture koje se koriste. Prije korištenja funkcija API-ja potrebno je uključiti zaglavne datoteke `sys/types.h`, `sys/socket.h` i `netinet/in.h`.

3.2.1. Klijent

Način kreiranja pristupne točke već je pojašnjen u prvoj laboratorijskoj vježbi. To se obavlja pozivom funkcije `socket(2)` sa odgovarajućim parametrima koji definiraju tip usluge koju želimo od mreže. Budući da želimo različitu uslugu u odnosu na prvu vježbu, to su i odgovarajući paremtri različiti. Potpunosti radi prototip funkcije socket ima sljedeći oblik:

```
#include <sys/socket.h>
#include <netinet/in.h>

int socket(int domain, int type, int protocol);
```

Kako bi mogli koristiti UDP protokol, argumenti trebaju imati sljedeće vrijednosti:

DON'T PANIC

- `domain`

  Kao vrijednost ovog argumenta potrebno je proslijediti konstantu `AF_INET` (ili ekvivalentno `PF_INET` čime naznačujemo da želimo koristiti grupu protokola baziranih na IPv4 mrežnom protokolu. Ovom prilikom može se spomenuti postojanje srodnog argumenta `AF_INET6`. Taj argument označava grupu protokola baziranih na IPv6 baziranom mrežnom sloju, tj. na idućoj generaciji IP-a. Što se aplikacija tiče jedina razlika u tim protokolima je u načinu adresiranja.

- `Type`

  Uz pomoć ovog argumeta određuje se vrsta transportnog protokola koji želimo koristiti unutar IPv4 porodice protkola. U ovom trenutku potreban nam je UDP, pa se za ovaj argument navodi konstanta `SOCK_DGRAM` koja označava bespojnu uslugu sa svim već navedenim karakteristikama.

- `Protocol`

  Konačno, ovaj argument bira protokol koji se koristi za implementaciju željene vrste komunikacije u danoj grupi komunikacijskih usluga. Budući da najčešće samo jedan protokol implementira određenu vrstu komunikacije to se za ovaj parametar najčešće navodi 0, a to znači da kernel i/ili funkcije u bibliotekama same odaberu odgovarajući protokol na osnovu prva dva parametra. U konkretnom slučaju TCP/IP porodice protokola jedino UDP implementira bespojnu uslugu bez garancije isporuke te će on zato biti automatski odabran.

Povratna vrijednost funkcije je kao što je već rečeno, pozitivan deskriptor koji koristimo tijekom komunikacije, odnosno -1 u slučaju greške.

Nakon što je kreirana, pristupna točka se u slučaju klijenta može početi koristiti. Konkretno moguće je slati poruke na poslužitelj upotrebom funkcije `sendto(2)`.

Prototip ove funkcije već je navođen, no ovdje je ponovo naveden radi potpunosti:

```
int sendto(int sockfd, const void *msg, size_t len, int flags,
                  const struct sockaddr *to, socklen_t tolen);
```

Razlika u odnosu na upotrebu te funkcije u podatkovnom sloju je u načinu adresiranja, tj. u zadnja dva argumenta. Iako se u samom prototipu ta razlika ne primjeti, čiji razlozi su već pojašnjavani, ipak ona postoji. Način adresiranja na ovom transportnom sloju pojašnjen je nešto kasnije.

Primanje paketa obavlja se funkcijom `recvfrom(2)`, koja je također već opisivana i čiji prototip je ovdje ponovo naveden:

```
int recvfrom(int sockfd, void *buf, size_t len, int flags,
                   struct sockaddr *from, socklen_t *fromlen);
```

Sve napomene dane za ovu funkciju prilikom njena opisa danog u prethodnoj vježbi vrijede i za ovu vježbu sa odgovarajućim različitim načinom adresiranja koji će biti pojašnjen kasnije u ovoj pripremi.

Nakon što više nemamo namjeru korištenja pristupne točke, već spomenutom funkcijom `close(2)` je zatvaramo.

3.2.2. Poslužitelj

Za razliku od klijenta koji može koristiti pristupnu točku odmah nakon njenog kreiranja, poslužitelj mora prvo vezati adresu uz nju i potom čekati da pristignu podaci. To se obavlja uz pomoć funkcije `bind(2)`. Nakon njenog poziva pristupna točka ima adresu, te klijent, u slučaju da poznaje tu adresu, može pristupiti poslužitelju.

Prototip ove funkcije ima sljedeći oblik:

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
```

Argumenti imaju sljedeća značenja:

- `sockfd`

  Pristupna točka kreirana funkcijom ***`socket`***.

- `my_addr`

  Unutar ove podatkovne strukture upisana je adresa koju želimo dodijeliti pristupnoj točki. Ta identična struktura koristi se i u funkcijama `sendto` i `recvfrom`, normalno sa različitom svrhom. Odgovarajuća polja te strukture i konstante i funkcije koje se koriste za inicijalizaciju opisana su u daljnjem tekstu.

- `addrlen`

  Veličina prethodne strukture, tj. `sizeof(addrlen)`

Povratna vrijednost funkcije je 0 u slučaju kada je adresa uspješno dodijeljena pristupnoj točki, a u suprotnom vraća se -1 kao indikator greške, te je varijabla `errno` postavljena na odgovarajuću vrijednost.

Ovom prilikom treba primjetiti da klijent iako ne koristi funkciju `bind` ipak dobiva privremenu adresu. Tu privremenu adresu dodjeljuje mu operacijski sustav i ta adresa koristi se kako bi klijent mogao dobiti odgovor od poslužitelja! Ako je iz nekog razloga potrebno da klijent ima točno određenu addresu tada je moguće i u klijentu koristiti funkciju `bind`.

3.2.3. Adresiranje na transportnom sloju Interneta

Za adresiranje na transportnom sloju Interneta koristi se struktura `sockaddr_in`, odnosno `sockaddr_in6` za IPv6 mrežnog protokola. U nastavku je opisana samo struktura za IPv4 budući da je ona potrebna za izradu ove vježbe.

Struktura `sockaddr_in` je struktura u koju se upisuje odgovarajuća internet adresa. Adresa se sastoji od IP adrese računala i pristupa. U skladu s tim ova struktura ima sljedeća polja:

- `sin_family`

  Ovaj član strukture sadrži konstantu koja određuje o kojoj vrsti adrese se radi. Kao što je već rečeno, taj član je neophodan budući da sve funkcije pristupnih točki rukuju sa generičkim oblikom adrese. Kako bi se tijekom izvršavanja funkcija moglo utvrditi o kojoj točno strukturi se radi koristi se ovo polje. U našem slučaju vrijednost ovog polja se uvijek inicijalizira na vrijednost ***`AF_INET`***.

- `sin_port`

DON'T PANIC

U slučaju poslužitelja ovo polje sadrži pristup na kojemu poslužitelj čeka pakete od klijenata. Ako se u klijentu koristi `bind` funkcija tada je to pristup na koji će klijent dobivati odgovore poslužitelja. To je 16-bitni nepredznačeni broj pri čemu se njegova vrijednost ne inicijalizira direktno, tj. jednostavnim pridodjeljivanjem, već je to potrebno učiniti preko već spomenute funkcije ***`htons()`***. Argument te funkcije je 16 bitni broj čiji okteti su u poretku računala na kojemu se izvršava program *(host byte order*), dok je povratna vrijednost 16 bitni broj čiji okteti su u standardom poretku definiranom za mrežu *(network byte order*).

- `sin_addr`

  Ovo je struktura sa samo jednim članom, i to `s_addr`. U slučaju klijenta ovo polje sadrži adresu računala kojemu se pristupa i na kojemu se očekuje poslužitelj. U slučaju poslužitelja (i klijenta kada se koristi `bind` funkcija) ovo polje sadrži adresu sučelja na koju se veže poslužitelj (klijent). Recimo, postavljanjem ovog polja na vrijednost 127.0.0.1 server bi primao pakete od klijenta isključivo na loopback sučelju bez obzira na postojanje dodatnih sučelja. U slučaju da želimo da poslužitelj prima pakete sa svih sučelja koristi se posebna konstanta `INADDR_ANY`.

  Ovo polje je veličine 32 okteta i kao u slučaju polja `sin_port` tako se i ovome vrijednosti moraju dodjeljivati koristeći pomoćnu funkciju, u ovom slučaju `htonl(3)`. Primjerice:

```
...
struct sockaddr_in recv_addr;
...
...
recv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
...
```

  Ako umjesto 32-bitnog broja imamo ASCII zapis IP adrese (u obliku aaa.bbb.ccc.ddd), u tom slučaju koristimo funkciju `inet_pton(3)` kako bi dobili 32-bitnu vrijednost. Ta funkcija ima sljedeći prototip:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);
```

  Ova funkcija je u stanju prevesti više različitih oblika adresa te se kao prvi argument navodi porodica adresa čiju konverziju je potrebno obaviti. U našem slučaju to je ponovo konstanta AF_INET koja označava internet protokole bazirane na IPv4. Drugi argument je ASCII zapis IP adrese i konačno, treći argument je pokazivač na polje `sin_addr` varijable tipa `sockaddr_in`. U slučaju uspješne konverzije povratna vrijednost je pozitivan broj veći od nule.

  Ovom prilikom spomenuti ćemo i funkciju `inet_ntop(3)` koja 32-bitni zapis IPv4 adrese pretvara u ASCII zapis pogodan za ispis i koja ima sljedeći prototip:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *src, char *dst,
                            size_t cnt);
```

Argumenti te funkcije slični su argumentima funkcije `inet_pton(3)`. Prvi argument predstavlja porodicu adresa čiju konverziju želimo obaviti. U slučaju ove laboratorijske vježbe to je `AF_INET` budući da se radi o IPv4 adresama. Drugi argument pokazivač je na odgovarajuću adresnu strukturu čiji tip ovisi o prvom argumentu, tj. o porodici adresa. Za slučaj ove laboratorijske vježbe to je struktura `sockaddr_in.sin_addr`. Konačno, predzadnji argument pokazivač je na spremnik u koji se upisuje ASCII oblik adrese, dok zadnji argument sadrži veličinu spremnika.

Napomena: Na starijim implementacijama pristupnih točaka umjesto funkcija `inet_pton(3)` i `inet_pton(3)` koristile se se funkcija `inet_aton(3)` i `inet_ntona(3)`. Te funkcija su zastarjele budući da se mogu koristiti isključivo sa IP adresama vezanim uz verziju 4 IP protokola. Njihov opis ovdje nije dan ali se kao i za sve ostale funkcije može naći u on-line uputstvima.

3.3.Korištenje DNS-a u programima

U zadatku vježbe postavljen je zahtijev da argument klijentu treba biti simboličko ime računala na kojemu se izvršava poslužitelj. Zbog toga potreban je mehanizam konverzije iz simoličkih imena u IP adrese. U ovom potpoglavlju opisane su funkcije za pretvaranje DNS imena u IP adrese. DNS je bio tema jedne od prethodnih laboratorijskih vježbi kada je bilo potrebno konfigurirati DNS poslužitelje. U toj vježbi DNS je bio detaljnije pojašnjen sa administrativne strane, dok su ovdje opisani detalji neophodni za obavljanje ove vježbe, tj. opis sa programerske strane.

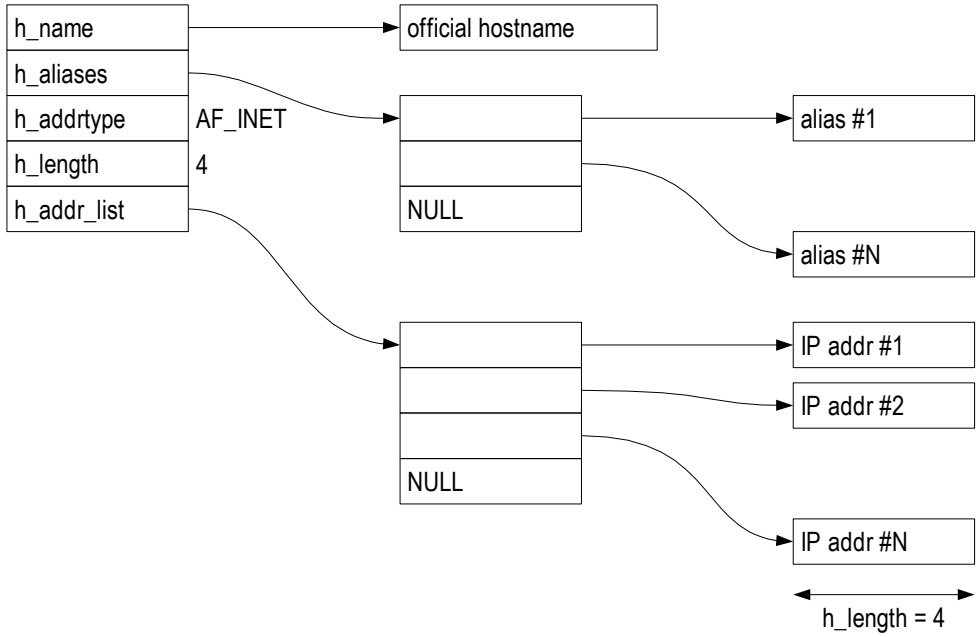
3.3.1. Funkcije i strukture podataka

Struktura `hostent` je temeljna struktura u koju se smještaju podaci o simboličkim imenima i IP adresama. Sve funkcije koje se koriste za pretvaranje simboličkih imena u IP adrese i obratno koriste tu strukturu. Kako bi se ta struktura i funkcije mogli koristiti potrebno je uključiti zaglavnu datoteku `netdb.h` u programu. Struktura ima sljedeća polja:

```
struct hostent
{
    char *h_name;       /* Official name of host. */
    char **h_aliases;   /* Alias list. */
    int h_addrtype;     /* Host address type. */
    int h_length;       /* Length of address. */
    char **h_addr_list; /* List of addresses from name server. */
};
```

Ta struktura je grafički prikazana na slici 12 za slučaj korištenja IPv4 porodice adresa. Slično izgleda za slučaj IPv6 porodice adresa, no taj slučaj nije ovdje posebno prikazan budući da se u ovoj laboratorijskoj vježbi ne koristi.

Pojedina polja sadrže slijedeće vrijednosti:



Slika 12. Grafički prikaz strukture hostent

- `h_name`

  Pokazivač na primarno ime računala zajedno sa domenom (npr. pinus.cc.fer.hr). Ime je terminirano sa karakterom `\0`.

- `h_aliases`

  Mnoga računala na Internetu imaju više od jednog imena od kojih se jedno naziva primarnim, a ostala su alternativna imena ili aliasi. Ovo polje je pokazivač na niz pokazivača. Svaki pokazivač u tom polju pokazuje na jedno alternativno ime računala. Npr., primarno ime poslužitelja na ZEMRIS-u je `gandalf.zemris.fer.hr` i na to ime će pokazivati prvo polje strukture. Međutim, jedno od aliasa poslužitelja je i ime `www.zemris.fer.hr`, te na njega pokazuje jedan od pokazivača u ovom polju. Kao i u prethodnom slučaju i ovdje je svako ime terminirano znakom `\0`.

- `h_addrtype`

  Tip vraćenih adresa. Moguće vrijednosti su `AF_INET` i `AF_INET6`. U našem slučaju koristi se samo porodica `AF_INET`.

- `h_length`

  Duljina pojedine adrese. Za slučaj IPv4 vrijednost ovog polja je 4 budući da su te adrese veličine četiri okteta, dok je za IPv6 vrijednost tog polja 16.

- `h_addr_list`

  Svako računalo osim što može imati više imena vezanih za jednu IP adresu, može također imati više IP adresa vezanih uz jedno ime. Zbog toga je ovo pokazivač na polje pokazivača od kojih svaki pokazuje na jednu IP adresu računala. Ta vrijednost se može direktno preslikati u polje `sin_addr.s_addr` strukture `sockaddr_in`. Treba primjetiti da polje `h_length` u stvari određuje duljinu svakog od elemenata na koji pokazuje ovo polje, `h_addr_list`.

DON'T PANIC

Strukturu `hostent` popunjavaju funkcije `gethostbyname()` i `gethostbyaddr()`. Prva funkcija, `gethostbyname()` ima sljedeći prototip:

