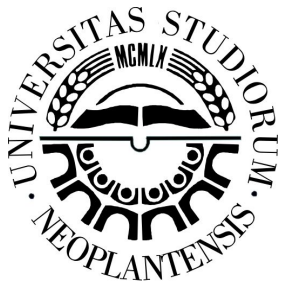



UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA
NOVI SAD


Kandidat: Lazar Stričević

# Pristup agregaciji mrežnih veza u operativnom sistemu sa mikrojezgrom

doktorska disertacija

Mentor: dr Miroslav Hajduković

Novi Sad 2016

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

# КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

| | | |
|---|---|---|
| Редни број, **РБР**: | | |
| Идентификациони број, **ИБР**: | | |
| Тип документације, **ТД**: | | Монографска публикација |
| Тип записа, **ТЗ**: | | Текстуални штампани документ/ЦД |
| Врста рада, **ВР**: | | Докторска дисертација |
| Аутор, **АУ**: | | Лазар Стричевић |
| Ментор, **МН**: | | др Мирослав Хајдуковић, редовни професор |
| Наслов рада, **НР**: | | Приступ агрегацији мрежних веза у оперативном систему са микројезгром |
| Језик публикације, **ЈП**: | | српски (латиница) |
| Језик извода, **ЈИ**: | | српски/енглески |
| Земља публиковања, **ЗП**: | | Србија |
| Уже географско подручје, **УГП**: | | Војводина |
| Година, **ГО**: | | 2016. |
| Издавач, **ИЗ**: | | Ауторски репринт |
| Место и адреса, **МА**: | | Факултет техничких наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | 7/90/107/0/27/0/2 |
| Научна област, **НО**: | | Електротехничко и рачунарско инжењерство |
| Научна дисциплина, **НД**: | | Примењене рачунарске науке и информатика |
| Предметна одредница/Кључне речи, **ПО**: | | агрегација мрежних веза, микројезгро, толеранција поремећаја, ослоњивост, поузданост, доступност, рачунарске мреже |
| **УДК** | | |
| Чува се, **ЧУ**: | | Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад |
| Важна напомена, **ВН**: | | |
| Извод, **ИЗ**: | | Теза се бави повећањем укупне ослонљивости модуларног микрокернел оперативног систем *MINIX* 3 кроз повећање поузданости његовог мрежног подсистема. То је постигнуто тако што је овом оперативном систему додата агрегација мрежних веза, чиме је подражана толеранција на поремећај комуникационих линија. На крају је дата анализа како додати део утиче на укупне мрежне перформансе. |
| Датум прихватања теме, **ДП**: | | |
| Датум одбране, **ДО**: | | |
| Чланови комисије, **КО**: | Председник: | др Душан Малбашки, редовни професор |
| | Члан: | др Александар Ердељан, ванредни професор |
| | Члан: | др Бранислав Атлагић, ванредни професор |
| | Члан: | др Жарко Живанов, доцент — Потпис ментора |
| | Члан, ментор: | др Мирослав Хајдуковић, редовни професор |

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

# KEY WORDS DOCUMENTATION

Образац **Q2.HA.04-05** - Издање 1

| | | | |
|---|---|---|---|
| Accession number, **ANO**: | | | |
| Identification number, **INO**: | | | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual material, printed/CD | |
| Contents code, **CC**: | | Ph.D. thesis | |
| Author, **AU**: | | Lazar Stričević | |
| Mentor, **MN**: | | Ph. D. Miroslav Hajduković, full professor | |
| Title, **TI**: | | Link aggregation approach to a microkernel operating system | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian/English | |
| Country of publication, **CP**: | | Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2016 | |
| Publisher, **PB**: | | Author's reprint | |
| Publication place, **PP**: | | Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 7/90/107/0/27/0/2 | |
| Scientific field, **SF**: | | Electrical and Computer Engineering | |
| Scientific discipline, **SD**: | | Applied Computer Science and Informatics | |
| Subject/Key words, **S**/**KW**: | | link aggregation, microkernel, fault tolerance, dependability, reliability, availability, computer networks | |
| **UC** | | | |
| Holding data, **HD**: | | Library of Faculty of Technical Sciences, D. Obradovića 6, 21000 | |
| Note, **N**: | | | |
| Abstract, **AB**: | | The thesis deals with the way to increase the dependability of the modular microkernel operating system *MINIX* 3 through the increase of the reliability of its network subsystem. This is achieved by adding link aggregation to this operating system, which added fault tolerance for the communication lines. At the end, the analysis is given of how new module affects the overall network performance. | |
| Accepted by the Scientific Board on, **ASB**: | | | |
| Defended on, **DE**: | | | |
| Defended Board, **DB**: | President: | Ph. D. Dušan Malbaški, full professor | |
| | Member: | Ph. D. Aleksandar Erdeljan, associate professor | |
| | Member: | Ph. D. Branislav Atlagić, associate professor | |
| | Member: | Ph. D. Žarko Živanov, assistant professor | Menthor's sign |
| | Member, Mentor: | Ph. D. Miroslav Hajduković, full professor | |

# Sadržaj

# Skraćenice

U tekstu su korišćene sledeće skraćenice:

API (engl. application programming interface) progamski interfejs za aplikacije
CPU (engl. *Central Processing Unit*) - centralni procesor
DMA (engl. direct memory access) direktan pristup memoriji
DMLT (engl. Distributed Multi-link Trunking) distribuirana verzija *Nortel* MLT tehnologije
DS (engl. Data Store) *MINIX3* server zadužen za imenovanje javnih podataka
FEC (engl. Fast EtherChannel) 100 Mbps verzija *Cisco EtherChannel* standarda
FT (engl. Fault tolerance) otpornost na poremećaje
GEC (engl. Gigabit EtherChannel) 1000 Mbps verzija *Cisco EtherChannel* standarda
IA-32 (engl. Intel Architecture, 32-bit) 32-bitna arhitektura procesora firme Intel
IC (engl. integrated circuit) integrisano kolo
IEEE (engl. The Institute of Electrical and Electronics Engineers) Institut inženjera elektrotehnike i elektronike
INET (engl Internet server) MINIX server zadužen za mrežnu komunikaciju
IO (engl. input-output) ulaz-izlaz
IOMMU (engl. input–output memory management unit) jedinica za povezivanje DMA IO uređaja na memorijsku magistralu
IP (engl. Internet protocol) internet protokol
IPC (engl. Interprocess communication) međuprocesna komunikacija
IPX (engl. Internetwork Packet Exchange) protokol za razmenu paketa
IRQ (engl. *Interrupt Request*) - prekid, interapt
ISA (engl. Industry Standard Architecture) 16-bitna magistrala za stare PC-AT kompatibilne računare
ISO (engl. the International Organization for Standardization) Međunarodna organizacija za standardizaciju
L1-L7 (engl Layer1, … , Layer7) - nivoi OSI modela
LACP (engl. Link Aggregation Control Protocol) protokol za kontrolu agregacije mrežnih veza
LAG (engl. Link Aggregation Group) grupa agregiranih mrežnih linkova
LAN (engl. Local Area Network) lokalna mreža
LB (engl. Load balancing) raspoređivanje mrežnog saobraćaja
LKM (engl. Linux kernel modules), *Linux* moduli jezgra
LoC (engl. lines of code) broj linija izvronog koda
MAC (engl. *Media Access Control*), kontrola pristupa medijumu
MBR (engl. Master boot record) glavni sekror za pokretanje OS
MDI (engl. media-dependent interface) interfejs zavisan od prenosnog medija
MII (engl. media-independent interface) interfejs nezavisan od prenosnog medija
MLT (engl. Multi-link Trunking) Nortel-ova tehnologija za agregaciju mrežnih veza
MMU (engl. memory management unit) jedinica za upravljanje memorijom
MPTCP (engl. Multi-path TCP) TCP sa više tokova podataka
MTBF (engl. *mean time between failures*), srednje vreme između otkaza

MTTF (engl. *mean time to failure*), srednje vreme do otkaza
MTTR (engl. *mean time to repair*), srednje vreme do popravke
NIC (engl. Network Interface Controller), mrežni interfejs
OS - operativni sistem
OSI (engl. *Open Systems Interconnection*), model otvorenog sistema povezivanja
OUI (engl Organizationally Unique Identifier), jedinstveni identifikator organizacije
PC (engl. *Personal Computer*), lični računar
PCI (engl. Peripheral Component Interconnect), brza lokalna magistrala računara za povezivanje spoljnih uređaja
PDU (engl. protocol data unit), protokolska jedinica podataka
PHY (engl. physical transciever), fizički primopredajnik/transiver
PM (engl. Process Manager), *MINIX3* server zadužen za rukovanje procesima
PPP (engl. Point-to-Point Protocol ) protokol
POLA (engl. principle of least authority) vidi POLP
POLP (engl. principle of least privilege) princip najmanje privilegije
POSIX (engl. Portable Operating System Interface) standard za međusobnu kompatibilnost operativnih sistema definisan od strane IEEE
RAM (engl. *Random Access Memory*) - operativna memorija
RS (engl. Resurrection Server) *MINIX3* server zadužen za upravljanje sistemskim procesima
SoC (engl. System on Chip) IC koje integriše kompletan računarski sistem na jednom čipu.
SEF (engl. System Event Framework) programski okvir (skup funkcija) MINIX-a za rukovanje sistemskim događajima koji koriste sistemski procesi
TCP (engl. Transmission control protocol) internet protokol za kontrolu prenosa
VLAN (engl. Virtual LAN) virtuelni LAN
VM (engl. Virtual Memory) *MINIX3* server zadužen za rukovanje virtuelnom memorijom
VFS (engl. Virtual File System) *MINIX3* server zadužen za rukovanje fajl sistemom
UID (engl. User Identification) identifikacioni broj korisnika

# 1. Opšti uvod

Današnji svet u sve većoj meri zavisi od računara. Jedan od najvećih problema prilikom njihove upotrebe je njihova pouzdanost, dostupnost i bezbednost. Iako su ove osobine u stvari različiti koncepti, sa stanovišta korisnika one su usko povezane i zajedno čine oslonjivost (engl. *dependability*) sistema [Avizienis2004].
Potreba za oslonjivošću prevazilazi obične PC računare kao što su recimo desktop, laptop ili serverski računari i u većoj meri je izražena kada su u pitanju mobilni uređaji (mobilni telefoni, tablet uređaji) ili ugrađeni (engl. embedded) sistemi poput sistema u automobilima, medicinskim uređajima, ili u elektronici široke potrošnje (npr. u frižiderima ili veš mašinama).
Problemi sa oslonjivošću na danas najčešće korićenim operativnim sistemima PC računara (kao što su *Linux*, *Windows* ili *MacOS*), predstavljaju svakodnevicu, pa otkazi u vidu "zamrzavanja" ili resetovanja sistema nisu neuobičajeni. Dobar primer je recimo *Windows*-ov čuveni "plavi ekran smrti", koji je dobro poznat problem [Keizer2014].
Nažalost, isti problemi se pojavljuju i na mobilnim uređajima i ugrađenim sistemima koji su sada dovoljno moćni da ih sada pogone "pravi" operativni sistemi iz PC sveta, kao što su *Windows Phone*, *Embedded Linux*, *Android* ili *iOS*. Dizajn ovih sistema se ne razlikuje značajno od njihovih PC varijanti, što znači da su mobilni i ugrađeni sistemi podložni istim ili sličnim problemima sa oslonjivošću kao i PC računari.
Ovo može da predstavlja ozbiljan problem obzirom na ekspanziju koju trenutno doživljava primena računara u najrazličitije svrhe. Primer za ovo je ugrađivanje računara sa senzorima u najrazličitije predmete svakodnevne upotrebe i njihovo sveopšte povezivanje pomoću tzv. "Interneta stvari" (engl. *Internet of Things*) [Atzori2010]. Pretpostavlja se da će do 2020. godine u svetu biti više od 50 milijardi ovakvih uređaja [Evans2011].

## 1.1 Osnovni pojmovi oslonjivosti

Da bismo mogli da govorimo o oslonjivosti sistema potrebno je da definišemo mere njenih najvažijih elemenata: pouzdanosti i dostupnosti [Lee1990]. Osnovne mere oslonjivosti su MTTF (engl. *mean time to failure*), srednje vreme do otkaza, koja predstavlja očekivanu vrednost funkcije gustine otkaza, i MTTR (engl. *mean time to repair*), srednje vreme do popravke, koja je očekivana vrednost funkcije gustine popravki. Srednje vreme između otkaza, MTBF (engl. *mean time between failures*) je suma ove dve vrednosti:

$$MTBF = MTTF + MTTR$$

Dostupnost (engl. *availability*) predstavlja spremnost za upotrebu, što je zapravo verovatnoća da će sistem ili servis biti operativan kada je potrebno, a računa se ovako:

$$A = MTTF / MTTR$$

Pouzdanost (engl. *reliability*) je kontinualnost servisa, što je zapravo verovatnoća da će sistem ili servis da bude i ostane operativan u određenom vremenskom periodu:

$$R(t) = P(\text{da nema otkaza u } [1, t] ) = 1 - Q(t)$$

Gde je Q(t) funcija raspodele otkaza.

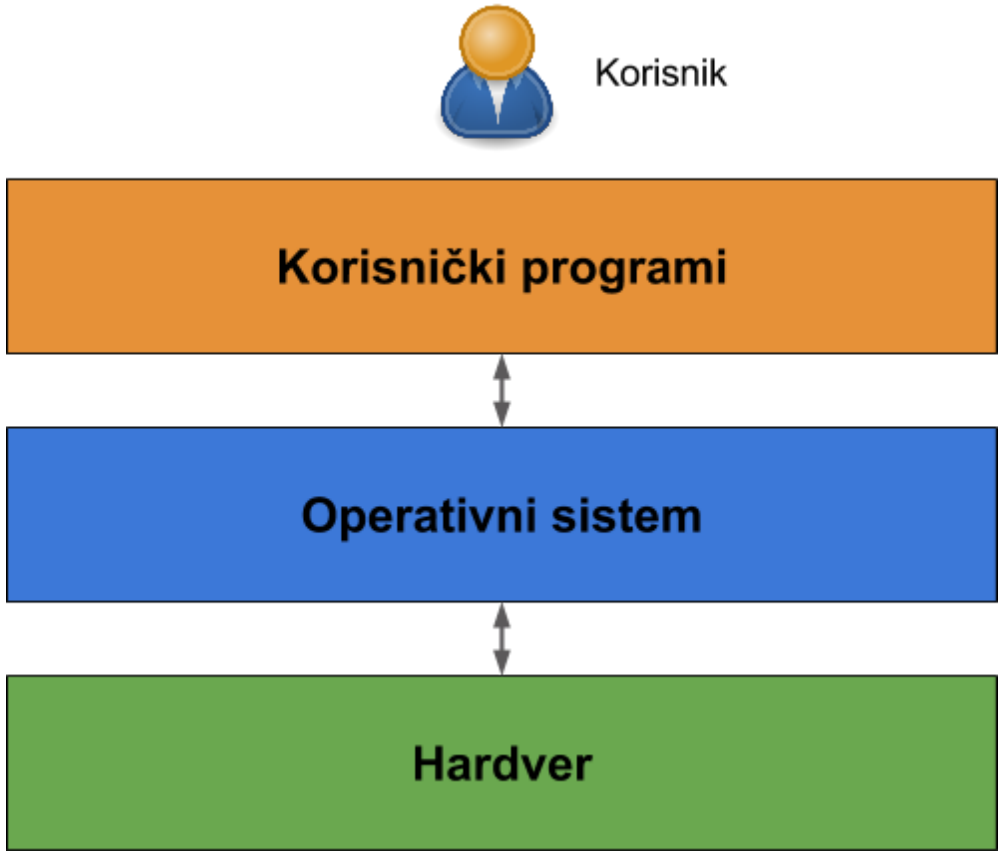
Relativna važnost dostupnosti i pouzdanosti zavise od primene datog servisa. Dostupnost je od primarnog značaja za transakcione servise kao što je HTTP servis. Dokle god server "uglavnom" funkcioniše, nije previše bitno i ako često otkazuje sve dok je MTTR vrlo kratko. Sa druge strane, pouzdanost je od velike važnosti za servise zasnovane na sesijama i konekcijama, kao što je telekonferencija. Da bi sesija mogla da bude od koristi, mora da bude operativna određeni vremenski period, što zahteva veliku vrednost MTTF.

## 1.2 Potreba za oslonjivošću operativnog sistema

Možda je potreba za oslonjivošću najbolje ilustrovana potencijalnim dalekosežnim posledicama softverskih otkaza. Na primer između 1985 i 1987 softverska greška poznata kao "utrkivanje" (engl. *race condition*) uzrokovala je više incidenata sa Therac-25 radiološkim uređajima koji su doveli do prekomernog ozračenja, do ozbiljnih povreda i tri žrtve [Leveson1993]. 1996. godine prekoračenje u celobrojnom računu je uzrokovalo uništenje rakete *Ariane-5* nedugo posle lansiranja, što je za posledicu imalo direktnu materijalnu štetu od najmanje 370 miliona dolara [Dowson1997]. Greška u sistemu za kontrolu ubrzavanja firme Tojota 2009. godine je uzrokovala saobraćajne nesreće sa ljudskim žrtvama [CBSNews2010] [Dunn2013]. Ovi događaji, kao i mnogi drugi [Wikipedia2016] ilustruju opštu potrebu za oslonjivim računarskim platformama.

Oslonjivost operativnog sistema je od posebne važnosti zbog fundamentalne uloge koju operativni sistem ima u skoro svakom računarskom sistemu. On predstavlja najniži sloj sistemskog softvera (u užem smislu zvanim jezgro ili kernel) koji posreduje između hardvera računara i aplikacija koje pokreće korisnik. Može se reći da operativni sistem posredovanjem prilikom pristupa hardveru, za korisničke programe obavlja dve osnovne uloge [Tanenbaum2014]:

- virtualizuje hardver računara nudeći zgodan programski interfejs za korisničke programe (engl *application programming interface* - *API*) zgodnije nego što bi to bio direktan pristup hardveru, čineći programe manje zavisnim od konkretnog hardvera.
- kontroliše i upravlja softverskim i hardverskim resursima, kako bi sprečio konflikte i sproveo bezbednosne politike.




Slika 1.1: Uloga operativnog sistema u računarskom sistemu: OS je najniži nivo sistemskog softvera koji upravlja resursima i komunicira sa aplikacijama

Zbog važne uloge koju operativni sistem ima u funkcionisanju računara, svaki problem sa operativnim sistemom računara će imati neposredne posledice po oslonjivost čitavog sistema kojeg dati računar opslužuje.
Bez obzira na važnu ulogu koji ima operativni sistem, njegovoj oslonjivosti se još uvek ne poklanja dovoljno pažnje. Većina neplaniranih zastoja (engl. *downtime*) uzrokovana greškama u sistemskom softveru, a ne otkazima hardvera [Xu1999]. Za razliku od ostalog softvera, otkazi operativnog sistema zaslužuju posebnu pažnju, zbog nihovih mogućih posledica. U slučaju pada (otkaza) korisničkog programa odnosno aplikacije, najčešće posledice su gubitak korisničkih podataka koji nisu bili snimljeni i prekid mrežnih sesija, što može da uzrokuje nekonzistentno stanje aplikacije prilikom ponovnog pokretanja. Postupak oporavka obično obuhvata ponovno pokretanje pomenute aplikacije, uz eventualno vraćanje stanja podataka u konzistentno stanje. Nasuprot tome, otkaz operativnog sistema je znatno ozbiljniji problem, jer uzrokuje otkaz svih trenutno pokrenutih aplikacija, što zatim uzrokuje prekid svih aktivnih mrežnih konekcija, gubitak svih nesnimljenih podataka uz moguće dovođenje svih aktivnih aplikacija u nekonzistentno stanje [Ganapathi2005]. Uz sve to, otkaz operativnog sistema može da u nekonzistentno stanje dovede i neke od njegovih delova, kao što je fajl sistem. To znači da oporavak od pada operativnog sistema mora da uključi ponovno pokretanje operativnog sistema uz eventualni oporavak nekih njegovih delova, kao i oporavak i ponovno pokretanje svih aplikacija koje su bile pokrenute u trenutku pada. Za vreme oporavka i ponovnog pokretanja čitav sistem nije dostupan korisniku, što uz moguć

gubitak podataka čini otkaz operativnog sistema ozbiljnim i vrlo nepoželjnim problemom za sve korisnike. Za pojedine kritične sisteme (avioni, automobili, vojni sistemi, medicinski uređaji, ...) je od velike važnosti da se ovakvi problemi izbegnu i zbog toga oslonjivost operativnog sistema ovde igra veliku ulogu.

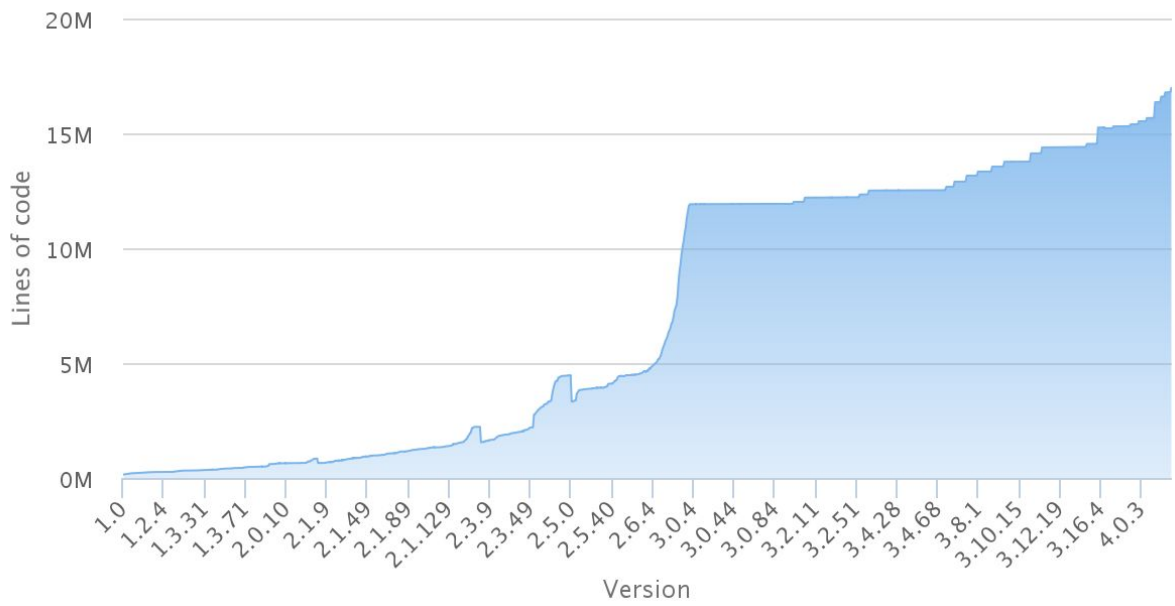
## 1.3 Uzroci otkaza operativnog sistema

Otkazi operativnog sistema mogu biti uzrokovani greškama u hardveru, firmveru ili softveru [Kinshuman2011]. Softverske greške su neizbežne pre svega zbog velike kompleksnosti programskog koda, kao i zbog česte nemogućnosti da se svaki deo programskog koda detaljno proveri pre početka operativne upotrebe.

### 1.3.1 Kompleksnost softvera

Priroda velikih softverskih sistema je da sadrže greške. Istraživanje gustine grešaka u softveru [Hatton1997] u nekoliko programskih jezika dovodi do zaključka da se čak i za najbolje napisan programski kod može očekivati da ima 1-6 grešaka na 1000 linija izvornog koda. Od toga nisu imuni ni operativni sistemi otvorenog koda kao što su Linux i FreeBSD [Dinh-Trong2005].

Rezultati istraživanja [Ostrand2005] pokazuju da ukupna gustina grešaka u softveru posle popravki i novih verzija ne pada (čak ni asimptotski) na nulu. Analiza [Dinh-Trong2005] daje podatak da *FreeBSD* jezgro ima gustinu grešaka od 1,89 grešaka na 1000 linija, što se smatra vrlo dobrim. Obzirom da FreeBSD jezgro (verzija 10.2) ima oko 5 miliona linija koda, to bi značilo da u jezgru postoji 9.450 grešaka, koje nisu otkrivene i od koje potencijalno mogu da dovedu do otkaza čitavog operativnog sistema.




Slika 1.2: Promena broja linija koda Linux jezgra po verzijama od nastanka do danas [Linuxcounter2016]

Kao što se vidi na slici 1.2, broj linija *Linux* jezgra je rastao sa svakom novom verzijom, pa je od verzije 1.0, koja je imala 150.867 linija, dogurao do skoro 16.977.804 linija u verziji 4.5

[Linuxcounter2016]. Slično važi i za jezgra ostalih operativnih sistema, jer svaka nova verzija dodaje nove mogućnosti, podršku za nov hardver i slično, uz očuvanje kompatibilnosti sa prethodnim verzijama. Sve ovo uzrokuje da svaka nova verzija unosi još novih, potencijalno fatalnih grešaka u programski kod jezgra sistema.

### 1.3.2 Problemi sa drajverima

Studija [Chou2001] je pokazala da drajveri za rad sa različitim uređajima čine preko 70% programskog koda operativnog sistema, kao i da je gustina grešaka u programskom kodu drajvera tri do sedam puta veća nego u ostalim delovima operativnog sistema. Ovo ne iznenađuje, jer je ukupan broj različitih uređaja vrlo veliki i svakodnevno se povećava. Analiza iz 2004. godine [Murphy2004] daje broj od oko 800.000 različitih vrsta postojećih uređaja i pojavu oko 1.500 novih uređaja dnevno, za koje je potrebno napisati drajvere. Obzirom na brzinu pojavljivanja novih uređaja, teško je očekivati da svi drajveri budu dobro dizajnirani, napisani i testirani. Zatim, drajvere često pišu ljudi kojima je primarna delatnost dizajn hardvera, a ne softvera. Dokumentacija hardvera, na osnovu koje se piše drajver, je često nepotpuna i sa greškama. Ovo su neki od razloga zbog čega su drajveri mesta na kojima se često nalaze greške. Bitno je napomenuti da u operativnim sistemima sa monolitnim jezgrom (kao što su *Linux*, *Windows*, *FreeBSD*, ...) drajveri postaju deo jezgra, pa se samim tim greška u drajveru smatra greškom u jezgru.

### 1.3.3 Problemi sa hardverom

Otkazi hardverskih komponenata mogu da uzrokuju greške u radu, a često i otkaze čitavog sistema. Ove vrste otkaza se obično prevazilaze redundancijom, odnosno dupliranjem kritičnih komponenti i funkcija sistema.

## 1.4 Život sa greškama u sistemu

U prethodnom delu smo videli da su greške u softveru, pa tako i u operativnom sistemu, neizbežne. Ovi razlozi daju povoda da se operativni sistem dizajnira na takav način da se greške u što većoj meri neutrališu ili bar ne dovedu do potpunog otkaza sistema. Način da se ovo postigne je modularizacija, odnosno podela na međusobno izolovane celine. Ovakav način rada se primenjuje u drugim inženjerskim granama, kao što je na primer brodogradnja: brod se sastoji od međusobno izolovanih odeljaka, tako da ako dođe do potapanja jednog odeljka, voda se tu zadržava i ne prodire u druge delove broda, pa je brod u stanju da ostane na površini. Ovaj princip može da se primeni i na operativni sistem. Ukoliko bi došlo do greške u jednom od međusobno izolovanih delova operativnog sistema, greška bi bi bila ograničena i izolovana u delu u kom je nastala i ne bi nužno vodila otkazu celog sistema.
Da bi izolacija bila moguća uvedeni su različiti režimi rada centralnog procesora računara. Tih režima mora da bude najmanje dva:

- privilegovani ili kernel režim u kom nema ograničenja: dozvoljeno je izvršavanje svih instrukcija i pristup svoj memoriji koja postoji u računaru,
- neprivilegovani, odnosno korisnički režim rada u kome je dozvoljeno izvršavanje samo “sigurnih” instrukcija i pristup samo rezervisanim delovima memorije.

Operativni sistem (sa monolitnim jezgrom kao što su *Linux*, *Windows*, ...) se štiti od grešaka u korisničkim aplikacijama tako što se korisničke aplikacije izvršavaju u tzv. korisničkom ili neprivilegovanom režimu, u kome su one ograničene po pitanju memorije kojoj smeju da pristupe i instrukcija koje smeju da izvrše. To znači da neka eventualna greška u aplikaciji može potencijalno da dovede do otkaza te aplikacije, ali zbog ograničenja korisničkog režima ne može da utiče na okolinu, pa ostali delovi sistema mogu da nesmetano nastave sa radom. Sa druge strane, jezgro operativnog sistema se izvršava u privilegovanom režimu procesora, kako bi moglo da opslužuje korisničke aplikacije. Ako se greška nalazi u jezgru operativnog sistema, u kome je zbog privilegovanog režima nema ograničenja u pristupu memoriji, ta greška može da dovede do otkaza jezgra sistema, jer može da utiče na bilo koji od njegovih delova. To je razlog zbog čega su greške u jezgru operativnog sistema daleko ozbiljnije, mogu da dovedu do raznih vrsta otkaza celog računarskog sistema. Zbog toga se programski kod jezgra naziva “kod od poverenja” ili TCB (engl. *Trusted Code Base*) [Rushby1981] i on bi trebalo da je rigorozno proveren. Pošto smo videli da gustina grešaka teško može da se spusti ispod nekog minimuma, rešenje (ili bar jedno od rešenja) je da se jezgro učini što manjim.

### 1.4.1 Mikrokernel

Kada se iz jezgra izbace svi delovi za koje nije apsolutno neophodno da se izvršavaju u privilegovanom režimu, ono što je preostalo (programiranje kontrolnih registara, preključivanje procesa,...) naziva se mikrojezgro odnosno mikrokernel. Delovi koji su izbačeni iz jezgra sada postaju zasebni, međusobno izolovani korisnički procesi od kojih svaki sada ima tačno određenu svrhu i naziva se server. Pošto se sada operativni sistem sastoji iz mikrojezgra i više servera rezličite namene, on se sada naziva multiserverski operativni sistem.

Prebacivanjem u korisnički režim pojedinih delova jezgra i njihovom izolacijom ne mora da se smanji broj grešaka koji je ranije postojao, ali sada su te greške izolovane, pa je njihov uticaj na ostatak sistema manje izražen. Najvažniji cilj ovog izmeštanja iz jezgra je izolacija drajvera koji su ranije identifikovani kao delovi jezgra sa najviše problema.

Deo koji je preostao u mikrojezgru da se izvršava u privilegovanom režimu procesora je znatno manji nego ranije (smanjen je TCB), tako da je sada taj programski kod lakše pregledati, pa je za očekivati da će gustina grešaka biti manja. Ako gustina grešaka i ostane ista, ukupan broj grešaka u TCB će biti manji, jer je smanjen broj linija izvornog koda.