```
struct hostent *gethostbyname(const char *name);
```

Parametar te funkcije je ime računala čiju IP adresu tražimo, no također može biti ASCII zapis IP adrese. S druge strane, funkcija `gethostbyaddr()` uzima IP adresu računala te vraća popunjenu strukturu `hostent`. Prototip te funkcije je:

```
struct hostent *gethostbyaddr(const char *addr, int len,
                  int type);
```

Prvi parametar je struktura `sockaddr_in`, drugi parametar je duljina proslijeđene adrese, a treći je vrsta adrese, tj. u slučaju ove laboratorijske vježbe IPv4 (`AF_INET`). Primjetite da je za korištenje ove funkcije potrebno dodatno uključiti i `sys/socket.h` zbog tih konstanti.

Obje funkcije u slučaju greške vraćaju `NULL`, a kôd greške postavljaju u varijablu `h_errno`. Neke od bitnijih grešaka su:

Prijavljena greška	***Značenje***
`HOST_NOT_FOUND`	Zadano računalo nije pronađeno
`NO_ADDRESS/NO_DATA`	Zadano računalo postoji, no nema IP adresu
`NO_RECOVERY`	Označava grešku bez mogućnosti oporavka.
`TRY_AGAIN`	Pojavila se greška privremenog karaktera, može se pokušati ponovo nakon nekog vremena.

4. Upute za izradu programa

Za testiranje programa pogodno je koristiti jednu od postojećih implementacija TFTP protokola. Dvije implementacije se mogu skinuti sa Interneta, a nalaze se na adresi ftp://ftp.mamalinux.com/pub/atftp/ ili http://www.kernel.org/pub/software/network/tftp/. Te implementacije rade na Unix operacijskom sustavu, a prije korištenja potrebno je programe prekompajlirati. Ti programi također se nalaze priloženi na CD-u.

Problem s barem jednom od tih implementacija je u tome što poslužitelj zahtijeva root ovlasti kako bi se prebacio na nekog neprivilegiranog korisnika. Iz tog razloga bolje je prvo razviti poslužitelj i testirati ga sa postojećim klijentom, a tek potom razvijati klijent.

Program na unix operacijskom sustavu koji može pomoći prilikom razvoja TFTP poslužitelja, a i bilo kojeg drugog programa koji koristi pristupne točke je `netstat(8)`. Taj program ispisuje sve aktivne portove na lokalnom računalu. Pokretanjem TFTP servera može se programom `netstat(8)` provjeriti da li se je on stvarno povezao na zadani pristup te da li očekuje zahtijeve na tom pristupu.

Vježba VI – Spojna usluga TCP/IP porodice protokola

1. Uvod

Daleko najčešće korištena mrežna usluga na Internetu je spojna usluga. Ova laboratorijska vježba služi za upoznavanje sa karakteristikama i načinom korištenja te usluge.

2. Zadatak

U prethodnoj laboratorijskoj vježbi zadatak je bio načiniti TFTP klijent i poslužitelj temeljen na bespojnoj usluzi TCP/IP porodice protokola, tj. korištenjem UDP transportnog sloja. Iako bespojna usluga ima svoju primjenu, za većinu aplikacija daleko je pogodnija spojna usluga.

U ovoj laboratorijskoj vježbi potrebno je isprogramirati web poslužitelj koji koristi dosta jednostavan oblik HTTP protokola. HTTP protokol je odabran zbog toga što se koristi kao transportni protokol ne samo za Web već i za neke druge servise i apsolutno je dominantan protokol na Internetu. Dodatno, potrebno je isprogramirati i jednostavan HTTP klijent.

2.1.Specifikacija klijenta

Klijent koji treba načiniti poziva se na sljedeći način:

```
http_client [IPadresa] <url>
```

Prvi argument je opcionalan i u pitanju je IP adresa na kojoj se izvršava poslužitelja. Drugi argument klijentu je standardni HTTP URL. On uvijek počinje sa znakovima `http://`, nakon toga dolazi ime računala. Nakon imena računala dolazi dvotočka te pristup na kojemu sluša poslužitelj. Iza toga dolazi putanja i datoteka koju je potrebno dohvatiti sa poslužitelja. Ako IP adresa nije zadana u komandnoj liniji tada klijent treba funkcijom `gethostbyname()` opisanoj u prethodnoj laboratorijskoj vježbi pronaći IP adresu računala.

Zaglavlje što ga klijent primi od poslužitelja treba ispisati na standardni izlaz za greške (`stderr`), dok tijelo odgovora treba ispisati na standardni izlaz (`stdout`).

Nekoliko primjera poziva klijenta:

```
http_client http://www.zemris.fer.hr/predmeti/mr/
http_client http://www.zemris.fer.hr:80/predmeti/mr
http_client 161.53.65.72 http://www.zemris.fer.hr:30001/index.html
```

Prvi i drugi primjer su potpuno identični budući da ako se ne navede pristup unutar URL-a podrazumijeva se da je on 80, što je standardni pristup web poslužitelja! U drugom primjeru od klijenta se traži da se spoji na IP adresu 161.53.65.72, pristup 30001 i tamo zatraži datoteku index.html sa polužitelja www.zemris.fer.hr.

2.2.Specifikacija poslužitelja

Web poslužitelj (u daljnjem tekstu samo poslužitelj) treba se konfigurira preko konfiguracijske datoteke koja se nalazi u istom direktoriju kao i poslužitelj. Primjer jedne konfiguracijske datoteke prikazan je na sljedećem ispisu:

DON'T PANIC

```
# Pristup na kojemu poslužitelj čeka na zahtijeve
Port 33001

# Datoteka u koju se bilježe svi zahtijevi koji su pristigli
# poslužitelju
LogFile /home/web/httpd.log

# Virtualni poslužitelji koje server prepoznaje. Prvi navedeni
# poslužitelj je ujedno i onaj koji se podrazumijeva u slučaju da
# browser ne navedene eksplicitno poslužitelj, ili u slučaju da ga
# naveden no on ne postoji.
Server www.server1.com /home/web/www.server1.com
Server www.server2.net /home/web/www.server2.net
Server www.server3.org /home/web/www.server3.org
```

Kao što je vidljivo iz primjera svaka linija koja započinje znakom # ili je prazna smatra se komentarom te se ignorira. U suprotnom linija započinje rezerviranom rječju sa odgovarajućim parametrima. Jedino rezervirana riječ `Server` traži dodatno pojašnjenje. Pomoću te konfiguracijske riječi definiraju se tzv. **virtualni poslužitelji**. Ti poslužitelji su virtualni zbog toga što se njihovo postojanje emulira jednim jedinim poslužiteljem, tj. jednom IP adresom. Prvi parametar te konfiguracijske naredbe je ime virtualnog poslužitelja, dok je drugi parametar direktorij u kojemu su smještene datoteke koje pripadaju tom virtualnom poslužitelju. Prilikom pokretanja poslužitelja čita se konfiguracijska datoteka, kreira se pristupna točka i na nju se veže adresa – pri čemu se koristi pristup zadan u konfiguracijskoj datoteci. Nakon toga se očekuju zahtijevi. Kada pristigne zahtijev potrebno je pokrenuti novu kopiju procesa (ili novu dretvu) koja će opslužiti samo taj zahtijev dok će glavni program čekati na nove zahtijeve za vezom. Novi proces (ili dretva) prihvaća zahtijev od pretraživača koji ima sljedeći oblik (izostavljeni su nebitni dijelovi zahtijeva):

```
GET /index.html HTTP/1.1\r\n
Host: www.server2.net\r\n
Connection: close\r\n
User-Agent: Mozilla/4.0\r\n
...
\r\n
```

Prvo je potrebno primjetiti da se svaka linija mora završiti sa ASCII znakovima CRLF što je ozačeno ekvivalentnom C sekvencom koja se koristi u `printf` funkciji. Dodatno se zahtijeva da iza zadnje linije zaglavlja mora doći jedna prazna linija koja označava kraj zaglavlja.

Zahtijev se sastoji od metode koja obavezno dolazi prva, te niza zaglavnih linija. U ovom slučaju upotrebljena je najčešća metoda GET kojom se traži datoteka `index.html`. Osim tražene datoteke klijent označava da razumije verziju 1.1 HTTP protokola. Kao što je rečeno, nakon metode dolazi niz zaglavnih linija koje pobliže definiraju zahtijev. Tako se u navedenom slučaju tražena datoteka nalazi na poslužitelju `www.server2.net`. Veza se nakon ispunjenja zahtijeva treba zatvoriti što je navedeno u liniji koja započinje sa rječju `Connection`. Konačno, klijent se predstavlja u liniji `User-Agent` na osnovu koje poslužitelj može prilagoditi odgovor. To je posebno korisno kada se stranice optimiraju za pojedine pretraživače (Mozilla, Opera, Internet Explorer, itd.).

Nakon što poslužitelj primi zahtijev, konstruira ime datoteke u datotečnom sustavu na osnovu podataka iz konfiguracijske datoteke i pristiglog zahtijeva. U ovom slučaju, ako se pretpostavi

konfiguracijska datoteka dana u gornjem primjeru, puno ime tražene datoteke u datotečnom sustavu je `/home/web/www.server2.net/index.html` te ju proces (dretva) treba vratiti pretraživaču. Odgovor se sastoji od zaglavlja nakon kojega dolazi sadržaj datoteke. Primjer zaglavlja odgovora je:

```
HTTP/1.1 200 OK\r\n
Connection: close\r\n
Date: Thu, 06 Aug 1998 12:00:15 GMT\r\n
Server: MojServer/0.9 (Unix)\r\n
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT\r\n
Content-Type: text/html\r\n
Content-Length: 456\r\n
\r\n
<sadrzaj datoteke>
```

Kao i u slučaju zahtijeva svaka linija zaglavlja mora završavati sa CR LF znakovima, a između zaglavlja i sadržaja mora se nalaziti jedna prazna linija. Odgovor započinje linijom koja definira da poslužitelj podržava HTTP/1.1, te da je tražena datoteka pronađena (***200 OK***). Nakon toga dolazi linija sa vremenom odgovora i poslužitelj se predstavlja. Format datuma je standardiziran dok format imena poslužitelja nije striktno definiran. Polje `Last-Modified` nosi informaciju kada je tražena datoteka zadnji puta mijenjana i ona se koristi u slučaju keširanja sadržaja. Nakon toga vraća se i tip datoteke. Tipovi su predefinirani. Konačno sa `Content-Length` označava se veličina tijela odgovora, tj. datoteke. U slučaju da datoteka nije pronađena, ili nije joj moguće pristupiti poslužitelj mora vratiti HTML dokument koji se nalazi u direktoriju gdje i poslužitelj te koja se zove `error.html`. Za kreiranje odgovora vrijede ista pravila kao i za sve ostale ispravne odgovore osim što prva linija u ovom slučaju ima sljedeći oblik:

```
HTTP/1.1 404 Not Found
```

Nakon što je odgovorio na zahtijev poslužitelj mora upisati podatke o zahtijevu u datoteku specificiranu sa konfiguracijskom linijom `LogFile`. Primjer dva zapisa je:

```
1038159809 127.0.0.1 32825 /home/web/www.server2.net/index.html 200
1038159815 127.0.0.1 32826 /home/web/www.server2.net/index.htm 404
```

Prva kolona je vrijeme zahtijeva. To vrijeme se dobija jednostavnim pozivom funkcije `time(2)`. Druga kolona je IP adresa sa koje je pristigao zahtijev. Potom dolazi pristup sa kojeg je došao zahtijev, datoteka koja je tražena unutar datotečnog sustava, te kod koji opisuje da li je zahtijev uspješno poslužen ili ne.

U nastavku tekstu ukratko je opisana spojna usluga, te su dane upute za korištenje funkcija pristupnih točaka kako bi se ta usluga mogla koristiti u aplikacijskim programima.

3. Korištenje spojna usluga putem pristupnih točka

Za razliku od UDP-a, TCP prije prijenosa podataka zahtijeva uspostavljanje veze između dva entiteta koja komuniciraju. Nadalje, TCP obavlja slijedeće funkcije o kojima je u slučaju UDP-a bilo potrebno da se brine sama aplikacija:

- **Pouzdanost** – svaki niz poslanih okteta će sigurno stići na odredište - osim kada to nije moguće zbog ispada mreže, greške na poslužitelju, itd., ali će u tom slučaju aplikacija biti obaviještena o pojavi greške.

DON'T PANIC

- **Kontrola toka** – TCP pruža kontrolu toka tako da nije moguće za pošiljatelja da zaguši primaoca.
- **Očuvani redoslijed isporuke** – redoslijed slanja je i redoslijed primanja podataka.
- **Ne postoji ekvivalent poruke kao u UDP protokolu** – TCP gleda na poslane podatke kao na niz okteta bez ikakve forme, a vrijeme slanja određuje čitav niz parametara od kojih je samo jedan veličina, tj. popunjenost međusprenika na predajniku.

Slično kao i UDP tako se i TCP aplikacije vežu za određeni pristup. Nužno je zapamtiti da UDP pristupi i TCP pristupi se međusobno **ne** preklapaju, tj. moguće je da jedna UDP aplikacija čeka na, primjerice, pristupu 3044, dok neka sasvim druga TCP aplikacija čeka na tom istom pristupu!

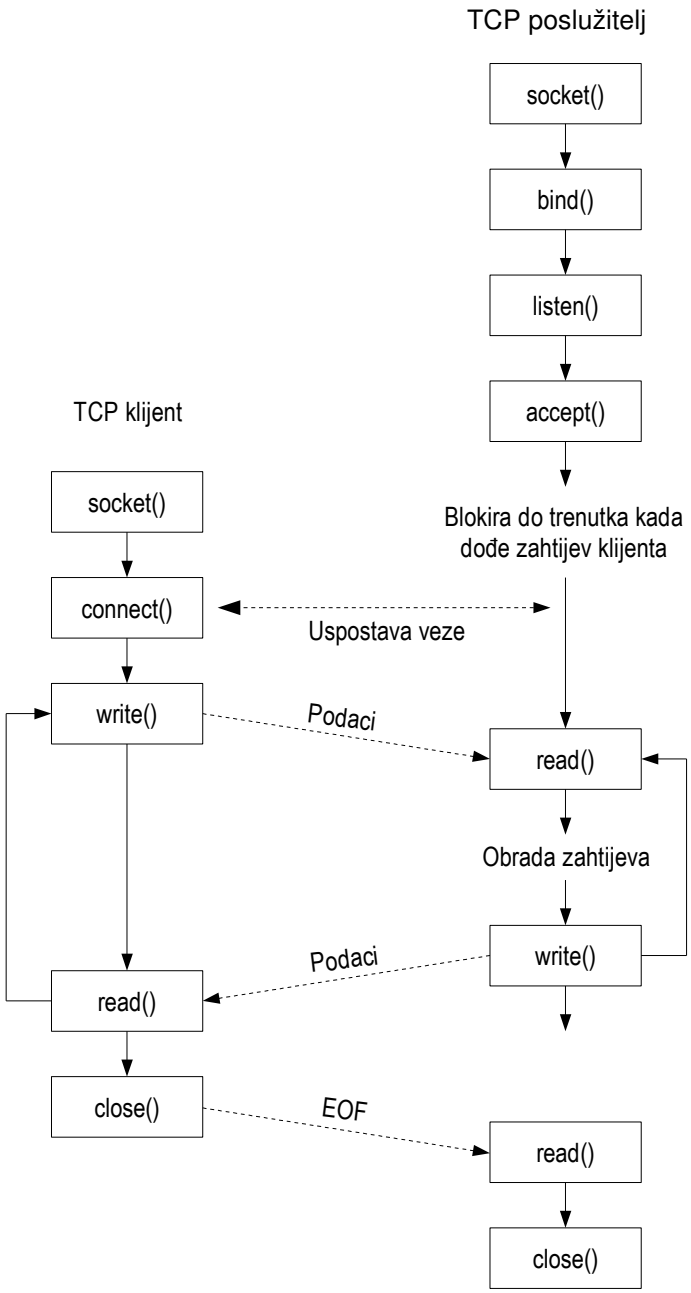
3.1.Programsko sučelje

Problem sa pristupima (port) je u činjenici da moramo unaprijed znati pristup na koji se moramo spojiti kako bi mogli pristupiti usluzi neke aplikacije preko mreže. To može biti problem značajan problem, jer ako primjerice znamo da se na nekom računalu nalazi web poslužitelj, ali taj web poslužitelj čeka na nekom slučajno izabranom pristupu tada nema načina da mu pristupimo osim da pogađamo broj pristupa! Taj problem se rješava na sljedeći način. Svaka aplikacija koja nudi neku uslugu (poslužitelj ili servis) koji se izvršava na računalu sluša na točno predodređenom pristupu (eng. *well-known port*). Ti točno određeni pristupi definirani su za popularne aplikacije, kao primjerice za web, ftp, DNS, itd. Uspostavljanje veze sada ide u sljedećem redoslijedu. Kada se pojavi zahtijev za vezom na tom pristupu, poslužitelj se prebacuje na neki drugi pristup i o tome obaviještava klijenta u odgovoru. Na taj način oslobođen je **poznati** pristup kako bi preko njega mogli pristupiti usluzi i drugi klijenti.

Na slici 1 prikazan je redoslijed poziva funkcija pristupnih točaka za izradu aplikacije koja koristi spojnu uslugu, dok je u nastavku teksta dan opis novih funkcija ili promjenjene semantike u odnosu na prethodne laboratorijske vježbe.

DON'T PANIC

3.1.1. Klijent



Slika 13. Redoslijed poziva funkcija pristupnih točaka za spojnu uslugu

Kako u ovoj laboratorijskoj vježbi trebamo spojnu uslugu to je funkciji `socket()` potrebno proslijediti odgovarajuće parametre kako bi od mreže dobili odgovarajuću uslugu. U prethodnoj laboratorijskoj vježbi funkcija `socket()` pozivana je na sljedeći način:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

U ovoj laboratorijskoj vježbi potrebno je koristiti ljedeći oblik:

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

Razlika je dakle samo u drugom argumentu. S tim drugim argumentom tražimo spojnu uslugu (stream).Treći argument opet označava da se protokol automatski odabere što će u ovom slučaju biti TCP.

Nakon što je kreirana pristupna točka možemo se spojiti na poslužitelj upotrebom funkcije `connect()`. Prototip te funkcije je sljedeći:

```
int connect(int sockfd, const struct sockaddr *serv_addr,
                  socklen_t addrlen);
```

Prvi parametar predstavlja pristupnu točku kreiranu sa funkcijom `socket`, drugi parametar je struktura `sockaddr_in` sa popunjenom adresom poslužitelja (IP adresa i port), dok je zadnji parametar duljina strukture `sockaddr_in`. U slučaju uspješnog uspostavljanja veze sa poslužiteljem ova funkcija vraća nulu, dok u slučaju greške vraća -1 i postavlja vrijednost varijable `errno` na odgovarajuću vrijednost.

Nakon što je uspostavljena veza sa poslužiteljem moguće je započeti sa prijenosom podataka. Za prijenos podataka može se koristiti više različitih funkcija. Prva varijanta su već spomenute funkcije `recvfrom()` i `sendto()`. Druga varijanta su funkcije `recv()` i `send()`. Te funkcije imaju sljedeći prototip:

```
int send(int s, const void *msg, size_t len, int flags);
int recv(int s, void *buf, size_t len, int flags);
```

Funkcije su identične funkcijama `sendto` i `recvfrom` osim što su im uklonjeni argumenti koji se odnose na adresiranje primaoca odnosno u kojima se pohranjuje adresa pošiljitelja. Semantika te dvije funkcije identična je već spomenutim funkcijama. Konačno, treća varijanta je korištenje standardnih funkcija za pisanje i čitanje datoteka, tj. `read` i `write`.

Kada je klijent gotov i želi zatvoriti vezu to može učiniti pozivom funkcije close(). Međutim, ta funkcija samo označava da je ta pristupna točka zatvorena i više se ne može koristiti. Operacijski sustav pokušava poslati sve neposlane podatke i u trenutku kada više nema podatka započinje proces zatvaranja jednog smjera veze. No, nakon toga čeka se da drugi kraj pokrene završetak TCP veze. Ako se želi trenutni prekid veze jedna od mogućnosti je korištenje funkcije shutdown čiji prototip je:

```
int shutdown(int s, int how);
```

Kao prvi argument navodi se pristupna točka koju je potrebno zatvoriti, a kao drugi argument može se proslijediti jedna od konstanti `SHUT_RD`, `SHUT_WR` ili `SHUT_RDWR`. `SHUT_R` onemogućava primanje podataka na pristupnoj točki, `SHUT_WR` onemogućava slanje podataka i `SHUT_RDWR` potpuno onemogućava pristupnu točku.

3.1.2. Poslužitelj

Poslužitelj prvo poziva funkciju `socket()` sa istim parametrima kao i klijent. Natkon toga, poziva funkciju `bind()` čija namjena je da pridruži adresu pristupnoj točki. U ovom slučaju vrijede isti komentari kao i kod poslužitelja koji koristi bespojnu uslugu. Način na koji se inicijaliziraju parametri funkcije `bind()` i način na koji se funkcija poziva su potpuno identični! Nakon pridruživanja adrese slijedi poziv funkcije `listen()`.

DON'T PANIC

Uz pomoć te funkcije poslužitelj definira maksimalnu veličinu reda koji sadrži zahtijeve za vezom. Svi zahtijevi koji pristignu kada je taj red pun će biti odbijeni. U određenim će slučajevima IP stog jednostavno ignorirati zahtijev, tj. neće obavijestiti klijenta o odbijanju veze te će u tom slučaju nakon isteka vremenskog ograničenja zahtijev biti ponovljen. Prototip funkcije je:

```
int listen(int s, int backlog);
```

`s` predstavlja deskriptor pristupne točke kreirane funkcijom `socket`, dok drugi parametar definira maksimalnu veličinu reda. Povratna vrijednost funkcije je 0 u slučaju uspjeha ili -1 u slučaju greške kada je `errno` postavljen na odgovarajući kod greške.

Nakon što je kreirana pristupna točka, pridodjeljena joj je adresa i određen maksimalan broj klijenata koji može čekati u redu za uspostavu veze poslužitelj čeka na klijente. To postiže upotrebom funkcije `accept()`. Ta funkcija blokira poslužitelj do trenutka kada se uspostavi veza sa nekim klijentom. U tom trenutku vraća se **novi** deskriptor kako bi originalna pristupna točka mogla nastaviti dalje prihvaćati zahtijeve za vezom. Nakon toga potrebno je pokrenuti novi proces uz pomoć funkcije `fork()` ili novu dretvu upotrebom funkcije `pthread_create()` koji će obrađivati zahtijeve tog klijenta, a glavni program ponovo čeka na nove zahtijeve za uspostavom veze. Prototip funkcije `accept()` je:

```
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

U drugom argumentu, tipa `sockaddr_in`, po izlasku iz funkcije pohranjena je adresa klijenta dok se u trećem argumentu nalazi duljina strukture `sockaddr`, tj. u ovom slučaju duljina strukture `sockaddr_in`.

Poslužitelj nakon uspostave veze sa klijentom koristi iste funkcije za prijenos podataka i prekid veze kao i klijent.

4. Upute za izradu vježbe

Tijekom izrade ove laboratorijske vježbe može biti određenih problema sa DNS-om! Konkretno, ako u konfiguracijskoj datoteci poslužitelja definiramo nepostojeća simbolička imena računala tada se poslužitelju neće moći pristupiti upotrebom klijenata kao što su Mozilla, Opera ili Internet Explorer – bar ne bez posebnih dodatnih podešavanja. S druge strane, ako se definiraju postojeća imema može se desiti da ta imena pokazuju na neka računala na Internetu koja nemaju nikakve veze sa našim poslužiteljem. U tom slučaju već spomenuti klijenti neće upće pristupati našem poslužitelju. Jedno moguće rješenje tog problema je upisivanje imena u DNS poslužitelj. No, to je u većini slučajeva nemoguće te to rješenje otpada.

Najpogodnije ješenje je da se prvo napravi HTTP klijent koji može pristupati nekoj IP adresi, a u zahtijevu proslijediti proizvoljno simboličko ime. U tom smislu je i specificiran klijent. Zbog tog razloga potrebno je prvo napraviti i ispitati ispravnost klijenta, a tek potom napraviti poslužitelj.

Tijekom kreiranja i čitanja upita i odgovora zgodno je pristupnu točku pretvoriti u strukturu tipa FILE te potom koristiti standardne operacije f* kako bi se čitati linija po linija upita ili odgovora. Za pretvaranje deskriptora u strukturu tipa FILE koristi se funkcija `fdopen(3)` čiji je prototip sljedeći:

```
FILE *fdopen(int fildes, const char *mode);
```

Prvi parametar je pristupna točka, a drugi parametar je način na koji je otvoren deskriptor. U slučaju ove laboratorijske vježbe može se koristiti `r+`, no i bilo koji drugi će također raditi.

DON'T PANIC

Vježba VII – Pozivi udaljenih procedura

1. Uvod

U prethodnim laboratorijskim vježbama za postizanje komunikacije između međusobno razdvojenih procesa bilo je potrebno poznavanje vrsta usluga koje nudi računalna mreža i API-ja uz pomoć kojega se te usluge mogu koristiti. Taj pristup ima više pozitivnih i negativnih posljedica. Kao prvo, omogućena je veća fleksibilnost a time i iskoristivost/efikasnost u korištenju mrežne usluge, dok sa druge strane to od korisnika zahtijeva da dio vremena potrebnog za razvoj same aplikacije utroši na ostvarivanje komunikacije. Nadalje, prilikom prenošenja podataka mreža nije ulazila u njihovu strukturu, već je te podatke tretirala kao "sirovi" niz okteta za koje je u najboljem slučaju garantirana sama isporuka i očuvan redoslijed. U slučaju heterogene okoline, što je u slučaju računalne mreže pravilo, u kojoj se nalaze računala različite duljine riječi i načina pohranjivanja tih riječi u memoriju korisnik je morao uložiti dodatan napor za prevladavanje tih različitosti.

Logičan nastavak je transparentno korištenje mrežnih usluga pri čemu se korisnika što više odvaja od mrežnih detalja te mu na taj način više vremena ostaje za izradu same aplikacije, a istovremeno se skraćuje razvoj aplikacije koja inherentno ima mogućnost distribuiranog rada. Postoji više različitih načina na koje se taj cilj može ostvariti dok će se u ovoj vježbi obraditi jedan od popularnijih - poziv udaljenih procedura (Remote Procedure Call – RPC).

2. Zadatak

U ovoj vježbi potrebno je ostvariti uslugu ispisa poruke na udaljenom terminalu u mreži, korištenjem poziva udaljene procedure. Parametri poziva klijenta uključuju ime čvora u kojemu je rezidentan poslužitelj, poruku koju je potrebno ispisati, te lokalno vrijeme generiranja poziva. Poslužitelj treba ispisati poruku sa vremenom generiranja poziva/poruke, te klijentu vratiti odgovor s parametrima koji obuhvaćaju broj ispisanih znakova poruke i lokalno vrijeme ispisa izvorne poruke. Za dobivanje vremena preporuča se koristiti funkcija `time()`, ali se može upotrijebiti i bilo koja druga.

3. Pozivi udaljenih procedura

Pozivi udaljenih procedura temelje se na klijent/poslužitelj modelu. Pionir u ovom području je Sun Microsystems. Oni su 1988. godine implementirali pozive udaljenih procedura (RPC) u svojem operacijskom sustavu SunOS te isti objavili kao otvoreni standard u nizu RFC-ova. Isti su potom koristili za niz mrežnih usluga od kojih su najpopularijni mrežni informacijski sustav (NIS), i dijeljenje diskova u mrežnom okruženju (NFS). Obje usluge su u međuvremenu dosta unaprijeđene iako se NIS i noviji NIS+ sve brže zamijenjuju LDAP-om, a koriste ih svi proizvođači Unix operacijskih sustava od kojih većina sudjeluje i u razvoju. Osim za Unix postoje i nezavisne implementacije za Windows porodicu. Osim Sun-ove, postojalo je još par implementacija od kojih je zanimljiva DCE-RPC iz 1997. koju je dijelom preuzeo Microsoft, poneštno izmjenio i nadogradio, te prozvao COM/DCOM. Ta implementacija integrirana je u Windows porodicu operacijskih sustava počevši (skromno) sa Windows 95, a vrhunac je doživjela u Windows NT 4. Sam RPC ima ekvivalent i u Javi u vidu RMI-ja. Najnoviji RPC standard ima puni naziv ONC RPC (Open Network Computing RPC) da se što više naglasi njegova otvorenost.

Kako bi se eliminirale razlike koje sa sobom povlači heterogena okolina uveden je univerzalni zapis podataka u vidu XDR-a (External Data Representation) koji je također dokumentiran u RFC-u. Svi podaci neposredno prije prijenosa pretvaraju se u taj format. Za transformaciju

primitivnih tipova podataka iz zapisa računala u univerzalni format i obratno postoje funkcije koje se nalaze u paketu sa ostalim funkcijama i programima koji čine kompletni RPC. Ovdje je potrebno naglasiti da ONC RPC i DCE RPC međusobno nisu interoperabilni.

Osnovna ideja ovog pristupa leži u tome da se pojedine procedure izdvoje iz programa i izvršavaju na zasebnim računalima (iako mogu biti i na lokalnom računalu). Sučelje prema tim procedurama (njihov naziv i ulazni i izlazni parametri) opisuju se u posebnom jeziku. Uz pomoć zasebnog programa (`rpcgen`) automatski se na osnovu opisanog sučelja generira sav potreban kod za pozivanje udaljene procedure sa programom iz kojeg se ona poziva. Taj kod istovremeno omogućava nezavisnost o transportnom sloju. Rezultat tog pristupa je da su potrebne minimalne izmjene u programu. Ipak postoje neki detalji koje je potrebno poštivati i razumijeti prilikom izrade programa koji koriste RPC.