Najveći problem sa ovim pristupom je taj što sada izdvojeni delovi jezgra nemaju više direktan pristup svim strukturama podataka kao ranije, nego sada moraju da komuniciraju sa ostalim delovima kako bi došli do podataka koji im trebaju, odnosno kako bi izvršili svoj posao. Ovo zahteva upotrebu međuprocesne komunikacije (engl. *interprocess communication* - IPC) i kopiranje podataka za većinu operacija, što je u opštem slučaju sporije od direktnog pristupa. Zato se mikrokernel operativni sistemi do danas često smatraju za inferiorna po pitanju performansi u odnosu na operativne sisteme sa monolitnim kernelom [Lameter2007]. Ipak novije generacije operativnih sistema zasnovanih na mikrokernelu pokazuju da je moguće da se postignu performanse koje zaostaju za monolitnim operativnim sistemima oko 5% [Haertig1997]. Za najveći broj primena ovo ne predstavlja veliki hendikep, obzirom da je to cena za postizanje značajno veće pouzdanosti.

Do razvoja operativnih sistema sa mikrokernelom je dovela formulacija i primena sledećih principa:

- princip razdvajanja mehanizma i politike,
- princip najmanje privilegije.

Princip razdvajanja mehanizma i politike [Levin1975] je princip koji se primenjuje na dizajn operativnih sistema. On kaže da mehanizmi ne bi trebalo da diktiraju (ili preterano ograničavaju) politike po kojima se donose odluke o autorizaciji operacija i alokaciji resursa. Pod mehanizmima se podrazumevaju delovi sistema koji kontrolišu autorizaciju operaciju i alokaciju resursa. Ovaj princip je nastao da bi se obezbedila fleksibilnost operativnom sistemu da podrži što veći broj bezbednosnih politika, jer postaje moguća promena politike bez promene mehanizma. Razdvajanje mehanizma od politike se obično postiže razdvajanjem na module, pa je tada moguće zameniti modul koji recimo diktira politiku raspoređivanje procesa, bez zamene modula koji vrši samo raspoređivanje procesa. Ideja o modularizaciji se dobro uklapa sa idejom mikrokernela, zbog čega se mikrokernel sistemi smatraju fleksibilnim.

Princip najmanje privilegije (engl. *principle of least privilege* - POLP) kaže da svaki deo sistema (modul, proces, program, korisnik) treba da imaju pristup samo informacijama i resursima koji su neophodni za posao koji legitimno obavljaju [Saltzer1974]. Ovaj princip se negde još naziva i princip najmanjeg autoriteta (engl. *principle of least authority* - POLA). Primena ovog principa uključuje postizanje otpornosti na poremećaje (engl. *fault tolerance*) [Denning1976]. Još jedan od rezultata primena principa najmanje privilegije je i nastanak sistema zasnovanih na sposobnostima (engl. *capability based systems*) [Levy1984]. Sposobnost (engl. capability) je prenosivo, ovlašćenje pristupa. To je vrednost koja referencira objekat sa pridruženim skupom prava pristupa. Zbog prednosti sistema zasnovanim na sposobnostima [Miller2003], mnogi mikrokernel sistemi koriste ovaj sigurnosni model.

Mikrokerneli prve generacije (kao što su *Mach* [Accetta1986] i *Chorus* [Rozier1988]) su evoluirali sa monolitnog dizajna. Zbog toga su u početku imali loše performanse u odnosu na monolitne sisteme, ali su njihove inherentne prednosti dovele do nastavka istraživanja. To je dovelo do druge generacije mikrokernela (kao što su *L4* [Liedtke1995] i *QNX* [Hildebrand1992]) koji su strožije težili minimalnosti i koji su bili dizajnirani “od nule”, što je rezultovalo znatno boljim performansama, čak svega 5% slabije od monolitnih sistema [Haertig1997].

Danas postoji mnoštvo projekata za razvoj operativnih sistema zasnovanih na mikrokernel paradigmi (*MINIX3*, *HelenOS*, *Hurd*, …) i većina ima za glavni cilj poboljšanje oslonjivosti operativnog sistema. Pojedini projekti su u tome stigli dosta daleko - *seL4* mikrokernel je prvi koji je čak formalno verifikovan [Klein2014].

## 1.5 Važnost mrežnih komunikacija

Još od svojih početaka povezivanje računara u mrežu za razmenu podataka se pokazalo kao vrlo korisno i postalo je ključno za mnoge oblasti računarske primene [Leiner2009].

Između ostalog, to je dovelo do pojave distribuiranih računarskih sistema [Tanenbaum2006]. U ovakvim sistemima prekid komunikacione veze može da uzrokuje probleme u radu ili čak prekid rada celokupnog računarskog sistema. Zbog velike važnosti funkcionisanja mreže definisan je pojam otpornosti mreže (engl. *network resilience*), koji predstavlja sposobnost mreže da obezbedi i održi prihvatljiv nivo usluge u prisustvu različitih poremećaja i izazova normalnom funkcionisanju. U ovu svrhu su definisani i okvirni (engl. *framework*) skupovi metoda i strategija, kao što je ResiliNets [Sterbenz2010].

Da bi se mrežni komunikacioni kanal zaštitio od otkaza (prekida veze) koristi se redundancija kao jedan od načina, odnosno uvodi se još najmanje jedan komunikacioni kanal koji preuzima komunikaciju u slučaju da glavni kanal zakaže. Redundancija u opštem smislu predstavlja dupliciranje kritičnih komponenti ili funkcija sistema u paraleli sa ciljem povećanja pouzdanosti sistema, najčešće u obliku rezervnog elementa u slučaju da primarni otkaže. Svaka duplicirana komponenta koja je dodata sistemu smanjuje verovatnoću otkaza prema formuli:

$$p = \prod_{i=1}^{n} p(i)$$

Ovde je $n$ broj komponenti, $p_i$ je verovatnoća otkaza komponente $i$, a $p$ verovatnoća otkaza sistema. Ova formula pretpostavlja nezavisnost događaja otkaza tj. da je verovatnoća otkaza komponente B je ista bez obzira da li je komponeta A otkazala ili ne. Takođe važi pretpostavka da je samo jedna komponenta neophodna da bi sistem funkcionisao.

Tehnika koja se koristi za upotrebu redundancije u pri mrežnim komunikacijama se nazivom agregacija mrežnih veza. Agregacija mrežnih veza (engl. *link aggregation*) je način za korišćenje više paralelnih fizičkih mrežnih veza između dva uređaja kao jedan logički kanal. u cilju povećanja pouzdanosti mrežne konekcije, a takođe i način za povećanje ukupnog protoka podataka raspoređivanjem po raspoloživim vezama.

## 1.6 Tema i cilj istraživanja

Ovaj rad se bavi problemom pouzdanosti i dostupnosti servisa operativnog sistema sa posebnim akcentom na servis mrežne komunikacije. Operativni sistem zasnovan na mikrojezgru je organizovan modularno, što predstavlja osnovu za rešavanje datog problema, jer mogućava ograničavanje štete koju greška može da napravi. U takvom okruženju se razmatra povećanje pouzdanosti i propusnosti mrežne komunikacije kroz agregaciju mrežnih veza na nivou operativnog sistema. Na ovaj način se stvaraju uslovi za značajno pojednostavljenje aplikacija kojima je ovakav servis neophodan. Namera je da se agregacija mrežnih veza u okruženju mikrojezgra uporedi sa rešenjem istog problema u okruženju monolitnog jezgra radi uočavanja prednosti i mana oba pristupa i izvođenja zaključaka o svojstvima pristupa mikrojezgra.

Cilj istraživanja je da se kroz analizu postojećih načina povećanja pouzdanosti i propusnosti računarske mreže, dođe do pristupa koji omogućuje njihovo poboljšanje na nivou mikrokernel operativnog sistema *MINIX 3*. Ideja je da se napravi agregacioni modul i integriše u *MINIX 3* operativni sistem bez njegovih izmena i tako se dođe do rešenja koje ima prihvatljive performanse, a ujedno učini mrežni servis propusnijim, pouzdanijim i dostupnijim, što znači da ima manji broj otkaza i da mu se može više verovati.

## 1.7 Struktura rada

U poglavlju 2 je dat kratak istorijat *MINIX 3* operativnog sistema, razlozi za njegov izbor za predstavnika mikrokernel klase operativnih sistema, a zatim je dat opis najvažnijih osobina ovog sistema.
U poglavlju 3 je opisan mrežni podsistem operativnog sistema i dat je pregled njegovog funkcionisanja na različitim nivoima.
U poglavlju 4 je objašnjen pojam agregacije mrežnih veza i dat prikaz implementacija na popularnim komercijalnim operativnim sistemima
U poglavlju 5 je opisana implementacija agregacije mrežnih veza na *MINIX 3* operativnom sistemu.
U poglavlju 6 je dat opis testiranja performansi implementiranog rešenja za agregaciju mrežnih veza na *MINIX*-u i prikaz dobijenih rezultata.
U poglavlju 7 su data zaključna razmatranja

U prvom prilogu je predstavljen modul za testiranje mrežnog drajvera, dok je u drugom dodatku dat opis pokretanja *MINIX 3* operativnog sistema.

## 1.8 Literatura

[Accetta1986] Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young. Mach: A new kernel foundation for UNIX development. In Proceedings of the Summer 1986 USENIX Conference, pages 93--112, Atlanta, GA, June 9--13, 1986. Usenix Association.

[Atzori2010] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[Avižienis2004] Avižienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. on Dependable and Secure Computing, 1(1):11–33, Jan.–Mar. 2004.

[CBSNews2010] CBSNews, “Toyota ‘Unintended Acceleration’ Has Killed 89,” CBSNews, 25-May-2010. [Online]. Available: http://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89/ [Accessed: 01-Apr-2016].

[Chou2001] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. 2001. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01). ACM, New York, NY, USA, 73-88.

[Denning1976] P. J. Denning, “Fault Tolerant Operating Systems,” ACM Comput. Surv., vol. 8, no. 4, pp. 359–389, Dec. 1976.

[Dennis1966] J. B. Dennis and E. C. Van Horn, “Programming Semantics for Multiprogrammed Computations,” Commun. ACM, vol. 9, no. 3, pp. 143–155, Mar. 1966.

[Dinh-Trong2005] T. T. Dinh-Trong and J. M. Bieman, "The FreeBSD project: a replication case study of open source development," in IEEE Transactions on Software Engineering, vol. 31, no. 6, pp. 481-494, June 2005. doi: 10.1109/TSE.2005.73

[Dowson1997] M. Dowson, “The Ariane 5 Software Failure,” SIGSOFT Softw. Eng. Notes, vol. 22, no. 2, p. 84–, Mar. 1997.

[Dunn2013] M. Dunn, “Toyota’s killer firmware: Bad design and its consequences,” EDN, 28-Oct-2013. [Online]. Available: http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences [Accessed: 01-Apr-2016].

[Evans2011] D. Evans, “The Internet of Things: How the Next Evolution of the Internet Is Changing Everything,” Cisco IBSG, White Paper BSG_0411, Apr. 2011.

[Ganapathi2005] Archana Ganapathi, David Patterson, "Crash Data Collection: A Windows Case Study," 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 280-285, 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005

[Haertig1997] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, and S. Schönberg, “The Performance of µ-kernel-based Systems,” in Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, 1997, pp. 66–77.

[Hatton1997] L. Hatton, "Reexamining the fault density component size connection," in IEEE Software, vol. 14, no. 2, pp. 89-97, Mar/Apr 1997. doi: 10.1109/52.582978

[Hildebrand1992] D. Hildebrand. An architectural overview of QNX. In 1st USENIX Workshop on Micro-kernels and Other Kernel Architectures, pages 113–126, Seattle, WA, April 1992

[Keizer2014] G. Keizer, “Microsoft urges customers to uninstall ‘Blue Screen of Death’ update,” Computerworld, 17-Aug-2014. [Online]. Available: http://www.computerworld.com/article/2491256/malware-vulnerabilities/microsoft-urges-customers-to-uninstall-blue-screen-of-death-update.html [Accessed: 01-Apr-2016].

[Kinshuman2011] Kinshuman Kinshumann, Kirk Glerum, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt. 2011. Debugging in the (very) large: ten years of implementation and experience. Commun. ACM 54, 7 (July 2011), 111-116.

[Klein2014] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser, “Comprehensive Formal Verification of an OS Microkernel,” ACM Trans. Comput. Syst., vol. 32, no. 1, pp. 2:1–2:70, Feb. 2014.

[Lameter2007] Lameter, Christoph. "Extreme High Performance Computing or Why Microkernels suck.", Proceedings of the Linux Symposium Volume One 2007

[Lee1990] P.A. Lee, T. Anderson, Fault Tolerance: Principles and Practice, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1990, ISBN: 0387820779.

[Leiner2009] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. 2009. A brief history of the internet. SIGCOMM Comput. Commun. Rev. 39, 5 (October 2009), 22-31.

[Leveson1993] Leveson, N. G. and Turner, C. S. An Investigation of the Therac-25 Accidents. IEEE Computer, 26(7):18–41, July 1993

[Levy1984] H. M. Levy, Capability-Based Computer Systems. Newton, MA, USA: Butterworth-Heinemann, 1984.

[LinuxCounter2016] C. Löhner, “Funny Statistics for the Linux Kernel - Lines of Code, Bad words, Good words - The Linux Counter Project - Statistics about Linux, its Users and more.” [Online]. Available: https://www.linuxcounter.net/ [Accessed: 01-Apr-2016].

[Miller2003] M. Miller, K.-P. Yee, and J. Shapiro, “Capability Myths Demolished,” 2003.
[Murphy2004] Brendan Murphy. 2004. Automating Software Failure Reporting. Queue 2, 8 (November 2004), 42-48.
[Ostrand2005] T. J. Ostrand, E. J. Weyuker and R. M. Bell, "Predicting the location and number of faults in large software systems," in IEEE Transactions on Software Engineering, vol. 31, no. 4, pp. 340-355, April 2005. doi: 10.1109/TSE.2005.49
[Rozier1988] M. Rozier, A. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser. CHORUS distributed operating system. Computing Systems, 1(4):305–370, 1988
[Rushby1981] J. M. Rushby, “Design and Verification of Secure Systems,” in Proceedings of the Eighth ACM Symposium on Operating Systems Principles, New York, NY, USA, 1981, pp. 12–21.
[Saltzer1974] J. H. Saltzer, “Protection and the Control of Information Sharing in Multics,” Commun. ACM, vol. 17, no. 7, pp. 388–402, Jul. 1974.
[Sterbenz2010] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, “Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines,” Computer Networks, vol. 54, no. 8, pp. 1245–1265, Jun. 2010.
[Tanenbaum2006] A. S. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigms (2Nd Edition). Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.
[Tanenbaum2014] Andrew S. Tanenbaum and Herbert Bos. 2014. Modern Operating Systems (4th ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
[Wikipedia2016] “List of software bugs,” Wikipedia, the free encyclopedia. 01-Apr-2016. https://en.wikipedia.org/wiki/List_of_software_bugs
[Xu1999] Jun Xu, Z. Kalbarczyk and R. K. Iyer, "Networked Windows NT system field failure data analysis," Dependable Computing, 1999. Proceedings. 1999 Pacific Rim International Symposium on, 1999, pp. 178-185.

# 2. Primer operativnog sistema zasnovanog na mikrojezgru: *MINIX 3*

*MINIX 3* operativni sistem je doneo nov, bezbedniji način rada sa drajverima, koji su postali korisnički procesi, a hardveru pristupaju uz pomoć kernelskih poziva. Iako je ovaj operativni sistem prilično detaljno opisan u [Tanenbaum2006], u međuvremenu je tokom daljeg razvoja došlo do nekoliko značajnijih izmena u njegovom načinu rada (na primer uvođenje asinhrone međuprocesne komunikacije). Verzija *MINIX 3* korišćena za ovu tezu je *MINIX 3.3.0* i zbog toga je u ovom poglavlju dat kratak pregled mehanizama ove verzije *MINIX*-a koji su vezani za pokretanje i rad drajvera.

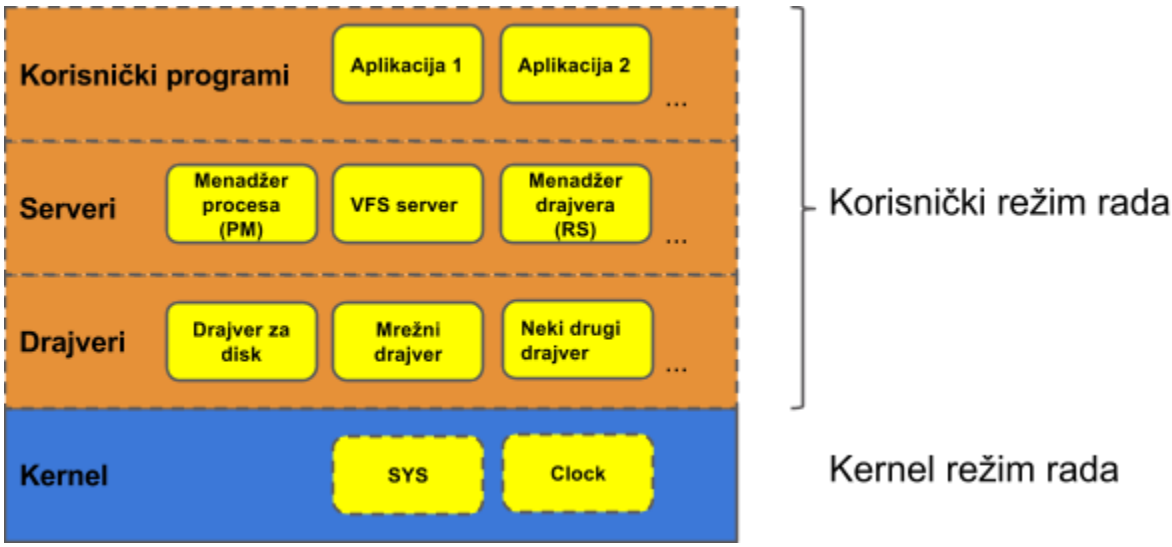
## 2.1 Kratka istorija MINIX-a

Kada je *UNIX* razvijan u okviru kompanije *AT&T* 60-tih i objavljen 70-tih godina prošlog veka [Ritchie1974], njegov izvorni kod je bio manje-više javno dostupan, pa su univerziteti širom sveta usvojili *UNIX* kao osnovu za nastavu iz oblasti operativnih sistema. Mnogi univerzitetski kursevi o operativnim sistemima su bili zasnovani na komentarima Verzije 6 UNIX-a [Lions1977]. Kada je kompanija *AT&T* shvatila komercijalnu vrednost *UNIX*-a kasnih 70-tih, promenjena je licenca sa ciljem zabrane svake eksterne upotrebe izvornog koda *UNIX* sistema, između ostalog i u nastavne svrhe. Ovo je uzrokovalo nastanak mnogih operativnih sistema, koji nisu bili zasnovani na izvornom kodu *AT&T UNIX*-a, ali su se ponašali kao *UNIX* tj. imali su dobar deo njegovih osobina. Ovakvi operativni sistemi su bili nazivani *UNIX* klonovi, *UN*X* ili engl. "*UNIX-like*".
Nedostatak malog i jednostavnog operativnog sistema pogodnog za edukaciju, je između ostalog inicirao i nastanak *MINIX*-a, novog *UNIX* klona, koji je objavljen 1987. godine na Vrije univerzitetu u Amsterdamu [Tanenbaum1987]. *MINIX* je bio funkcionalno kompatibilan sa *UNIX*-om, ali je imao više modularnu strukturu i implementirao je delove funkcionalnosti izvan kernela. Tačnije, rukovalac procesima (proces menadžer) i fajl server su bili pokretani kao nezavisni korisnički procesi, dok su svi drajveri bili u kernelu (zbog čega se ne može nazvati mikrokernelom). Zajedno sa pratećom knjigom i dostupnim celokupnim izvornim kodom, *MINIX* je ubrzo postao široko korišćen u nastavi na mnogim univerzitetima.

Iako je izvorni kod MINIX-a bio dostupan, modifikacija i redistribucija su bili ograničeni, što je stvorilo prostor za nastanak Linux jezgra 1991. godine [Torvalds1991; Torvalds2001]. Nasuprot *MINIX*-u, *Linux* kernel se vratio monolitnoj strukturi, što je uzrokovalo čuvenu debatu na *comp.os.minix Usenet* grupi između Tanenbauma i Torvaldsa [DiBona1999]. *Linux* od svog nastanka nije značajnije promenio svoju arhitekturu i još uvek ima formu monolitnog kernela, koje je u međuvremenu veoma naraslo. *MINIX* je međutim ostao mali i jednostavan. Najznačajnije verzije su bile *MINIX* 1.5, koja je portovana na nekoliko različitih arhitektura, kao i *MINIX* 2, koja je dodala *POSIX* kompatibilnost [IEEE1990], ali je osnovna arhitektura uglavnom ostala ista.

### 2.1.1 Mikrokernel

koncept mikrokernela je razvijen 80-tih godina prošlog veka, sa idejom da se što više servisa kernela, a pre svega drajveri, razdvoje kao zasebni moduli i pokreću kao procesi u neprivilegovanom režimu. Ova modularizacija je omogućila lakši nezavisan rad na pojedinačnim servisima, kao i na preostalom kernelskom kodu. Istovremeno, modularizacija je činila sistem i bezbednijim, jer je greška u nekom od neprivilegovanih modula bila izolovana, pa je ostatak sistema mogao da nesmetano nastavi rad.
U pokušaju da napravi pouzdan operativni sistem, profesor Tanenbaum 2005. godine okuplja grupu razvojnih programera sa ciljem razvoja verzije 3 *MINIX*-a [Tanenbaum2006], ovog puta potpuno zasnovanom na multiserverskoj mikrokernel paradigmi.




Slika 2.1: Multiserverska struktura MINIX-a 3

U verziji *MINIX* 3 , svi drajveri su izbačeni iz kernela i pretvoreni u nezavisne korisničke procese. Time je *MINIX* 3 postao potpuno modularan sistem koji izvršava svaki sistemski server i drajver kao zaseban proces (slika 2.1).
Od 2010. godine fokus *MINIX* 3 projekta je prebačen na ugrađene (engl *embedded*) sisteme, pre svega na računare zasnovane na ARM procesorima u obliku “sistema na čipu” ili SoC (engl. *system on chip*). Početkom 2013. godine razvoj MINIX 3 se usmerava na kompatibilnost sa *NetBSD* operativnim sistemom, kako bi se na taj način uvećao broj raspoloživih programa za *MINIX* 3. Ipak, osnovni cilj nastanka i razvoja verzije 3 MINIX-a je da se pokaže da je moguće da se napravi upotrebljiv operativni sistem, koji se može smatrati pouzdanim [Herder2006]. Ovo istraživanje predstavlja nastavak napora u cilju poboljšanja oslonjivosti operativnog sistema povećavanjem otpornosti na poremećaje.

## 2.2 Struktura operativnog sistema *MINIX* 3

*MINIX* 3 operativni sistem je pravljen da se izvršava u 32-bitnom zaštićenom režimu procesora x86 arhitekture (*IA-32*)[1], da bi mogao da koristi mogućnosti hardverske zaštite, kao što su virtuelna memorija i zaštitni prstenovi [Intel2015]. Zaštitni prstenovi omogućavaju izvršavanje programa sa različitim stepenima privilegija, a stepen sa najviše privilegija se naziva "privilegovanim". Privilegovan režim rada procesora i privilegovani memorijski prostor se obično vezuju za kritične delove sistema kao što je jezgro tj. kernel pa se zato nazivaju kernel prostor i kernel režim. Mikrokernel dizajn je minimalistički i striktno razdvaja politiku *(policy*) i mehanizme. U mikrokernelu se implementiraju samo najosnovniji mehanizmi, koji zahtevaju privilegije kernel režima, a sva politika se određuje od strane servera i drajvera [Liedtke1995]. MINIX 3 sledi ovaj pristup. Iako se svi procesi jednako tretiraju, razaznaju se logički slojevi, kao što se vidi na slici 2.1. Na najnižem nivou je malo mikrojezgro od oko 10 hiljada linija izvornog koda tj. LoC (engl. *Lines of Code*), koje izvodi privilegovane operacije. Mikrokernel presreće hardverske prekide, hvata izuzetke u korisničkim procesima, rukuje komunikacijom između procesa i podešava CPU (engl. *central processing unit*) i MMU (engl. *memory management unit*) kako bi izvršavanje procesa bilo moguće. Mikrokernel takođe sadrži posebne kernelske procese koji se nazivaju "taskovi" (engl. *task* - zadatak) i koji se normalno raspoređuju kao i ostali procesi. Prvi task je "*CLOCK*" koji obrađuje prekid sistemskog sata, vodi računa o sistemskom vremenu, obavlja raspoređivanje procesa i upravlja tajmerima i alarmima. Iako bi neke od ovih funkcionalnosti mogle da se implementiraju na korisničkom nivou, to bi bilo vrlo neefikasno. Drugi sistemski task (*SYS* sa slike) pruža mali skup kernel poziva za podršku autorizovanim serverima i drajverima korisničkog nivoa u izvršavanju svojih poslova. U suštini ovaj task daje privilegovanu funkcionalnost, koja ne može da postoji na korisničkom nivou.

Sledeći nivo iznad sadrži drajvere uređaja. Postoji po jedan drajver za svaki od glavnih uređaja, uključujući drajvere za diskove, mrežu, video, itd. Dodatno, sloj drajvera sadrži drajvere za fajl sisteme i mrežne protokole, kao što su fajl server i mrežni server. Svaki drajver je korisnički proces ograničen na isti način kao i ostali obični procesi. Drajveri su posebni samo u smislu da im je dozvoljeno da zovu mali broj kernel poziva, da bi kernel za njih izvršio privilegovane operacije, kao što su postavljanje rukovaoca prekida (engl. *interrupt handler*), čitanje i pisanje U/I uređaja ili kopiranje memorije između adresnih prostora.

Iznad sloja drajvera nalazi se sloj servera. Serveri su zasebni procesi, od kojih svaki ima određenu ulogu u operativnom sistemu. Na primer, serveri PM (*Proces Manager*) i VFS (*Virtual File System*) implementiraju POSIX API i obezbeđuju upravljanje procesima i memorijom, kao i virtuelnim fajl sistemom. Server DS (*Data Store*) obezbeđuje servis imenovanja javnih podataka koji može da se koristi kako bi se sačuvalo i posle vratilo stanje pojedinih procesa. Postoji još nekoliko servera, kao npr. IS (*Information Server*), koji čuva "dampove" sistemskih struktura, koje kasnije mogu da se koriste za debagovanje.
Serveri i drajveri se zajednički nazivaju **sistemski procesi** ili servisi i deo su operativnog sistema, iako su zapravo zasebni procesi. Razlike u odnosu na obične korisničke procese su sledeće:

[1] U međuvremenu je nastala i verzija za 32 bitne ARM procesore, koji takođe podržavaju privilegovan režim rada.

- ima ih ograničen broj (na *MINIX* verziji 3.3.0 ovaj broj je 64),
- dozvoljeno im je da pristupaju nekim od kernel poziva,
- dozvoljeno im je da komuniciraju sa nekim sistemskim procesima,
- imaju definisan memorijski i I/O prostor koji mogu da koriste uz pomoć kernel poziva.

Sve ove informacije o sistemskim procesima održava kernel u svojim za to namenjenim strukturama podataka. Kernel posreduje u pristupima svim resursima tako da nije moguć pristup resursu na koji dati sistemski proces nema pravo. Ovakav dizajn pravi mali overhed, ali omogućava preciznu kontrolu rada sistemskih procesa, što znači da greška u nekom sistemskom procesu neće da sruši ceo sistem. Roditeljski (engl. *parent*) proces za sve sistemske procese je server koji se naziva RS (*Reincarnation Server*). On može da pokreće nove sistemske procese, kao i da u hodu restartuje one koji su se "zaglavili".

Iznad serverskog sloja nalaze se obični, neprivilegovani aplikacioni programi, kao što su komandni interpreter, editor teksta i sl. Kada se sistem podigne, `init` je prvi aplikacioni proces, koji pokreće `getty` procese, koji posle uspešnog logovanja startuju komandni interpreter (engl. *shell*). On dozvoljava pokretanje ostalih aplikacija. Jedina razlika između ovog i ostalih *UNIX* sistema je da bibliotečke procedure za sistemske pozive zapravo šalju poruke serverskom sloju. Iako se interno šalju poruke, sistemske biblioteke programeru pružaju standardan *POSIX* aplikacioni interfejs (*POSIX API*).

## 2.3 Međuprocesna komunikacija (IPC)

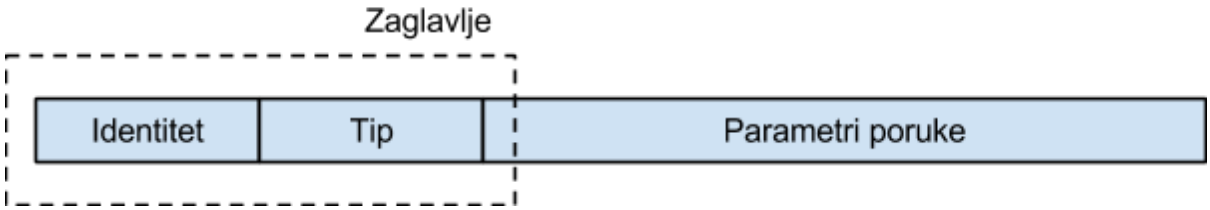
Pošto *MINIX* 3 izvršava sve servere i drajvere kao nezavisne procese, oni ne mogu da direktno pristupaju tuđim funkcijama ili strukturama podataka. Umesto toga, proces mora da pravi udaljeni poziv odnosno *RPC* (engl. *remote procedure call*) kada hoće da sarađuje sa drugim procesom. Tačnije, kernel obezbeđuje usluge međuprocesne komunikacije ili IPC (engl. *interprocess communication*) zasnovane na prenošenju poruka (engl. *message passing*). Ako dva procesa treba da komuniciraju, pošiljalac sastavlja poruku u svom adresnom prostoru i vrši IPC poziv da bi je poslao drugoj strani. Deo koda, koji se linkuje sa aplikacijom, stavlja IPC parametre na stek i izvršava odgovarajuću privilegovanu instrukciju, uzrokujući softverski prekid koji predaje kontrolu kernelovom IPC podsistemu. Kernel zatim proverava IPC parametre i izvršava odgovarajući deo za rukovanjem IPC. Poruke se ne baferuju u kernelu, nego se kopiraju ili mapiraju od pošiljaoca ka primaocu, ubrzavajući tako IPC i eliminišući mogućnost da nestane bafera.