Sve udaljene procedure grupirane su u programe. Svaki program ima dodijeljenu verziju i jedinstveni broj, a također i svaka procedura unutar programa ima svoju verziju. Prilikom pokretanja poslužioca on registrira svoj jedinstveni broj, verziju te sve procedure i njihove verzije sa tzv. *portmapper*-om. Portmapper je usluga koja čeka na portu 111 i povezuje klijente sa poslužiteljima. Za pregled programa i procedura koje su registrirane na određenom računalu služi naredba `rpcinfo`, primjer izvršavanja te naredbe na jednom računalu:

```
$ /usr/sbin/rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp   1024  status
    100024    1   tcp   1024  status
    391002    2   tcp   1025  sgi_fam
 536870913    1   udp   1025
 536870913    1   tcp   1047
```

Kao što je već rečeno generiranje svog potrebnog koda vezanog za mrežu obavlja program `rpcgen`. Taj program na osnovu ulaza u kojemu je opisano sučelje poslužitelja generira sav potreban mrežni kod. Primjer opisa jednog sučelja dan je u sljedećem primjeru:

```
/*
 * dir.x: Remote directory listing protocol
 *
 * This example demostrates the functions of rpcgen
 */

const MAXNAMELEN = 255; /* Max. length of directory entry */

typedef string nametype<MAXNAMELEN>;    /* directory entry */
typedef struct namenode *namelist;      /* link in the listing */

/*
 * A node in the directory listing.
 */
struct namenode {
        nametype name;          /* name of direcotry entry */
        namelist next;          /* next entry */
};
```

DON'T PANIC

```
/*
 * The result of a READDIR operation
 *
 * a truly portable application would use an agreed upon list of
 * error codes rather than (as this sample program does) rely upon
 * passing UNIX errno's back.
 *
 * In this example: The union is used here to discriminate between
 * successful and unsuccessful remote calls.
 */

union readdir_res switch(int rpcerrno) {
        case 0:
                namelist list;          /* No error: return directory
listing */
        default:
                void;                   /* Error occured: nothing
else to return */
};

/*
 * The directory program definition
 */
program DIRPROG {
        version DIRVERS {
                readdir_res READDIR(nametype) = 1;
        } = 1;
} = 0x20000076;
```

Sintaksa opisa sučelja dosta je slična programskom jeziku C. U prethodnom primjeru definirana je funkcija koja vraća sadržaj danog direktorija, a također je definirano da je to prva verzija i programa i funkcije. Jedinstveni broj programa je 0x20000076. Taj broj pripada nerezerviranom području (od 0x200000000 do 0x3fffffff) iz kojega se slobodno može odabrati određen broj no bez garancije da taj broj ne koristi neki drugi program. Ovdje možemo sumirati neke karakteristike tog opisnog jezika te samih procedura:

- Uobičajeno je da se definicija sučelja pohranjuje u datoteku sa ekstenzijom `.x` iako se to ne zahtijeva od strane samih alata.
- Preporuča se imena programa i funkcija pisati sa velikim slovima.
- Svaka funkcija može imati isključivo jedan ulazni i jedan izlazni (povratni) parametar. U slučaju da je nužno proslijediti više argumenata tada se oni grupiraju u strukturu. Taj uvijet je uklonjen u novijim verzijama `rpcgen` alata no u ovim laboratorijskim vježbama će biti rađeno po pravilima koje diktira starija verzija.
- Kako bi razlikovali povratnu vrijednost od indikacije greške korištena je unija, a kao indikator koji element unije je važeći koristi se varijabla `rpcerrno`.
- Za definiranje nizova znakova (stringova) ne koristi se notacija `char *` poznata iz C-a budući da je ona dvosmislena već je uveden poseban tip za tu namijenu `string`. U gornjem primjeru je vidljivo i kako se definira maksimalna duljina niza.

Kod poslužitelja - bivše lokalne procedure - prikazan je u sljedećem ispisu:

```
/*
 * dir_proc.c: remote readdir implementation
 */
#include <dirent.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include "dir.h"                /* Created by rpcgen */

extern char *malloc();

readdir_res *readdir_1_svc(nametype *dirname, struct svc_req *req)
{
        DIR *dirp;
        struct dirent *d;
        namelist nl;
        namelist *nlp;
        static readdir_res res;         /* Must be static */

        /*
         * Open directory
         */
        dirp = opendir (*dirname);
        if (dirp == (DIR *)NULL) {
                res.rpcerrno = errno;
                return (&res);
        }
```

DON'T PANIC

```
        /*
         * Free previous result
         */
        xdr_free (xdr_readdir_res, &res);

         /*
          * Collect directory entries. Memory allocated here is
          * free by
          * xdr_free the next time readdir_1 is called.
         */
         nlp = &res.readdir_res_u.list;
         while (d = readdir(dirp)) {
                  nl = *nlp = (namenode *)malloc(sizeof(namenode));
                  if (nl == (namenode *)NULL) {
                           res.rpcerrno = EAGAIN;
                           closedir(dirp);
                           return (&res);
                  }
                  nl->name = strdup(d->d_name);
                  nlp = &nl->next;
         }

         *nlp = (namelist)NULL;

        /*
         * Return the result
         */
        res.rpcerrno = 0;
        closedir(dirp);
        return (&res);
}
```

U prethodnom kodu možemo primjetiti par specifičnosti vezanih za izradu poslužitelja:

- Nužno je uključiti posebnu zaglavnu datoteku koju generira `rpcgen` kako ćemo to naknadno biti opisano. Ta zaglavna datoteka ima isto ime kao i datoteka u kojoj je opisano sučelje servera s razlikom da je ekstenzija odgovarajuće izmjenjena.
- Ime funkcije je ime definirano u IDL datoteci (datoteci u kojoj je opisano sučelje - `dir.x`) pisano sa svim malim slovima i dodanim sufiksom `_1_svc`. U nekim implementacijama traži se samo sufiks `_1`. Brojka u sufiksu je verzija funkcije!
- Ulazni parametar je pointer na argument proslijeđen od klijenta. Drugi parametar je specifičan za RPC i u njemu se mogu saznati podaci o klijentu.
- Varijabla u koju se pohranjuje povratna vrijednost mora biti statički definirana!
- Varijable tipa unije definirane u IDL datoteci se u C izvornom kodu ne definiraju sa `union readdir_res res`, već sa `readdir_res res`.
- Memoriju rezerviranu sa `malloc` potrebno je osloboditi, ALI nakon što se rezultati vrate klijentu. U prethodnom primjeru prilikom drugog i svakog idućeg poziva oslobađa se memorija zauzeta prethodim pozivom poslužitelja. Za oslobađanje memorije koristi se funkcija `xdr_free` čiji prvi argument je ime funkcije koja se koristi za konverziju tipa definiranog u IDL datoteci, dok je drugi argument pointer na varijablu u kojoj treba osloboditi memoriju. O funkciji za konverziju biti će kasnije riječi. Potrebno je primjetiti da se rekurzivno oslobađa memorija na koju se pokazuje proslijeđena varijabla (drugi argument) ali se ne oslobađa prostor što ju zauzima sama varijabla.

- Povratna vrijednost iz funkcije je pokazivač na statički definiranu varijablu!

Konačno, klijent odnosno glavni program ima sljedeći oblik:

```
/*
 * rls.c: Remote directory listing client
 */

#include "dir.h"

extern int errno;

main (int argc, char **argv)
{
         CLIENT *clnt;
         char *server;
         char *dir;
         readdir_res *result;
         namelist nl;

         if (argc != 3) {
                  fprintf (stderr, "usage: %s  \n", argv[0]);
                  exit (1);
         }

         server = argv[1];
         dir = argv[2];

         /*
          * Create client "handle" used for calling MESSAGEPROG
          * on the server designated on the command line.
          */
         clnt = clnt_create (server, DIRPROG, DIRVERS, "tcp");
         if (clnt == (CLIENT *)NULL) {
                  clnt_pcreateerror(server);
                  exit(1);
         }

         result = readdir_1(&dir, clnt);
         if (result == (readdir_res *)NULL) {
                  clnt_perror (clnt, server);
                  exit(1);
         }

         /*
          * Okay, we successfully called the remote procedure.
          */
         if (result->rpcerrno != 0) {
         /*
          * Remote system error. Print error message and die.
          */
                  errno = result->rpcerrno;
                  perror(dir);
                  exit(1);
         }

         /*
```

```
         * Successfully got a directory listing. Print it.
         */
        for (nl = result->readdir_res_u.list; nl != NULL;
                                              nl = nl->next) {
                printf ("%s\n", nl->name);
        }

        xdr_free(xdr_readdir_res, result);
        clnt_destroy(clnt);
        exit(0);
}
```

Kod klijenta možemo primjetiti sljedeće specifičnosti:

- Kao i kod servera potrebno je uključiti zaglavnu datoteku čije ime nastaje od imena IDL datoteke sa odgovarajuće zamijenjenom ekstenzijom.
- Povezivanje sa serverom obavlja se uz pomoć funkcije `clnt_create` čiji parametri su ime računala na kojemu se nalazi server, verzije programa i procedure koju želimo koristiti i vrsta transportne usluge koju želimo koristiti. Konstante koje označavaju verziju programa i procedure definirane su u zaglavnoj datoteci spomenutoj u prethodnoj točki, a njihove vrijednosti i imena dolaze iz IDL-a. Funkcija `clnt_create` vraća *handle* uz pomoć kojega komuniciramo sa serverom, slično kao što operacije sa nekom datotekom obavljamo preko deskriptora. U slučaju kakve greške povratna vrijednost je `NULL`, a tekst greške dobija se uz pomoć funkcije `clnt_pcreateerror`.
- Poziv udaljene procedure obavlja se sa navođenjem njenog imena, a argumenti u pokazivač na ulazne parametre i handle dobijen pozivom funkcije `clnt_create`.
- Povratna vrijednost je pokazivač na rezultat. Povratna vrijednost `NULL` je indikator da je došlo do greške u komunikaciji sa serverom, a tekst greške dobija se pozivom funkcije `clnt_perror`.
- Kao i kod servera tako je i ovdje potrebno osloboditi memoriju u kojoj se nalazi pohranjena povratna vrijednost. Nužno je primjetiti da to nije ista memorija koja se nalazi na serveru u koju je pohranjen rezultat.
- Konačno, kako bi se oslobodila memorija zatvara se handle koji povezuje klijenta sa poslužiteljem upotrebom funkcije `clnt_destroy`.

Prevođenje prethodnih programa obavlja se u tri koraka. Prvo se pokretanjem `rpcgen`-a iz IDL datoteke dobijaju sljedeće četiri datoteke:

- `dir.h`

  Ovo je zaglavna datoteka koju je potrebno uključiti u kod klijenta i poslužitelja. U općenitom slučaju ime ove datoteke je isto kao i datoteke u kojoj se nalazi definicija sučelja samo je ekstenzija promijenjena.

- `dir_svc.c`

  Datoteka koja sadrži kod nužan da bi server mogao raditi preko mreže. Kao što se primjećuje iz ispisa servera nigdje nema koda vezanog za komunikaciju preko mreže. Taj kod se nalazi u ovoj datoteci.

- `dir_clnt.c`

  Slično kao što je mrežno zavisni kod za server automatski generiran u zasebnu datoteku

DON'T PANIC

tako je i za klijenta generiran i smješten u ovu datoteku.

- `dir_xdr.c`

  Ova datoteka je opcionalna i u njoj se nalaze XDR funkcije za konverziju tipova podataka. Za svaki korisnički definirani/spomenuti tip podataka u IDL datoteci generira se funkcija čije ime ima prefix `xdr_` te potom dolazi ime tipa. Za prethodni primjer ova datoteka sadrži sljedeće funkcije: `xdr_nametype`, `xdr_namelist`, `xdr_namenode`, `xdr_readdir_res`. Te funkcije obavljaju tzv. serijalizaciju podataka neophodnu kako bi se podaci mogli prenijeti preko mreže. Nadalje, svaka od tih funkcija vraća logičku vrijednost `TRUE` u slučaju uspješne konverzije i `FALSE` u slučaju kakve greške.

Na kraju, potrebno je još prevesti sve u izvršne programe: klijent i poslužitelj. To se obavlja upotrebom sljedećih naredbi:

```
$ gcc -o dir_proc dir_proc.c dir_svc.c dir_xdr.c
$ gcc -o rls rls.c dir_clnt.c dir_xdr.c
```

Literatura

1. Internet Software Consortium, *BIND 9 Administrator Reference Manual*, 2001.
2. Kunihiro Ishiguro, et. al., *Quagga*, August 2003.
   http://www.quagga.net/docs/docs-multi/index.html
3. Rusty Russell, *Linux 2.4 NAT HOWTO*, 2002.
   http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html
4. Rusty Russell, Linux 2.4 Packet Filtering HOWTO, 2002.
   http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html
5. Richard Stevens, *UNIX Network Programming: Networking APIs: Sockets and XTI*, Volume 1, Prentice Hall, Englewood Cliffs 1997.
6. Alexey N. Kuznetsov, *IP Command Reference*, Institute for Nuclear Research, Moscow, 1999.
7. VMWare, Inc., *Workstation 4, User's Manual*, Palo Alto CA, 2003.
8. Oskar Andreasson, *Iptables Tutorial 1.1.19*, 2003.
   http://iptables-tutorial.frozentux.net/iptables-tutorial.html

Dodatak A – Osnovni elementi Interneta

1. Pristup Internetu i lokalne računalne mreže

Tri su često korištena načina spajanja računala na mrežu, odnosno, na Internet. To su preko analognog modema, ISDN modema, te uz pomoć lokalnih mreža. Osim tih načina postoji još čitavo mnoštvo drugih od kojih je bitniji ADSL. Svaka veza može biti privremena i stalna. Veze preko analognih i ISDN modema su uglavnom privremene, dok su veze preko lokalnih mreža uglavnom stalne. U ovom tekstu razmatranje je ograničeno samo na lokalne mreže budući da je to okruženje u kojemu se obavljaju vježbe, no većina osnovnih pojmova vrijedi neovisno o načinu spajanja. Ostali načini priključivanja na Internet detaljnije se obrađuju na predavanjima, a puno dodatnih informacija može se naći u knjigama ili na Internetu. Neki pokazivači na dodatne materijale nalaze se i na stranicama predmeta.

Spajanje računala na lokalnu mrežu (LAN – Local Area Network) obavlja se upotrebom mrežnih kartica (FastEthernet, i stariji način Ethernet). Samo spajanje izvodi se UTP kabelom kategorije 5e sa konektorima koji imaju oznaku RJ45 a sliče telefonskim konektorima no nešto su veći od njih. Osim UTP-a, kao zaostatak prošlosti dosta često se sreće i BNC kabel sa tzv. T-konektorima. BNC kabel se inače upotrebljava za prijenos video signala.

Nakon postavljanja kartice u računalo i njenog priključivanja na mrežu (ili preciznije rečeno na uređaj zvan koncentrator odn. preklopnik) potrebno je još instalirati odgovarajuće pogonske programe (*driver*) te podesiti parametre operacijskog sustava kako bi aplikacijski programi mogli koristiti mrežu, tj. mogli komunicirati.

Unix operacijski sustav predstavlja sučelje korisniku preko kratkog imena koje se sastoji od dva dijela. Prvi dio označava fizičko sučelje, dok je drugi dio redni broj tog sučelja. Kao primjer možemo uzeti mrežno sučelje ***eth0*** koje označava prvu Ethernet karticu u računalu. U slučaju da postoji još koja Ethernet kartica ona bi imala oznaku ***eth1***. Osim `eth` sučelja, susreću se dosta često i ***ppp*** mrežna sučelja koja se odnose na serijsku vezu i koja će se također koristiti u ovim laboratorijskim vježbama. Za `ppp` sučelja vrijede ista pravila numeracije kao i za Ethernet sučelja.