Može se razlikovati nekoliko različitih IPC režima interakcije. Prvi, sinhroni IPC je dvosmerna komunikacija, gde se inicirajući proces blokira dok druga strana ne postane spremna. Kada su obe strane spremne, poruka se kopira ili mapira od pošiljaoca ka primaocu, posle čega obe strane mogu da nastave izvršavanje. Termin *IPC roundtrip* se koristi ukoliko pošiljalac sinhrono očekuje odgovor posle slanja zahteva. Drugi, asinhroni IPC znači da pošiljalac može da nastavi sa izvršavanjem bez blokiranja od strane kernela, a IPC podsistem preuzima brigu o poruci i isporučuje je u ime pošiljaoca prvom prilikom. I sinhroni i asinhroni IPC mogu da se kombinuju sa tajmautima kako bi IPC bio opozvan ukoliko nije uspeo da se izvrši u zadatom vremenskom intervalu. Ipak, pošto nalaženje odgovarajuće vrednosti tajmauta nije trivijalno, obično se koristi vrednost nula ili beskonačno. Prvi (kada je vrednost tajmauta nula) se naziva neblokirajući IPC, jer se isporuka pokuša jednom i status odmah biva vraćen. Drugi (vrednost tajmauta beskonačno) se zove blokirajući IPC i odgovara normalnoj sinhronoj IPC interakciji.

| **Operacija** | **Značenje** | **Skladištenje** | **Režim** | **Blokiranje** |
|---|---|---|---|---|
| SEND | Blokiraj dok se poruka ne pošalje | kod pošiljaoca | sinhroni | blokirajuća |
| RECEIVE | Blokiraj dok poruka ne stigne | - | sinhroni | blokirajuća |
| SENDREC | Pošalji poruku i čekaj odgovor | kod pošiljaoca | sinhroni | blokirajuća |
| SENDNB | pošalji ako primaoc očekuje | kod pošiljaoca | sinhroni | neblokirajuća |
| SENDA | Baferovana isporuka | kod pošiljaoca | asinhroni | neblokirajuća |
| NOTIFY | Mehanizam signaliziranja događaja | u kernelu | asinhroni | neblokirajuća |

Slika 2.2: Operacije sa rad sa porukama MINIX-ovog IPC mehanizma

Operacije sa porukama i format poruka koje koristi *MINIX* 3 IPC podsistem su dati na slikama 2.2 i 2.3. Najosnovnije IPC operacije su sinhroni blokirajući pozivi SEND i RECEIVE. Argumenti SEND poziva su IPC identitet (engl. *endpoint*) i pokazivač na bafer poruke. Argument RECEIVE poziva je pokazivač na bafer poruke. Pri SEND pozivu, baferovanje tj. skladištenje poruke nije potrebno, jer je poziv blokiran sve dok se poruka ne iskopira u memorijski prostor procesa koji je prima. SENDREC poziv kombinuje funkcionalnosti prethodna dva poziva, tako što radi sinhroni SEND, pa zatim implicitni RECEIVE. Ovo ne samo da izbegava dodatni kernel poziv, već i sprečava pojavu "utrkivanja" (engl. *race condition*) u slučaju da primaoc odgovori neblokirajućim IPC pozivom (SENDA). Ostali IPC pozivi su neblokirajući. SENDNB je neblokirajuća varijanta sinhronog SEND, koja vraća grešku ako druga strana nije spremna u vreme poziva. SENDA podržava asinhroni IPC sa baferovanjem poruka sa strane pošiljaoca. Argumenti su pokazivač i veličina tabele sa baferima poruka za slanje. Svako mesto u tabeli sadrži sve informacije potrebne za isporuku poruke, kao što su flegovi, IPC identitet, sama poruka i statusna reč. Pošiljaoc ne biva blokiran, nego se poziv odmah vraća, a kernel pretražuje tablelu sa porukama za slanje, garantujući isporuku što je pre moguće. Na kraju tu je još i asinhroni NOTIFY poziv, koji podržava signalne događaje servera i drajvera. Poziv je neblokirajući i ako notifikacija ne može da se direktno isporuči, kernel ga označava kao notifikacija u čekanju (engl. *pending*) u bit mapi u strukturi procesa primaoca. Čekajuće notifikacije istog tipa se spajaju i bivaju isporučene samo jednom.




Slika 2.3: Format IPC poruke

Zbog jednostavnosti MINIX3 koristi samo male poruke fiksne dužine. Ove poruke su strukture sa zaglavljem koje sadrži izvorište poruke i njen tip, kao i uniju različitih formata koji sadrže argumente poruke. Ova struktura je prikazana na slici 2.3. Veličina poruka zavisi od procesorske arhitekture, određuje se u vreme kompajliranja i predstavlja maksimum veličina svih tipova u uniji. Tip poruke i sadržaj može slobodno da postavlja pošiljalac, ali kernel prilikom isporuke upisuje IPC identitet (krajnju tačku pošiljaoca), tj. polje izvorišta poruke. Na ovaj način primaoc uvek pouzdano zna ko je poslao poruku.

## 2.4 Upravljanje drajverima

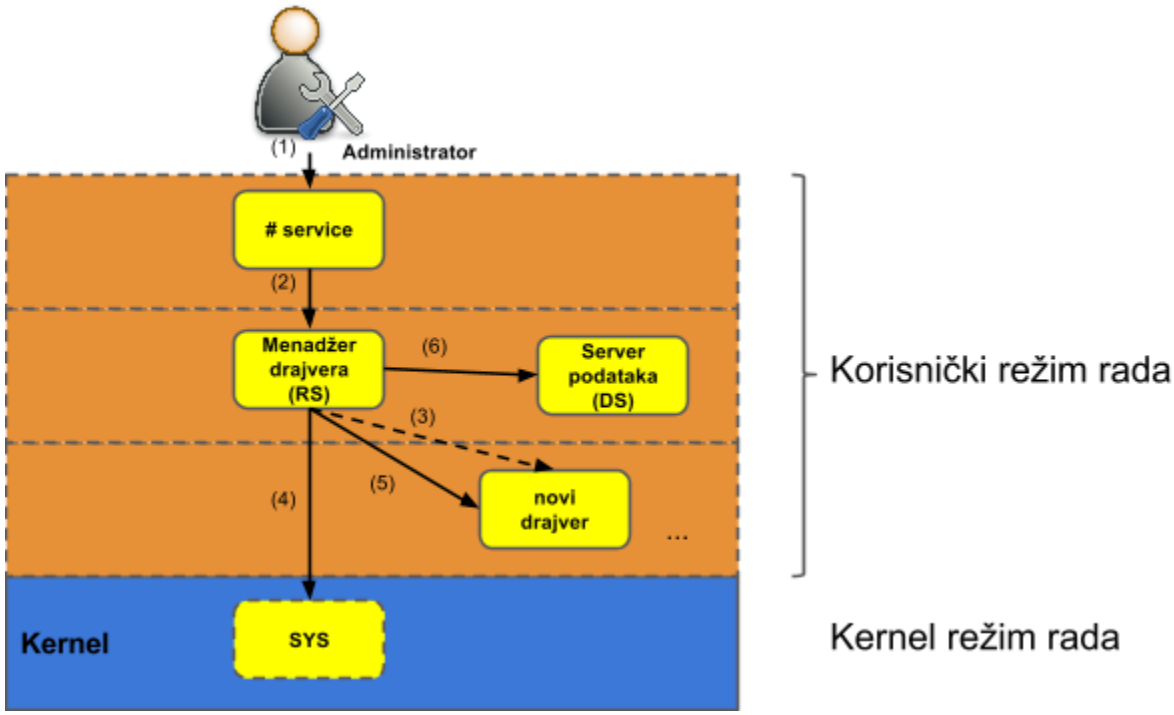
Pošto su svi serveri i drajveri obični korisnički procesi, oni mogu da se kontrolišu kao i svaka druga obična aplikacija. Iako su drajveri prvobitno bili deo MINIX but imidža, sada postoji podrška za pokretanje i zaustavljanje drajvera u hodu. Tačnije, za upravljanje ovim mehanizmom se koristi server RS (drajver menadžer), koji upravlja svim sistemskim procesima, odnosno serverima i drajverima u sistemu. Administrator sistema može da obavlja radnje kao npr. pokretanje novog drajvera, korišćenjem programa "`service`", koji ovaj zahtev prenosi serveru RS. Osnovna komanda za pokretanje novog drajvera bi glasila ovako:

```
$ service up <driver binary> –dev <device node>
```

Slično tome, restartovanje ili zaustavljanje drajvera može da se obavi komandama "`service refresh`" i "`service down`". Takođe postoji podrška za zamenu drajvera novom verzijom bez prestanka funkcionisanja. Ova procedura se naziva "*dynamic update*", a pokreće se komandom "`service update`". Pored `-dev` parametra, program "`service`" komanda podržava još nekoliko naprednih parametara za konfiguraciju sistema. Tako parametar `-args` dozvoljava prosleđivanje niza argumenata komponenti koja se pokreće, slično mehanizmu za prosleđivanja parametara komandne linije aplikaciji. Parametar `-dev` se koristi isključivo za drajvere i uzrokuje da VFS server bude informisan o inodu koji je pridružen datom uređaju. Parametar `-label` omoućava da se navede posebno ime. S obzirom da svaki proces ima jedinstveni IPC identitet izvorišta koji generiše kernel, sistemski procesi ne mogu lako da pronađu jedan drugog. Zbog toga su uvedeni stabilni identifikatori, koji se sastoje od imena čitljivog za ljude i opcionog broja, što se objavljuje u DS (*Data Store*) serveru.

Ostali parametri mogu da se koriste da se postavi posebna politika za mehanizme tolerancije na poremećaje sistema. Za početak drajverima je pridružena politika izolacije koja kontroliše koje privilegovane operacije oni mogu da izvrše. Podrazumevano mesto na kome

se navode privilegije drajvera je fajl `/etc/system.conf`, ali drugi konfiguracioni fajl može da se specificira parametrom `-config`. Da bi proverio funkcionisanje svih drajvera, RS može periodično da zahteva (*heartbeat*) poruku. Parametar `-period` dozvoljava fino podešavanje frekvencije provere. RS može automatski da restartuje komponente koje su otkazale, ali ako je potrebna veća fleksibilnost, moguće je specificirati skript sa naprednijom politikom oporavka. Konačno, parametar `-c` kaže RS serveru da li da pravi memorijsku kopiju izvršnog koda drajvera. Ova mogućnost se koristi npr. za oporavak disk drajvera koji je otkazao, jer u slučaju otkaza disk drajvera nije moguće pristupiti disku da bi se sa njega učitao kod drajvera, radi njegovog ponovnog pokretanja.




Slika 2.4: Pokretanje sistemskog procesa/drajvera

Procedura za pokretanje drajvera je strogo definisan niz događaja, kao što je pokazano na slici 2.4. Na početku (1) administrator odlučuje o politici drajvera i poziva program `service`, koji proverava ispravnost poziva i (2) prosleđuje poziv RS-u. RS proverava autorizaciju poziva i čuva zapis (bit-mapu) koja opisuje datu politiku. Zatim RS pokreće sam drajver tako što (3) kreira novi proces za drajver i traži IPC endpoint koji jedinstveno identifikuje proces, (4) informiše ostale delove sistema o novom procesu i privilegijama koje su mu dodeljene izolacionom politikom i (5) po ostvarivanju izolacije, izvršava binarni kod drajvera. Na kraju se detalji o upravo pokrenutom drajveru objavljuju u DS-u i (6) zavisne komponente se obaveštavaju o novoj konfiguraciji sistema.

Server DS (*data store*) omogućava dinamičku konfiguraciju sistema, bez određivanja unapred svih zavisnosti. U svojoj suštini, DS je server - baza podataka, koji dozvoljava sistemskim komponentama da korišćenjem imena kao ključa pišu i čitaju celobrojne vrednosti, stringove, ili čak kompletne delove memorije. Sistemski procesi mogu da smeste podatke javno (mogu svi da ih čitaju) ili privatno (može da čita samo dati proces). Posebna

osobina DS-a je da nudi mehanizam objave i pretplate za sve javne podatke. Ova osobina omogućava da se razdvoje proizvođači i potrošači (podataka). Komponente mogu da se pretplate na određene događaje, navođenjem odgovarajućeg imena ili regularnog izraza. Kadgod je neki podatak objavljen ili obnovljen, DS automatski obaveštava sve komponente koje su se na taj podatak pretplatile. Ove osobine čine DS veoma pogodnim za ulogu servera imenovanja (engl. *naming*). Po učitavanju novog servera ili drajvera, RS objavljuje stabilno ime i IPC identitet na DS-u, kako bi o tome automatski obavestio sve zainteresovane komponente. Npr. mrežni server je pretplaćen na ključ "`drv.net.*`" da bi primao obaveštenja o sistemskim mrežnim drajverima. Zato je mrežni server automatski obavešten kad god se mrežni drajver startuje ili restartuje, tako da može da pokrene (re)inicijalizaciju ili oporavak.

# 2.5 Izolacija drajvera

*MINIX3* obezbeđuje mehanizme za izolaciju delova operativnog sistema (sistemskih procesa), kako greška u jednom od delova ne bi "srušila" ceo sistem. Prilikom konstrukcije *MINIX3* sistema je primenjen princip najmanje nadležnosti tj. POLA (engl. *Principle Of Least Authority*), u literaturi još poznat i kao princip najmanje privilegije tj. POLP (engl. *Principle Of Least Privilege*). Ukratko, ovaj princip kaže da svaka komponenta treba da može da pristupi samo resursima koji su potrebni za njenu legitimnu svrhu. Malo kompletnija definicija kaže: "Svaki program i svaki korisnik sistema treba da funkcioniše koristeći najmanji skup privilegija neophodan da se obavi dati posao. Primarno, ovaj princip ograničava štetu koja može da nastane slučajnom greškom. Takođe smanjuje na minimum broj potencijalnih interakcija među privilegovanim programima koje su neophodne za ispravno funkcionisanje, tako da je nenamerna, neželjena ili nedozvoljena upotreba privilegije manje verovatna." [Saltzer75]. Skup mehanizama koji ovo obezbeđuju naziva se izolaciona arhitektura. Iako su mehanizmi izolacione arhitekture implementirani softverski, oni se takođe oslanjaju na podršku u hardveru (režimi rada CPU, MMU, IOMMU, ...). Ovo je sve zajedno definisano kao poverljiva računarska osnova ili TCB (engl. *Trusted Computing Base*) [Lampson1992]. Da bi se smanjila verovatnoća pojave greške u TCB, on treba da bude što manji. Jedino TCB deo sistema sme da izvršava privilegovane operacije (operacije koje mogu da uzrokuju pad sistema).

## 2.5.1 Klasifikacija privilegovanih operacija

U [Herder2009] su identifikovane četiri ortogonalne klase privilegovanih operacija koje se mapiraju na osnovne komponente svakog računarskog sistema: procesor, memoriju, periferijske uređaje i sistemski softver.

### 2.5.1.1 Klasa I: upotreba procesora

Proces koji se izvršava u kernel režimu ima pristup celom skupu privilegovanih CPU instrukcija koje mogu da budu upotrebljene kako bi se zaobišli mehanizmi zaštite višeg nivoa. Drajver koji se izvršava u ovom režimu može, na primer, da resetuje tabele stranica, da radi U/I, onemogući prekide ili čak da zaustavi procesor. Ove funkcije su vitalne za ispravno funkcionisanje operativnog sistema i ne mogu biti na raspolaganju nepouzdanom kodu bez ugrožavanja celog sistema. Ipak, zbog neophodnosti rada na niskom nivou, drajveri moraju da budu u stanju da obavljaju određene privilegovane operacije, koje nisu

normalno dostupne aplikacijama korisničkog nivoa, što može da stvori problem. Drajver može da uzurpira procesor, što može da dovede do problema sa preformansama ili čak da "sruši" sistem. Na primer drajver može da nehotično sadrži beskonačnu petlju i da time blokira procesor i obori sistem, jer drajver niskog nivoa obično nije pod kontrolom raspoređivača procesa.

#### 2.5.1.2 Klasa II: pristup memoriji

Pošto drajveri često imaju potrebu da razmenjuju podatke sa sistemskim serverima i aplikativnim programima, posebno važna pretnja je mogućnost neovlašćene izmene memorije putem neovlašćenog pristupa. Na primer, drajver koji je zadužen za ulaz/izlaz uređaja mora da kopira podatke iz/u određene bafere koji se nalaze u različitim adresnim prostorima. Pokazivač iz virtuelnog adresnog prostora ne može da bude korišćen direktno od strane drajvera i uređaja, nego pre toga mora da bude preveden u fizičke adrese. Greške u prevođenju adresa ili kopiranje više podataka od kapaciteta bafera uzrokuje neovlašćene izmene memorije.

Direktan pristup memoriji (DMA) je specijalan slučaj U/I operacije nad uređajem. DMA operacija mora da bude ograničena da bi se sprečilo prepisivanje proizvoljnog dela memorije. Uređaj, koji podržava DMA može da direktno prebaci podatke na proizvoljnu lokaciju u fizičkoj memoriji bez intervencije CPU i MMU, zaobilazeći hardverske i softverske zaštitne mehanizme. Stari ISA uređaji se oslanjaju na DMA kontroler na matičnoj ploči, dok PCI uređaji mogu da imaju ugrađena DMA svojstva. Termin "*bus-mastering DMA*" se koristi ako uređaj može da preuzme kontrolu nad magistralom i pokrene DMA prenos. Kernel ne kontroliše situaciju tokom DMA prenosa i ne može da zna da li tražena operacija može da bude izvršena bezbedno. Dodatno, uređaj koristi U/I adrese koje MMU hardver ne može da proveri. Tako pogrešan ili zlonameran drajver čiji je uređaj sposoban za DMA može da neovlašćeno izmeni bilo koji deo fizičke memorije koristeći pogrešne U/I adrese.

#### 2.5.1.3 Klasa III: Ulaz/Izlaz

Važno je ograničiti pristup U/I portovima, registrima i memoriji uređaja, kako bi bili sprečeni neovlašćeni pristupi. Na primer, mrežni drajver bi smeo da pristupa samo mrežnoj karti za koju je namenjen, ali ne i ostalim uređajima. Ukoliko se drajver nalazi u kernelu, on može da pristupi bilo kog uređaja. Ukoliko više drajvera nezavisno pristupa istom uređaju, konflikti su vrlo verovatni. To može da uzrokuje greške u podacima ili čak prestanak funkcionisanja sistema. Programiranje hardvera uređaja je podložno greškama zbog niskog nivoa pristupa koji često nije dovoljno dobro dokumentovan.

Rukovanje prekidima takođe donosi probleme, jer su prekidi inherentno asinhroni i zbog performansi moraju da budu opsluženi na najnižem nivou. Kada je uređaju potrebno opsluživanje, on izazove prekid (IRQ) da bi kernel preuzeo kontrolu. Tom prilikom kernel aktivira obrađivač prekida drajvera pridruženog uređaju, kako bi ga opslužio. Obrađivači prekida se obično izvršavaju na višem nivou prioriteta i moraju da zadovolje specijalna ograničenja specifična za OS, što čini rukovanje prekidima podložno greškama.

#### 2.5.1.4 Klasa IV: IPC

Međuprocesna komunikacija (IPC) donosi nekoliko pretnji sistemskom softveru. Iako se IPC često dovodi u vezu sa multiserverskim dizajnom, IPC je takođe uobičajeno sredstvo za komunikaciju između jezgra i proširenja sistema. Zato IPC podsistem mora da bude maksimalno robustan. Ako nepouzdan programski kod pravi IPC pozive (često na hiljade u

sekundi), greške u pozivima ili argumentima (kao što su pogrešan IPC identitet ili pogrešna adresa bafera) ne mogu biti sprečene. Takođe, pošto i pouzdani i nepouzdani delovi sistema zavise od IPC-a, mogu da se pojave i neautorizovani pokušaji pristupa. To se desi kada na primer neispravan drajver želi da koristi kernel pozive za upravljanje procesima. Na kraju, neodmereno korišćenje globalnih resursa može da dovede do nedostatka nekog od njih kada jedan ili više klijenata zajedno naprave previše IPC zahteva. Ako se baferi poruka dinamički alociraju, IPC podsistem može da ostane bez memorije i da ne bude u stanju da opslužuje nove zahteve.
Čak iako IPC sistem radi pouzdano, neočekivane interakcije između procesa pošiljaoca i procesa primaoce poruke može da potencijalno poremeti rad sistema. Na primer, loš drajver može da pokvari sadržaj poruke, pošalje odgovor na pogrešnoj strani, izazove blokiranje zbog ciklične zavisnosti ili da jednostavno ne odgovori na zahtev. Posebno, odnosi sa asimetričnim poverenjem prave probleme kada se koristi sinhroni IPC. Na primer, nepouzdani klijent može da blokira server ako ne primi njegov odgovor. Slično, drajver u ulozi nepouzdanog servera može da blokira klijente, pa klijent ostaje blokiran sve dok drajver ne primi ili odgovori na IPC poziv.

Zahtevanj privilegovane operacije posredstvom IPC-a može da bude problematično. Zato drajveru, kome nije dozvoljeno da obavi neku operaciju direktno, takođe mora biti zabranjeno da zahteva od nekog drugog (privilegovanog) procesa da za njega obavi tu operaciju. Na primer, izolovanje drajvera u privatnom adresnom prostoru ne vredi mnogo, ako on može posredstvom kernela da pristupi proizvoljnom delu memorije. Slično tome, čak i kada neprivilegovani drajveri ne mogu direktno da obavljaju U/I, takva ograničenja nisu delotvorna ako postoji način da kernel taj posao obavi u ime drajvera bez provere prava.

## 2.5.2 Izolaciona arhitektura

Izolacija modula se na MINIX3 postiže kombinacijom nekoliko tehnika. Konkretno, drajveri se izoluju na tri različita načina:

- sistemska strukturna zaštita,
- statička politika izolacije za svaki drajver pojedinačno i
- mehanizmi dinamičke kontrole pristupa.

Osnova strukturne izolacije je činjenica da je svaki drajver nezavisan sistemski proces. Ovo čini drajver bezopasnim po ostatak sistema. Da bi drajver mogao da pristupa resursima koji su mu potrebni, statički mehanizmi dozvoljavaju fino podešavanje prava pristupa sistemskih procesa. Dinamički mehanizmi podržavaju sigurnu razmenu podataka između sistemskih procesa, omogućavanjem kontrolisanog pristupa jednog procesa memoriji drugog.

### 2.5.2.1 Sistemska strukturna zaštita

Upotreba multiserver modela zasnovang na mikrokernelu, modularizuje operativni sistem i donosi nekoliko prednosti kada je reč o oslonjivosti. Za početak smanjenje veličine kernela smanjuje njegovu kompleksnost, a samim tim i verovatnoću za pojavu grešaka. Mala veličina kernela može da učini izvodivim ručnu ili formalnu verifikaciju koda [Klein 2009], što daje dobru osnovu za dalju izgradnju pouzdanog operativnog sistema.
Za izolaciju eventualnih grešaka klučno je da su moduli (aplikacija, server ili drajver) enkapsulirani u zasebne procese sa zasebnim adresnim prostorom.

- Pošto se drajveri više ne izvršavaju u privilegovanom režimu procesora, oni ne mogu da direktno izvrše privilegovane i potencijalno opasne instrukcije i ne mogu da zaobiđu mehanizme restrikcije koje implementira ostatak operativnog sistema.
- Zato što MMU hardverski sprovodi striktno razdvajanje adresnog prostora procesa, mnogi problemi sa neovlašćenim izmenama memorije su sprečeni samom strukturom sistema. Ako npr. neki memorijski pokazivač pokaže van predviđenog adresnog prostora, to će izazvati CPU/MMU izuzetak (engl. *exception*).
- Drajveri se takođe izvršavaju sa neprivilegovanim UID-om (engl. *user identification*), pa zaštitni mehanizmi operativnog sistema mogu da im ograniče korišćenje resursa i pristup sistemskim pozivima.

Procesi operativnog sistema su ograničeni u tome šta sve mogu da urade. Da bi serveri i drajveri mogli da obavljaju svoju funkciju, kernel im omogućava pozivanje kernel poziva koji dozvoljavaju izvršavanje privilegovanih operacija na kontrolisan način. Takođe serveri i drajveri mogu da traže i usluge jedni od drugih, razmenom IPC poruka.

#### 2.5.2.2 Statička politika izolacije drajvera

Pošto svaki server ili drajver ima različite zahteve, svakom se dodeljuje zasebna izolaciona politika. Moguće je definisati različite politike za svaki drajver koje dozvoljavaju pristup samo onim resursima koje su datom drajveru neophodne za obavljanje njegovog redovnog posla. Na slici 2.5 je data lista resursa koji mogu biti ograničeni na *MINIX* 3 sistemu.

| **Oznaka u `system.conf`** | **Resurs** | **Sprovodilac politike** | **Objašnjenje** |
|---|---|---|---|
| `uid` | korisničke privilegije | RS server | Servis se izvršava sa pravima datog korisnika |
| `ipc` | IPC odredište | IPC podsistem kernela | Ograničava koja IPC odredišta mogu biti pozivana |
| `system` | IPC kernel | Kernel task | ograničava korišćenje kernel poziva |
| `vm` | VM pozivi | VM server | Lista VM poziva koji su dostupni |
| `control` | kontrola servisa | RS server | Lista sistemskih servisa koji mogu da kontrolišu dati sistemski servis |
| `io` | IO opseg | Kernel task | Specificira listu IO opsega koje servis |

| | | | |
|---|---|---|---|
| | | | sme da koristi |
| `irq` | IRQ | Kernel task | Specificira listu IRQ opsega koje servis sme da koristi |
| `sigmgr` | upravljanje signalima | Kernel task | Specifikacija upravljača signalima za dati servis |
| `scheduler` | raspoređivanje procesa | sched server | Specificira raspoređivač kojem je dati servis pridružen |
| `priority` | prioritet servisa | sched server | Specifikacija prioriteta tj. prioritetskog reda za čekanje |
| `quantum` | veličina kvantuma | sched server | Specificira veličinu vremenskog kvantuma koji bi servis trebao da dobije |
| `pci device` | Tip PCI uređaja | PCI drajver | Specificira PCI ID uređaja kojeg kontroliše drajver |
| `pci class` | Klasa PCI uređaja | PCI drajver | Specificira PCI klasu uređaja kojeg kontroliše drajver |

Slika 2.5: Tabela sa listom opcija za ograničenje servisa koje mogu biti postavljene u `system.conf` fajlu

Izolacione politike su upisane u jednostavnim tekstualnim fajlovima kao što je `/etc/system.conf` ili fajlovi u `/etc/system.conf.d` direktorijumu. Ove fajlove čita program `service` i prenosi ih serveru RS koji ih postavlja prilikom pokretanja datog servisa. Potrebna je pažnja prilikom konfiguracije da bi drajver bio sprečen da zaobiđe mehanizme zaštite i dobije veća ovlašćenja nego što je to potrebno. Na *MINIX 3* sistemu odgovornost za to ima administrator koji ručno treba da napiše konfiguraciju, ali moguće je da će u budućnosti ovaj mehanizam biti dopunjen nekom vrstom automatizacije.
Sprovođenje politike vrši server RS, kao i ostali serveri i drajveri “od poverenja”. Učitavanje drajvera i postavljanje njegove izolacione politike se obavlja u tri koraka. Prvo server RS stvara novi proces i traži njegov IPC identitet, koga generiše kernel. Ovaj identitet se koristi za identifikaciju novog procesa. Zatim RS informiše kernel i odabrane sistemske procese o izolacionoj politici za novi proces, da bi ona mogla biti primenjena. Ovo se radi tako što se prosleđuje IPC identitet novog procesa zajedno sa listom dodeljenih resursa. Konačno, kada

je izolaciona politika postavljena, novi proces se pokreće i sada može bezbedno da izvršava mašinske instrukcije drajvera. Kada drajver želi da napravi privilegovani poziv, kernel proverava politiku koju je postavio RS i proverava da li onaj što pokušava da izvrši poziv ima na to pravo.

#### 2.5.2.3 Mehanizmi dinamičke kontrole pristupa

Mehanizmi strukturne zaštite, kao što je razdvajanje adresnih prostora procesa stvara problem kada je potrebno da proces pristupi memorijskom prostoru drugog procesa. Da bi ovo bilo moguće, u *MINIX 3* sistemu postoje dva dinamička mehanizma koja to omogućavaju.

- pristup memoriji jednog procesa od strane drugog je podržan mehanizmom memorijskih dozvola (engl. *memory grants*). Proces koji želi da dozvoli drugom procesu pristup delu svog memorijskog prostora, kreira dozvolu, koja sadrži IPC identitet, početak i kraj memorijskog prostora, kao i prava pristupa. Dozvola se smešta u tabelu poznatu kernelu i može da se pošalje drugom procesu slanjem indeksa dozvole u pomenutoj tabeli. Drugi proces zatim može da pristupa memoriji za koju važi dozvola, kopirajući njen sadržaj u svoj lokalni bafer ili obrnuto korišćenjem odgovarajućih kernel poziva. Parametri ovog kernel poziva su indeks dozvole i adresa lokalnog bafera. Kernel na osnovu indeksa nalazi dozvolu u tabeli i proverava da li dati proces ima prava na traženu operaciju kopiranja memorije. Zatim obavlja operaciju ako je dozvoljena, a u suprotnom je odbija. Takođe postoji podrška i za tzv. "*zero-copy"* protokole.
- *MINIX* 3 se oslanja na hardversku podršku da bi se zaštitio od uređaja koji koriste DMA (engl. *direct memory access*). DMA je način da pojedini uređaji direktno pristupaju sistemskoj memoriji, zaobilazeći zaštitu koju pruža procesor i MMU hardver. Na sreću, DMA može da bude kontrolisan upotrebom IOMMU (engl. *I/O memory management unit*) koji održava tabele sa memorijskim opsezima koji su dostupni uređajima. IOMMU radi slično kao i običan MMU, sa razlikom da MMU obezbeđuje zaštitu adresama vidljivim procesoru, dok IOMMU radi isto za adrese vidljive uređajima. Ako neki drajver želi da koristi DMA, mora prethodno da traži od IOMMU drajvera da mu to dozvoli za željeni adresni prostor. IOMMU drajver zatim proverava zahtev i, ako je zahtev odobren, postavlja IOMMU tabele za dati uređaj. Uređaju se dozvoljava samo pristup memorijskom prostoru drajvera. Ipak, moguće je i izvođenje DMA direktno do adresnog prostora aplikacije, upotrebom bezbednog memorijskog mapiranja zasnovanog na dozvolama.

Trenutno *MINIX* 3 ima drajver za AMD-Vi IOMMU pod imenom `amddev` (engl. *AMD Device Exclusion Vector*).

### 2.5.3 Ograničavanje pristupa memoriji

Pristup memoriji je ograničen korišćenjem kombinacije softverske i hardverske zaštite. Svaki proces ima privatni adresni prostor, koji je zaštićen pomoću MMU i IOMMU hardvera. Da bi bila moguća razmena podataka između procesa, bilo kroz kopiranje ili mapiranje, MINIX3 operativni sistem ima mehanizam za bezbedno davanje memorijskih dozvola u toku rada sistema.