2. Internet

Internet je spoj mnoštva lokalnih mreža i računala, a ovisno o tome što je konkretno spojeno možemo razgraničiti četiri pojma:

1. **Internet** – Internet (sa velikim početnim slovom I) označava konkretnu svjetsku mrežu koju svatko od nas koristi kako bi slao elektroničku poštu, tražio informacije na web-u, itd.
2. **internet** – za razliku od Interneta, pod internetom (sa malim početnim slovom i) se označava bilo koje povezivanje lokalnih mreža pri čemu se upotrebljavaju iste tehnologije kao i na Internetu (TCP/IP porodica protokola).
3. **intranet** – pod intranetom se podrazumijeva internet (primjetite malo slovo i) koji je vlasništvo neke ustanove (uglavnom poduzeća), a nije nužno povezan s Internetom. Dosta često intranetu nije dopušten pristup osobama i računalima koja mu ne pripadaju, tj. koja se ne nalaze unutar same organizacija. S druge strane često je kontroliran i pristup s intraneta na Internet.
4. **extranet** – ovaj pojam komplementaran je pojmu intranet i označava sve mrežne elemente koji ne pripadaju intranetu.

Vidimo dakle da se pod ta četiri pojma podrazumijevaju četiri različite stvari između kojih ima određenih preklapanja, ali svima je zajednička ista tehnologija, tj. porodica protokola, koja se zove

TCP/IP porodica protokola.

2.1.Adresiranje na internetu

Kada priključimo računalo na Internet ono mora imati svoju adresu kako bi ostala računala mogla sa njim komunicirati. To ime je jedan 32-bitni broj, koji se radi lakšeg pamćenja te manipuliranja piše u sljedećem obliku:

```
aaa.bbb.ccc.ddd
```

`aaa`, `bbb`, `ccc` i `ddd` su decimalne vrijednosti pojedinog okteta 32-bitne riječi pri čemu je oktet `aaa` najveće težine, a `ddd` oktet najmanje težine. Očito su vrijednosti pojedinih elemenata adrese u rasponu od 0 do 255. Npr., adresu računala koje ima heksadecimalni oblik:

```
0xA135410B
```

pišemo u sljedećem obliku:

```
161.53.65.11
```

Prilikom dodjeljivanja adrese nekom računalu taj broj se ne može odabrati proizvoljno već postoje strogo određena pravila po kojima se on bira, a ta pravila diktira mreža na koju se računalo priključuje.

Uz sam broj, tj. internet adresu, vezan je još jedan parametar koji se zove **mrežna maska**. Radi lakše izvedbe usmjeravanja sve internet adrese podijeljene su u tzv. **mrežni dio** i **računalni dio**. Mrežni dio identificira neku određenu mrežu na Internetu, dok računalni dio određuje pojedino računalo u toj mreži. I opet kao i kod same internet adrese i za mrežnu masku postoje određena pravila po kojima se ona određuju te se ne može odabrati proizvoljno. Za gornji primjer adrese `161.53.65.11` mrežna dio je predodređen i čini ga gornjih 24 bita, tj. od 31. bita sve do (i uključivši) 8. bit, dok se računalni dio sastoji od donjih osam bitova, tj. od 7. do 0. bita. Ta činjenica može se pisati na dva ekvivalentna načina:

```
161.53.65.11/255.255.255.0
```

ili

```
161.53.65.11/24
```

Očito je da se u prvom primjeru mrežni dio označava pomoću broja čiji oblik je sličan samoj adresi pri čemu su u tom broju pojedine binarne znamenke postavljene na 1 ako pripadaju mrežnom dijelu, a na 0 ako pripadaju dijelu koje određuje pojedino računalo. U drugom slučaju, koji je daleko kompaktniji i češće se koristi, upotrebljava se zapis kod kojega se nakon kose crte piše broj bitova koji pripadaju mrežnom dijelu pri čemu se podrazumijeva da se ti bitovi nalaze na najvišim mjestima 32-bitne riječi. U navedenom primjeru radi se o lokalnoj mreži čija adresa je `161.53.65` dok računalo ima broj 11. Prilikom označavanja mreže izostavlja se računalni dio ako nije bitan pa se za mrežu u gornjem primjeru onda piše `161.53.65/24`.

Konkretno, adresa `161.53.65/24` je adresa lokalne mreže na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave (ZEMRIS), a računalo je glavni poslužitelj Zavoda (broj računala je 11).

Svaka mrežna kartica u računalu, a općenito i bilo koji mrežni uređaj uz pomoć kojega se računalo priključuje na internet, mora imati barem jednu adresu i mrežnu masku u obliku sličnom gore

navedenom – no može ih imati i više, tj. jedna mrežna kartica može imati više IP adresa. U tom slučaju govori se o *alias* adresama. Konfiguracija uređaja (mrežne kartice) obavlja se ili automatski ili ručno, što ovisi o samoj mreži.

Postoji određeni niz adresa koje imaju posebno značenje. Neke od tih adresa su sljedeće:

1. Svako računalo ima poseban "uređaj" koji se zove **loopback** i koje ima posebnu adresu, 127.0.0.1/8. To je virtualni uređaj u smislu da fizički ne postoji, tj. operacijski sustav emulira njegovo ponašanje. Nadalje, sama mreža 127/8 ne postoji, tj. ne smije se pojaviti na Internetu!
2. Adresa 0.0.0.0/0 je tzv. nespecificirana adresa i biti će detaljnije objašnjena prilikom opisivanja načina usmjeravanja.
3. Mrežne adrese 10/8, 172.16/12 i 192.168/16 su tzv. privatne adrese i nikada se ne smiju pojaviti na Internetu, no smiju se pojaviti na internetu ili na intranetu! Drugim riječima namijenjene su za lokalne mreže koje nikada neće biti priključene na Internet ili su na njega vezane indirektno.
4. Unutar svake mreže postoje posebne adrese. Za primjer možemo uzeti već spomenutu mrežu 161.53.65/24. U toj mreži adresa 161.53.65.255 je tzv. broadcast adresa. Očito je da se broadcast adresa dobija tako da se bitovi koji pripadaju računalnom dijelu svi postave na 1. Osim te adrese, posebna je i adresa kod koje su svi bitovi na 0.

Dodjeljivanje brojeva mrežama se obavlja preko odgovarajućih institucija koje su hijerarhijski organizirane. Tako npr. CARNet je dobio mrežnu adresu 161.53/16 koja je dodatno podijeljena od strane samog CARNet-a u blokove od po 256 adresa koje se dodjeljuju pojedinim akademskim institucijama i zavodima. Npr. ZEMRIS ima već spomenutu mrežnu adresu 161.53.65/24, dok Računski centar FER-a ima dodijeljenu adresu 161.53.73/24. Dodijeljivanje brojeva pojedinim računalima obavlja mrežni administrator unutar odgovarajuće administrativne jedinice.

2.1.1. IPv6

Gornji oblik adresa su tzv. IPv4 adrese i u upotrebi su od početka Interneta, a danas su apsolutno dominantne.

Naziv IPv4 sastoji se od dva dijela. Prvi dio je skraćenica IP od *Internet Protocol*, a drugi dio označava verziju koja je u ovom slučaju 4 – slovo v označava riječ *Version*. Problem sa tim adresama je njihov mali broj. Iako je teoretski moguće imati 2^{32} adresa (ili brojeva što je ekvivalentno), dobar dio tih brojeva je neupotrebljiv iz razno-raznih razloga. Internet brzo raste, i na njega se osim stolnih računala priključuju i raznorazni drugi uređaji (ručna računala, kućanski uređaji, itd.) a i kompletna telekomunikacijska infrastruktura prelazi na Internet (VoIP telefoni, mobilni telefoni, itd). Očito je problem manjka adresa vrlo ozbiljan iako ne i jedini koji IPv4 ima. Zahvaljujući raznoraznim poboljšanjima i dodacima nestanak IP adresa odgođen je do otprilike 2010. godine, no ipak rješenje već postoji i zove se IPv6, tj. *Internet Protocol Version 6* i uređaji na Internetu polako prelaze na taj protokol. U tom protokolu adrese se sastoje od 128 bita umjesto dosadašnjih 32. Na Internetu već postoji određeni broj mreža koji je prešao na novi protokol ili paralelno koristi i stari i novi protokol. Budući da je nagli prijelaz nemoguć postoje definirani načini postupnog prelaska sa IPv4 na IPv6.

Primjer IPv6 adrese je:

```
fe80:0000:0000:0000:02c0:dfff:fe22:4b85/10
```

To je potpuna IPv6 adresa od 128 bita. Kao i kod IPv4 i ovdje postoji mrežni i računalni dio, a

način označavanja mrežnog broja bitova isti je kao i kod IPv4. Ono što je različito je da se umjesto decimalnog koristi heksadecimalni zapis pri čemu se svih 128 bita dijeli u grupe od po 16 bitova. Grupe se međusobno razdvajaju dvotočkom, a sa lijeve strane se nalaze teži bitovi. Zbog velikog broja nula uveden je i skraćeni način zapisivanja IPv6 adrese. Dva su pravila koja se pri tome koriste i koja je potrebno

1. Vodeće nule se u pojed resa iz primjera ima sljedeći oblik:

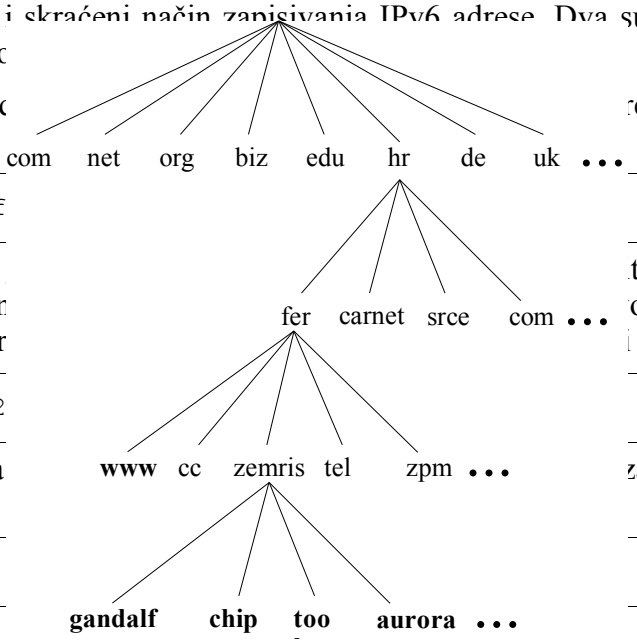
```
fe80:0:0:0:2c0:dfff
```

2. Nadalje, uzastopni niz ti i na to mjesto se stavljaju dvije uzastopn o jednom u cijelom broju! Primjenom tog pr oblik:

```
fe80::2c0:dfff:fe22
```

Kao i u slučaju IPv4 adresa za sada bitna jedino **loopback** adresa čiji oblik je:

```
::1/128
```



Slika 14. Prikaz dijela DNS hijerarhije na Internetu

2.2.Simboličke adrese

Kako je ljudima dosta teško pamtiti brojeve uvedena su simbolička imena. Imena su hijerarhijski organizirana u tzv. **domene**, slično kao što je datotečni sustav hijerarhijski organiziran u direktorije. Cijeli sustav je distribuiran u smislu da se za svaku domenu brine posebna organizacija. Na slici 1 je prikazana hijerarhija domena koja vodi do već spomenutog poslužitelja na ZEMRIS-u. Masnim slovima označena su računala.

Za razliku od datotečnog sustava slika se iščitava odozdo prema gore, te se kao separator upotrebljava točka umjesto kose crte. Tako čitajući punu adresu računala Gandalf, ona ispada:

```
gandalf.zemris.fer.hr.
```

Za web poslužitelj FER-a (čija IP adresa je 161.53.72.111) je to

```
www.fer.hr.
```

Ta puna adresa još se označava i kao FQDN od engleskog izraza *Fully Qualified Domain Name*. Zadnja točka predstavlja tzv. root domenu i može se ispustiti. Prilikom pisanja imena ne postoji razlika između malih i velikih slova, tj. potpuno je ekvivalentno gandalf i GANDALF. U ovom potpunom imenu prva komponenta predstavlja računalo, dok svaka iduća predstavlja mrežu. To je slično (iako obratno) kao i kod IP adresa koje se sastoje od mrežnog i računalnog dijela.

O svakoj domeni brine se posebna organizacija. Tako se o zemris domeni brine administrator na ZEMRIS-u, o FER domeni se brine računski centar, o HR domeni se brine CARNet, itd. Zaduženje administracije je dodjela imena te otvaranje novih poddomena.

Računalima ime ne znači ništa te ona svako ime koje se upiše moraju prevesti u gore spomenuti broj. To prevođenje obavljaju posebni poslužitelji, a aplikacije komuniciraju sa poslužiteljima uz pomoć protokola koji se zove *Domain Name Service* (skraćeno **DNS**). Poslužitelji su također organizirani hijerarhijski, tj. poslužitelj za ZEMRIS domenu nalazi se na ZEMRIS-u, poslužitelj za FER domenu nalazi se u računskom centru, poslužitelj za HR domenu nalazi se u CARNet-u,

itd. Pri pisanju programa postoje posebne funkcije koje to prevođenje obavljaju transparentno pa za korištenje te usluge nije potrebno poznavanje protokola, iako će detalji biti spomenuti u posebnoj vježbi.

Dodatak B – Upute za korištenje VMWare programa

U ovom dodatku dane su osnovne upute za korištenje VMWare programa dovoljne za obavljanje predviđenih laboratorijskih vježbi. Sve dodatne detalje moguće je pronaći u uputama za program koje je moguće skinuti sa Interneta. Te upute nalaze se priložene i na CD-u koji dolazi sa uputama za laboratorijske vježbe.

U laboratorijima je na svakom računalu instaliran program VMWare. Taj program omogućava da se fizičko sklopovlje jednog PC računala istovremeno dijeli sa više operacijskih sustava koji se međusobno ponašaju kao potpuno nezavisna računala. Svako emulirano računalo zasebna je cjelina i skoro potpuno neovisno o fizičkom računalu, te se unutar njega može instalirati bilo koji operacijski sustav. U nastavku teksta će se operacijski sustav koji se izvršava na fizičkom PC-ju zvati *OS domaćin* (host operating system – HOS). Računala koja emulira VMWare nazivati će se *virtualnim računalima*, a operacijski sustav koji se izvršava unutar virtualnog PC-ja nazivati će se *gost operacijskim sustavom* (guest operating system – GOS). Na laboratorijskim vježbama iz Mreža računala kao gost operacijski sustav koristi se isključivo Linux. Kako bi se olakšalo izvođenje laboratorijskih vježbi i izbjegla dugotrajna instalacija operacijskih sustava unaprijed su pripremljene tri instalacije koje se jednostavnim kopiranjem mogu umnožiti proizvoljan broj puta. Te instalacije su u arhivama koje se nalaze na priloženom CD-u u direktoriju *images*. Točna procedura umnožavanja opisana je u nastavku isto kao i način izvođenja potrebnih modifikacija za pojedine vježbe.

1. Kreiranje virtualnog PC računala

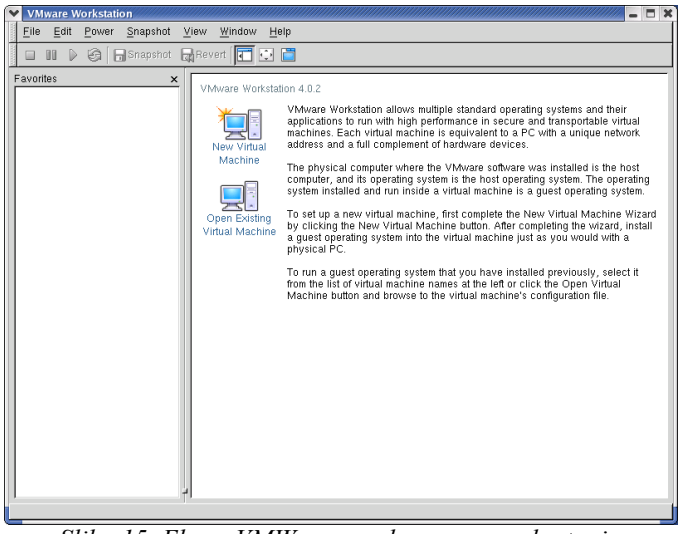
Prva pripremljena instalacija nalazi se u arhivi sa imenom `devel_tpl_rh9.zip` te se u njoj nalazi preinstalirani Linux operacijski sustav sa svom neophodnom programskom podrškom za razvoj programske podrške. Druga instalacija nalazi se u arhivi `router_tpl_mdk9.zip` i u njoj je instalacija koja se koristi za kreiranje usmjernika. Treća datoteka je `srv_tpl_rh9.zip` i koristi se u slučajevima kada je potreban poslužitelj. Poslužitelj je primjerice potreban prilikom konfiguriranja DNS i web usluga.