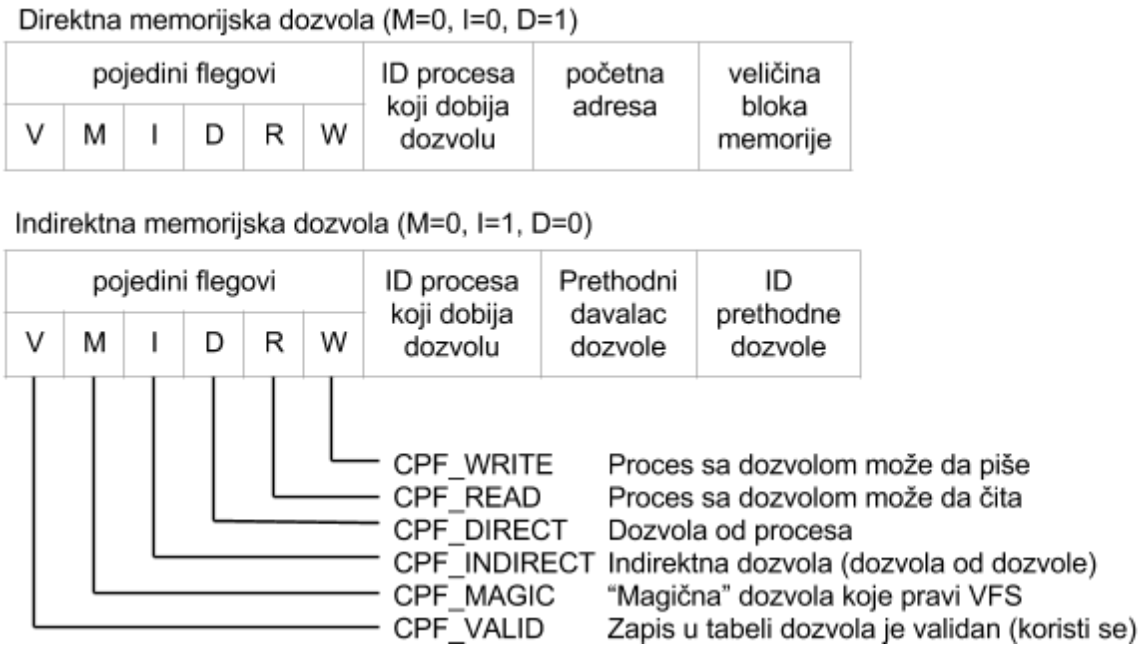
#### 2.5.3.1 Memorijske reference

*MINIX* 3 se oslanja na MMU hardversku zaštitu da sprovede striktno razdvajanje adresnog prostora. Svaki drajver ima privatan virtuelni adresni prostor koji zavisi od zahteva prilikom kompajliranja i izvršavanja drajvera. Adresni prostor sadrži izvršni kod drajvera (tekst segment), globalne i statičke podatke, hrpu (engl. *heap*) i izvršni stek. Prilikom startovanja procesa granice određuje PM (engl. *process manager*) server koji daje zahtev kernelu da u skladu sa tim programira MMU. MMU prevodi virtuelne adrese vidljive procesu u fizičke adrese koristeći MMU tabele koje je definisao kernel. Izuzetke straničenja (engl. *page fault*) presreće kernel i servisira ih transparentno za proces. Ali neovlašćen pristup memoriji, izvan adresnog prostora drajvera, daje MMU izuzetak koji obično navodi kernel da "ubije" drajver koji je uzrokovao dati izuzetak.

Drajveri, koji žele da razmenjuju podatke sa drugim sistemskim procesima bi mogli da koriste deljenje stranica, što omogućuje na primer *System V* i *POSIX* deljena memorija, ali ovi načini ne pružaju fleksibilnost i preciznu zaštitu koja je potrebna za izolaciju drajvera niskog nivoa. Ovde je identifikovano nekoliko nedostataka: Prvo, zaštita je zasnovana na UID (engl. *user identification*) i GID (engl. *group identification*), umesto na individualnim drajverima. Drugo, deljenje stranica koisti veličinu stranice kao granulaciju zaštite dok je za male strukture potrebna zaštita na nivou bajta. Treće, delegacija pristupnih prava nije moguća. Četvrto, pristupna prava se ne poništavaju automatski u slučaju "pucanja" procesa koji deli memoriju. Zato *MINIX* 3 ima svoj mehanizam autorizacije za pristup memoriji.

#### 2.5.3.2 Kopiranje i deljenje memorije

*MINIX* 3 dozvoljava bezbednu razmenu podataka putem mehanizma memorijskih dozvola [Herder2009-2]. Memorijska dozvola može da bude posmatrana kao sposobnost (engl. *capability*) koja može da bude prenesena drugoj strani kako bi ta strana dobila precizno definisan pristup memoriji. Nasuprot tome mehanizam liste kontrole pristupa (engl. *access control list*) ne podržava delegiranje i grublji je. Svaka dozvola definiše memorijski region sa preciznošću na nivou bajta i daje određenom drugom procesu dozvolu da čita i/ili piše odgovarajuće podatke. Proces, koji želi da dozvoli drugom procesu da pristupi svom adresnom prostoru, mora da napravi tabelu dozvola, u kojoj će čuvati memorijske dozvole. Prilikom prve upotrebe kernel mora da bude informisan o lokaciji i veličini tabele dozvola korišćenjem SETGRANT kernel poziva. Memorijska dozvola može da bude prosleđena drugom procesu slanjem identifikatora (ID) dozvole, koji zapravo predstavlja indeks dozvole u tabeli. Tako je dozvola jednoznačno identifikovana pomoću IPC identiteta procesa koji daje dozvolu i ID dozvole. Ove vrednosti mogu da budu poslate kernelu pozivom za obavljanje privilegovanih memorijskih operacija.




Slika 2.6: Struktura memorijske dozvole

Struktura memorijske dozvole je data na slici 2.6. Flegovi dozvole označavaju da li je dozvola u upotrebi, tip dozvole i vrste pristupa koji su predviđeni. Direktna dozvola (fleg CPF_DIRECT) znači da proces A dozvoljava procesu B ograničeni pristup memorijskom regionu u svom adresnom prostoru. Način dozvoljenog pristupa definišu R/W flegovi (čitanje i/ili pisanje). Primalac direktne dozvole (u prethodnom primeru to je proces B) može da tu dozvolu prosledi trećem procesu C pomoću indirektne dozvole (fleg CPF_INDIRECT). Memorijski region pokriven indirektnom dozvolom je definisan prethodnom dozvolom, čiji identifikator data indirektna dozvola sadrži. Specificirani memorijski prostor je uvek u adresnom prostoru procesa na početku lanca dozvola (koji je izdao direktnu dozvolu). Na ovaj način je delegacija memorijskih dozvola podržana indirektnim dozvolama i čini specifičnu hijerarhijsku strukturu.

Kada proces hoće da pristupi memorijskom prostoru za koji ima dozvolu, mora da pozove kernel, koji proverava validnost dozvole i obavlja traženu operaciju. Kernel poziv SAFECOPY služi za kopiranje podataka između lokalnog adresnog prostora drajvera i memorijskog prostora za koji je dobijena dozvola od drugog procesa. Po prijemu zahteva, kernel izdvaja IPC identitet i identifikator dozvole, pronalazi odgovarajuću memorijsku dozvolu i verifikuje da je identitet procesa, koji traži operaciju kopiranja, naveden u dozvoli. Indirektne dozvole se obrađuju rekurzivnom pretragom sve dok se ne dođe do direktne dozvole na koju se indirektna dozvola odnosi. Zatim se proverava da li zahtev za kopiranje odgovara prostoru i pravima definisanim u direktnoj dozvoli. Ako je zahtev u redu, kernel nalazi fizičke adrese izvora i odredišta i kopira zahtevanu količinu podataka.

Memorijske dozvole mogu da formiraju isključivo sistemski procesi. Ako korisnički proces pozove recimo bibliotečki sistemski poziv `read()` dajući kao parametar pokazivač na bafer u svom adresnom prostoru, on će biti pretvoren u poruku virtuelnom fajl serveru (VFS). Da bi VFS mogao da pokazivač prosledi kome treba (recimo drajveru uređaja koji se čita), on

kreira specijalnu vrstu memorijske dozvole, tzv. “magičnu” memorijsku dozvolu. Ova dozvola sadrži identitet procesa koji ju je kreirao (može biti samo VFS), identitet procesa kome se daje dozvola, pokazivač na memorijski prostor za koji dozvola važi i dužina ovog prostora. Pokazivač može da pokazuje na bilo koje mesto u memoriji. O bezbednosti ovakvog načina rada brine VFS, za koji se smatra da pripada TCB-u.

## 2.5.4 Ograničavanje ulaza/izlaza uređaja

Pristup uređajima i obrada prekida su ograničeni korišćenjem izolacione politike za svaki drajver ponaosob. Politike su definisane od strane administratora i smeštene u obične tekstualne fajlove. Server RS, koji upravlja svim sistemskim procesima, obaveštava kernel i skup važnih servera o politikama posle učitavanja drajvera, ali pre njegovog pokretanja, kako bi ograničenja bila primenjena za vreme rada drajvera.

### 2.5.4.1 Pristup uređaju

Server RS koristi izolacione politike kako bi osigurao da svaki drajver može da pristupi samo svom uređaju. Prilikom učitavanja novog drajvera, RS server prvo poredi izolacionu izolacionu politiku datog drajvera sa politikama drajvera koji su već aktivni. Ako je uređaj, za koga se učitava drajver, već zauzet, pokretanje se obustavlja. Između ostalog, moguće je statički ograničiti ulazno/izlazni adresni prostor kome drajver može da pristupi (opcija “`io`”) i kojim hardverskim prekidom može da rukuje (opcija “`irq`”). Takođe je ovo moguće dinamički definisati zadavanjem PCI broja (opcija “`pci device`”) i klase uređaja (opcija “`pci class`”), pa se onda na osnovu toga utvrđuje ulazno/izlazni adresni prostor i broj interapta uređaja. Ove i ostale opcije, koje mogu da se ograniče, su date na slici 2.5.
Zatim RS kernel pozivom PRIVCTL informiše kernel o prostoru u kome je drajveru dozvoljen rad. Za uređaje, koji koriste memorijski mapiran ulaz/izlaz, drajver može da zahteva da se fizička memorija koja pripada uređaju mapira u njegov virtuelni prostor pozivom UMAP ili VUMAP (vektorska verzija UMAP poziva). Za uređaje, koji zahtevaju programski U/I, postoji precizna kontrola pristupa portovima i registrima uređaja. Drajver poziva DEVIO kernel poziv (odnosno njegovu vektorsku verziju VDEVIO) i ako je poziv dozvoljen, kernel ga izvršava.

### 2.5.4.2 Obrada prekida

Kada je uređaju potrebna reakcija drajvera, on aktivira svoju IRQ liniju na magistrali što izaziva hardverski prekid procesora. Ovo dalje izaziva obustavljanje dotadašnjeg rada procesora i aktiviranje kernela. Iako obradu prekida na najnižem nivou mora da obavi kernel, svu obradu specifičnu za dati uređaj obavlja drajver. To znači da generički obrađivač prekida reaguje na svaki prekid i prosleđuje ga odgovarajućem drajveru, koristeći neblokirajuću IPC poruku obaveštenja (engl. *notification*), a zatim drajver obavlja sav posao vezan za obradu prekida. Na ovaj način se greške obrade prekida izoluju unutar procesa drajvera.
Drajver može da se registruje za dobijanje IPC obaveštenja za određeni hardverski prekid koristeći IRQCTL kernelski poziv. Pre nego što registruje drajver, kernel proverava njegovu politiku i obavlja registraciju je ukoliko je registracija dozvoljena.

## 2.5.5 Ograničavanje međuprocesne komunikacije (IPC)

Ograničavanje IPC-a za drajvere je postignuto dizajnom IPC podsistema, koji kao parametar uzima politiku koja se definiše za dati drajver.

#### 2.5.5.1 IPC primitive

U delu o IPC su opisane comunikacione primitive kao i mehanizam koji ograničava njihovu upotrebu. Dodatno, sistemski serveri koriste unapred definisane IPC protokole za komunikaciju sa nepouzdanim drajverima.

Pouzdanost IPC podsistema je postignuta tako što kernel u potpunosti kontroliše sve što se događa tokom IPC poziva. *MINIX* 3 kernel garantuje sledeće osobine [Herder2008]:

- atomnost: IPC poruka je ili isporučena u celosti ili nije isporučena
- pouzdana isporuka: poruke se ne gube, pošiljalac je informisan o ishodu slanja
- poverljivo adresiranje : IPC krajnje identiteti ne mogu da budu falsifikovani
- izolacija: višestruki IPC pozivi su nezavisni
- poverljivost: prisluškivanje IPC poziva nije moguće

Atomnost se obezbeđuje tako što kernel jednostavno ne vraća kontrolu procesu dok se IPC ne završi.

Pouzdana isporuka je postignuta tako što kernel kopira celu poruku odredišnom procesu, pa se na taj način čuva i integritet poruke. Strukturom se sprečava iscrpljivanje resursa pošto IPC podsistem koristi samo statički alocirane resurse, a poruka se baferuje lokalno kod procesa pošiljaoca.

Polje poruke koje sadrži izvorišni identitet (identitet procesa koji šalje poruku) popunjava kernel prilikom isporuke poruke, tako da falsifikovanje nije moguće. Izolacija se garantuje time što se višestruki IPC pozivi zasebno obrađuju, pa time ni uvid u IPC saobraćaj drugog procesa nije moguć.

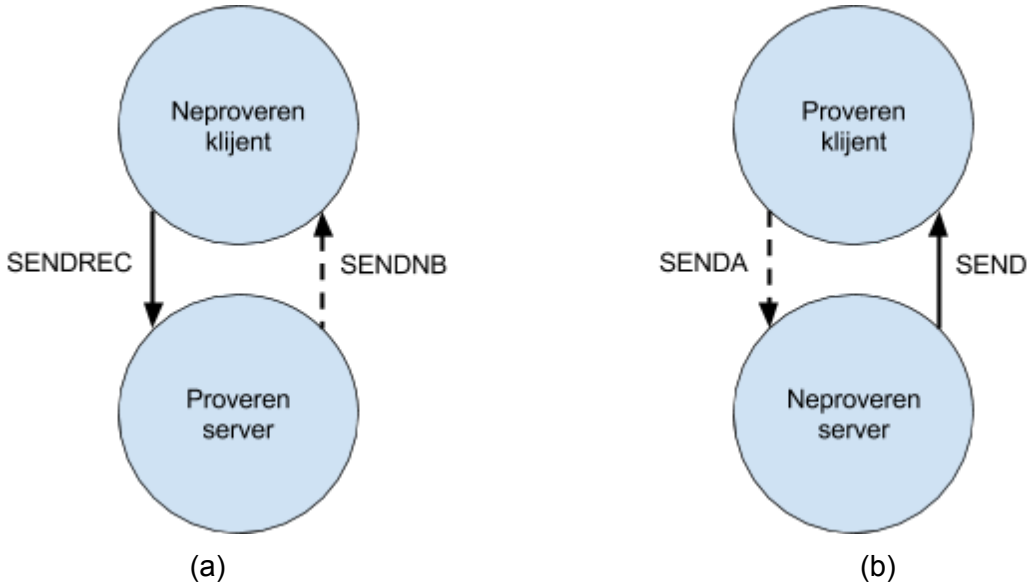
IPC podsistem ima mehanizme za kontrolu upotrebe IPC-a. Zahvaljujući postojanju bit maske slanja svaki proces je ograničen u tome kom sistemskom procesu može da šalje poruke. Bit maska slanja ima onoliko bitova koliko maksimalno može da bude sistemskih procesa (64 bita u verziji *MINIX* 3.3). Maske slanja su definisane kao simetrična relacija. To znači da ako proces A može da šalje poruke procesu B, maska slanja procesa B se automatski podešava kako bi proces B mogao da šalje odgovore procesu A. Prijem poruka nije ograničen, budući da to ima značaj samo ako neki ovlašćen proces pošalje poruku.

Još jedan zaštitni mehanizam je mehanizam IPC identiteta. Kako bi se sprečila zbrka oko procesa koji mogu da tokom vremena zauzimaju isti slot u kernelovoj tabeli procesa, pribeglo se posebnom načinu konstrukcije IPC identiteta. IPC identitet je 32 bitni broj koji sadrži 16 bitni indeks u kernel tabelu i 16 bitni broj, koji se inkrementira svaki put kada proces ponovo koristi dati slot. Alokacija slotova se vrši kružno (engl. *round robin*), kako bi do ponovne upotrebe istog slota došlo što kasnije. Na ovaj način se osigurava da IPC poruka upućena procesu, koji nije živ, ne može da dođe procesu, koji ponovo koristi isti slot. Takođe prilikom završetka rada sistemskog procesa, mehanizam pretplate DS (engl. *Data Store*) servera momentalno obaveštava sve pretplaćene procese o ovom događaju, kako bi znali da dati identitet više nije u funkciji.

#### 2.5.5.2 Obrasci komunikacije sa nepouzdanim partnerom

IPC komunikacija nekog od servera sa neproverenim procesima može da dovede do uzajamnog blokiranja (engl. *deadlock*), što bi izazvalo otkaz tog servera, a moguće i celog sistema. Da bi se ovo sprečilo, implementirane su mere zaštite. Zaštita od uzajamnog blokiranja pri IPC komunikaciji sa neproverenim procesima je, na operativnom sistemu *MINIX*3, implementirana tako što koristi asinhrone i neblokirajuće IPC primitive prilikom slanja ka neproverenom procesu. Ovaj način komunikacije zahteva pamćenje stanja u kom

se komunikacija trenutno nalazi, što se implementira modelom konačnog automata (engl. *state machine*). Na *MINIX* 3 se asinhroni i neblokirajući IPC koriste na tačkama razdvajanja sistema, gde (provereno ispravni) sistemski serveri moraju da komuniciraju sa u opštem slučaju neproverenim drajverima.




Slika 2.7: Obrazac komunikacije sa neproverenim partnerom u slučaju kada je neproveren klijent (a) i kada je neproveren server (b)

IPC obrasci, koji se na *MINIX* 3 koriste u komunikaciji sa nepouzdanim partneriom, su prikazani na slici. Kada komunicira sa neproverenim klijentima, provereni serveri koriste neblokirajuću SENDNB primitivu da bi klijentu poslali odgovor (slika 2.7a). Ovo se radi da server ne bi ostao blokiran u slučaju da neprovereni klijent ne pozove RECEIVE. Da bi odgovor mogao da stigne, klijent mora da komunicira sa serverom pomoću sinhrone SENDREC primitive, jer u suprotnom rizikuje da ne dobije odgovor. U slučaju kada provereni klijent inicira komunikaciju sa neproverenim serverom, on koristi asinhronu SENDA primitivu, kako ne bi ostao blokiran u slučaju da neprovereni server ne pozove RECEIVE (slika 2.7b). Server mora da odgovori sa sinhronim SEND, jer ne može da zna da li je klijent spreman za prijem odgovora. Na ovaj način se provereni sistemski serveri izoluju od neproverenih kao što su drajveri.

Kernel se brine isključivo o prosleđivanju poruka od jednog procesa ka drugom i ne proverava ispravnost sadržaja poruka. To znači da ograničenja sadržaja poruka (npr. dozvoljene vrste zahteva) moraju biti sprovođena od strane primaoca poruke. Ovaj problem je najkritičniji kad je u pitanju SYSTEM task kernela (kernelski proces), koji omogućava mnoštvo osetljivih operacija, kao što je kreiranje procesa ili postavljanje privilegija. Zbog toga je komunikacija sa kernelom izdvojena iz osnovnog IPC podsistema i prebačena u podsistem kernelskih poziva. Kernelski pozive mogu da pozivaju samo sistemski procesi, a skup kernelskih poziva koje sistemski proces može da pozove je definisan u skupu ostalih ograničenja za dati proces u tekstualnom `/etc/system.conf` fajlu. U skladu sa principom najmanjeg ovlašćenja (POLA) svaki sistemski proces (uključujući i drajvere) dobija pristup samo onim kernelskim pozivima koji su mu neophodni za obavljanje poverenog mu posla, kao što su ulaz/izlaz ili operacije sa memorijom. Listu dozvoljenih poziva sa ostalim

ograničenjima kernelu saopštava server RS kernelskim pozivom PRIVCTL, posle kreiranja, a pre pokretanja datog sistemskog procesa. Svaki put kada dati sistemski proces izvrši kernelski poziv, kernel proverava da li procesu to dozvoljeno.
Treba naglasiti da za pojedine kernelske pozive postoje više nivoa zaštite. Recimo, ako je sistemski proces odnosno drajver ovlašćen da koristi npr. poziv SAFECOPY, on mora da poseduje i validnu memorijsku dozvolu, u kojoj je specificirana adresa i dužina bafera.

## 2.6 Literatura

[DiBona1999] DiBona, C. and Ockman, S. Open Sources: Voices from the Open Source Revolution. O'Reilly, 1st ed., Jan. 1999. Appendix A: The Tanenbaum-Torvalds Debate.

[Haertig1997] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, and S. Schönberg, "The Performance of µ-kernel-based Systems," in Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, 1997, pp. 66–77.

[Herder2006] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "MINIX 3: A Highly Reliable, Self-repairing Operating System," SIGOPS Oper. Syst. Rev., vol. 40, no. 3, pp. 80–89, Jul. 2006.

[Herder2008] Herder, J.N.; Bos, H.; Gras, B.; Homburg, P.; Tanenbaum, A.S., "Countering IPC Threats in Multiserver Operating Systems (A Fundamental Requirement for Dependability)," in Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium on , vol., no., pp.112-121, 15-17 Dec. 2008

[Herder2009] Herder, J.N.; Bos, H.; Gras, B.; Homburg, P.; Tanenbaum, A.S., "Fault isolation for device drivers," in Dependable Systems & Networks, 2009. DSN '09. IEEE/IFIP International Conference on , vol., no., pp.33-42, June 29 2009-July 2 2009

[Herder2009-2] Memory Sharing Revisited, Jorrit N. Herder, Herbert Bos, Arun Thomas, Ben Gras, Andrew S. Tanenbaum, 4th European Conference on Computer Systems (EUROSYS '09) 2009

[IEEE1990] IEEE Information Technology-Portable Operating System Interface (POSIX)—Part 1: System Application: Program Interface (API) [C Language]. IEEE, 1st ed., Oct. 1990. Also known as IEEE Std. 1003.1-1990. Approved as Int'l Std. ISO/IEC 9945-1:1990.

[Liedtke1995] Liedtke, J. On µ-Kernel Construction. In Proc. 15th ACM Symp. on OperatingSystems Principles, pp. 237–250, Dec. 1995.

[Intel2015] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture, 253665-055US, 2015

[Klein2009] Klein, Gerwin. "Operating system verification—an overview." Sadhana 34.1 (2009): 27-69.

[Klein2009-2] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. seL4: formal verification of an OS kernel. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09). ACM, New York, NY, USA, 207-220.

[Lampson1992] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. 1992. Authentication in distributed systems: theory and practice. ACM Trans. Comput. Syst. 10, 4 (November 1992), 265-310.

[Lions1977] Lions, J. A Commentary on the Sixth Edition UNIX Operating System. Dept. of Computer Science, The University of New South Wales, 1977.

[Ritchie1974] Ritchie, D. M. and Thompson, K. The UNIX Time-Sharing System. Comm. of the ACM, 17(7):365–375, July 1974.

[Tanenbaum1987] Tanenbaum, A. S. A UNIX Clone with Source Code for Operating Systems Courses. ACM SIGOPS Operating System Review, 21(1):20–29, Jan. 1987.

[Tanenbaum2006] Tanenbaum, A. S. and Woodhull, A. S. Operating Systems Design and Implementation. Prentice Hall, 3rd ed., Jan. 2006.

[Torvalds1991] Torvalds, L., What Would You Like to See Most in MINIX?, Message to the Usenet Newsgroup comp.os.minix, Aug. 1991.

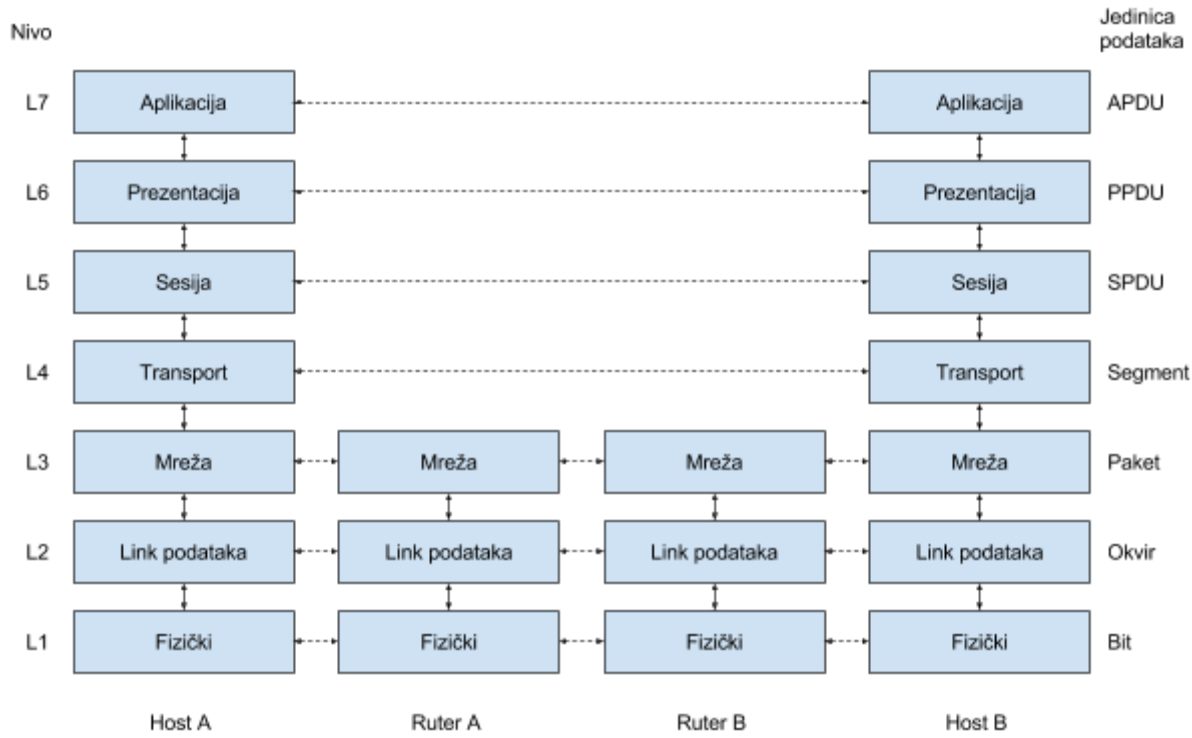
[Torvalds2001] Torvalds, L. and Diamond, D. Just for Fun: The Story of an Accidental Revolutionary. HarperCollins, 1st ed., May 2001.

# 3. Funkcionisanje mrežnog podsistema

Računarska komunikacija počiva na tehnologiji računarskih mreža. u ovom poglavlju će biti predstavljen način funkcionisanja mrežnog podsistema na operativnom sistemu sa monolitnim jezgrom i mikrojezgrom. Kao predstavnik familije sistema sa monolitnom strukturom jezgra je uzet *Linux*, dok će funkcionisanje mreže na mikrokernel sistemima biti opisano na primeru *MINIX*-a.

## 3.1 Nivoi mrežne komunikacije

Računarska mreža se sastoji od mnoštva hardverskih i softverskih komponenti. Radi lakšeg razumevanja razvijeni su modeli koji mrežnu komunikaciju apstrahuju u više slojeva kako bi se razumeli njeni različiti aspekti. Jedan od najviše upotrebljavanih modela je definisala Međunarodna organizacija za standarde - ISO (engl. *International Standards Organization*) i zove se OSI (engl. *Open Systems Interconnection*) referentni model [Day1995].




Slika 3.1: Šema OSI referentnog modela

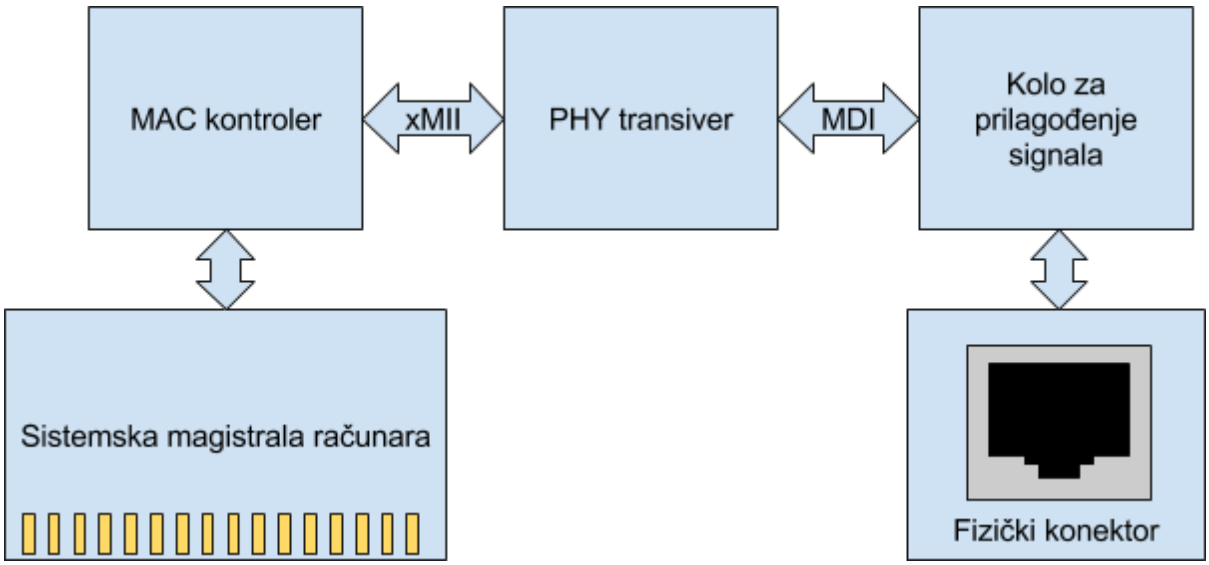
Na slici je prikazan šematski prikaz OSI referentnog modela. Podaci se prenose od najvišeg (aplikativnog) nivoa do najnižeg (fizičkog), koji predstavlja komunikacioni kanal. Logički, svaki od nivoa komunicira sa odgovarajućim nivoom sa druge strane komunikacionog kanala. Pod komunikacijom se podrazumeva razmena podataka, a jedinica razmene podataka se različito definiše u zavisnosti od nivoa komunikacije i zbog toga se naziva protokolska jedinica podataka, odnosno PDU (engl. *Protocol Data Unit*). Tako se PDU za aplikacioni nivo naziva aplikacioni PDU tj. APDU, PDU za nivo prezentacije PPDU i tako

redom. Svaki od nivoa prilikom slanja preuzima PDU nivoa iznad, dodaje svoje zaglavlje (enkapsulira) i prosleđuje ga nivou ispod. Prilikom prijema postupak je obrnut. Za najniža četitri nivoa PDU imaju posebna imena: PDU transportnog nivoa se naziva segment, PDU mrežnog nivoa se naziva paket, PDU nivoa linka podataka okvir (engl. *frame*), a fizičkog bit. Najniža tri nivoa se obično implementiraju u obliku hardverskih uređaja (npr. uređaj koji radi na mrežnom nivou se naziva ruter), što je takođe prikazano na slici 3.1. U tekstu će nivoi OSI modela često biti označavani oznakama L1 do L7.

Kada se govori o mreži i mrežnim protokolima, u literaturi se koristi termin “oktet”, koji se odnosi na grupu od osam bitova. U literaturi koja se bavi operativnim sistemima i programiranjem se ista osmbobitna grupa naziva “bajt”. U ovom tekstu se oba termina ravnopravno koriste.