Tijekom izvođenja laboratorijskih vježbi javlja se potreba za odgovarajućim brojem virtualnih PC računala. Postupak kreiranja virtualnog PC računala uvijek je isti i počinje sa raspakiravanjem odgovarajuće arhive. Primjerice, ako je za izvođenje vježbe potreban određen broj usmjernika tada je potrebno koristiti arhivu `router_tpl_mdk9.tar.gz`. Preporučljivo je arhivu raspakirati u za to posebno kreiran direktorij. Kao ime direktorija zgodno je koristiti naziv virtualnog računala. Recimo, ako nam je potreban usmjernik nazvan `routerA`, tada je zgodno kreirati direktorij `routerA`. Nakon što je arhiva raspakirana potrebno je pokrenuti VMWare program. Slika 15 prikazuje uvodni ekran programa nakon njegova prvog pokretanja.

Upotrebom ikone *Open Existing Virtual Machine*, ili ekvivalentno odabirom opcije *File* → *Open...*, otvara se standardni dijalog koji omogućava dodavanje nekog već postojećeg virtualnog PC računala. Potrebno je pronaći direktorij kreiran u prethodnom koraku i u njemu odabrati datoteku sa ekstenzijom `vmx`. Odabirom te datoteke otvara se nova stranica (*tab*) sa nazivom virtualnog računala. Površina pojedinog virtualnog računala podijeljena je u dva dijela. U lijevom dijelu nalaze se opcije uz pomoć kojih je moguće pokrenuti virtualno računalo, odnosno mijenjati njegovu konfiguraciju. Na desnoj strani nalazi se ispisana konfiguracija virtualnog računala.

Budući da se sva virtualna računala kreiraju iz istog uzorka potrebno je promijeniti naziv kako bi se međusobno razlikovala. Promjena naziva virtualnog PC-ja obavlja se odabirom opcije *Edit virtual machine settings*. U novootvorenom dijalogu potrebno je odabrati stranicu (*tab*) *Options* i

promijeniti polje sa nazivom *Virtual machine name*. Nakon toga mijenja se i natpis na imenu stranice (*tab-a*).



Slika 15. Ekran VMWare-a nakon prvog pokretanja

2. Rad s virtualnim PC računalom

Ispod menija nalaze se kontrole virtualnog PC-a, tj. kontrole uz pomoć kojih se on može upaliti/ugasiti, pokrenuti iz suspendiranog stanja, suspendirati i resetirati. Kontrole su predstavljene standardnim ikonama koje podsjećaju na simbole na audio uređajima. Osim tih kontrola postoji i kontrola *Full Screen* uz pomoć koje je moguće da GOS preuzme kompletni monitor. To je izuzetno praktično ako GOS radi u grafičkom načinu sa velikom rezolucijom (recimo 1024x768) te je nepraktično prikaz imati u prozoru koji nije vidljiv u potpunosti. Tijekom rada s GOS-om VMWare preuzima miša i tipkovnicu kada se klikne mišom unutar GOS-a koji se trenutno izvršava. Od tog trenutka sav ulaz proslijeđuje se gost operacijskom sustavu. Vraćanje tipkovnice i miša HOS-u, isto kao i vraćanje iz stanja u kojemu GOS drži cijeli monitor, obavlja se istovremenim pritiskom na tipke ALT i CTRL.

Pokretanje pojedinog GOS-a obavlja se odabirom njegove stranice (*tab*) te pritiskom tipke *Power On* nakon čega slijedi uobičajna inicijalizacija virtualnog PC-a te pokretanje GOS-a. Instalirani GOS nije svjestan da se on izvršava na virtualnom PC-ju. U slučaju da je potrebno pokrenuti i drugi virtualni PC tada se jednostavno odabere njegov TAB te se odabire tipka *Power On*.

Tipka *Suspend* korisna je kada je iz nekog razloga potrebno zaustaviti izvršavanje virtualnog PC-a ali se ne želi izgubiti trenutno stanje. Recimo, ako se neka laboratorijska vježba ne može obaviti u jednom terminu tada je moguće suspendirati računalo, te se idući puta rad može nastaviti kao da nije prekidan.

Virtualno računalo **ne smije** se naglo isključiti odabirom tipke *Power Off*. Prije gašenja virtualnog računala potrebno se je ulogirati kao root te zadati naredbu **`halt`**. Ta naredba započinje gašenje računala i u trenutku kada se ispiše **`System halted`** može se isključiti virtualno računalo. U slučaju da se ipak zbog nekog razloga računalo ne ugasi propisanim načinom prilikom pokretanja operacijskog sustava to će biti dojavljeno te je potrebno dovesti sustav u ispravno stanje ili napraviti novi primjerak sustava.

2.1.Popravljanje sustava

Prilikom pokretanja sustava nakon njegova naglog gašenja potrebno ga je dovesti u konzistentno

stanje. To znači da je potrebno pokrenuti provjeru konzistencije datotečnog sustava. Uglavnom se ta provjera pokreće automatski prilikom starta ali u slučaju većih oštećenja automatski postupak provjere ne mora proći i potrebno je ručno pokrenuti provjerku konzistencije. U svakom slučaju nakon što sustav nakon provjere završi sa inicijalizacijom preporučljivo ga je resetirati korištenjem naredbe `restart`.

2.2.Spremanje konfiguracije

U slučaju prestanka rada u laboratoriju te planiranog kasnijeg nastavka obavljanja vježbe javlja se potreba za spremanjem svih napravljenih izmjena na virtualnom PC-ju. To je nužno budući da u slučajevima generičkog korisničkog računa na kojma se rade labosi bilo tko može pokrenuti virtualni PC i na njemu imati root ovlasti. Ipak, i uprkos vlastitom korisničkom računu preporučljivo je napraviti rezervnu kopiju. Postoji nekoliko načina na koje je to moguće izvesti, no svi oni podrazumijevaju prvo arhiviranje konfiguracijskih ili programskih datoteka. Prvo je objašnjen način na koji se može arhivirati neki direktorij ili datoteka, a potom su objašnjeni načini na koji se može prebaciti arhiva na neko sigurno mjesto.

Arhiviranje se pod Linux-om može obaviti upotrebom programa `tar`. Sintaksa poziva tog programa je za kreiranje arhive je sljedeća:

```
tar czf <ime arhive>.tar.gz <popis datoteka i direktorija>
```

Ta naredba kreira arhivu sa ektenzijom `.tar.gz` te u nju pohranjuje sve navedene datoteke, direktorije. Prilikom pakiranja direktorija također je u arhivu uključen i sav sadržaj direktorija.

Kako bi se arhiva raspakirala u tekućem direktoriju potrebno je koristiti se sljedećim oblikom naredbe `tar`:

```
tar xzf <ime arhive>.tar.gz
```

Nakon što je arhiva kreirana postoje sljedeće varijante uz pomoć koje se ta arhiva može prebaciti na sigurno mjesto:

1. Ako virtualno PC računalo ima disketnu jedinicu moguće je koristiti m-naredbe da se ta arhiva prebaci na disketu. m-naredbe su u stvari naredbe koje imaju ista imena kao u DOS-u ali im svima prethodi slovo m. Primjerice, naredba za kopiranje je `mcopy`, naredba za brisanje je `mdel`, itd. Sve te naredbe prihvaćaju sličnu sintaksu kao i ekvivalentne DOS naredbe. Primjer kopiranja datoteke u tekući direktorij na disketu u disketnoj jedinici A:

   ```
   mcopy datoteka.tar.gz a:
   ```

   Kopiranje sa diskete u lokalni direktorij:

   ```
   mcopy a:datoteka.tar.gz .
   ```

   Ako virtualno računalo nema disketnu jedinicu tada ga je potrebno propisno ugasiti (naredba halt), te potom u konfiguraciji računala dodati disketnu jedinicu. Nakon što je disketna jedinica dodana potrebno je ponovo uključiti virtualno računalo.

2. Jedna od mrežnih kartica se putem NAT-a priključi na mrežnu karticu računala domaćina (kako se to obavlja opisano je u dodatku koji se bavi VMWare programom). Nakon toga potrebno je u virtualnom PC-ju pokrenuti naredbu `dhclient` i nakon što se ta naredba izvrši moguće je pristupiti računalu domaćinu i bilo kojem računalu na Internetu. U tom

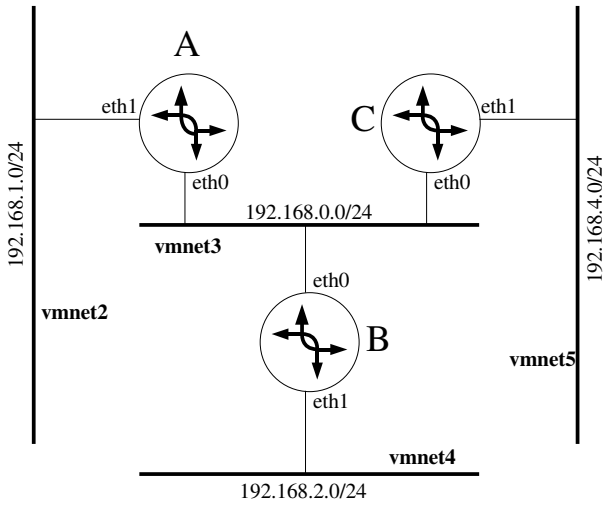
slučaju može se putem FTP klijenta prebaciti arhiva na neko drugo računalo na mreži (primjerice *pinus.cc.fer.hr*).

3. Mrežne postavke virtualnih PC računala

Tijekom izvođenja laboratorijskih vježbi javlja se potreba za odgovarajućim brojem Ethernet mrežnih priključaka i serijskih sučelja na usmjernicima. U uzorku usmjernika kreiran je maksimalan mogući broj serijskih sučelja i ethernet mrežnih sučelja. Veći dio prilagodbe svodi se na brisanje nepotrebnih sučelja,

3.1.Ethernet priključci

VMWare može emulirati do 9 međusobno odvojenih Ethernet mreža. Zbog načina na koji rade pojedini protokoli koji se koriste u laboratorijskim vježbama potrebno je pripaziti da se ne pomješaju dvije mreže koje bi trebale biti fizički različite. Kao primjer, uzmimo konfiguraciju prikazanu na slici 16. Na toj slici nalaze se četiri različita Ethernet segmenta, tj. četiri različite Ethernet mreže. Svaka podebljana linija predstavlja jednu Ethernet mrežu. Na slici je odmah naznačen i raspon IP adresa koji pripada pojedinoj mreži kao i sučelja uz pomoć kojih su usmjernici spojeni sa pojedinom Ethernet mrežom.



Slika 16. Računalna mreža sa četiri različita Ethernet segmenta

Prilikom kreiranja te konfiguracije potrebno je pripaziti da se promet na ta četiri segmenta ne miješa – na to određeni protokoli mogu biti osjetljivi. To se obavlja tako da se svakom segmentu pridruži druga virtualna mreža. Na slici je odmah za svaki segment prikazano kojoj virtualnoj mreži pripada. Npr., `eth1` sučelje usmjernika A pripada vmnet2 mreži, dok sučelje `eth0` usmjernika C pripada vmnet3 mreži. Mreže vmnet0, vmnet1 i vmnet8 imaju posebno značenje no tijekom izrade laboratorijskih vježbi ono se može ignorirati.

Dodavanje Ethenet sučelja opisano je u idućem potpoglavlju. Promjena parametara Ethernet sučelja i način njegova uklanjanja jednostavan je proces koji dijelom slijedi korake dodavanje novog sučelja. Iz tog razloga ti postpuci nisu zasebno pojašnjavani.

3.1.1. Dodavanje novog Ethernet sučelja

Dodavanje sučelja obavlja se dok je virtualno računalo ugašeno. Odabirom opcije *Edit virtual machine settings* otvara se dijalog unutar kojega se definiraju sklopovske karakteristike virtualnog računala. Za dodavanje novog sklopa potrebno je odabrati tipku *Add...* Odabirom te opcije otvara

se novi prozor. Prvo je potrebno odabrati tip sklopovlja koje je potrebno dodati virtualnom računalu. U našem slučaju to je *Network adapter*. Odabirom opcije *Next* omogućeno nam je da se definira virtualna mreža na koju je potrebno spojiti adapter. Nude se sljedeće mogućnosti:

1. *Bridged*

   Ova konfiguracija povezuje virtualni adapter sa fizičkim i omogućava mu direktu vezu na stvarnu lokalnu mrežu. Kako bi to ispravno radilo nužno je imati važeću i nekorištenu IP adresu lokalne mreže na kojoj se nalazi fizičko računalo.

2. *NAT*

   Ova opcija slična je prethodnoj sa razlikom da virtualni adapter indirektno preuzima postojeću IP adresu fizičkog sučelja. Ipak, nužno je dati privatnu adresu sučelju koja mora biti identična privatnoj mreži dodjeljenoj virtualnom sučelju fizičkog računala.

3. *Host-only*

   Odabirom ove opcije virtualni adapter ne može pristupiti fizičkoj mreži, ali može komunicirati sa računalom domaćinom.

4. *Custom*

   Ova opcija omogućava odabir virtualne mreže na kojoj će se nalaziti adapter, te je ta opcija za ove laboratorijske vježbe najzanimljivija.

Maksimalan broj Ethernet sučelja koje pojedini virutalni PC može imati je 4. Do tri sučelja moguće je kreirati korištenjem gore opisanog načina, dok je četvrto sučelje moguće dodati isključivo mijenjanjem konfiguracijske datoteke virtualnog PC-ja.

3.2.Serijska sučelja

Vrlo često se udaljene mreže povezuju serijskim sučeljem. U ovim laboratorijskim vježbama koriste se serijska sučelja virtualnih PC računala na kojima se podaci prenose putem PPP protokola. Inicijalizacija serijskog sučelja ponešto je kompliciranija zbog toga što osim dodavanja samog sklopovlja virtualnom PC-ju uključuje i određena podešavanja u samom operacijskom sustavu.

Prvi korak prilikom spajanja dva usmjernika putem serijske veze je dodavanje virtualnih serijskih sučelja usmjernicima. To se obavlja na sljedeći način:

1. Dok je virtualni PC isključen potrebno je odabrati opciju *Edit virtual machine settings*, te u dijalogu koji se pojavi pritisnuti tipku *Add*.
2. U novootvorenom dijalogu treba označiti opciju *Serial port* i pritisnuti tipku *Next*.
3. Odabrati opciju *Use named pipe*, te potom ponovo pritisnuti tipku *Next*.
4. U polje ispod natpisa *Named pipe* navesti puno ime i putanju do objekta na datotečnom sustavu koji će se koristiti kao posrednik u komunikaciji. U listi ispod tog natpisa odabrati opciju *This end is the client*. U slučaju Windows operacijskih sustava ime cjevovoda mora imati poseban oblik: `\\.\pipe\<ime cjevovoda>` jer u suprotnom Windows-i pokušavaju kreirati običnu datoteku.
5. Pritisnuti tipku *Finish*.