# 3.2 Mrežni interfejs

Mrežni interfejs ili mrežni kontroler ili NIC (engl. *Network Interface Controller*) je računarski hardver koji omogućava povezivanje računara na računarsku mrežu. Najčešće se proizvodi u obliku kartice koja se spaja direktno na sistemsku magistralu na matičnoj ploči računara, a danas je čest slučaj da se mrežni interfejs ugrađuje na samu matičnu ploču. On se sastoji iz elektronskih kola neophodnih za komunikaciju, koja koriste određene standarde fizičkog nivoa i nivoa veze podataka (1. i 2. nivoa OSI modela) kao što su *Ethernet* [IEEE2014] ili Wi-Fi [IEEE2013]. Ove standarde definiše IEEE (engl. *Institute of Electrical and Electronics Engineers*).




Slika 3.2: Šematski prikaz elemenata IEEE 802.3 mrežnog interfejsa

Jedan od danas najviše korišćenih standarda za mrežno povezivanje je *Ethernet* odnosno IEEE 802.3 standard [IEEE 2014]. Ovaj standard omogućava povezivanje računara brzinama 10, 100, 1000 i 10000 Mbit/s, a u pripremi su proširenja standarda za još veće

brzine. Skup kola koji čini ovakav interfejs (slika 3.2) obuhvata integrisano kolo MAC (engl. *Media Access Control*) kontrolera, integrisano kolo PHY primopredajnika (engl. *physical transciever - transmitter/receiver*), kolo za prilagođenje signala i fizički konektor. Integrisano kolo MAC kontrolera implementira programski model mrežnog interfejsa koji vidi drajver. Sadrži interfejs ka sistemskoj magistrali računara, prijemne i predajne bafere, registre i upravljačku logiku kao i interfejs ka različitim prenosnim medijima MII (engl. *media-independent interface*). Integrisano kolo PHY primopredajnika prilikom slanja kodira podatke dobijene od MAC kontrolera u oblik za prenos, odnosno dekodira primljene podatke i prenosi MAC kontroleru. Obuhvata MII interfejs za komunikaciju sa MAC kontrolerom i MDI (engl. *media-dependent interface*) interfejs, koji se preko kola za prilagođenje signala i fizičkog konektora vezuje na prenosni medijum.
Kolo za prilagođenje signala pretvara logičke signale standardnog naponskog nivoa (npr. 5V ili 3.3V) u signale za prenos. Na primer, kada se radi o bakarnom prenosnom mediju, kao kolo za prilagođavanje se koristi tzv. magnetski modul. PHY primopredajnik, kolo za prilagođenje signala i fizički konektor spadaju u fizički nivo OSI modela i zavise od konkretnog prenosnog medijuma. MAC kontroler spada u nivo veze podataka. Danas se često sreću kontroleri gde su MAC i PHY komponente integrisane u jedno kolo (čip), a kolo za prilagođenje se obično integriše u fizički konektor. Ovakav mrežni interfejs predstavlja osnovu za potpuni mrežni stek protokola što dozvoljava komunikaciju među računarima, od malog broja računara u lokalnoj mreži (LAN) do komunikacije na svetskom nivou.

Svaki MAC kontroler ima jedinstveni 48-bitni identifikator, koji se zove fizička adresa ili MAC adresa. Jedinstvenost MAC adrese je osigurana tako što zainteresovana strana mora da od IEEE kupi 24-bitni blok jedinstvenih adresa. Svaki blok se identifikuje 24-bitnim prefiksom OUI (engl. *Organizationally Unique Identifier*). Prilikom proizvodnje svakom interfejsu se dodeljuje po jedna adresa iz dobijenog bloka adresa. Ranije je MAC adresa interfejsa bila nepromenljiva, ali na većini današnjih NIC-ova ju je moguće menjati.

### 3.2.1 Princip rada NIC

Rad NIC-a je zasnovan na tome da centralni procesor računara (CPU) obavlja ulaz odnosno izlaz u trenucima kada je to potrebno. Da bi znao kada da opslužи NIC, CPU treba da bude obavešten o sledećim događajima:

- o prijemu novog (jednog ili više) mrežnog paketa/datagrama u NIC bafer, kako bi sadržaj bafera bio preuzet u operativnu memoriju
- da je slanje prethodno poslatog (jednog ili više) paketa završeno, kako bi mogao da šalje nove.

Procesor može o ovim događajima da bude obavešten korišćenjem sledećih tehnika:

- prozivanje (engl. *polling*), kada CPU povremeno ispituje stanje NIC-a čitajući njegove registre, kako bi znao da li je potreban ulaz/izlaz.
- korišćenje prekida, kada NIC prekida redovan rad CPU, kako bi ga obavestio da je potreban ulaz/izlaz.

Prebacivanje paketa iz bafera NIC-a u operativnu memoriju i obrnuto može da se radi na sledeće načine:

- programski U/I, kada CPU prebacuje podatke koristeći instrukcije ulaza i izlaza

- pomoću DMA (engl. *Direct Memory Access*), gde inteligentna periferija preuzima kontrolu nad sistemskom magistralom, da bi pristupala memoriji direktno. Ovo oslobađa CPU posla oko ove vrste prenosa i NIC ne mora da ima veliki memorijski bafer za pakete, ali zahteva da NIC bude "inteligentniji", odnosno da poseduje DMA kontroler.

Pojedini proizvođači u svoje MAC kontrolere dodaju i podršku za protokole višeg nivoa (npr. IP, TCP) kako bi se centralni procesor rasteretio tog dela posla.

# 3.3 Mrežni drajver

Jedna od glavnih funkcija operativnog sistema je kontrola svih U/I (ulazno/izlaznih) uređaja računara. On mora da upravlja uređajima, obrađuje prekide i izlazi na kraj sa greškama. Takođe bi trebao da obezbedi interfejs između uređaja i ostatka sistema koji je jednostavan i lak za korišćenje. Koliko god je to moguće, interfejs bi trebao da bude isti za sve uređaje. Uređaji se međusobno razlikuju, a drajveri su delovi operativnog sistema koji obavljaju prilagođavanje U/I interfejsa konkretnog uređaja standardnom programskom interfejsu. Obzirom da danas postoji veliki broj uređaja koje operativni sistem treba da podrži pomoću odgovarajućih drajvera, drajveri čine značajan deo njegovog programskog koda.

## 3.3.1 Pojam drajvera

Drajver je softverska komponenta koja omogućava komunikaciju između operativnog sistema (odnosno njegovog jezgra - kernela) i hardverskog uređaja. Drajver konfiguriše uređaj, upravlja njime i pretvara zahteve kernela u zahteve hardveru. [Kadav2012].

Drajveri počivaju na tri interfejsa: 1. interfejsu između drajvera i kernela (za prenošenje zahteva i pristupanju servisima OS), 2.interfejsu između drajvera i uređaja (za izvršavanje operacija), i 3. interfejsu između drajvera i magistrale (za upravljanje komunikacijom sa uređajem). To znači da drajveri igraju ulogu prevodioca prilikom komunikacije između programa i uređaja. Svaki uređaj ima svoj specijalan skup komandi, koje samo drajver poznaje. Nasuprot tome, kernel i programi pristupaju uređaju koristeći generički skup komandi, odnosno programski interfejs. Drajver prihvata generičke komande programskog interfejsa i prevodi ih u specijalizovane komande uređaja. Takođe, specijalne odgovore uređaja prevodi u generičke signale koje kernel i programi razumeju. Komunikacija drajvera sa uređajem zavisi od vrste i tipa uređaja, a programski interfejs drajvera zavisi od operativnog sistema.

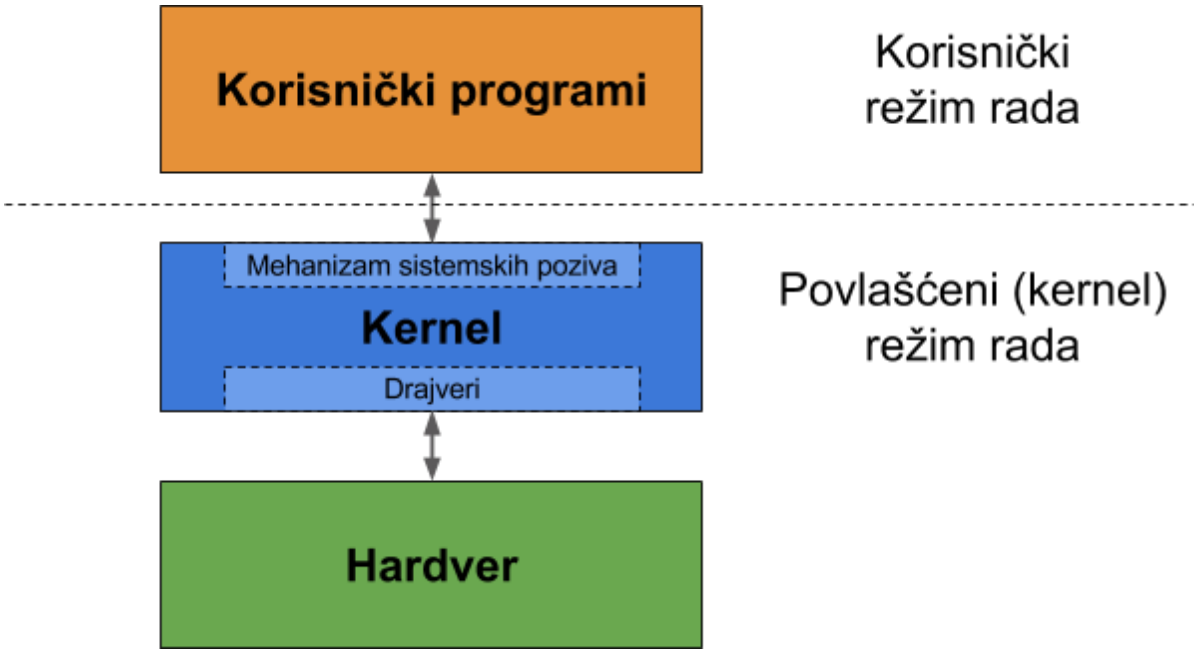
Drajveri se obično implementiraju kao moduli, koji omogućavaju kernelu i programima pristup i upravljanje raspoloživim hardverom. Moduli predstavljaju način da se proširi funkcionalnost operativnog sistema. Oni su delovi programskog koda sa dobro definisanim programskim interfejsom prema ostatku operativnog sistema, koji sakriva detalje funkcionisanju uređaja sa kojim se komunicira. Obično se moduli dizajniraju tako da mogu biti učitani i/ili izbačeni, prema potrebi, i proširuju funkcionalnost kernela bez potrebe za resetovanjem celog sistema.

## 3.3.2 Specifičnosti mrežnog drajvera

Osnovni zadatak mrežnog drajvera je da šalju na NIC pakete koje dobiju od mrežnog podsistema OS, kao i da mrežnom podsistemu isporuče pakete koji su preko NIC-a primljeni sa mreže. Ove aktivnosti pripadaju drugom nivou OSI modela. Da bi mogao da izvršava svoj zadatak, drajver prilikom svog pokretanja mora da inicijalizuje mrežni interfejs (NIC) na sistemskoj magistrali i da ga pripremi za prenos i prijem paketa. U ovom tekstu se ravnopravno koriste termini “mrežni uređaj”, “mrežni interfejs” i “NIC”, iako u nekim drugim kontekstima postoji bitna razlika između ovih termina.
Interfejs između kernela i drajvera zavisi od strukture operativnog sistema i predstavlja jedan od kriterijuma za klasifikaciju operativnih sistema. Današnji operativni sistemi se po ovom kriterijumu mogu podeliti na dve osnovne klase: monolitni i mikrokernel sistemi.

## 3.3.3 Mrežni drajver na monolitnom sistemu (*Linux*)

*Linux* predstavlja jednu od modernih inkarnacija operativnog sistema *UNIX* nastalog 70-tih godina prošlog veka [Ritchie1974] i od njega je između ostalog nasledio i monolitnu strukturu jezgra. Monolitno jezgro sistema je napisano kao kolekcija procedura, od kojih svaka može da pozove bilo koju drugu, kad god je to potrebno. Da bi ovo funkcionisalo, procedure imaju definisan interfejs u smislu potrebnih parametara i rezultata. Procedure jezgra se kompajliraju i linkuju u jedinstveni izvršni fajl - kernel. Pošto su sve procedure i strukture podataka unutar kernela vidljive svim ostalim kernel procedurama, među njima nema nikakve izolacije niti skrivanja informacija. Ipak, izolacija postoji između jezgra, koje se izvršava u povlašćenom “kernel” režimu procesora, i korisničkih procesa, koji se izvršavaju u neprivilegovanom tzv. “korisničkom” režimu. Da bi podržali ovu vrsu izolacije, moderni procesori imaju privilegovan režim i najmanje jedan neprivilegovan režim rada . U povlašćenom režimu rada, procesor može da izvršava sve operacije koje dozvoljava njegova arhitektura, kao što je izvršavanje svih instrukcija ili pristup svim delovima operativne memorije. U korisničkom tj. neprivilegovanom režimu važe pojedina ograničenja nametnuta hardverom, kao što je zabrana izvršavanja pojedinih (privilegovanih) instrukcija i pristup samo ograničenom delu memorije. Neprivilegovanih režima može biti više, što zavisi od konkretne arhitekture procesora.
Svrha ove izolacije je da se operacije, koje su kritične za sistem, obavljaju isključivo unutar jezgra operativnog sistema, kako bi se onemogućio pad sistema zbog grešaka ili zle namere u korisničkim programima. Kada korisničkiom procesu zatreba neka od privilegovanih operacija, on se obraća operativnom sistemu, tako što izvršava specijalnu instrukciju (tzv. “sistemski poziv”) sa unapred definisanim parametrima. Struktura jednog ovakvog sistema je prikazana na slici 3.3.




Slika 3.3: Šema monolitnog operativnog sistema

Pravljenje jezgra, koje će da funkcioniše za sve namene i u svakom hardverskom okruženju, je nemoguće bez mehanizma proširenja. U monolitnim operativnim sistemima se obično funkcionalnost kernela proširuje mehanizmom za dodavanje kernelskih modula. Na *Linux*-u se ovaj mehanizam naziva LKM (engl. *Linux Kernel Modules*). Ovaj mehanizam se koristi i za učitavanje i rad drajvera. Na taj način kernel može da učitava samo drajvere za uređaje koji su deo konfiguracije računara. Kao i ostali delovi kernela, i drajveri se izvršavaju u kernel režimu. Drajveri se obično koncipiraju kao unapred definisan skup funkcija koje ostali delovi kernela (u ime korisničkih programa) pozivaju zbog komunikacije sa određenim U/I uređajem. Drajveri takođe mogu da pozivaju druge kernel funkcije. Prednost ovakvog pristupa je što je pisanje drajvera relativno jednostavno, jer se pišu samo funkcije definisane programskim interfejsom kernel-drajver, ali zato nema izolacije drajvera. Zbog odsustva izolacije između drajvera i ostatka kernela, greška u nekom od drajvera može da uzrokuje pad čitavog operativnog sistema.

#### 3.3.3.1 “Sve je fajl”

Još jedna od stvari koju je *Linux* nasledio od *UNIX*-a je organizacija načina pristupa drajverima od strane korisničkih programa. Da bi se standardizovao rad sa drajverima, oni su predstavljeni u obliku specijalnog fajla u “`/dev`” direktorijumu. Pored redovnih osobina fajla, kao što u ime i prava pristupa, drajverski fajl ima tzv. “glavni” i “sporedni” broj, koji služe da povežu dati specijalni fajl sa instancom drajvera u memoriji.
Da bi se standardizovao skup funkcija koje drajver treba da realizuje zarad komunikacije sa nekim uređajem, uređaji se svrstavaju u klase uređaja sa sličnim osobinama. U/I uređaji (odnosno drajveri za njih) na *Linux*-u i ostalim *UNIX*-olikim operativnim sistemima mogu grubo da se podele na dve osnovne klase: na blok i znakovne uređaje. Blok uređaji čuvaju (čitaju i pišu) informacije u blokovima fiksne veličine, gde svaki blok ima svoju adresu. Znakovni uređaji isporučuju ili prihvataju niz znakova (bajtova), ne vodeći računa o njihovoj strukturi. Ovi podaci nemaju adresu, nego im se pristupa isključivo sekvencijalno (redom).

Svaka od klasa drajvera ima definisan određeni skup funkcija koje drajver koji pripada toj klasi treba da implementira. Ove funkcije se u žargonu nazivaju “udice” (engl. *hooks*) za koje se “kače” pozivi kernela. One implementiraju operacije nad drajverom, odnosno njegovim uređajem, kao što su inicijalizacija, čitanje ili pisanje. Kernel ove operacije vezuje za operacije nad specijalnim drajverskim fajlom koji je pridružen datom drajveru. Na taj način korisnički program, koji želi da koristi uređaj, može da radi sa njim na isti način kao da koristi običan fajl.
Nažalost, klasifikacija na znakovne i blok drajvere nije potpuna, budući da dosta U/I uređaja nije moguće svrstati ni u jednu od navedenih klasa [Kadav2012]. Ovde spadaju i mrežni uređaji, čiju upotrebu omogućavaju i čijim radom upravljaju mrežni drajveri. Zbog toga mrežni uređaji prema nekim izvorima [Corbet2005] čine treću, zasebnu klasu uređaja.

### 3.3.3.2 Kako rade mrežni drajveri na *Linux*-u?

Za razliku od znakovnih i blok drajvera, mrežni drajveri na *Linux*-u nemaju svoj specijalni fajl, kao ni “glavni” ni “sporedni” broj. Uloga mrežnog interfejsa i njegovog drajvera unutar sistema je da na mrežu šalje i sa mreže prima pakete, odnosno okvire na zahtev kernela, putem odgovarajuće funkcije. Mrežni interfejs, odnosno njegov drajver, mora da se prijavi kernelu popunjavajući određene strukture podacima, kako bi mogao da bude pozivan kada se mrežni paketi razmenjuju sa spoljnim svetom.
Mrežni interfejs nema specijalni fajl u “`/dev`” direktorijumu, jer primena klasičnih fajl-operacija (čitanje, pisanje, ...) na mrežne interfejse nema smisla, tako da na njih nije moguće primeniti pristup “sve je fajl”, koji karakteriše *UNIX*-olike operativne sisteme. Zbog toga, mrežni interfejsi postoje u svom sopstvenom imenskom prostoru (engl. *namespace*) i pružaju skup operacija koji je drugačiji od ostalih grupa drajvera.
Najznačajnija razlika od ostalih tipova drajvera je da mrežni drajveri asinhrono primaju pakete od spolja. To npr. znači da kod mrežnih drajvera, sam drajver može da traži da kernelu prosledi pristigli paket. Zbog toga je interfejs između mrežnog drajvera i kernela pravljen da podrži taj specifičan način funkcionisanja.
Mrežni drajveri takođe moraju da budu spremni da podrže brojne administrativne poslove, kao što su postavljanje mrežne adrese, promena parametara prenosa ili održavanje statistike mrežnog saobraćaja (kolilčina podataka, broj grešaka, …). Tokom interakcije između mrežnog drajvera i kernela obrađuju se paketi jedan po jedan, što omogućava da detalji protokola budu sakriveni od drajvera, a da detalji fizičkog prenosa budu sakriveni od mrežnog podsistema.

### 3.3.3.3 Implementacija mrežnog drajvera na *Linux*-u

Da bi mrežni interfejs mogao da se koristi, mora da bude prepoznat od strane kernela i pridružen odgovarajućem drajveru. Drajver u svojim strukturama podataka čuva sve informacije, koje su potrebne za rad sa mrežnim interfejsom, kao i za interakciju sa ostalim komponentama kernela kojima je potrebno da koriste NIC.
Kada se modul mrežnog drajvera učita u kernel, poziva se njegova funkcija za inicijalizaciju, koja integriše drajver sa kernelom i inicijalizuje uređaj za upotrebu. Tom prilikom se od kernela traže i zauzimaju resursi kao što su memorijski i U/I adresni prostor, broj prekida uređaja (IRQ) i nude usluge kernelu inicijalizacijom pokazivača funkcija uređaja u posebnoj strukturi podataka.

Struktura podataka kojom je opisan svaki pojedinačan mrežni interfejs je struktura `net_device`, koja je deklarisana u fajlu `<linux/netdevice.h>`. Ova struktura čuva sve informacije koje se tiču određenog mrežnog uređaja, odnosno interfejsa. Polja koja sadrži struktura `net_device` mogu biti klasifikovana prema sledećim kategorijama [Benvenuti 2006]:

- konfiguracija uređaja (ime i tip uređaja, resursi koje zauzima,...),
- statistiku rada,
- trenutni status uređaja,
- upravljanje mrežnim saobraćajem,
- upravljanje spregnutim listama,
- specifičnosti konkretnog uređaja,
- generička polja i
- pokazivači na funkcije uređaja

Pokazivači na funkcije predstavljaju osnovni mehanizam kojim kernel komunicira sa drajverom. Funkcije se prema nameni dele na funkcije za:

- prenos i prijem okvira/paketa,
- dodavanje i parsiranje zaglavlja drugog nivoa,
- promenu konfiguracije,
- informisanje o statistici i
- posebnu namenu specifičnu za dati uređaj

Među funkcijama koje je potrebno prijaviti kernelu, je i funkcija obrađivač prekida za dati mrežni uređaj (engl. *interrupt handler*). Drajver treba da čuva pokazivač na `net_device` strukturu svoga uređaja. Posle završene inicijalizacije, potrebno je strukturu `net_device` datog uređaja registrovati kernelu, kako bi ostatak sistema mogao da koristi dati mrežni uređaj. Način na koji se mrežni drajver registruje kernelu se razlikuje u odnosu na registraciju drajvera za znakovne i blok uređaje, jer nema “glavnog” ni “sporednog” broja drajvera. Umesto toga, mrežni drajver (po završetku inicijalizacije) u globalnu kernel listu mrežnih uređaja ubacuje posebnu strukturu podataka za svaki od NIC uređaja za koji je nadležan (jedan drajver je najčešće nadležan za jedan uređaj).Od trenutka registracije kernel može da počne da koristi drajver, odnosno da poziva funkcije kojie drajver implementira. Glavni korisnik funkcija mrežnog drajvera je mrežni podsistem kernela, koji koristi drajver pre svega za slanje i primanje mrežnih paketa.
Drajver je takođe dužan da obezbedi i funkciju za svoje zaustavljanje odnosno prestanak rada. Pre zaustavljanja drajver je dužan da zaustavi uređaj i da vrati sistemu sve resurse koje je u toku rada zauzeo. Detalji načina funkcionisanja *Linux* mrežnog drajvera su dati u [Benvenuti 2006].

## 3.3.4 Mrežni drajver na mikrokernelu (*MINIX*)

Za razliku od *Linux*-a i ostalih monolitnih operativnih sistema, *MINIX3* je operativni sistem sa mikrojezgorom. To znači da drajveri (uključujići i mrežni drajver) na *MINIX*-u predstavljaju zasebne celine tj. procese koji se izvršavaju u korisničkom režimu (imaju svoju `main()` funkciju), a sa ostatkom sistema komuniciraju razmenom poruka. Svaki od uređaja ima svoju instancu drajvera. Drajveri spadaju u posebnu vrstu *MINIX* procesa, koji se nazivaju sistemski procesi.

### 3.3.4.1 Sistemski procesi

Drajveri i serveri se nazivaju sistemskim procesima i deo su operativnog sistema. Oni ne pripadaju nijednom korisniku i mnogi (ako ne svi) su aktivirani pre nego što se prvi korisnik prijavi na sistem. Formalno se zahteva da svaki proces, pa i sistemski, ima svoj UID (engl. *user identification*) broj. Trenunto u prosečnoj *MINIX3* (verzija 3.3) konfiguraciji sistemski procesi obično imaju UID 0 (korisnik `root`) ili UID 12 (korisnik `service`), ali to nije pravilo. Obično sistemski procesi imaju viši prioritet izvršavanja od ostalih procesa, a drajveri obično imaju viši prioritet od servera. Ipak, ovo nije pravilo jer se prioritet određuje za svaki proces ponaosob. Moguće je da neki drajver, koji opslužuje neki spori uređaj, ima niži prioritet od pojedinih servera, koji moraju da se brzo odazivaju na zahteve.

Za svaki sistemski proces mora da postoji unapred definisana lista ovlašćenja, kao npr. sa kim dati servis može da komunicira, koje kernel pozive može da koristi i (u slučaju drajvera) kojim U/I portovima može da pristupa. Skup ovih ovlašćenja se obično nalazi u fajlu `/etc/system.conf`.

Određen deo ključnih sistemskih procesa se nalazi u fajlu koji predstavlja sliku memorije u početnom stadijumu rada. Ovaj fajl se zove početna slika (engl. "*boot image*"), a njegov sadržaj se prilikom podizanja sistema učitava na određeno mesto u memoriji. Taj sadržaj su zapravo instrukcije i podaci neophodni da bi *MINIX*3 jezgro tj. operativni sistem mogao da počne sa radom.

Učitavanje, odnosno pokretanje preostalih sistemskih procesa obično radi proces `init` izvršavajući "`service up`" komandu unutar `/etc/rc` skripte. Komanda `service` je zapravo način za komunikaciju reinkarnacionim serverom (RS), koji pokreće i zaustavlja sve sistemske procese. Samim tim, ovaj server je i "roditelj" svih sistemskih procesa i zato je u stanju da primeti ukoliko neki od njegove "dece" iznenada prestane da funkcioniše.

Da bi drugi procesi mogli da komuniciraju sa datim sistemskim procesom, moraju prethodno da saznaju njegov IPC identitet. Za ovo između ostalog služi DS (Data Store) server, kome sistemski proces oglasi svoje prisustvo pomoću skupa funkcija, kao što su `ds_publish_u32()`, `ds_publish_str()` i slične [MINIX-DS2015].

### 3.3.4.2 Funkcionisanje drajvera na *MINIX*-u

Zadatak drajvera je da prima poruke od nadređenih mu servera i da na njih odgovara. Obično se ovo implementira kao beskonačna petlja (petlja događaja) unutar funkcije `main()` koja počinje sa prijemom IPC poruke (funkcije `receive()`). Primljena poruka se potom interpretira pomoću C-ovskog "*switch-case*" iskaza i, u zavisnosti od vrste poruke, poziva se odgovarajuća funkcija, koja radi ono što data poruka zahteva. Ovde je bitno naglasiti da se drajver ne izvršava u kernel režimu i ne može da samostalno pristupa U/I portovima, niti da pristupa memorijskom prostoru koji nije njegov. Pošto su mu ove funkcionalnosti neophodne da bi mogao da pristupa uređaju i da komunicira sa serverima koji zahtevaju prenos informacija iz uređaja i na uređaj, drajver mora da pozove kernel svaki put kada nešto od ovoga treba da uradi, a kernel to uradi ili ne, u zavisnosti od definisanih ovlašćenja za dati drajver.

### Instanca drajvera

Svaki drajver je odgovoran za tačno jedan NIC. Računar može da ima više NIC istog tipa i u tom slučaju se pokreće više kopija istog drajvera. Svakoj od kopija se daje jedinstveni broj instance (0 za prvi, 1 za drugi, itd.)
Na drajveru je da odluči kojoj instanci će da dodeli koju mrežnu kartu. To mapiranje bi trebalo da bude stabilno posle višestrukih podizanja sistema, kako bi se korisnička podešavanja za pojedinačnu instancu odnosila uvek na isti interfejs. Najčešće se ovo radi tako što se za instancu N drajvera linearno čitaju PCI uređaji koji odgovaraju drajveru, i preskoči N-1 odgovarajućih pre nego što se rezerviše uređaj .

### Inicijalizacija

Inicijalizacija drajvera se obavlja po učitavanju. U aktuelnoj verziji *MINIX* 3 (verzija 3.3.0) se učitavanje drajvera je jedan od poslova komandnog skripta `rc`, po uzoru na BSD sisteme [McKusick1996], kojeg po startovanju izvršava prvi "korisnički" proces `init` pomoću komandnog interpretera. Konkretno pokretanje mrežnog drajvera se obavlja pomoću komande "`service up`" zajedno sa pokretanjem i ostalih elemenata mreže iz skripta `/usr/rc`.

Mrežni drajver se pokreće iz `/usr/etc/rc` skripta koji prethodno iz fajla `/etc/inet.conf` preuzima ime i broj instance drajvera:

```
service up lance -label lance_0 -args "instance=0" -period 5HZ
service up /service/lance -label lance_0 -args instance=0 -period
5HZ
```

Zatim se pokreće mrežni server:

```
service up /service/lwip -script /etc/rs.inet -dev /dev/ip
```

ili

```
service up /service/inet -script /etc/rs.inet -dev /dev/ip
```

Prilikom inicijalizacije drajver mora da sazna svoj broj instance. On se prosleđuje drajveru kao jedan od argumenata komandne linije u obliku `instance=N`, gde je N decimalna cifra, koja predstavlja broj instance. Čitanje ovog broja se obično radi pomoću funkcija `env_setargs()` i `env_parse()`, koje su deklarisane u <minix/sysutil.h>.
Drajver treba da pokuša da inicijalizuje uređaj neposredno po startovanju i, ukoliko inicijalizacija bude neuspešna, to odmah i oglasi (npr. pomoću funkcije `panic()` ). Po uspešnom pokretanju, drajver mora da oglasi svoje prisustvo na DS serveru. Ovo bi trebalo da se uradi pozivom funkcije `netdriver_announce()` iz biblioteke *libnetdriver*.

### Interakcija sa sistemom

Sve sistemske procese, pa i drajvere, pokreće i kontroliše server RS (*Reincarnation Server*). Kada se drajver pokrene, RS mu šalje inicijalizacionu poruku sa uputstvom kako da se pravilno inicijalizuje. Drajver bi trebao da odgovori sa OK (uspela inicijalizacija) ili sa kodom greške (nije uspela). U slučaju greške RS će oboriti drajver i ponovo ga pokrenuti. Ovaj inicijalizacioni protokol je sakriven kroz upotrebu SEF-a (*System Event Framework*) koji drajverima nudi bibliotečke funkcije koje olakšavaju inicijalizaciju i čine je efikasnijom. SEF je komponenta sistemske biblioteke koja se bavi sistemskim događajima (engl. *events*) na

centralizovan način, što olakšava rad sa njima. SEF se tako automatski brine i o interakciji sa RS serverom. Da bi drajver mogao da reaguje na signale (na drugačiji način od podrazumevanog), treba da registruje funkciju za rukovanje signalima (engl. *signal handler callback function*) za čega se takođe koristi SEF. Na primer, drajver bi trebao da završi rad kada dobije SIGTERM signal, a pre izlaska, drajver bi trebao da zaustavi rad svog uređaja.
U sklopu inicijalizacije, drajver takođe mora da uradi pretragu/pronalaženje svog uređaja i rezervaciju hardverskih resursa. Ove aktivnosti se vrše u interakciji sa serverom koji je nadležan za magistralu na koji je uređaj priključen (npr. PCI).

### Protokol komunikacije

Pošto je mrežni drajver zaseban proces, on sa mrežnim podsistemom (INET server) komunicira pomoću IPC-a. Osnovni protokol za komunikaciju sa mrežnim drajverom se sastoji od zahteva INET servera ka drajveru i odgovora drajvera. Polje poruke `m_type` sadrži tip poruke, bilo da je reč o zahtevu ili odgovoru. Svi tipovi poruka, kao i njihova polja su definisani u `<minix/com.h>`.
Komunikacija sa drajverom uvek započinje porukom za konfiguraciju interfejsa - DL_CONF, iako još drugih DL_CONF zahteva može da bude izdato kasnije. INET server neće slati novi zahtev dok ne dobije odgovor na prethodni. Zbog toga drajver mora uvek odmah da odgovori na svaki zahtev. U slučaju prenosa podataka koriste se poruke DL_READV_S i DL_WRITEV_S čije izvršenje može da potraje. Zato drajver, prilikom prijema zahteva za prenos podataka, odmah šalje odgovor da potvrdi zahtev i još jedan odgovor kada se zahtev završi. Pomenuti DL_CONF zahtev i preostali DL_GETSTAT_S traže striktnu komunikaciju tipa “zahtev-odgovor” (engl. *request-reply*).
Za vršenje IPC-a, mrežni drajveri bi trebali da poruke primaju pozivom funkcije `netdriver_receive()` iz biblioteke `libnetdriver` (koja poziva funkciju `sef_receive status()` ), a funkcija `ipc_send()` (sinhroni SEND poziv) bi trebala da se koristi za slanje odgovora INET serveru. S druge strane, INET server za slanje poruka drajveru koristi funkciju `asynsend()` (asinhroni SENDA poziv). Razlog za ovakav način komunikacije je sprečavanje nepouzdanog drajvera da eventualnom greškom u implementaciji blokira pouzdani i provereni INET server [Herder2008].

### Prenos podataka

Kao i sve druge zahteve mrežnom drajveru, zahteve za prenos podataka su takođe inicira INET server. INET će poslati DL_READV_S zahtev kada je spreman da primi paket, odnosno DL_WRITEV_S paket kada ima paket spreman za slanje. Moguće je da drajver nije uvek u stanju da istog momenta zadovolji zahteve za slanjem ili primanjem paketa, u slučaju da recimo nema paketa u prijemnom redu/baferu ili zato što je red za slanje pun. Ako je drajver u mogućnosti da zahtev zadovolji odmah, treba da pošalje DL_TASK_REPLY poruku, sa odgovarajućim DL_PACK_ flegom postavljenim u DL_FLAGS polju (i u slučaju prijema, sa odgovarajuće postavljenim DL_COUNT). Ako drajver nije u mogućnosti da odmah zadovolji zahtev, treba da pošalje DL_TASK_REPLY poruku sa vrednošću DL_NOFLAGS (0) u DL_FLAGS polju, kako bi signalizirao da je zahtev primljen ali da se čeka na njegovo izvršenje. Kada drajver uspe da izvrši zahtev koji je bio na čekanju (zato što je stigao novi paket ili se pojavilo slobodno mesto u redu za slanje), treba da pošalje još jednu DL_TASK_REPLY poruku sa odgovarajućim DL_PACK_ flegom postavljenim u DL_FLAGS polju (i odgovarajućim DL_COUNT u slučaju prijema). Ovo se obično događa prilikom odgovora na prekid. Ako isti prekid istovremeno zadovoljava i zahtev za slanjem i

zahtev za prijemom koji su na čekanju, jedna DL_TASK_REPLY poruka može da potvrdi oba zahteva. U tom slučaju DL_STAT bi sadržao binarno OR-ovanu kombinaciju DL_PACK_SEND i DL_PACK_RECV flegova, a DL_COUNT bi sadržala veličinu primljenog paketa. Drugim rečima, iako zahtev za slanjem ili prijemom mora da bude odmah potvrđen odgovorom, dati zahtev ostaje u stanju čekanja (engl. *pending*) sve dok odgovarajući DL_PACK_ fleg ne bude setovan u odgovoru. Drajver nikada neće primiti drugi zahtev istog (DL_READV_S, DL_WRITEV_S) tipa, dok prethodni zahtevi još nisu u potpunosti završeni i potvrđeni.

### 3.3.4.3 Opis poruka protokola za komunikaciju sa mrežnim drajverom

#### Konfiguracija

Poruka DL_CONF ima dvojaku ulogu. Prvo, ona specificira u koji režim rada (koji može biti promiskuitetni/multicast/broadcast) treba da se prebaci *Ethernet* kartica. Drugo, zahteva hardversku adresu (MAC) kartice. Drajver može da dobije više DL_CONF poruka u toku svog života. DL_CONF zahtev izgleda ovako:

| | | |
|---|---|---|
| Zahtev | DL_CONF | Zahtev za konfiguraciju i postavljanje režima rada |
| Polje | DL_MODE | polje sa flegovima režima rada |

Polje DL_MODE je bit kombinacija sledećih mogućih flegova:

| Alias | Vrednost | Značenje |
|---|---|---|
| DL_PROMISC_REQ | 0x1 | promiskuiterni režim |
| DL_MULTI_REQ | 0x2 | multikast režim |
| DL_BROAD_REQ | 0x4 | broadkast režim |

Ovi flegovi označavaju kakvi će paketi biti primani, osim paketa koji su adresirani direktno na fizičku adresu mrežnog interfejsa. Alias DL_NOMODE predstvlja vrednost 0 i koristi se kada nijedan od flegova nije postavljen.

Mrežni drajver bi trebao da promeni režim rada mrežnog uređaja prema flegovima, koji su postavljeni u zahtevu i da odgovori na sledeći način:

| | | |
|---|---|---|
| Odgovor | DL_CONF_REPLY | Podaci o konfiguraciji |
| Polje | DL_STAT | kod rezultata |
| Polje | DL_HWADDR | MAC adresa |

Rezultata ima svoj kod, koji mora biti postavljen na vrednost OK da bi signalizirao uspešnost operacije, ili na negativni kod greške. Ako drajver ne može da rezerviše uređaj i da komunicira sa njim, očekivano je da kod rezultata ima vrednost ENXIO. Ukoliko je

konfigurisanje bilo uspešno, treba smestiti hardversku (MAC) adresu u polje DL_HWADDR odgovora. Drajver može da iskoristi DL_CONF poruku da inicijalizuje hardver, ali se ipak preporučuje da se inicijalizacija obavi odmah po startovanju drajvera.

### Statistika

Od kada se uređaj konfiguriše porukom DL_CONF, INET server može, kada kod mu je to potrebno, da od drajvera traži statistiku o broju prenetih paketa i statistiku grešaka.

| Zahtev | DL_GETSTAT_S | Zahtev za statistikom komunikacije |
|---|---|---|
| Polje | DL_GRANT | dozvola (WRITE) za strukturu `eth_stat_t` |

Po prijemu ovog zahteva, drajver mora da popuni polja strukture `eth_stat_t` najbolje što može, da je iskopira koristeći `sys_safecopyto()` sistemski poziv i dozvolu koja je stigla u zahtevu, i pošalje odgovor:

| Zahtev | DL_STAT_REPLY | Odgovor sa statistikom komunikacije |
|---|---|---|
| Polje | nema | |

### Prenos podataka

Osnovna namena mrežnog uređaja je da prima i šalje mrežne pakete.

#### Prijem paketa

Kada mrežni server želi da primi paket sa mreže, on mora da obezbedi memorijski prostor za prijem paketa i da drajveru pošalje zahtev koji sadrži vektor memorijskih dozvola za dati prostor (tačnije dozvolu za vektor dozvola). Zahtev za prijem paketa izgleda ovako:

| Zahtev | DL_READV_S | Primi paket |
|---|---|---|
| Polje | DL_GRANT | grant (READ) za vektor iovec_s_t |
| | DL_COUNT | broj elemenata vektora |

Zahtev dolazi sa grantom za vektor elemenata tipa `iovec_s_t`. Ovaj tip koji specificira dozvole i veličine prijemnih bafera. Struktura tipa `iovec_s_t` je deklarisana u fajlu `type.h`:

```
typedef struct {
  cp_grant_id_t iov_grant;    /* grant ID of an I/O buffer */
  vir_bytes iov_size;         /* sizeof an I/O buffer */
}
```

Drajver paket kopira tako što prvo kod sebe iskopira vektor koristeći kernelski poziv `sys_safecopyfrom()`, i zatim za svaki od elemenata vektora kopira deo paketa u bafer

za koji je data dozvola sadržana u datom elementu vektora koristeći kernelski poziv `sys_safecopyto()`, sve dok se ne iskopira ceo paket.

#### Slanje paketa

Od mrežnog servera se očekuje da paket u potpunosti pripremi za slanje, sa svim poljima uključujući odredišnu i izvorišnu MAC adresu. Jedino se CRC polje ne zadaje, nego ga računa sam NIC. Kada je paket spreman za slanje, mrežni server drajveru šalje zahtev za slanje, koji sadrži vektor memorijskih dozvola na delove paketa. Taj zahtev izgleda ovako:

| | | |
|---|---|---|
| Zahtev | DL_WRITEV_S | Pošalji paket |
| Polje | DL_GRANT | dozvola (READ) za vektor `iovec_s_t` |
| | DL_COUNT | broj elemenata vektora |

Slično kao i zahtev za prijem, zahtev za slanje paketa sadrži vektor koji sadrži elemente tipa `iovec_s_t`. Svaki od elemenata specificira dozvole i veličine za bafere koji sadrže delove paketa za slanje. Proces njihovog kopiranja u paket za slanje se sastoji od kopiranja vektora i kopiranja sledećih `iov_size` bajtova paketa za slanje sledeće `iov_grant` dozvole sadržane u elementima vektora.

### Odgovor na zahteve za prijem i slanje paketa

Poruka za odgovor za DL_READV_S i DL_WRITEV_S je ista. Odgovor može da potvrdi prijem zahteva i/ili da označi da je poslati zahtev ispunjen.

| | | |
|---|---|---|
| Odgovor | DL_TASK_REPLY | Potvrda čekanja ili završetka prenosa podataka |
| Polje | DL_FLAGS | flegovi završetka |
| Polje | DL_COUNT | Ako je fleg prijema setovan, ovde se nalazi veličina primljenog paketa |

Flegovi završetka izvršenja zahteva DL_FLAGS mogu da budu binarne kombinacije sledećih flegova:

| Alias | Vrednost | Značenje |
|---|---|---|
| DL_PACK_SEND | 0x1 | zahtev za slanjem je izvršen |
| DL_PACK_RECV | 0x2 | zahtev za prijemom je izvršen |

Drajver uvek mora da odgovori neposredno po prijemu zahteva kako bi potvrdio njegov prijem. Ukoliko je zahtev primljen ali nije izvršen, odgovor ne sadrži postavljene flegove (sadrži konstantu DL_PACK_NONE, čija je vrednost 0x0). Kada označava završetak

zahteva za prijemom, drajver mora da specificira veličinu primljenog paketa u polju DL_COUNT. Odgovor na zahtev slanja nema ovo polje.

# 3.4 Mrežni podsistem operativnog sistema (mrežni i transportni protokoli)

Implementacije protokola trećeg (mrežnog) i četvrtog (transportnog) nivoa OSI modela takođe zahtevaju podršku operativnog sistema. Da bi se omogućila međuprocesna komunikacija između procesa, svi UNIX-oliki operativni sistemi omogućavaju procesima kreiranje krajnje tačke za ovu vrstu komunikacije, koja se naziva utičnica ili soket (engl. *socket*). U skladu sa pomenutom UNIX filozofijom "sve je fajl", procesi sokete koriste kao da su fajl deskriptori, upotrebom sistemskih poziva `read()` za čitanje, odnosno `write()` za pisanje. Da bi sistem znao od koga da dobavi podatke za čitanje tj. kome da prosledi podatke koji se upišu na soket, prilikom kreiranja proces mora da saopšti sa kojim drugim procesom želi da komunicira. Ukoliko se proces sa kojim se želi komunikacije nalazi na drugom računaru, komunikacija se ostvaruje putem mreže, pa se u tom slučaju soket naziva mrežni soket. Svaki mrežni soket mora da sadrži:

- mrežnu adresu koja identifikuje dati računar i
- port, komunikacioni identifikator procesa na datom računaru.

Mrežna adresa predstavlja identifikator koji pripada protokolu trećeg OSI nivoa (kao što je IP), a port pripada protokolu četvrtog, transportnog nivoa (kao što su TCP ili UDP). Svaki od procesa koji želi da učestvuje u mrežnoj komunikaciji mora da kreira mrežni soket. To znači da su za komunikaciju između dva procesa potrebna dva soketa.
Iako korisnički procesi, odnosno aplikacije koriste sistemske pozive `read()` i `write()` kada koriste sokete, ovi pozivi se izvršavaju nad softverskim objektima, a ne nad mrežnim interfejsom. Nekoliko stotina soketa može da bude multipleksirano na istom fizičkom interfejsu.
Internet protokol odnosno IP [Postel 1981a] je protokol mrežnog nivoa koji je danas najviše zastupljen.

## 3.4.1 Mrežni podsistem *Linux* jezgra

Mrežni podsistem *Linux* jezgra je dizajniran da bude nezavisan od vrste protokola koji se koristi. To se odnosi i na protokole mrežnog nivoa (IP, IPX, …), kao i na protokole nivoa veze podataka (*Ethernet*, *Token ring*, *PPP*, …). Interakcija između mrežnog drajvera i kernela se bavi jednim mrežnim paketom u datom trenutku, što dozvoljava specifičnostima protokola da budu sakrivene od drajvera, a da fizički prenos bude sakriven od protokola. Pri tome se "zaglavlje" (engl. *header*) je skup bajtova tj. okteta, koji se dodaju paketu dok prolazi kroz slojeve mrežnog podsistema. Na primer, kada aplikacija pošalje blok podataka kroz TCP [Postel 1981b] soket, mrežni podsistem deli te podatke na pakete i na početak svakog paketa dodaje TCP zaglavlje, koje opisuje koji je dati paket po redu unutar dobijenog niza paketa. Niži nivo zatim ispred svega dodaje IP zaglavlje koje se koristi za nalaženje odredišta paketa. Ako taj paket ide preko nekog Ethernet medija, pre svih ostalih zaglavlja se dodaje Ethernet zaglavlje koje kasnije interpretira hardver. Mrežni drajveri (obično) ne

moraju da brinu o zaglavljima višeg nivoa, nego samo o kreiranju zaglavlja hardverskog nivoa. Način funkcionisanja *Linux*-ovog mrežnog podsistema je detaljno obrađen u [Benvenuti2006] i [Rosen2013].

### 3.4.2 Mrežni podsistem *MINIX*-a

Za mrežnu komunikaciju *MINIX* sistema je još od prvih verzija standardno zadužen je server *INET*. Ovaj server podržava mrežni protokol IP, kao i transportne protokole TCP i UDP. Način na koji mrežni server komunicira sa mrežnim drajverima je opisan u prethodnom delu poglavlja o drajverima, a njegova interna struktura je opisana u [Wong2003] i [Shang2003].

Zbog modularne strukture *MINIX*-a moguće je lako zameniti ovaj server nekim drugim, pod uslovom da implementiraju iste interfejse ka ostatku sistema. Tako je za *MINIX* napravljena implementacija *LwIP* [Dunkels 2001].

Pokazano je da je moguće zameniti jedinstveni mrežni server grupom servera od kojih svaki obavlja različitu mrežnu funkciju [Hruby2012]. Na ovaj način je moguće iskoristiti više procesorskih jezgara istovremeno i ovakvim paralelizmom ostvariti dobitak na mrežnim performansama.

## 3.5 Literatura

[Benvenuti2006] Christian Benvenuti. 2006. *Understanding Linux Network Internals*, O'Reilly Media, Inc.

[Corbet2005] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. 2005. *Linux Device Drivers*, "O'Reilly Media, Inc."

[Day1995] J. Day. 1995. The (un)revised OSI reference model. *SIGCOMM Comput. Commun. Rev.* 25, 5 (October 1995), 39–55.

[Dunkels2001] Adam Dunkels. 2001. *Design and Implementation of the lwIP TCP/IP Stack*, Swedish Institute of Computer Science.

[Herder2008] Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, and Andrew S. Tanenbaum. Countering IPC Threats in Multiserver Operating Systems (A Fundamental Requirement for Dependability). In *2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*. IEEE, 112–121.

[Hruby2012] Tomas Hruby, Hruby Tomas, Vogt Dirk, Bos Herbert, and Andrew S. Tanenbaum. 2012. Keep net working - on a dependable and fast networking stack. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. DOI:http://dx.doi.org/10.1109/dsn.2012.6263933

[IEEE2013] IEEE. 2013. *Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Piscataway, NJ, USA: IEEE.

[IEEE2014] IEEE. 2014. *ISO/IEC/IEEE International Standard for Ethernet*, Piscataway, NJ, USA: IEEE.

[Kadav2012] Asim Kadav and Michael M. Swift. 2012a. Understanding modern device drivers. *SIGARCH Comput. Archit. News* 40, 1 (April 2012), 87.

[McKusick1996] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S.

Quarterman. 1996. *The Design and Implementation of the 4.3 BSD UNIX Operating System*, Addison-Wesley.

[Postel1981a] J. Postel. 1981a. *Internet Protocol*, RFC Editor.

[Postel1981b] J. Postel. 1981b. *Transmission Control Protocol*, RFC Editor.

[Ritchie1974] Dennis M. Ritchie and Ken Thompson. 1974. The UNIX Time-sharing System. *Commun. ACM* 17, 7 (July 1974), 365–375.

[Rosen2013] R. Rosen, Linux Kernel Networking: Implementation and Theory, 1 edition. New York, NY: Apress, 2013.

[Shang2003] Mingdong Shang, Inet Help Sheet For Minix, 2003 http://www.cis.syr.edu/~wedu/seed/Documentation/Minix2/minix2_inet.pdf

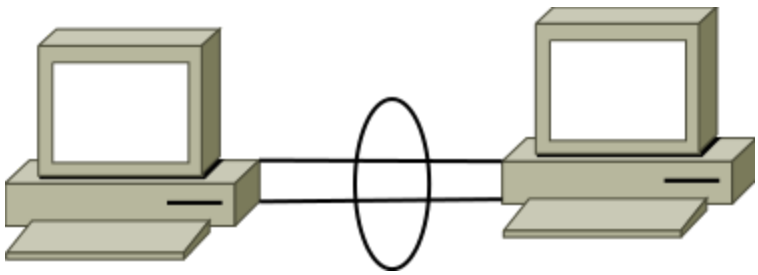
[Wong2003] Charlie Tsz-Hong Wong, Minix Networking Documentation, 2003. http://www.nyx.net/~ctwong/minix/

# 4. Agregacija mrežnih veza

Agregacija mrežnih veza (engl. *link aggregation*) je način za korišćenje više paralelnih mrežnih veza između dva uređaja kao jedan kanal. u cilju povećanja pouzdanosti mrežne konekcije, a takođe i način za povećanje ukupnog protoka podataka raspoređivanjem po raspoloživim vezama. U ovom poglavlju se daje pregled najčešće korišćenih načina agregacije, kao i nekoliko primera implementacije.

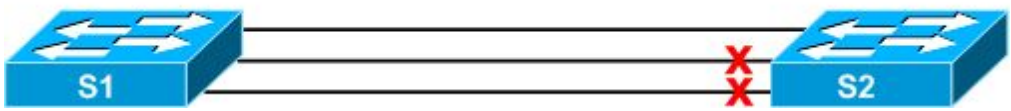
## 4.1 Uvod

On nastanka računarskih mreža postoji potreba da se uveća njihova propusna moć i pouzdanost. Jedan od načina da se to postigne je agregacija mrežnih veza. Agregacija mrežnih veza predstavlja skup metoda za korišćenje paralelnih veza između dva uređaja kao da su jedan jedinstveni komunikacioni kanal [Seifert2008]. Na ovaj nači se postiže veća dostupnost i veća propusna moć novostvorenog komunikacionog kanala između uređaja bez promene mrežne tehnologije. Skup veza koje se zajedno koriste se naziva agregaciona grupa ili LAG (engl. *Link Aggregation Group*). Ova grupa može da bude predstavljena klijentima viših nivoa kao jedan jedinstveni logički interfejs za komunikaciju sa ostatkom mreže [IEEE2008]. Na slici 1 je prikazan primer šeme agregirane veze dva računara. Oba prikazana računara imaju po dva mrežna interfejsa, gde je prvi interfejs prvog računara spojen mrežnom vezom (linija, vod) na prvi interfejs drugog računara, a drugi interfejs prvog računara spojen na drugi interfejs drugog računara. Ove mrežne veze učestvuju u agregacionoj grupi, koja formira komunikacioni kanal između računara.



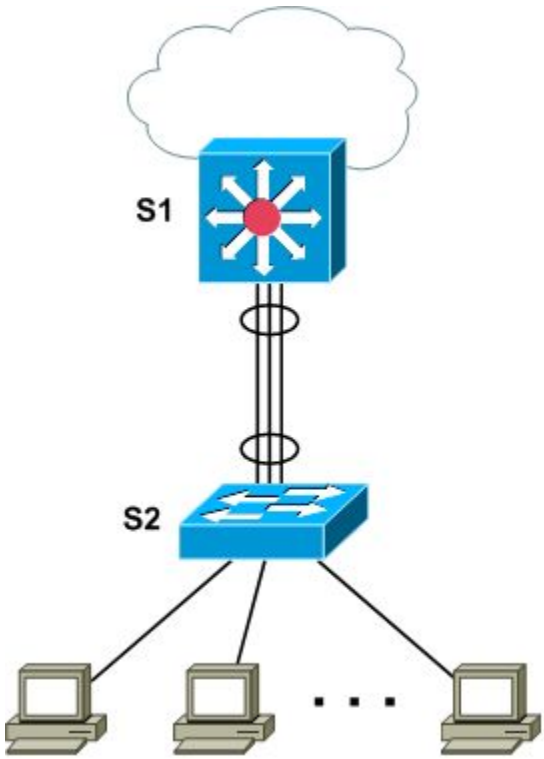
Slika 4.1: Šematski prikaz agregacije mrežnih veza između dva računara

Važan razlog za agregaciju mrežnih veza je mogućnost da se obezbedi nastavak komunikacije i u slučaju kada jedna od veza otkaže. Ova osobina se naziva otpornost na otkaz (engl. *fault tolerance*), a prebacivanje komunikacije sa aktivne na rezervnu liniju zove se fejlover (engl *failover*) (slika 4.2).




Slika 4.2: Agregacija linkova između svičeva S1 i S2 omogućava nastavak komunikacije iako su otkazale dve od tri agregirane veze

Za komunikaciju je dovoljno postojanje bar jednog komunikacionog kanala odnosno veze, a postojanje više raspoloživih paralelnih mrežnih veza predstavlja redundantnost kojom se omogućava tolerisanje eventualnog gubitka neke od veza.
Budući da svaka od redundantnih mrežnih veza pojednačno ima svoj propusni opseg, veći mrežni protok je moguće postići raspoređivanjem protoka na raspoložive veze (engl. *load balance*). Na slici 4.3 je prikazana lokalna mreža računara koji su povezani na mrežni svič S2. Na svič S2 je takođe povezan i ruturajući svič S1, koji povezuje datu lokalnu mrežu sa ostalim mrežama tj. sa internetom. Pošto se pretpostavlja da više računara istovremeno ima potrebu za vezom sa internetom, veza između svičeva S1 i S2 će u tom slučaju trpeti najveće opterećenje, zbog čega je ona pojačana trostrukom agregiranom vezom. Konekcije računara iz lokalne mreže sa računarima sa interneta bivaju raspoređeni po mrežnim vezama iz agregacione grupe.




Slika 4.3: Raspoređivanje protoka između svičeva S1 i S2, da bi se omogućila brža komunikacija računara u lokalnoj mreži povezanih na svič S2 sa internetom (preko sviča S1).

Važno je napomenuti da raspoređivanje protoka automatski znači i otpornost na poremećaje, jer otkaz neke od veza članova agregacione grupe uzrokuje da se njen saobraćaj rasporedi na ostale članove grupe.

## 4.2 Nivoi agregacije

Agregacija može da se napravi na logičkim vezama različitih nivoa OSI referentnog modela, a poznate su implementacije:

- na nivou linka podataka
- na transportnom nivou
- na mrežnom nivou
- na aplikativnom/midlver nivou

# 4.3 Agregacija mrežnih veza na nivou linka podataka

Agregacija se najčešće obavlja na što nižem nivou, kako bi što je moguće manji deo sistema morao da toga "bude svestan" tj. da bude umešan u njenu implementaciju. Zbog toga se ovaj postupak najčešće implementira na najnižim nivoima OSI. Na nivou linka podataka, agregacija se vrši tako što se PDU (engl. *protocol data unit*) sa ovog nivoa šalje na odgovarajući mrežni interfejs iz agregacione grupe. Način odnosno algoritam na koji se ovo radi se zove **režim agregacije**.

Agregacija na nivou linka podataka se pre svega koristi na mrežama sa *IEEE 802.3 Ethernet* [IEEE2012] slojem linka podataka. Ovakve mreže su trenutno defakto najčešće korišćeni način mrežne komunikacije. Zbog toga će se u ovom delu teksta govoriti pre svega o agregaciji korišćenjem mrežnih veza po ovom standardu.

Postoji mnoštvo različitih komercijalnih implementacija agregacije mrežnih veza koje se nazivaju raznim imenima, kao što su engleski nazivi "*port trunking*", "*link bundling*", "*Ethernet/network/NIC bonding*", ili "*NIC teaming*"[WP2013].

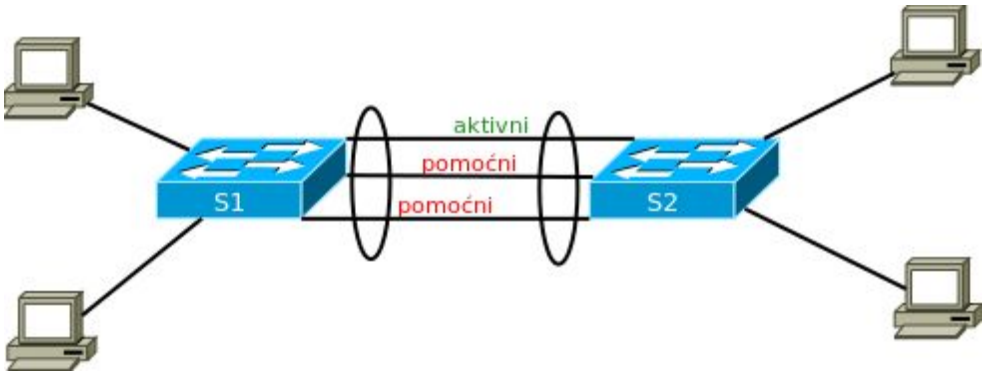
Za neke vrste agregacije postoji podrška u firmveru pojedinih mrežnih uređaja, a neka rešenja su implementirana u okviru operativnog sistema.

## 4.3.1 Režimi agregacije

Osnovni razlozi za agregaciju mrežnih veza su otpornost na poremećaje (FT) i raspoređivanje protoka (LB). U skladu sa tim napravljeni su režimi odnosno algoritmi za agregaciju. Važi napomena da LB algoritmi imaju i FT osobine, jer se u slučaju otkaza nekog od linkova iz grupe, protok raspoređuje na ispravne linkove. Zato se podela pravi prema tome da li je maksimalan protok agregiranog linka veći od kapaciteta jednog fizičkog linka, koji je član agregacione grupe. Ukoliko jeste, taj režim agregacije se smatra LB, a u suprotnom je samo FT. Sledi opis režima agregacije za koje postoji implementacija. Nazivi su preuzeti iz implementacije za *Linux*, jer su na tom sistemu implementirani svi ovde opisani načini agregacije.

### 4.3.1.1 Aktivni-pomoćni ("active-backup")

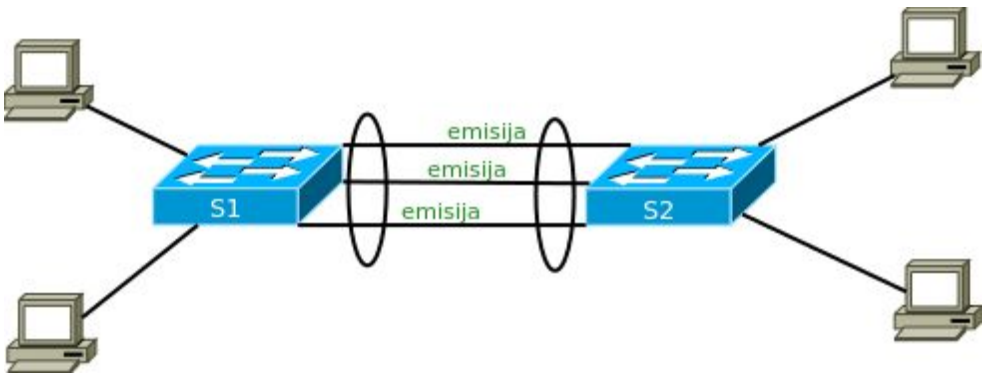
U prvom režimu agregacije samo jedan od agregiranih interfejsa je aktivan, a u slučaju njegovog otkaza aktivira se neki od rezervnih. Ukoliko se implementira na host uređaju (računaru), svi linkovi iz grupe dobijaju istu MAC adresu, ali se to spolja vidi samo na aktivnom linku. Ovaj režim se naziva "aktivni-pomoćni" ("*active-backup*") i prikazan je na slici 4.4. Svrha ovog režima agregacije je da se obezbedi otpornost na otkaz aktivne veze. Ovo je FT (*fault-tolerance*) režim i nema raspoređivanja protoka. Osnovna prednost je jednostavnost implementacije, jer nije potrebno analizirati pakete prilikom prenosa. Mane su neiskorišćenost propusnog opsega pomoćnih linkova i gubitak veze dok se prekid aktivnog linka ne uoči.




Slika 4.4: Agregacija “aktivni-pomoćni”. Samo aktivni link prenosi podatke, a ako dođe do otkaza, jedna od pomoćnih veza preuzima komunikaciju.

#### 4.3.1.2 Emisija (“broadcast”)

U drugom režimu agregacije zvanom “emisija” (engl. *broadcast*) svaki mrežni paket se šalje na sve raspoložive interfejse (slika 4.5). Ukoliko se implementira na host uređaju (računaru), svi linkovi iz grupe dobijaju istu MAC adresu. Ovde se koristi sav raspoloživ propusni opseg, ali za prenos istog sadržaja. Tako rezultujući propusni opseg i dalje nije veći od propusnog opsega jednog kanala, ali se sada veza agregacije ne prekida, bez obzira na prekid neke od veza koje je čine.



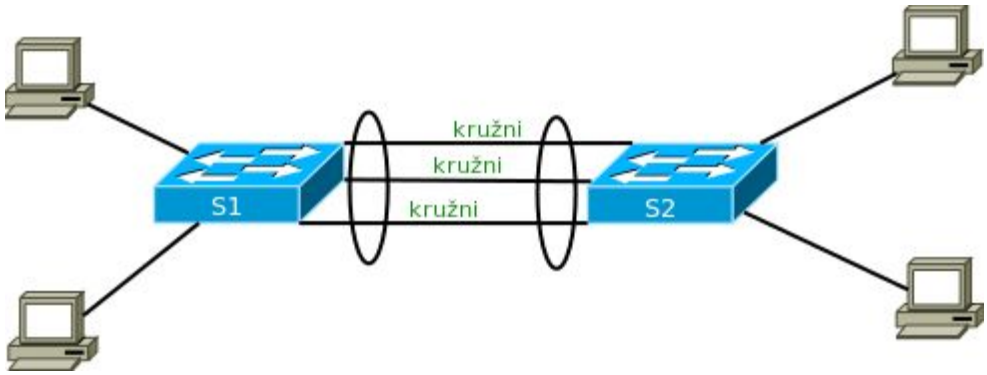

Slika 5. Agregacija “emisija”, gde se isti sadržaj šalje na sve raspoložive linkove

Ova dva režima su isključivo namenjeni za FT primenu, budući da je ukupan rezultujući protok agregiranih veza jednak protoku pojedinačne mrežne veze.