U tom trenutku virtualnom PC računalu dodano je serijsko sučelje. Drugom virtualnom računalu

serijsko sučelje dodaje se na identičan način osim što se umjesto opcije *This end is the client* odabire opcija *This end is the server*.

Nakon što je "sklopovski" dio rješen, potrebno je još prije korištenja inicijalizirati sučelje pod operacijskim sustavom. Inicijalizacija se obavlja pokretanjem programa `pppd`. Nakon prijave, na oba usmjenika potrebno je izvršiti sljedeću naredbu:

```
[root@localhost root]# pppd ttyS0 passive 192.168.2.1: \
                                    netmask 255.255.255.252
```

Parametri te naredbe imaju sljedeće značenje:

- `ttyS0`

  serijska sučelja na Linux operacijskom sustavu imaju imena `ttyS0` do `ttyS4`. Prilikom pokretanja `pppd` programa potrebno mu je reći na koje sučelje se treba povezati. U našem slučaju povezivanje se obavlja preko prvog serijskog pristupa.

- `passive`

  ova opcija označava `pppd` programu da u slučaju neuspješnog pokušaja uspostave veze sa drugim krajem ne prekida rad već pasivno čeka da druga strana inicijalizira vezu.

- `192.168.2.1:`

  Sa ovim parametrom definirana je IP adresa lokalnog sučelja.

- `netmask 255.255.255.252`

  Mrežna maska IP adrese lokalnog sučelja.

Da li je pokretanje naredbe pppd uspješno obavljeno može se provjeriti ispisom aktivnih procesa (naredba ps x), te ako se program `pppd` nalazi u ispisu vrlo vjerojatno je sve prošlo bez problema.

Kako provjeriti da li su ispravno spojena dva serijska porta?

Prilikom rada sa serijskim portovima javlja se potreba za provjerom da li su ispravno spojeni. Recimo da treba provjeriti da li su dva `ttyS0` sučelja međusobno dobro spojena. To se obavlja na sljedeći način. Na jednoj strani potrebno je izvršiti naredbu:

```
cat /dev/ttyS0
```

a na drugoj strani

```
echo "RADI" > /dev/ttyS0
```

Ako se na prvoj strani pojavi niz pod navodnicima tada je sve u redu.

Dodatak C – Mrežne naredbe Linux operacijskog sustava

1. ip

Pomoću naredbe `ip` postavljaju se adrese sučelja, rute i ostali parametri mrežnog podsustava na Linux operacijskom sustavu. Naredba `ip` specifična je za Linux. Standardna naredba za postavljanje parametara sučelja na Unix porodici operacijskih sustava je naredba `ifconfig`, za postavljanje ruta to je naredba `route` i za pregledavanje ruta naredba `netstat`. Iako se te naredbe nalaze na svakom Unix operacijskom sustavu, i sa izmjenjenim imenima na Windows operacijskim sustavima, sintakse njihova poziva mogu se značajno razlikovati. Kako je naredba `ip` daleko bolje prilagođena mrežnom podsustavu Linux operacijskog sustava, te je konzistentnija s parametrima prilikom poziva, to se ona primarno koristi tijekom ovih laboratorijskih vježbi. U nastavku su dane upute kako postići određene radnje koristeći tu naredbu.

1.1.Pregled sučelja

Kako bi se vidjelo koja sve sučelja kernel vidi, te status tih sučelja koristi se sljedeći oblik naredbe `ip`:

```
ip link sh
```

Ta naredba izlistati će sva postojeća sučelja, no u slučaju da želimo podatke samo o određenom sučelju koristi se sljedeći njen oblik:

```
ip link sh <sucelje>
```

1.2.Pregled, postavljanje i uklanjanje adrese sučelja

Popis dodjeljenih IP adresa sučeljima dobija se izvršavanjem sljedećeg oblika naredbe ip:

```
ip addr sh
```

Postavljanje adrese nekog sučelja obavlja se sljedećim oblikom naredbe ip:

```
ip addr add <ip adresa>[/<mrezna maska>] dev <mrezni uredaj>
```

U slučaju da je sučelje aktivno tada se dodavanjem adrese automatski dodavaju i rute za tu adresu u tablicu usmjeravanja. U suprotnom, ako sučelje nije aktivno, rute se ne mijenjaju. Ako se izostavi mrežna maska tada se podrazumijeva vrijednost 32.

Primjer postavljanja adrese 192.168.2.1 sa mrežnom maskom 255.255.255.0 mrežnom uređaju `eth0`:

```
ip addr add 192.168.2.1/24 dev eth0
```

Uklanjanje adrese sučelja obavlja se istim oblikom naredbe `ip` osim što je umjesto riječi `add` potrebno koristiti riječ `del`.

1.3.Aktiviranje i deaktiviranje sučelja

Postavljanje adrese sučelju ne aktivira sučelje. To je potrebno učiniti koristeći sljedeći oblik naredbe `ip`:

```
ip link set <mrežni uredaj> up
```

Za deaktiviranje sučelja potrebno je umjesto riječi `up` koristiti riječ `down`.

Aktiviranjem mrežnog uređaja automatski se dodaju rute u tablicu usmjeravanja, dok se deaktiviranjem rute uklanjaju.

1.4.Dodavanje rute

Točan oblik poziva naredbe ovisi o vrsti rute koja se dodaje. Prvi slučaj je kada ruta upućuje na točno određeno računalo ili usmjernik spojen na neko sučelje. U tom slučaju koristi se sljedeći oblik naredbe:

```
ip route add <mreza>/<mrezna_maska> via <iduca_ip_adresa>
```

To znači da kada pristigne neki paket čija se odredišna adresa poklapa sa mrežom navedenom u gornjoj naredbi tada se taj paket šalje na iduću IP adresu. Ovaj oblik uglavnom se koristi u slučajevima kada odredište nije direktno spojeno na usmjernik. Drugim riječima paket mora proći još barem jedan usmjernik na svom putu do odredišta. Prilikom korištenja tog oblika naredbe nužno je da je zadana IP adresa `iduca_ip_adresa` dostupna.

U slučajevima kada su odredišta direktno spojena na sučelje usmjernika koristi se sljedeći oblik naredbe

```
ip route add <mreza>/<mrezna_maska> dev <sucelje>
```

Primjer kada bi se koristila ta naredba je slučaj kada dodajemo rutu za lokalnu Ethernet mrežu.

Poseban slučaj je tzv. *default* ruta odnosno podrazumijevana ruta. Ta ruta se uglavnom koristi na računalima koja imaju samo jednu vezu prema ostatku interneta. Oblik te rute je 0/0, a može se upotrijebiti i rezervirana riječ `default`. Način dodavanja podrazumijevane rute je:

```
ip route add default via <IP adresa usmjernika>
```

Usmjernik kojemu se proslijeđuje paket mora biti na lokalnoj mreži!

2. ping

Naredba `ping(8)` je najelementarnija i najčešće korištena naredba za traženje greške u mreži. Ta naredba radi na mrežnom sloju ISO/OSI modela, a za svoj rad koristi ICMP protokol. Osnovni način pozivanja te naredbe je sljedeći:

```
# ping <ip adresa>
```

Ako se naredba pozove prethodno navedenim načinom tada se ona izvršava dok god se njeno izvršavanje ne prekine pritiskom na tipke Ctrl-C. Uspješnim izvršavanjem te naredbe sigurni smo da mrežni sloj ispravno funkcionira, tj. da IP paketi bez problema prolaze od računala na kojemu se naredba `ping(8)` izvršava, pa do odredišne IP adrese. Tada je također sigurno da su problemi – ako postoje – na transportnom ili nekom od viših slojeva.

U slučaju da naredba ne radi tada je problem izoliran od fizičkog pa do mrežnog sloja. Primjerice, moguće je da ne radi lokalno sučelje, da je negdje prekinut fizički vodič, da neki od usmjernika ne radi, itd.

Treba primjetiti da u slučaju da se koristi simboličko ime kao parametar naredbi `ping(8)` te ako naredba `ping(8)` ne radi moguć je i problem u DNS-u. DNS se isključuje ako se koristi IP adresa.

Primjer izvršavanja naredbe:

```
# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) from 10.0.0.2 : 56(84) bytes of data.
64 bytes from 10.0.0.2 : icmp_seq=1 ttl=64 time=0.286 ms
64 bytes from 10.0.0.2 : icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2 : icmp_seq=3 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2 : icmp_seq=4 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2 : icmp_seq=5 ttl=64 time=0.093 ms

---  10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% loss, time 3999ms
rtt min/avg/max/mdev = 0.093/0.145/0.286/0.071 ms
#
```

Kao što je rečeno, izvršavanje naredbe `ping(8)` mora se prekinuti sa Ctrl-C. Kada naredba `ping(8)` detektira zahtijev za prekidom izvršavanja ispisuje ukupnu statistiku. U prethodnom primjeru može se vidjeti da su svi primljeni paketi bili duljine 64 okteta. Prvi paket je stigao do odredišta i natrag za 0.286 ms, drugi za 0.120 ms, itd. U ukupnoj statistici se može vidjeti da je poslano pet paketa i da je primljeno pet odgovora (0% gubitaka). Prikazano je i minimalno, maksimalno i prosječno vrijeme potrebno za slanje paketa te za prijem odgovora.

3. traceroute

Naredbe `ping(8)` je u stanju detektirati da li postoji povezanost između računala na kojoj se naredba izvršava i odredišnog računala zadanog kao parametra. Za razliku od nje naredba `traceroute` pokazuje put koji prolazi paket na putu od polaznog računala pa do odredišta. Sintaksa izvršavanja te naredbe ista je kao za naredbzu `ping(8)`, tj.

```
traceroute <odredisna_ip_adresa>
```

Primjer izvršavanja te naredbe:

Za izvršavanje naredbe `ping(8)` nisu potrebne posebne dozvole, no naredbu `traceroute(8)` može izvršavati isključivo root.

4. tcpdump

Uz pomoć ovog programa može se pratiti sav promet koji prolazi mrežnom karticom. Naredba uključuje jednostavan, ali vrlo dobar jezik uz pomoć kojega se mogu specificirati izrazi koji selektiraju određen promet koji se prati. Primjeri tih izraza dani su u nastavku za promatranje ARP, UDP i TCP prometa.

4.1.ARP

Promatranje ARP prometa obavlja se pokretanjem sljedećeg oblika tcpdump naredbe:

```
tcpdump -i eth0 arp
```

4.2.UDP

Pretpostavimo da poslužitelj čeka na pristupu 10000. Također, pretpostavimo da se sva komunikacija obavlja preko loopback sučelja, tj. sučelja lo. U tom slučaju kako bi se promatrao UDP promet generiran od strane TFTP klijenta i poslužitelja potrebno je izvršiti sljedeću naredbu kao root:

```
/usr/sbin/tcpdump -i lo 'port 10000 and udp'
```

Na naredba kaže `tcpdump` programu da se spoji na sučelje `lo` te prati sve pakete UDP protokola koji pristupaju pristupu 10000 ili dolaze sa tog pristupa.

4.3.TCP

Praćenje TCP prometa slično je UDP osim što se navodi ključna riječ tcp. Primjer praćenja svih veza koje dolaze na pristup 80 ili idu na njega na nekom drugom računalu:

```
/usr/sbin/tcpdump -i eth0 'port 80 and tcp'
```

Efektivno se korištenjem prethodne naredbe prati promet sa Web poslužiteljima.

5. iptraf

Ovo je program analogan `tcpdump(8)` programu, ali koristi nešto bolje sučelje.

6. netstat

`netstat` je naredba koja daje popis otvorenih pristupa na računalu. Primjerice, kako bi se saznali svi TCP pristupi koji su na računalu otvoreni i u stanju slušanja (LISTEN) koristi se sljedeći oblik naredbe:

```
netstat -lt
```

Za pregled uspostavljenih veza potrebno je ukloniti opciju l, tj. naredba se poziva na sljedeći način:

```
netstat -t
```

Ako je potrebno vidjeti pristupne točke koje pripadaju UDP protokolu umjesto opcije `t` koristi se opcija `u`.

7. nslookup/host/dig

Inicijalizacija DNS-a

Kao što je već navedeno u uvodu pripreme IP adrese nisu naročito pogodne za korisnike internet-a te su zbog toga uvedena simbolička imena hijerarhijski organizirana po domenama. Kako bi računalo moglo pretvoriti upisano ime u IP adresu potrebno mu je reći koji server (ili više njih) treba kontaktirati kako bi obavili tu pretvorbu. Na Unix operacijskom sustavu to se obavlja upisivanjem sljedećih linija u datoteku `/etc/resolv.conf`

```
domain zemris.fer.hr
search fer.hr hr
```

```
nameserver 161.53.65.11
```

Linija koja započinje riječju `nameserver` određuje server koji će biti kontaktiran za pretvorbu imena u IP adresu. Server mora biti zadan preko IP adrese! U datoteci se može nalaziti više linija koje definiraju DNS servere i u tom slučaju će serveri biti sinkrono kontaktirani u navedenom redoslijedu do trenutka dok prvi od njih ne odgovori. Linija koja započinje sa rječju `domain` daje informaciju o domeni unutar koje se nalazi računalo na kojemu se nalazi `/etc/resolv.conf`. Ta linija ako nije zadana određuje se posredno, no ona omogućava da se koriste kratka imena. U datoteci se može pojaviti maksimalno jedna takva direktiva. Zahvaljujući toj direktivi moguće je, između ostalog, umjesto gandalf.zemris.fer.hr pisati samo gandalf. Linija koja započinje sa `search` određuje koje domene će se pretraživati u slučaju da se upotrebljava kratko ime i ono se ne nalazi u lokalnoj domeni. Lista se pretražuje u navedenom redoslijedu dok se ne dobije prvi odgovor. U datoteci je moguće imati više tih direktiva.

Provjera ispravnosti DNS-a obavlja se koristeći narebu `nslookup` ili `host` i `dig`. Ovdje je opisana starija naredba `nslookup` koja se nalazi instalirana na svakom operacijskom sustavu, dok su `host` i `dig` novije i opisane su u on-line upustvima. Kako bi vidjeli da li DNS radi može se pokušati saznati IP bilo kojeg računala na lokalnoj mreži. Recimo da nam je poznato da se računalo Soc nalazi na lokalnoj mreži te želimo provjeriti njegovu IP adresu:

```
# nslookup soc
Server:         161.53.65.11
Address:        161.53.65.11#53

Name:   gandalf.ZEMRIS.FER.HR
Address: 161.53.65.21
#
```

Naredba `nslookup` je kontaktirala DNS server sa IP adresom 161.53.65.11 i dobila odgovor da je IP adresa gandalf-a 161.53.65.11, a njegovo puno ime je gandalf.ZEMRIS.FER.HR. Ako je DNS server ispravno konfiguriran (što bi morao biti), tada možemo saznati IP adresu bilo kojeg računala na Internetu. Recimo, adresa od Amazona je:

```
# nslookup www.amazon.com
Server:         161.53.65.11
Address:        161.53.65.11#53

Non-authoritative answer:
Name:   www.amazon.com
Address: 207.171.185.16
#
```

Dobijen je odgovor da Amazonov web ima IP adresu 207.171.185.16.