#### 4.3.1.3 Kružno balansiranje (“balance-rr”)

Da bi se iskoristio propusni opseg svih raspoloživih linkova, korisi se neki od LB (engl. *load-balancing*) režima od kojih je najjednostavniji režim “kružnog balansiranja” (engl. “*round robin*”). Ovaj način agregacije podrazumeva da se prilikom slanja paketi podataka sekvencijalno (kružno) raspoređuju od prvog do poslednjeg raspoloživog mrežnog interfejsa (slika 4.6). Na primer, ako treba preneti *n* paketa preko dva na ovaj način agregirana interfejsa, onda će prvi paket ići preko prvog, drugi preko drugog interfejsa, treći opet preko prvog, četvrti preko drugog i tako do poslednjeg paketa. Ukoliko se ovaj način agregacije

implementira na host uređaju (računaru), svi linkovi iz grupe dobijaju istu MAC adresu. I kod ovog režima nema potrebe za analizom zaglavlje paketa, što je olakšava implementaciju. Ovaj režim je jedini u kom je za pojedinačnu TCP [IETF1981-RFC793] konekciju moguće postići veću brzinu prenosa od brzine jednog fizičkog linka. Najveća mana je moguća isporuka mrežnih paketa van redosleda (engl. *out-of-order delivery*), jer paketi putuju različitim fizičkim vezama. Ovo može da poremeti redovnu komunikaciju, jer pojedini protokoli višeg nivoa (npr. TCP) mogu da podrazumevaju da je izgubljen paket koji nije stigao u redosledu i da izvrše njegovu retransmisiju, čime se gubi na brzini prenosa [Stricevic2012-2] [Stricevic2012-3].




Slika 4.6: Agregacija “kružno balansiranje”. Paketi se šalju u krug preko prvog sledećeg linka.

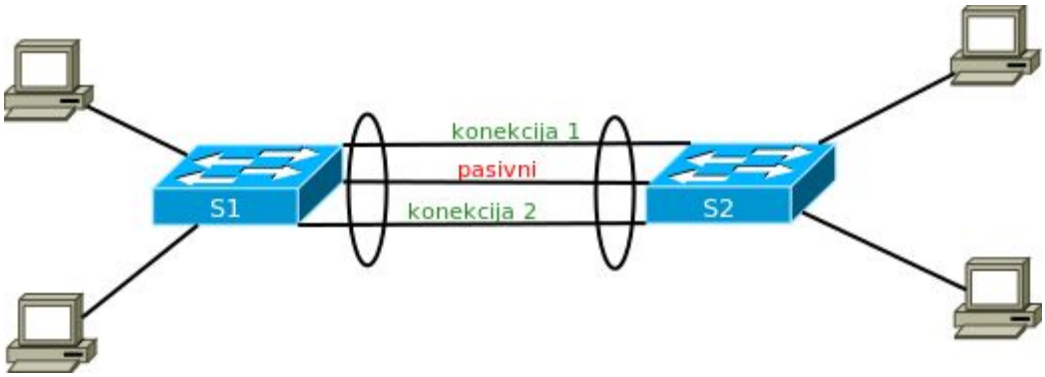
#### 4.3.1.4 EksILI balansiranje (“balance-xor”)

Ostali LB režimi pokušavaju da izbegnu isporuku van redosleda, najčešće tako što se komunikacija između dva konkretna hosta ograničava na jednu fizičku vezu (slika 4.7). Ovo se, na primer, postiže tako što se paketi sa istom izvorišnom i odredišnom fizičkom adresom (MAC) šalju uvek na isti link. Ako su veze koje čine agregaciju numerisane brojevima od 0 do n, režim “*balance-xor*” šalje paket na link određen logičkim izrazom:

**[(izvorni_MAC ^ odredisni_MAC) % broj_linija]**

gde je “^” operacija ekskluzivno ILI, a “%” operacija modula (ostatka pri deljenju), kao u notaciji programskog jezika C. Na primer, ako paket ima ima izvorišnu adresu 00:00:00:00:01:02, odredišnu adresu 00:00:00:00:03:04 i ako postoje 3 agregirana linka (numerisani brojevima 0, 1 i 2) onda će paket biti prosleđen kroz link sa brojem 2 (0x206%3=2).

Osim MAC adrese, ovaj režim agregacije može u svrhu određivanja linka da koristiti i mrežnu (IP) adresu + TCP port. Ovaj način agregacije je vrlo popularan i uključen je npr. u *EtherChannel* [Conlan2009] ili 802.3ad [IEEE2008] standarde za agregaciju.




Slika 4.7: LB režimi agregacije raspoređuju konekcije po raspoloživim linkovima

#### 4.3.1.5 Balansiranje odlaznog i dolaznog opterećenja ("balance-tlb", "balance-alb")

Ovi režimi se koriste isključivo za host uređaje (najčešće servere). U režimu "*balance-tlb*" (engl. *transmit load balancing*), odlazni saobraćaj se raspoređuje na interfejse shodno njihovom opterećenju. Kada se pojavi novi komunikacioni partner (sa novom MAC i/ili IP adresom), sav njemu upućen saobraćaj ide na interfejs koji je do tada bio najmanje opterećen. Interfejsi članovi agregacione grupe zadržavaju svoje MAC adrese. Ukoliko neki od linkova otkaže, drugi link iz grupe preuzima njegovu MAC adresu. Mana ovog algoritma je da ne uzima u obzir opterećenje dolaznog saobraćaja. Zbog toga je napravljen režim "balance-alb" (engl. *adaptive load balancing*), koji radi sve što i "balance-tlb", ali pokušava da i dolazni saobraćaj rasporedi po raspoloživim linkovima. Balansiranje dolaznog saobraćaja se za IPv4 [IETF-RFC791] mrežni saobraćaj postiže presretanjem ARP [IETF1982-RFC826] komunikacije. Naime, presreću se ARP odgovori IP modula sistema i u njega ubacuje MAC adresa najmanje opterećenog mrežnog interfejsa, kako bi druga strana kontaktirala baš taj interfejs i preko njega uspostavila komunikaciju.

### 4.3.2 Upotreba agregacije na host i mrežnim uređajima

Ako se koriste na hostu, režimi agregacije "*broadcast*", "*balance-rr*" i "*balance-xor*" svi agregirani interfejsi imaju istu MAC adresu. To može da zbuni pojedine mrežne uređaje (svičeve), pa je neophodno da i ti uređaji podržavaju grupisanje portova (kreiranje agregacione grupe tj. LAG).

Režimi "*active-backup*", "*balance-tlb*" i "*balance-alb*" mogu da rade bez podrške ostatka mrežne opreme, jer kod ovih režima nema više interfejsa koji u istom trenutku emituju pakete sa istom MAC adresom.

### 4.3.3 Statička i dinamička agregacija

U praksi se prilikom konfigurisanja i funcionisanja agregacije javljaju problemi kao što je npr. loše povezivanje kablova ili nemogućnost utvrđivanja otkaza tj. "pada" linka (ako je recimo link povezan preko konvertora medijuma). Zbog toga se definišu protokoli za kontrolu agregacije (kao što je npr. LACP [IEEE2008]), koji olakšavaju rešavanje ovakvih problema. Ukoliko se prilikom agregacije koristi agregacioni protokol, kažemo da se radi o dinamičkoj agregaciji, a u suprotnom kažemo da je agregacija statička.

Firme koje imaju svoja rešenja za agregaciju obično imaju i sopstvene protokole za podršku dinamičkoj agregaciji. Problem može da nastane, ako dva mrežna uređaja koja se povezuju,

ne podržavaju isti način agregacije.

### 4.3.4 Izvedba agregacije 2. nivoa na aktivnoj mrežnoj opremi

Mnogi proizvođači aktivne mrežne opreme su svojim mrežnim svičevima i ruterima dodavali mogućnost agregacije mrežnih veza, a svojim zaštićenim tehnologijama su davali komercijalna imena. U deljem tekstu su dati kratki opisi komercijalnih tehnologija koje su se prema mišljenju autora pokazale kao najznačajnije.

#### 4.3.4.1 *EtherChannel*

*EtherChannel* [Conlan2009] se smatra za jedan od prvih standarda za agregaciju mrežnih linkova na tržištu *Ethernet* mreža. Ubrzo posle kreiranja prvih mrežnih svičeva 1990. godine, američka kompanija *Kalpana* je kreirala svoj standard za agregacija *Ethernet* linkova pod nazivom "*EtherChannel*". Kako bi ušla na tržište *Ethernet* svičeva, kompanija *Cisco* je 1994 preuzela kompaniju *Kalpana* (zajedno sa kompanijama *Grand Junction*, *Crescendo Communications*), pa je tako i *EtherChannel* promenio vlasnika.

Način agregacije koji se ovde koristi je "*balance-xor*" koji može da obuhvati izvorne i odredišne MAC i IP adrese, TCP portove i eventualni VLAN tag (ukoliko ga ima i ukoliko je agregirani link VLAN trank port). Konfiguracijom portova na mrežnom uređaju se statički definišu članovi agregacione grupe (*Link Aggregation Group* - LAG) koji učestvuju u agregaciji. *Cisco* je za različite tipove linkova definisao različite nazive *EtherChannel* agregacije (*Fast EtherChannel - FEC, Gigabit EtherChannel - GEC*). *EtherChannel* ne dozvoljava da se linkovi iz agregacione grupe nalaze na fizički različitim uređajima. Ovo je donekle moguće prevazići stekovanjem (*stacking*) mrežnih uređaja, ali to obično zahteva da uređaji budu identični.

Za dinamičku agregaciju *EtherChannel* linkova razvijen je *PAgP* protokol, koji je kasnije poslužio kao uzor za *IEEE LACP* protokol. Danas je u *EtherChannel* standardu, koji je ugrađen u operativni sistem *Cisco IOS* (*Internetwork Operating System*), moguće korišćenje oba ova protokola.

#### 4.3.4.2 MLT (*Multi-link Trunking*)

Kompanija *Nortel* je 1999. godine razvila MLT tehnologiju agregacije linkova [Avaya2014], koja je tada ugrađena u operativni sistem za njihove svičeve BoSS (Baystack Operating System Switching Software). 2009. godine je ovu kompaniju zajedno sa svim njenim tehnologijama preuzeo njen današnji vlasnik, kompanija *Avaya*.

MLT je po funkcionalnosti slična *Cisco*-voj *EtherChannel* tehnologiji i takođe koristi "*balance-xor*" način agregacije. U kasnijim poboljšanjima, kao što su DMLT (*Distributed Multi-link Trunking*), SMLT (*Split Multi-link Trunking*) i DSMLT (*Distributed Split Multi-link Trunking*), postaje moguća agregacija linkova koji se nalaze na različitim mrežnim uređajima, tj. svičevima. Ova osobina (uz bolji LB) omogućava postizanje vrlo visoke pouzdanosti mrežne konekcije i predstavlja prednost u odnosu na *EtherChannel*.

#### 4.3.4.3 IEEE 802.3ad

IEEE 802.3 grupa je 2000. godine donela otvoreni standard pod imenom IEEE 802.3ad u okviru kog je specificiran i LACP (*Link Aggregation Control Protocol*) za dinamičku

konfiguraciju [IEEE2013]. Kako osnova za IEEE 802.3ad standard je poslužio *Cisco*-ov *EtherChannel* standard sa svojim PAgP (*Port Aggregation Protocol*) protokolom. Zato je i IEEE 802.3ad takođe zasnovan na "*balance-xor*" načinu agregacije. Ovaj standard je zbog slaganja sa ostalim IEEE protokolima 2008 godine prebačen u IEEE 802.1 grupu sa novim imenom IEEE 802.1AX-2008 [Law2006], ali se češće može sresti staro ime IEEE 802.3ad, tako da će ovaj naziv biti korišćen u ovom tekstu. Pošto je IEEE 802.3ad otvoren standard, danas ga podržava dobar deo aktivnih *Ethernet* mrežnih uređaja raznih proizvođača, kao i mnogi operativni sistemi opšte namene.

## 4.3.5 Izvedba agregacije u operativnim sistemima opšte namene

Agregacija linkova u operativnim sistemima se najčešće izvodi na nivou linka podataka pomoću drajvera koji kreira novi virtuelni mrežni interfejs. Ovaj novi interfejs se konfiguriše tako da kombinuje dva ili više fizičkih mrežnih interfejsa, sa ciljem povećanja raspoloživog propusnog opsega (engl. *load balancing*), kao i da se obezbedi nastavak komunikacije u slučaju da jedna od mrežnih veza otkaže (engl. *fault tolerance*). U zavisnosti konkretnih potreba, bira se odgovarajući režim agregacije. Najčešće su podržani statički načini agregacije, a od dinamičkih je po pravilu podržan IEEE standard 802.1AX-2008 (poznatiji pod svojim starim imenom IEEE 802.3ad) [IEEE2008]. Konkretne implementacije agregacije mrežnih veza se razlikuju od jednog do drugog operativnog sistema, ali ipak postoje i neke sličnosti. Na primer, na *UNIX*-olikim operativnim sistemima (kao što su *Linux*, *NetBSD*, *FreeBSD*, *OpenBSD*, *OS X*, …) agregacija se obično izvodi na nivou neposredno iznad NIC drajvera. Pošto je teško jednim tekstom pokriti sve moguće implementacije agregacije, sledi kratak pregled implementacija na (prema mišljenju autora) najznačajnijim operativnim sistemima opšte namene.

### 4.3.5.1 Agregacija na *Linux*-u: drajver "*bonding*"

*Linux* je jedan od prvih operativnih sistema koji je dobio podršku za agregaciju mrežnih veza. Za to je zaslužan Donald Beker koji je 1993. godine napisao agregacioni kernel drajver pod nazivom ***bonding*** za potrebe klaster sistema *Beowulf* [Jackson2005] . Ovaj drajver omogućava agregaciju više mrežnih interfejsa na nivou veze podataka (engl. *data link*) koje ostatku sistema prikazuje kao jedan logički interfejs. Prilikom konfiguracije je potrebno podesiti konfiguracione fajlove da informišu drajver koje interfejse treba da preuzme i u kom režimu novonastali agregirani interfejs treba da radi. Drajver *bonding* podržava sledeće režime agregacije [Davis2011]:

1. "***balance-rr***" (kružni "*round robin*" balansiranje opterećenja)
2. "***active-backup***" (pri otkazu aktivnog linka, komunikacija se prebacuje na pomoćni)
3. "***balance-xor***" (način agregacije kompatibilan sa *EtherChannel* i IEEE *802.3ad draft v1* standardima)
4. "***broadcast***" (svaki paket se šalje na sve raspoložive interfejse)
5. "***802.3ad***" (dinamička agregacija u skladu sa standardom IEEE 802.3ad [IEEE2008] uz korišćenje LACP protokola)
6. "***balance-tlb***": (balansiranje odlaznog saobraćaja*transmit load balancing*)
7. "***balance-alb***": (*adaptive load balancing* - obuhvata sve osobine "*balance-tlb*", a radi i raspoređivanje dolaznog saobraćaja pomoću ARP poruka)

Agregacija podrazumeva kreiranje jednog logičkog interfejsa, kome se dodeljuje mrežna

adresa (IP) i fizička (MAC) adresa. Svi fizički interfejsi, koji su agregirani, dobijaju fizičku adresu logičkog interfejsa.
U slučaju da je akcenat agregacije na ubrzavanju mrežne komunikacije među nodovima klastera, za izbor načina agregacije treba uzeti režim koji omogućava iskorišćavanje propusnog opsega više interfejsa (*"load balance"*). Većina ovakvih režima raspoređuje saobraćaj na osnovu fizičke (MAC) adrese odredišta, što za veliki broj komunikacionih nodova daje dobre prosečne rezultate. Nažalost, ovo znači da bi komunikacija između dve mašine bila ograničena na brzinu jednog komunikacionog kanala. Samo jedan od režima omogućava da raspoloživi komunikacioni opseg između dve mašine pređe ovu granicu, a to je "*balance-rr*" režim [Mohamed2006]

#### 4.3.5.2 *BSD*: drajveri "*lagg*", "*trunk*" i "*agr*"

Na BSD sistemima takođe postoji sistemski drajver koji takođe daje mogućnost agregacije više mrežnih interfejsa u jedan virtuelni, kako bi se omogućio LB (*load-balancing*) i FT (*fault-tolerance*). Podržani su sledeći režimi rada, slični onima na Linux-u:

1. *"**failover***" ("*active-backup*")
2. "***fec***" (*balance-xor* kompatibilan sa EtherChannel uređajima)
3. "***loadbalance***" (alias "fec" režima)
4. "***lacp***" - ("*802.3ad*")
5. "***roundrobin***" - ("*balance-rr*")
6. "***none***" - (agregacija je isključena)

Navedeno važi za FreeBSD implementaciju agregacije pomoću drajvera po imenom "*lagg*" [FreeBSD]. Ovo važi i za OpenBSD implementaciju (od verzije 3.8) gde postoji drajver "*trunk*" [OpenBSD].
Za NetBSD (od verzije 4.0) postoji drajver "*agr*" [NetBSD] koji implementira isključivo IEEE 802.3ad način agregacije.

#### 4.3.5.3 *Mac OS X*

OS X podržava isključivo agregaciju po IEEEE 802.3ad standardu. 2004. godine je firma *Small Tree Communications* napravila IEEE 802.3ad agregacioni drajver za svoje NIC interfejse [DiBenedetto2004], a 2005. je ova vrsta agregacije ugrađena u serversku verziju *MacOS X v10.4 "Tiger"* [Regan2006].

#### 4.3.5.4 *Windows*

Sam po sebi, operativni sistem *Windows* nije podržavao agregaciju mrežnih linkova sve do verzije *Windows Server 2012* [Microsoft2012]. U ovoj verziji hipervizor sistema (*Hyper-V*) podržava konfiguraciju "*Switch dependant*" i "*Switch independant*".
Režimi koji su na ovaj način podržani su:
"Active/Standby" ("*active-backup*") za "Switch independant" konfiguracije
IEEE 802.3ad statički ("*balance-xor*") i IEEE 802.3ad dinamički (LACP) režimi agregacije za "*Switch dependant*" konfiguraciju.
Pojedini proizvođači mrežnih interfejsa (npr Intel [Intel2006]) su i ranije omogućavali agregaciju pomoću drajvera koji omogućava kreiranje agregacione grupe, za interfejse koji

su pod njegovom kontrolom. Ovaj način agregacije pod *Windows*-om nije moguća za sve kartice, nego je neophodno da kartica ima drajver koji to omogućava.

## 4.4 Agregacija na mrežnom nivou

Ukoliko agregacija treba da "izađe" van lokalne mreže, nivo linka podataka više nije dovoljan, već toga mora da bude svestan i mrežni OSI sloj. Primeri protokola mrežnog nivoa koji ovo podržavaju su *Mobile IP*, HIP (*Host Identity Protocol*) , SHIM6 (*Site Multihoming by IPv6 Intermediation*). Mana svih ovih rešenja za agregaciju na mrežnom nivou je da transportni protokol (kao što je TCP pre svega) nije svestan da se put paketa promenio i teško može da prilagodi svoj mehanizam kontrole zagušenja (*congestion control*) novonastaloj situaciji.

## 4.5 Agregacija na transportnom nivou

Agregaciju na transportnom nivou implementiraju protokoli transportnog OSI nivoa koji su "svesni" postojanja više različitih mrežnih interfejsa sa različitim mrežnim adresama. Primeri ove vrste agregacije su SCTP [Stewart2001] i *MultiPath TCP* (MPTCP) [UCL2012, Paasch2014]. Oni omogućava simultanu upotrebu više IP interfejsa. SCTP je poseban protokol transportnog nivoa, dizajniran za ovu namenu. Nov protokol, kao što je SCTP, ne uspeva da se nametne kao standardno rešenje za agregaciju na transportnom nivou, jer ima API različit od TCP protokola, koji je više decenija važi za standard u mrežnoj komunikaciji i na koji su programeri navikli.

Da bi se prevazušao ovaj problem napravljen je MPTCP, koji uz pomoć modifikovanog TCP sloja daje regularnu TCP uslugu, odnosno API interfejs aplikacijama, dok u procesu komunikacije deli protok podataka na nekoliko podtokova. Prilikom slanja standardni TCP tok se deli na podtokove, dok se prilikom prijema podtokovi agregiraju u jedan prijemni tok i tako isporučuju prijemnoj apllikaciji.

## 4.6 Agregacija na aplikativnom/midlver nivou

Nekada je najbolje prepustiti samoj aplikaciji da se brine o onome što joj treba [Saltzer1984]. Ukoliko je aplikaciji neophodna agregacija mrežnih veza, recimo radi postizanja FT osobina, ona ne može da računa da će to da podržava svaki sistem na kom se bude izvršavala (npr. pojedine verzije Windows-a). Zbog toga je dobro ako se agregacijom se bavi deo zadužen za komunikaciju u okviru aplikacije. Taj deo se obično implementira u okviru zasebne biblioteke odnosno midlvera. Jedan primer specifikacije standarda za ovakav midlver je CORBA (engl. *Common Object Request Broker Architecture*) [OMG2012], za koju postoje brojne implementacije.

U okviru CORBA standarda postoji deo pod nazivom "FT-CORBA" (*Fault Tolerant CORBA*), koji se bavi problemima otpornosti na otkaze (engl. *fault tolerance*), uključujući i otkaz mreže. "FT-CORBA" se u praksi pokazala kao vrlo kompleksna i teška za implementaciju, tako da upotrebljivih implementacija, kao npr. MEAD (*Middleware for Embedded Adaptive Dependability*) [Narasimhan2002] ima veoma malo i koriste se za specifične namene. Ipak pojedine implementacije CORBA midlvera uspevaju da postignu otpornost na otkaze pomoću agregacije mrežnih veza, iako nemaju "FT-CORBA" implementaciju [Stricevic2012]

## 4.7 Literatura

[Avaya2014] Switch Clustering using Split MultiLink Trunking (SMLT) with VSP 9000, VSP 8000, VSP 7000, ERS 8600/8800, and ERS 5000 Technical Configuration Guide

[Conlan2009] Conlan, P. J. 2009. Cisco Network Professional's Advanced Internetworking Guide. SYBEX Inc., Alameda, CA, USA.

[Davis2011] Davis, T; Linux Ethernet Bonding Driver HOWTO, 2011 http://www.kernel.org/doc/Documentation/networking/bonding.txt

[DiBenedetto2004] J. DiBenedetto; Small Tree Communications Brings Ieee 802.3 Link Aggregation To The Apple Market, Enabling Enterprise Class Network Solutions; 2004 http://blog.small-tree.com/press-releases/small-tree-communications-brings-ieee-802-3-link-aggregation-to-the-apple-market-enabling-enterprise-class-network-solutions/

[IEEE2008] IEEE Std 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks — Link Aggregation. IEEE Standards Association. 2008-11-03. p. 30.

[IEEE2012] IEEE Standard for Ethernet IEEE Std 802.3™-2012

[IEEE2013] IEEE 802.3ad Link Aggregration Task Force, retrieved 09.09.2013 http://www.ieee802.org/3/ad/

[IETF1981-RFC791] Internet Protocol ; IETF; RFC 791:, 1981. http://www.ietf.org/rfc/rfc791.txt

[IETF1981-RFC793] Transmission Control Protocol ; IETF; RFC 793:, 1981. http://www.ietf.org/rfc/rfc793.txt

[IETF1982-RFC826] An Ethernet Address Resolution Protocol -- or -- Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware; IETF; RFC 826:, 1982. http://www.ietf.org/rfc/rfc826.txt

[Intel 2006] Intel Advanced Network Services Software Increases Network Reliability, Resilience and Bandwidth, Intel White Paper, 2006 http://www.intel.com/network/connectivity/resources/doc_library/white_papers/254031.pdf

[Jackson2005] Jackson Joab, “Donald Becker | The inside story of the Beowulf saga”, Government computer news, ISSN:0738-4300, Apr 13, 2005 https://gcn.com/Articles/2005/04/13/Donald-Becker--The-inside-story-of-the-Beowulf-saga.aspx

[Law2006] Law, D.; "IEEE 802.3 Maintenance" (PDF). p. 9., 2006, retrieved 09.09.2011 http://www.ieee802.org/3/maint/public/maint_open_1106.pdf

[Microsoft2012] Windows Server 2012 NIC Teaming (LBFO) Deployment and Management, 2012, http://www.microsoft.com/en-us/download/details.aspx?id=30160

[Mohamed2006] Mohamed, N et al.; High-performance message striping over reliable transport protocols. J. Supercomput. 38, 3 (December 2006), 261-278.

[Narasimhan2002] Narasimhan, Priya, et al. "Middleware for embedded adaptive dependability." IEEE Workshop on Large Scale Real-Time and Embedded Systems, Austin, TX. 2002.

[OMG2012] - Object Management Group, (2012),Common Object Request Broker Architecture (CORBA) Specification, Version 3.3

[Paasch2014] Christoph Paasch and Olivier Bonaventure. 2014. Multipath TCP. Commun.

ACM 57, 4 (April 2014), 51-57.

[Regan2006] Regan, Schoun; Mac OS X Server Essentials; Peachpit Press 2006; ISBN-13: 978-0321357588

[Saltzer1984] J. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end arguments in system design. ACM Trans. Comput. Syst. 2, 4 (November 1984), 277-288.

[Seifert2008] Rich Seifert and James Edwards. 2008. The All-New Switch Book: The Complete Guide to LAN Switching Technology (2 ed.). Wiley Publishing.

[Stewart2001] Stewart, R. and Xie, Q. Stream Control Transmission Protocol: A Reference Guide. Addison-Wesley, 2001.

[Stricevic2012] L. Stričević and P. Rakić; “Otpornost na otkaz mreže pomoću CORBA agregacije mrežnih veza”, YU INFO (18; Kopaonik; 2012 ), 365-369

[Stricevic2012-2] L. Stricevic, P. Rakic and M. Hajdukovic, "Finite strip method construction analysis program execution speed improvement on an MPI cluster by using multiple network links," *Telecommunications Forum (TELFOR), 2012 20th*, Belgrade, 2012, pp. 1405-1408.

[Stricevic2012-3] M. Hajdukovic, D. D. Milasinovic, M. Nikolic, P. S. Rakic, Z. Zivanov, and L. Stricevic, “Scope of MPI/OpenMP/CUDA parallelization of harmonic coupled finite strip method applied on large displacement stability analysis of prismatic shell structures,” Computer Science and Information Systems, vol. 9, no. 2, pp. 741–761, 2012.

[UCL2012] MultiPath TCP - Linux Kernel implementation, http://mptcp.info.ucl.ac.be/

[WP2013] Linux Aggregation;2011; Wikipedia, Wikimedia Foundation Inc., retrieved 09.09.2011. http://en.wikipedia.org/wiki/Link_aggregation

# 5. Primer implementacije agregacije mrežnih veza: *MINIX* 3

U ovom poglavlju su dati detalji implementacije agregacije na operativnom sistemu MINIX 3 uz obrazloženje odluka vezanih za dizajn implementacije. Pošto je cilj da se agregacija integriše sa operativnim sistemom, odlučeno je da njena implementacija bude na drugom nivou OSI (na nivou veze podataka). Kao uzor za *MINIX* 3 implementaciju agregacionog modula je poslužila implementacija agregacionog bonding drajvera sa Linux-a, koji je detaljnije opisan u [Aust2006].
Agregacija na drugom nivou OSI treba da obuhvati sledeće:

- mogućnost da se za agregaciju izdvoji grupa fizičkih interfejsa (link-agregaciona grupa - LAG),
- mogućnost da se šalju i primaju paketi na svim interfejsima koji učestvuju u agregaciji (LAG) i
- mogućnost da se detektuje otkaz bilo kog linka iz LAG (engl. *link monitoring*)

## 5.1 Mogući modeli implementacije agregacije mrežnih veza na nivou linka podataka

Pre same implementacije agregacije mrežnih veza u okviru MINIX 3 operativnog sistema, razmatrani su mogući načini da se to uradi. Moguće je da se agregacija ugradi u neki od mrežnih drajvera, da se ugradi u postojeći mrežni TCP/IP sloj (INET) ili da se implementira kao zaseban modul tj. agregacioni drajver. Sledi diskusija svake od ovih mogućnosti.

### 5.1.1 Opcija 1: izmena drajvera

Moguća je implementacija drajvera koji će da zameni dva (ili više) mrežnih drajvera, kako bi podržao agregaciju. Ovo rade pojedini proizvođači mrežnih interfejsa [Intel2006]. U ovom slučaju bi u konfiguraciji trebalo reći drajveru koje NIC treba da uzme i koji režim agregacije da primeni. Ako se menja format fajla `inet.conf`, treba menjati i mrežni server (INET ili LwIP) i uvesti zaseban konfiguracioni fajl (npr. `/etc/agrtl8139.conf`). U inicijalizaciji bi trebalo inicijalizovati NIC-ove i postaviti istu MAC adresu za sve njih.

Prednost ovog pristupa je mogućnost korišćenja svih hardverskih mogućnosti NIC-a u svrhu agregacije, pošto drajver ima direktan pristup hardveru.
Mana ovog pristupa je da se u agregaciju mogu uključiti samo oni mrežni interfejsi koje dati drajver podržava i da posle izmena treba testirati sve funkcije mrežnog drajvera, uključujući i one koje nemaju veze sa agregacijom.

### 5.1.2 Opcija 2: izmena TCP/IP sloja

Moguće je izmeniti mrežni server kako bi podržao agregaciju na nižem nivou. Budući da je mrežni TCP/IP server direktno nadređen svim mrežnim drajverima, moguća je i implementacija agregacije direktno unutar TCP/IP servera. Sličan princip se primenjuje kod implementacije lokalnog "*loopback*" interfejsa. Pošto je za mrežnu komunikaciju na MINIX3 OS-u je zadužen INET server, a od MINIX-a 3.2, kao opcioni server za mrežnu komunikaciju se nudi i LwIP server [LwIP2014], trebalo bi menjati neki od ta dva servera. Da bi se takvom

serveru dodala i funkcionalnost agregacije, potrebno je da se obezbedi:

- dodavanje načina konfigurisanja agregacije,
- dodavanje strukture podataka koje bi opisale agregaciju i
- definisanje algoritma agregacije.

Prednost ovog prisupa je mogućnost direktnog pristupa TCP baferima, čime se dobija na brzini, a osnovna mana proizilazi iz činjenice da mora da se menja mrežni server. To znači da je potrebno je vršiti kompletno testiranje svih funkcija mrežnog servera i da bilo koja dalja izmena mrežnog servera može da zahteva izmenu u kodu koji podržava agregaciju.