Drugi argument naredbe `nslookup` može biti DNS server kojega želimo da ta naredba kontaktira umjesto onoga postavljenoga u datoteci `/etc/resolv.conf`! Recimo da želimo saznati adresu web poslužitelja sa SRCE-a i to direktno od DNS poslužitelja sa SRCE-a. DNS poslužitelj SRCE-a je dns.srce.hr, pa prema tome naredba ima sljedeći oblik:

```
# nslookup www.srce.hr dns.srce.hr
Server:         dns.srce.hr
```

```
Address:        161.53.3.7#53

www.srce.hr     canonical name = regoc.srce.hr.
Name:   regoc.srce.hr
Address: 161.53.2.69#
```

U ovom primjeru ipak je kontaktiran lokalni DNS poslužitelj. Problem je što smo DNS koji treba pitati za pretvorbu zadali u simbličkom obliku. Taj simbolički oblik treba prvo pretvoriti u IP adresu što je obavio lokalni DNS poslužitelj, a zatim ta IP adresa kontaktirana sa upitom za pretvorbu simboličkog imena www.srce.hr u odgovarajuću IP adresu.

Dodatak D – Specifikacija TFTP protokola

Network Working Group K. Sollins
Request For Comments: 1350 MIT
STD: 33 July 1992
Obsoletes: RFC 783

THE TFTP PROTOCOL (REVISION 2)

Status of this Memo

This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

Acknowlegements

The protocol was originally designed by Noel Chiappa, and was redesigned by him, Bob Baldwin and Dave Clark, with comments from Steve Szymanski. The current revision of the document includes modifications stemming from discussions with and suggestions from Larry Allen, Noel Chiappa, Dave Clark, Geoff Cooper, Mike Greenwald, Liza Martin, David Reed, Craig Milo Rogers (of USC-ISI), Kathy Yellick, and the author. The acknowledgement and retransmission scheme was inspired by TCP, and the error mechanism was suggested by PARC's EFTP abort message.

The May, 1992 revision to fix the "Sorcerer's Apprentice" protocol bug [4] and other minor document problems was done by Noel Chiappa.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661.

1. Purpose

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) [2] so it may be used to move files between machines on different networks implementing UDP. (This should not exclude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data.

Three modes of transfer are currently supported: netascii (This is ascii as defined in "USA Standard Code for Information Interchange" [1] with the modifications specified in "Telnet

Protocol Specification" [3].) Note that it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.); octet (This replaces the "binary" mode of previous versions of this document.) raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. (The mail mode is obsolete and should not be implemented or used.) Additional modes can be defined by pairs of cooperating hosts.

Reference [4] (section 4.2) should be consulted for further valuable directives and suggestions on TFTP.

2. Overview of the Protocol

Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes signals termination of a transfer. If a packet gets lost in the network, the intended recipient will timeout and may retransmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet. The sender has to keep just one packet on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Notice that both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, the other sends acknowledgments and receives data.

Most errors cause termination of the connection. An error is signalled by sending an error packet. This packet is not acknowledged, and not retransmitted (i.e., a TFTP server or user may terminate after sending an error message), so the other end of the connection may not get it. Therefore timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events: not being able to satisfy the request (e.g., file not found, access violation, or no such user), receiving a packet which cannot be explained by a delay or duplication in the network (e.g., an incorrectly formed packet), and losing access to a necessary resource (e.g., disk full or access denied during a transfer).

TFTP recognizes only one error condition that does not cause termination, the source port of a received packet being incorrect. In this case, an error packet is sent to the originating host.

This protocol is very restrictive, in order to simplify implementation. For example, the fixed length blocks make allocation straight forward, and the lock step acknowledgement provides flow control and eliminates the need to reorder incoming data packets.

3. Relation to other Protocols

As mentioned TFTP is designed to be implemented on top of the Datagram protocol (UDP). Since Datagram is implemented on the Internet protocol, packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. As shown in Figure 1, the order of the contents of a packet will be: local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header. On the other hand, the source and destination port fields of the Datagram header (its format is given in the appendix) are used by TFTP and the length field reflects the size of the TFTP packet. The transfer identifiers (TID's) used by TFTP are passed to the Datagram layer to be used as ports; therefore they must be between 0 and 65,535. The initialization of TID's is discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the formats of the various types of packets are discussed further in the section on TFTP packets.

Figure 1. Order of Headers

4. Initial Connection Protocol

A transfer is established by sending a request (WRQ to write onto a foreign file system, or RRQ to read from it), and receiving a positive reply, an acknowledgment packet for write, or the first data packet for read. In general an acknowledgment packet will contain the block number of the data packet being acknowledged. Each data packet has associated with it a block number; block numbers are consecutive and begin with one. Since the positive response to a write request is an acknowledgment packet, in this special case the block number will be zero. (Normally, since an acknowledgment packet is acknowledging a data packet, the acknowledgment packet will contain the block number of the data packet being acknowledged.) If the reply is an error packet, then the request has been denied.

In order to create a connection, each end of the connection chooses a TID for itself, to be used for the duration of that connection. The TID's chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate succession is very low. Every packet has associated with it the two TID's of the ends of the connection, the source TID and the destination TID. These TID's are handed to the supporting UDP (or other datagram protocol) as the source and destination ports. A requesting host chooses its source TID as described above, and sends its initial request to the known TID 69 decimal (105 octal) on the serving host. The response to the request, under normal operation, uses a TID chosen by the server as its source TID and the TID chosen for the previous message by the requestor as its destination TID. The two chosen TID's are then used for the remainder of the transfer.

As an example, the following shows the steps used to establish a connection to write a file. Note that WRQ, ACK, and DATA are the names of the write request, acknowledgment, and data types of packets respectively. The appendix contains a similar example for reading a file.

1. Host A sends a "WRQ" to host B with source= A's TID, destination= 69.
2. Host B sends a "ACK" (with block number= 0) to host A with source= B's TID, destination= A's TID.

At this point the connection has been established and the first data packet can be sent by Host A with a sequence number of 1. In the next step, and in all succeeding steps, the hosts should make sure that the source TID matches the value that was agreed on in steps 1 and 2. If a source TID does not match, the packet should be discarded as erroneously sent from somewhere else. An error packet should be sent to the source of the incorrect packet, while not disturbing the transfer. This can be done only if the TFTP in fact receives a packet with an incorrect TID. If the supporting protocols do not allow it, this particular error condition will not arise.

The following example demonstrates a correct operation of the protocol in which the above situation can occur. Host A sends a request to host B. Somewhere in the network, the request packet is duplicated, and as a result two acknowledgments are returned to host A, with different TID's chosen on host B in response to the two requests. When the first response

arrives, host A continues the connection. When the second response to the request arrives, it should be rejected, but there is no reason to terminate the first connection. Therefore, if different TID's are chosen for the two connections on host B and host A checks the source TID's of the messages it receives, the first connection can be maintained while the second is rejected by returning an error packet.

5. TFTP Packets

TFTP supports five types of packets, all of which have been mentioned above:

opcode	*operation*
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgment (ACK)
5	Error (ERROR)

The TFTP header of a packet contains the opcode associated with that packet.

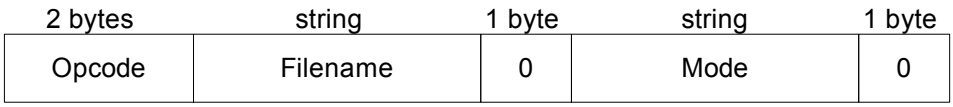


Figure 2. RRQ/WRQ packet

RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format shown in Figure 2. The file name is a sequence of bytes in netascii terminated by a zero byte. The mode field contains the string "netascii", "octet", or "mail" (or any combination of upper and lower case, such as "NETASCII", NetAscii", etc.) in netascii indicating the three modes defined in the protocol. A host which receives netascii mode data must translate the data to its own format. Octet mode is used to transfer a file that is in the 8-bit format of the machine from which the file is being transferred. It is assumed that each type of machine has a single 8-bit format that is more common, and that that format is chosen. For example, on a DEC-20, a 36 bit machine, this is four 8-bit bytes to a word with four bits of breakage. If a host receives a octet file and then returns it, the returned file must be identical to the original. Mail mode uses the name of a mail recipient in place of a file and must begin with a WRQ. Otherwise it is identical to netascii mode. The mail recipient string should be of the form "username" or "username@hostname". If the second form is used, it allows the option of mail forwarding by a relay computer.

The discussion above assumes that both the sender and recipient are operating in the same mode, but there is no reason that this has to be the case. For example, one might build a storage server. There is no reason that such a machine needs to translate netascii into its own form of text. Rather, the sender might send files in netascii, but the storage server might simply store them without translation in 8-bit format. Another such situation is a problem that currently exists on DEC-20 systems. Neither netascii nor octet accesses all the bits in a word. One might create a special mode for such a machine which read all the bits in a word, but in which the receiver stored the information in 8-bit format. When such a file is retrieved from the storage site, it must be restored to its original form to be useful, so the reverse mode must also be implemented. The user site will have to remember some information to achieve this. In both of these examples, the request packets would specify octet mode to the foreign host, but the local host would be in some other mode. No such machine or application specific modes have been specified in TFTP, but one would be compatible with this specification.

It is also possible to define other modes for cooperating pairs of hosts, although this must be done with care. There is no requirement that any other hosts implement these. There is no central authority that will define these modes or assign them names.

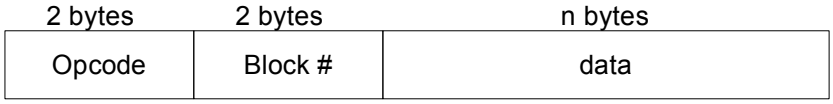


Figure 3. DATA packet

Data is actually transferred in DATA packets depicted in Figure 3. DATA packets (opcode = 3) have a block number and data field. The block numbers on data packets begin with one and increase by one for each new block of data. This restriction allows the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; if it is from zero to 511 bytes long, it signals the end of the transfer. (See the section on Normal Termination for details.)

All packets other than duplicate ACK's and those used for termination are acknowledged unless a timeout occurs [4]. Sending a DATA packet is an acknowledgment for the first ACK packet of the previous DATA packet. The WRQ and DATA packets are acknowledged by ACK or ERROR packets, while RRQ and ACK packets are acknowledged by DATA or ERROR packets. Figure 4 depicts an ACK packet; the opcode is 4. The block number in an ACK echoes the block number of the DATA packet being acknowledged. A WRQ is acknowledged with an ACK packet having a block number of zero.

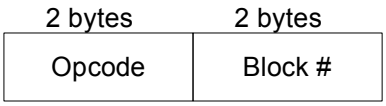


Figure 4. ACK packet

Figure 5. ERROR packet

An ERROR packet (opcode 5) takes the form depicted in Figure 5. An ERROR packet can be the acknowledgment of any other type of packet. The error code is an integer indicating the nature of the error. A table of values and meanings is given in the appendix. (Note that several error codes have been added to this version of this document.) The error message is intended for human consumption, and should be in netascii. Like all other strings, it is terminated with a zero byte.

6. Normal Termination

The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e., Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand, dallying is encouraged. This means that the host sending the final ACK will wait for a while before terminating in order to retransmit the final ACK if it has been lost. The acknowledger will know that the ACK has been lost if it receives the final DATA packet again. The host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully,

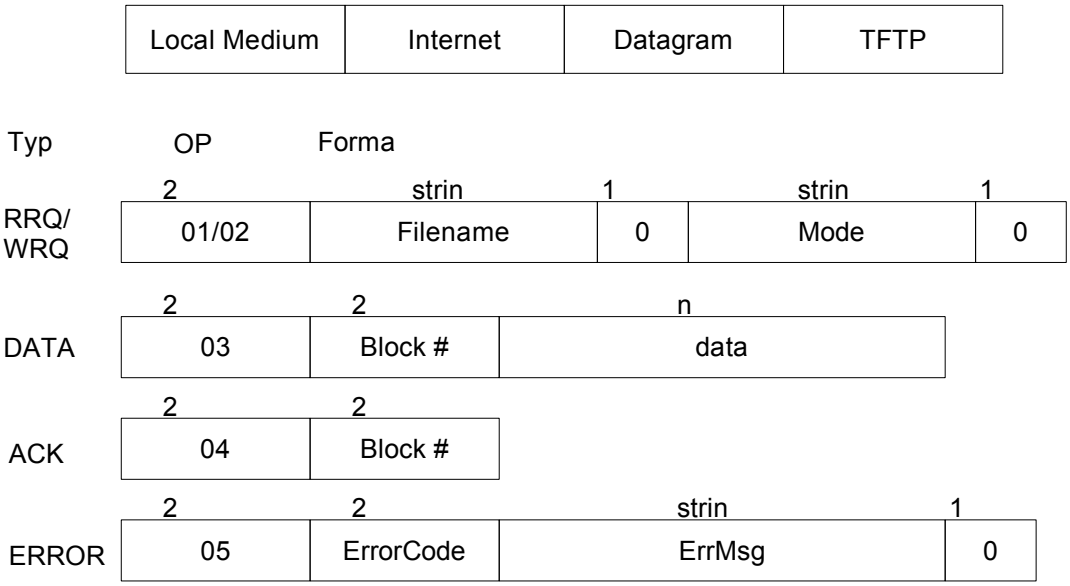
after which the acknowledger or network may have experienced a problem. It is also possible in this case that the transfer was unsuccessful. In any case, the connection has been closed.

7. Premature Termination

If a request can not be granted, or some error occurs during the transfer, then an ERROR packet (opcode 5) is sent. This is only a courtesy since it will not be retransmitted or acknowledged, so it may never be received. Timeouts must also be used to detect errors.

I. Appendix

Order of Headers



Initial Connection Protocol for reading a file

1. Host A sends a "RRQ" to host B with source= A's TID, destination= 69.
2. Host B sends a "DATA" (with block number= 1) to host A with source= B's TID, destination= A's TID.

Error Codes

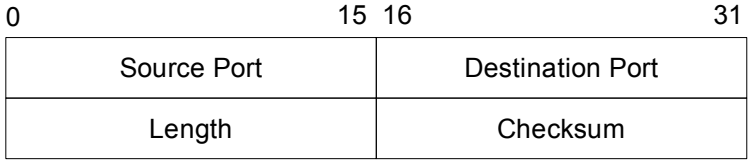
Value	*Meaning*
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

Internet User Datagram Header

(This has been included only for convenience. TFTP need not be implemented on top of the

Internet User Datagram Protocol.)

Format



Values of Fields

Source Port	Picked by originator of packet.
Dest. Port	Picked by destination machine (69 for RRQ or WRQ).
Length	Number of bytes in UDP packet, including UDP header.
Checksum	Reference 2 describes rules for computing checksum. (The implementor of this should be sure that the correct algorithm is used here.) Field contains zero if unused.

Note: TFTP passes transfer identifiers (TID's) to the Internet User Datagram protocol to be used as the source and destination ports.

References

1. USA Standard Code for Information Interchange, USASI X3.4-1968.
2. Postel, J., "User Datagram Protocol," RFC 768, USC/Information Sciences Institute, 28 August 1980.
3. Postel, J., "Telnet Protocol Specification," RFC 764, USC/Information Sciences Institute, June, 1980.
4. Braden, R., Editor, "Requirements for Internet Hosts -- Application and Support", RFC 1123, USC/Information Sciences Institute, October 1989.

Security Considerations

Since TFTP includes no login or access control mechanisms, care must be taken in the rights granted to a TFTP server process so as not to violate the security of the server hosts file system. TFTP is often installed with controls such that only files that have public read access are available via TFTP and writing files via TFTP is disallowed.

Author's Address

Karen R. Sollins
Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139-1986
Phone: (617) 253-6006
EMail: SOLLINS@LCS.MIT.EDU

Dodatak E – Don't panic disclaimer

Ako ste došli do ovdje a labosi još ne rade, tada

PANIC