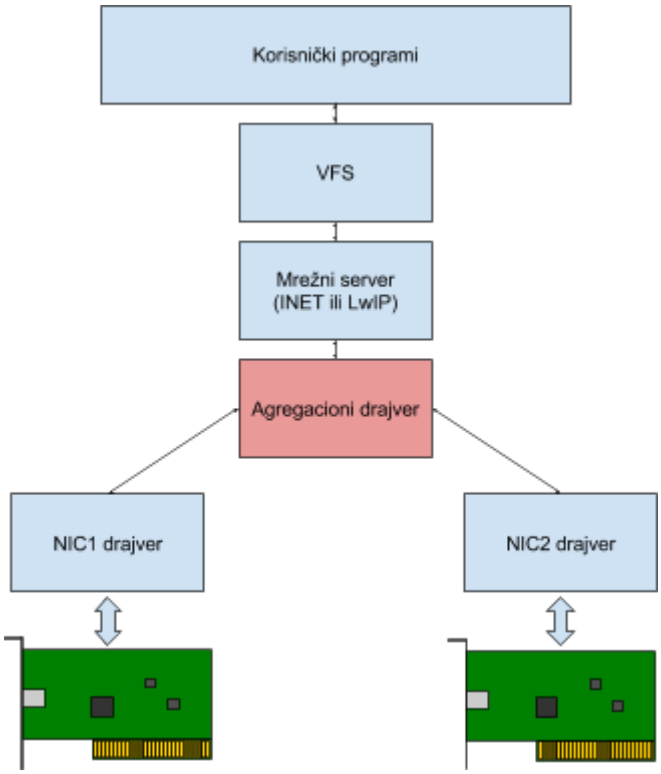
### 5.1.3 Opcija 3: Implementacija agregacionog drajvera

Agregacija bi mogla da bude implementirana u vidu posebnog agregacionog drajvera, koji bi za više nivoe operativnog sistema bio vidljiv kao i svaki drugi mrežni drajver (po uzoru na "*bonding*" drajver na *Linux*-u, "*lagg*" drajver na *FreeBSD*-u i "*agr*" drajver na *NetBSD*-u).
Prednosti ovog pristupa su da on ne zahteva izmene na ostatku operativnog sistema i da radi za bilo koji mrežni server i bilo koji drajver.
Mana ovog pristupa je da on uvodi dodatni sloj u mrežnoj komunikaciji, što neizbežno unosi dodatni overhed, pa samim tim i smanjuje maksimalnu brzinu komunikacije.

Ipak pošto ovaj pristup zahteva najmanje izmena na ostatku operativnog sistema i nudi fleksibilnost rada sa različitim drajverima i mrežnim serverima, on je odabran za implementaciju agregacije na *MINIX* 3 operativnom sistemu. U implementaciji je neophodno minimizirati overhed koji ovaj pristup implicira.

## 5.2 Implementacija agregacionog drajvera

Agregacioni drajver se logički nalazi između mrežnog servera servera mrežnoih drajvera agregiranih interfejsa (slika 5.1). Ovaj virtuelni interfejs ima svoj agregacioni drajver. U slučaja otkaza neke od mrežnih veza (vezanih na odgovarajući mrežni interfejs, koji kontroliše odgovarajući mrežni drajver), agregacioni drajver će saobraćaj da preusmeri na neki od drajvera čiji interfejs ima vezu koja je ispravna.
Konkretna implementacija agregacionog drajvera je napravljena za *MINIX* 3.3.0. (588a35b) verziju operativnog sistema. Pošto ovaj agregacioni drajver zahteva minimalne izmene ostatka operativnog sistema ograničene na minimum, on bi trebao da radi na svim verzijama MINIX-a na kojima funkcioniše isti IPC mehanizam i protokol za komunikaciju sa mrežnim drajverom (engl. *data link protocol*)




Slika 5.1: Logička pozicija agregacionog drajvera

### 5.2.1 Prosleđivanje saobraćaja

Agregacioni drajver posreduje u komunikaciji između mrežnog servera (kao što je INET) i mrežnih drajvera, tako što se predstavi mrežnom serveru predstavi kao drajver, a sa mrežnim drajverima komunicira kao da je mrežni server (slika 5.1). Za svaki paket koji mrežni server pošalje, agregacioni drajver ga prosledi jednom od mrežnih NIC drajvera, a paketi koji pristignu sa mreže tj. od mrežnog drajvera se prosleđuju mrežnom serveru. Zbog toga agregacioni drajver, kao i ostali sistemski procesi (pogotovo drajveri), ima strukturu programa vođenog događajima (engl. *event-driven*). To znači da postoji beskonačna glavna petlja, koja se naziva "petlja događaja" (engl. *event-loop*), u kojoj se primaju poruke od ostalih delova sistema i u zavisnosti od koga poruka dolazi i od njene sadržine, obavljaju se odgovarajuće funkcije (engl. *event-handlers*) koje taj događaj obrađuju.

Pošto na *MINIX* 3 ne postoji način da dva procesa dele memoriju, podaci se prenose kontrolisanim kopiranjem koje obavlja kernel (što je opisano u poglavlju 2) [Herder2009] . Da bi kopiranje bilo moguće, mora se napraviti "dozvola" za podatke koji treba da se kopiraju. Ta dozvola se zatim, putem poruke, dostavlja drugom procesu koji treba da od kernela traži da se podatak (za koji je upravo dobio dozvolu) kopira. MINIX 3 mrežni server (INET) podatke koji čine jedan paket kopira u više delova, a dozvole na svaki od delova prenosi u obliku vektora, koji je definisan u obliku tipa `iovec`. Dakle, prilikom prenosa paketa (recimo od INET servera do drajvera), prvo se prenese vektor dozvola, a potom svaki od delova.

### 5.2.2 IPC komunikacija

Da bi se izbegli problemi u komunikaciji sa nepouzdanim drajverom [Herder2008], agregacioni drajver sa NIC drajverima komunicira asinhrono (funkcijom `asynsend()`) dok sa mrežnim serverom (INET) komunicira sinhrono (funkcijom `ipc_send()`) .

### 5.2.3 Nadzor veze

Pored prosleđivanja mrežnog saobraćaja, agregacioni drajver mora da vrši i nadzor mrežnih veza (engl. *link monitoring*) koje učestvuju u agregaciji. Ovo se radi u cilju utvrđivanja eventualnog prekida neke od njih, kako bi saobraćaj, namenjen prekinutoj vezi, bio preusmeren na neku od ispravnih veza.
U praksi se koriste sledeći načini za utvrđivanje ispravnosti veze:

- MII (engl. *media indpendant interface*) nadzor predstavlja proveru stanja MII registara kartice koja može da otkrije postojanje/nepostojanje nosećeg signala (engl. *carrier*) na prijamnom delu interfejsa. Ukoliko noseći signal ne postoji, prenos podataka nije moguć i link sasvim sigurno nije u funkciji. Ukoliko noseći signal postoji na prijemu, ne postoji garancija da je ostatak mreže funkcionalan (npr. možda je interfejs priključen na konvertor medija, koji daje signal bez obzira na stanje sa druge strane).
- ARP nadzor funkcioniše tako što se na link šalju ARP upiti za jedan ili više sistema i ako se dobije odgovor, to se interpretira kao znak da link funkcioniše.
- Dinamička agregacija korišćenjem nekog od protokola za kontrolu agregacije između ostalog obezbeđuje i nadzor veze. Druga strana u komunikaciji mora da podržava dati protokol.

### 5.2.4 Delovi agregacionog drajvera

Agregacioni drajver se sastoji od sledećih delova:

- inicijalizacija SEF (engl. *System Event Framework*)
- registracija kod DS-a, pretraga da li su NIC drajveri već učitani i registrovani, kao i nalaženje IPC identiteta drajvera
- slanje DL_CONF poruke drajveru kako bi se saznala/postavila njegova fizička adresa
- inicijalizacija bafera i dozvola
- prijem paketa od INET servera i njegovo slanje/prosleđivanje odgovarajućem drajveru
- prijem paketa od mrežnog drajvera i njegovo prosleđivanje INET serveru

Na MINIX3 sistemu, svaki drajver i sistemski server mora da koristi SEF [MINIX_SEF]. SEF (engl. *System Event Framework*) je komponenta sistemske biblioteke (LIBSYS) koja omogućava rukovanje sistemskim događajima na centralizovan i jednostavan način (ili bar jednostavniji nego da ove komponente nema). Svaki sistemski servis (drajver ili server) mora da pozove funkciju `sef_startup()` prilikom pokretanja, zbog inicijalizacije SEF-a, i `sef_receive()` prilikom prijema poruke. Moguće je registrovati povratne (engl. *callback*) funkcije da bi se obezbedilo rukovanje za svaku od raspoloživih vrsta događaja. Ako se ne registruje povratna funkcija za određeni događaj, radi se podrazumevana aktivnost.

Prilikom inicijalizacije se obavlja registracija kod DS (engl. *DataStore*) servera [MINIX_DS]. Ovaj server omogućava ostalim komponentama da smeštaju podatke i da ih kasnije ponovo preuzimaju, kada im je to potrebno, npr. prilikom ponovnog pokretanja. Ovaj mehanizam može da se koristi i za razmenu podataka između komponenti. Tako se od svakog mrežnog drajvera očekuje da upiše svoje ime i IPC identitet kao podatke u DS. Agregacioni drajver glumi mrežni drajver, pa tako upisuje svoje ime i IPC identitet u podatke DS-a.

Inicijalizacija memorijskih dozvola se takođe obavlja prilikom inicijalizacije agregacionog drajvera. Potrebno je inicijalizovati dozvole za vektor kao i za njegove članove, kako bi se podaci iz njih mogli razmenjivati sa drugim komponentama.
Poruke od mrežnog servera se prosleđuju mrežnom drajveru tako što se u postojeću primljenu poruku postavi IPC identitet drajvera, a direktne dozvole iz poruke se zamenjuju indirektnim. Ovo se radi kako bi mrežni drajver bio u stanju da čita/piše direktno u bafere INET servera. Takođe, poruke pristigle od mrežnog drajvera se na sličan način prosleđuju INET serveru. Struktura podataka neophodna da podrži indirektno prosleđivanje (bez dodatnog kopiranja) mora da sadrži:

- IPC identitete svih mrežnih drajvera koji učestvuju u agregaciji
- IPC identitet mrežnog (INET) servera
- Indirektnu dozvolu za poruku DL_GETSTAT_S
- direktnu dozvolu za vektor - `iovec`, niz indirektnih dozvola i prazan `iovec` za kopiranje dobijenog `iovec` za poruku DL_WRITEV_S
- direktnu dozvolu za `iovec`, niz indirektnih grantova i prazan vektor (`iovec`) za kopiranje dobijenog `iovec` za poruku DL_READV_S

Ovakav indirektan način prosleđivanja paketa je brz, jer nema dodatnog kopiranja podataka u/iz memorijski prostor agregacionog drajvera, ali to znači i da agregacioni drajver ne može da čita podatke (pakete) koje prosleđuje. To može da predstavlja problem ako se planira da agregacioni drajver podržava neki od protokola (koji zahtevaju razmenu paketa). Zato je ostavljena mogućnost prosleđivanja podataka pomoću direktnih dozvola i sa kopiranjem svakog paketa u memorijski prostor agregacionog drajvera.
Prilikom startovanja agregacioni drajver treba da preuzme konfiguraciju fižičkih interfejsa kako bi način agregacije odgovarao željama administratora. Konfiguracija agregacije se čita iz tekstualnog fajla koji je sličan fajlu za konfiguraciju mreže (`/etc/inet.conf`).
Ovaj konfiguracioni fajl sadrži:

- ime agregacionog interfejsa
- naziv režima agregacije
- ime mrežnog drajvera prvog fizičkog interfejsa
- ime mrežnog drajvera drugog fizičkog interfejsa

Za sada agregacioni drajver podržava agregaciju dva interfejsa, a u budućnosti se planira podrška za više njih.

Agregacioni drajver za sada omogućava dva osnovna režima agregacije:

- prvi režim je “aktivni-pomoćni” ili **“failover”** režim agregacije: mrežni saobraćaj ide isključivo kroz jedan, aktivni link odnosno mrežni interfejs. Ukoliko aktivni link postane nedostupan, sledeći ispravan link iz agregacionog skupa postaje aktivan i preuzima na sebe sav saobraćaj. Prilikom kreiranja agregacionog skupa, prvi dodati interfejs je u početku aktivan, dok su svi ostali pasivni.
- drugi režim je “kružni” ili **“roundrobin”**: odlazni saobraćaj (paketi) se raspoređuje po svim interfejsima iz agregacionog skupa i prihvata sa bilo kog interfejsa iz skupa.

U budućnosti se planira implementacija i drugih režima rada, kao što su *LACP* ili *EtherChannel*.
Agregacioni drajver mora da bude u stanju da koristi *MINIX3* IPC mehanizam i da sarađuje

sa serverom mrežnog protokol steka. To znači da to mora da bude sistemski proces *MINIX*-a. Prema tome njegova prava moraju da budu opisana u `/etc/system.conf` fajlu (ili zasebnom fajlu u `/etc/system.conf.d/` direktorijumu), a on se pokreće pomoću komande "`service up`".

### 5.2.5 Konfiguracija

Prilikom redovnog podizanja *MINIX* 3 sistema, drajveri se pokreću iz skripta `/usr/etc/rc` , a imena mrežnih drajvera koje treba da pokrene taj skript nalazi u fajlu `/etc/inet.conf`. Kasnije rc skript pokreće skript `/etc/rc.net` koji konfiguriše IP adrese interfejsa. Ideja za (privremenu) konfiguraciju agregacije je da se agregacioni drajver navede kao prvi i "*default*" drajver u fajlu `/etc/inet.conf`, a da se drajveri fizičkih NIC-ova navedu posle njega. Na ovaj način će agregacioni drajver biti pokrenut prvi, a svoje "robove" (mrežne drajvere koji se koriste za agregaciju) će da nađe kada se ovi budu oglasili na DS-u.
Primer fajla `/etc/inet.conf`:

```
eth0 lnprox 0 { default; } ;
eth1 lance 0 { } ;
```

Problem neće praviti TCP/IP server (INET ili LwIP) ako se korisnički programi ne obraćaju direktno fizičkim mrežnim interfejsima, jer ih ni TCP/IP server ih neće dirati, pa neće praviti problem u komunikaciji agregacionog drajvera sa NIC drajverom. Zbog toga će se u skriptu `/etc/rc.net`, gde se konfiguriše mrežna adresa interfejsa (a kog poziva inicijalizacioni skript `/usr/etc/rc`) konfiguriše samo mrežna adresa agregiranog interfejsa. Zatim se čita konfiguracioni fajl `/etc/lnprox.conf` koji sadrži sledeća polja:

- IP adresa za koju se šalju ARP paketi (ako je nema, ne radi se provera ispravnosti linka i ostalli parametri se ne gledaju),
- interval slanja ARP poruka (podrazumevana vrednost je 1s) i
- broj intervala bez primljenog paketa za koji se smatra da je link otkazao (podrazumevana vrednost 2)

Primer izgleda jednog konfiguracionog fajla za agregacioni drajver `/etc/lnprox.conf` je:

```
slave1=rtl8169_0
slave2=rtl8169_1
aggmode=failover  # slave1 je aktivan
arp_interval=1
arp_iptarget=192.168.1.1
```

### 5.2.6 Izvođenje

Osnovni delovi, koje sadrži agregacioni drajver su deo za inicijalizaciju i glavna petlja unutar koje reaguje na razne događaje i poruke:
U inicijalizaciji treba:

- inicijalizovati memorijske dozvole
- naći parametre za ARP paket:
    - naći MAC adresu agregiranog interfejsa i postaviti u ARP paket
    - naći ip adresu agregiranog interfejsa kada se postavi i postaviti u ARP paket
    - očitati parametre iz konfiguracionog fajla i postaviti u internu strukturu podataka

- inicializovati sinhroni tajmer
  - naći broj sistemskih “tikova” u sekundi pomoću funkcije `sys_hz()`
  - postaviti sinhroni tajmer funkcijom `sys_setalarm()`

Unutar glavne petlje su predviđene reakcije na notifikacije iz sledećih izvora:

- CLOCK:
  - resetovati brojač primljenih paketa
  - resetovati tajmer funkcijom sys_setalarm()
  - poslati novi ARP paket preko aktivnog linka
- DS_PROC_NR: (DS-događaj)
  - reakcija kao kod LwIP servera
- TTY_PROC_NR:
  - pritisnut je taster za debag-damp

Unutar glavne petlje su predviđene reakcije na sledeće tipove primljenih poruka:

- DL_CONF
  - sačuvati IPC identitet servera
  - proslediti poruku drajveru (funkcijom `asynsend()`) bez izmena
- DL_CONF_REPLY
  - proslediti poruku serveru (funkcijom `ipc_send()`) bez izmena
- DL_GETSTAT_S, od direktnog DL_GRANT-a sa `cpf_setgrant_indirect()` napraviti indirektanu dozvolu na `eth_stat_t` strukturu i proslediti drajveru
- DL_STAT_REPLY proslediti serveru (funkcijom `ipc_send()`) bez izmena
- DL_READV_S:
  - napraviti lokalni `iovec_s_t` vektor sa DL_COUNT brojem elemenata,
  - sa `safecopyfrom()` u njega iskopirati sadržaj na koji pokazuje DL_GRANT-a (granta originalnog iovec)
  - napraviti drugi lokalni vektor takođe sa DL_COUNT brojem elemenata
  - svaki element prvog vektora (direktni grant na segment paketa) pretvoriti u indirektni grant sa `cpf_setgrant_indirect()` i upisati u drugi vektor
  - napraviti grant na drugi vektor sa `cpf_setgrant_direct()`
  - poslati poruku drajveru (funkcijom `asynsend()`) sa dozvolom na drugi vektor i DL_COUNT
- DL_WRITEV_S:
  - napraviti lokalni `iovec_s_t` vektor sa DL_COUNT brojem elemenata,
  - sa `safecopyfrom()` u njega iskopirati sadržaj na koji pokazuje DL_GRANT-a (dozvola originalnog `iovec`)
  - napraviti drugi lokalni vektor takođe sa DL_COUNT brojem elemenata
  - svaki element prvog vektora (direktnu dozvolu na segment paketa) pretvoriti u indirektnu dozvolu sa `cpf_setgrant_indirect()` i upisati u drugi vektor
  - napraviti dozvolu na drugi vektor sa `cpf_setgrant_direct()`
  - poslati poruku drajveru sa dozvolom na drugi vektor i DL_COUNT
- DL_TASK_REPLY:
  - ukoliko se radi o primljenom paketu (DL_PACK_RECV flag je setovan), uvećati brojač primljenih paketa za dati link
  - proslediti serveru bez izmena

Ovde je prikazana implementacija prijema i predaje bez kopiranja paketa. Za poruke

DL_READV_S i DL_WRITEV_S je moguće umesto prosleđivanja frejmova bez kopiranja (pomoću indirektnih dozvola) potrebno implementirati njihovo kopiranje kroz lokalni bafer. U tom slučaju reakciju na ove dve poruke je potrebno implementirati na sledeći način:

- DL_READV_S: (polja su DL_GRANT za `iovec_s_t` vektor, DL_COUNT broj elemenata vektora)
  - napraviti lokalni `iovec_s_t` vektor sa DL_COUNT brojem elemenata,
  - sa `safecopyfrom()` u njega iskopirati sadržaj na koji pokazuje DL_GRANT (dozvola originalnog `iovec`)
  - sačuvati vektor i DL_COUNT polje
  - napraviti lokalni bafer maksimalne veličine paketa
  - napraviti drugi lokalni vektor sa samo jednim elementom
  - napraviti direktna dozvola na lokalni bafer pomoću `cpf_setgrant_direct()` i upisati ga kao jedini element drugog vektora
  - napraviti dozvolu na drugi vektor sa `cpf_setgrant_direct()`
  - poslati poruku drajveru (funkcijom `asynsend()`) sa dozvolom na drugi vektor sa jednim elementom
- DL_WRITEV_S:
- DL_TASK_REPLY:
  - ukoliko se radi o primljenom paketu (DL_PACK_RECV flag je setovan):
    - sadržaj paket bafera kopirati sa `safecopyfrom()` redom na dozvole iz poslednjeg sačuvanog prijemnog vektora
    - uvećati brojač primljenih paketa za dati link
  - proslediti serveru bez izmena

U slučaju slanja drajveru, šalje se onom drajveru koji je aktivan (režim "*failover*") odnosno onom koji je na redu (režim "*roundrobin*").

### 5.2.7 Kompajliranje

Kompajliranje se obavlja kao i kompajliranje bilo kog drugog *MINIX* 3 sistemskog procesa/modula, što znači da se koriste usluge biblioteke LIBSYS. Pošto se agregacioni drajver predstavlja INET serveru kao mrežni drajver, potrebna je i biblioteka LIBNETDRIVER. Za nadzor veze se koriste *MINIX* 3 sinhroni alarmi, pa je za kompajliranje neophodna i biblioteka LIBTIMERS. Iz navedenog sledi da fajl `Makefile`, koji se koristi za kompajliranje agregacionog drajvera (i koji je deo celokupnog *MINIX* 3 sistema za kompajliranje) izgleda ovako:

```
# Makefile for the lnprox driver.
PROG= lnprox
SRCS= lnprox.c

DPADD+=    ${LIBCHARDRIVER} ${LIBSYS} ${LIBTIMERS}
LDADD+=    -lchardriver -lsys -ltimers

.include <minix.service.mk>
```

### 5.2.8 Alat za testiranje

Za potrebe testiranja agregacionog drajvera je razvijen poseban pseudo-mrežni server (LNTAP). Ovo je učinjeno, jer korisnički procesi nisu u stanju da direktno komuniciraju sa drajverima. O ovome će više biti reči u poglavlju o testiranju, a sam LNTAP server je opisan u dodatku A.

## 5.3 Plan za budući rad

Mehanizam koji je napravljen ovom implementacijom može da posluži za dobijanje servisa agregacije linkova poput onih u komercijalnim operativnim sistemima. Ovde se pre svega misli na dodavanje podrške za *Etherchannel* i IEEE 802.3ad načine agregacije. Takođe bi sličan princip mogao da se primeni i na konstrukciju drajvera za virtuelne interfejse za druge namene, kao što su IEEE 802.1Q VLAN, alias fizičkog interfejsa, pa čak i *Ethernet* most (engl. *bridge*) ili IP-IP tunel interfejs.

### 5.3.1 Predlog modifikacije postojećih mrežnih drajvera

Uočeno je da trenutna koncepcija MINIX 3 drajvera ograničava implementaciju agregacije.To se pre svega odnosi na sledeće aspekte:

1. Agregirani interfejs, koji nastaje prilikom agregacije dva ili više fizičkih veza bi trebao da ima svoju jedinstvenu fizičku (MAC) adresu. Svi fizički interfejsi koji učestvuju u agregaciji, bi trebali da koriste ovu adresu kao svoju prilikom prijema ili slanja paketa. To znači da drajver mora da ima definisanu operaciju postavljanja odgovarajuće fizičke adrese.

2. Potrebno je da postoji način da viši slojevi operativnog sistema budu u stanju da nadziru stanje linkova i eventualno reaguju prilikom otkaza nekog od njih, kako bi problem otkaza mogao da bude prevaziđen. Jedan od načina za nadzor linkova je čitanje MII registara kontrolera koji sadrže podatke o stanju linka (link aktivan/neaktivan, brzina linka, dupleks/poludupleks veza, …). Pošto pristup registrima kontrolera ima samo mrežni drajver, on mora da podržava mogućnost prenosa ovih informacija višim slojevima.

Aktuelni MINIX mrežni drajveri ne podržavaju ove osobine, pa se predlaže njihovo dodavanje u drajver, recimo na sledeći način:

1. Korišćeni nterfejs ima mogućnost da se u memorijski mapirane registre direktno upiše MAC adresa, pa je ta mogućnost dodata i u drajver. Poruka DL_CONF se proširi kako bi prilikom konfiguracije bila preneta i nova MAC adresa.

2. Problem sa očitavanjem statusa linka može da se reši proširivanjem strukture koja se vraća kao odgovor na DL_GETSTAT_S (struktura `eth_stat_t` koja je deklarisana u fajlu `/src/minix/include/net/gen/eth_io.h`) i dodavanjem u drajvere dela koji popunjava prošireni deo strukture.

## 5.4 Literatura

[Aust2006] S. Aust, Jong-Ok Kim, P. Davis, A. Yamaguchi, and S. Obana. 2006. Evaluation of Linux Bonding Features. In *Communication Technology, 2006. ICCT '06. International Conference on*. 1–6.

[Herder2008] Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, and Andrew S. Tanenbaum. Countering IPC Threats in Multiserver Operating Systems (A Fundamental Requirement for Dependability). In *2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*. IEEE, 112–121.

[Herder2009] Herder, J. N., Bos, H., Thomas, A., Gras, B., & Tanenbaum, A. S. Memory Sharing Revisited (Work in Progress), Proceedings EuroSys 2009.

[LwIP2014] lwIP - A Lightweight TCP/IP stack - Summary, http://savannah.nongnu.org/projects/lwip/

[MINIX_BUILD] MINIX Documentation Wiki, Rebuilding the System, http://wiki.minix3.org/DevelopersGuide/RebuildingSystem

[MINIX_DS] MINIX Documentation Wiki, DataStore, http://wiki.minix3.org/DevelopersGuide/DataStore

[MINIX_NETC] MINIX Documentation Wiki, Networking Configuration http://wiki.minix3.org/en/NetworkingConfiguration

[MINIX_SEF] MINIX Documentation Wiki, The System Event Framework (SEF), http://wiki.minix3.org/DevelopersGuide/SEF

# 6. Testiranje agregacionog drajvera

Ovo poglavlje govori o tome kako je implementirano rešenje agregacionog drajvera testirano sa stanovišta funkionalnosti i sa stanovišta performansi.

## 6.1 Testiranje ispravnosti agregacionog drajvera

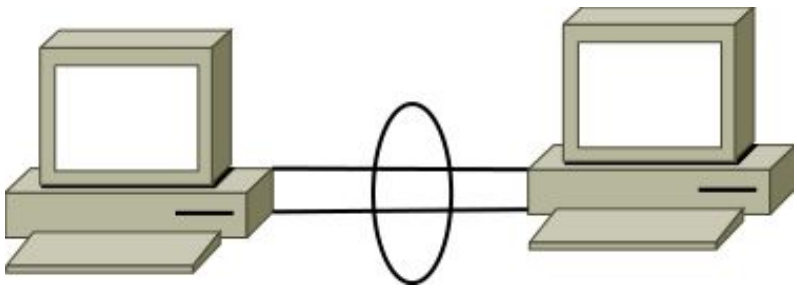
Da bi moglo da se tvrdi da novonastala implementacija agregacionog drajvera funkcioniše, potrebno je izvršiti testiranje prihvatanja (engl. *acceptance testing*) [Black2009]. Ispravnost implementacije agregacionog drajvera je testirana metodom “crne kutije” (engl. *black-box testing*) [SIGST2001], odnosno tako što je testirano da li agregacioni drajver zadovoljava specifikaciju svojih funkcionalnosti. Glavni razlog nastanka ovog agregacionog drajvera je povećanje pouzdanosti i dostupnosti mrežne veze. Zato su testiranjem obuhvaćene sledeće funkcionalnosti:

- Komunikacija između dva računara funkcioniše kada su ispravne sve agregirane veze
- Komunikacija između dva računara nastavlja da funkcioniše kada dođe do prekida jedne agregirane veze.

Oprema koja je korišćena za ovo testiranje je sledeća:

- 2 računara (CPU Intel E2160)
- po 2 PCI mrežne kartice po 1000BASE-T standardu (sa Realtek 8169 mrežnim kontrolerom) za svaki od računara
- po 2 PCI mrežne kartice po 100BASE-TX standardu (sa Realtek 8139 mrežnim kontrolerom) za svaki od računara
- UTP Cat5e mrežni kablovi za povezivanje računara.

Oprema je povezana na način koji prikazuje šema na slici 6.1, a na slici 6.2 je data fotografija opreme na kojoj je testiranje izvedeno.



Slika 6.1: Šema povezivanja računara prilikom eksperimenta

Slika 6.2: Oprema za izvođenje eksperimenta

Testiranje funkcionisanja komunikacije se ispituje pomoću programa *ping*, koji koristi ECHO i ECHO_REPLY poruke ICMP protokola [Postel1981a] da bi utvrdio da mrežna veza funkcioniše. Ovakvo testiranje je pokazalo da komunikacija funkcioniše ako je operativna najmanje jedna komunikaciona veza.

## 6.2 Ocena performansi agregacije

Za mikrokernel operativne sisteme se smatra da su po performansama slabiji u odnosu na sisteme sa monolitnim kernelom, jer se procesorska snaga troši na komunikaciju između modula. Zbog toga realizovani agregacioni drajver za *MINIX* 3 svakako unosi dodatni overhed u proces mrežne komunikacije iako je drajver dizajniran i implementiran sa ciljem da ovaj overhed bude što manji. Da bi ovo bilo provereno, osmišljen je eksperiment kojim bi se izmerila veličina overheda koji unosi agregacioni drajver, odnosno razlika u brzini prenosa koja nastaje kada se instalira pomenuti drajver. Za utvrđivanje efikasnosti agregacionog drajvera, potrebno proveriti:

- kako se implementirani agregacioni drajver ponaša u redovnoj upotrebi
- kolika je razlika u performansama između običnog *MINIX* 3 sistema i sistema koji koristi agregacioni drajver, odnosno koliko se obara performanse upotrebom agregacionog drajvera
- kakva je razlika u performansama mreže između *MINIX* 3 sistema (sa i bez agregacije) i nekog monolitnog sistema, npr. *Linux*-a.

Pojam “performansa mreže” se koristi u raznim kontekstima, ali u ovom slučaju će on podrazumevati mrežni protok (engl. *throughput*). Da bi se proverilo gorenavedeno potrebno je obaviti odgovarajuća merenja, čiji je cilj da se utvrdi u kojoj meri je prosečna brzina

prenosa smanjena kada se u *MINIX* 3 sistem instalira implementirani agregacioni drajver. Oprema koja je korišćena za testiranje performansi je ista kao i oprema korišćena za testiranje funkcionalnosti:

- 2 računara (CPU Intel E2160)
- po 2 PCI mrežne kartice po 1000BASE-T standardu (sa Realtek 8169 mrežnim kontrolerom) za svaki od računara
- po 2 PCI mrežne kartice po 100BASE-TX standardu (sa Realtek 8139 mrežnim kontrolerom) za svaki od računara
- UTP Cat5e mrežni kablovi za povezivanje računara

### 6.2.1 Metodologija merenja

Prenos podataka preko mreže aplikacija preko obično se obavlja pomoću TCP/IP protokola [Postel1981b] , koji je na *MINIX* 3 sistemu implementiran u okviru INET mrežnog servera. Kada korisnička aplikacija želi da koristi mrežni podsistem *MINIX*-a (recimo da otvori TCP konekciju), ona podatke šalje VFS serveru koji ih prosleđuje INET serveru, kojiih prosleđuje drajveru. Svo ovo prosleđivanje utiče na brzinu mrežnog protoka. Zbog toga je eksperiment napravljen tako da se mrežni protok meri na dva načina:

- test prenosa TCP (koji koristi VFS i INET servere) i
- test direktnog prenosa drajveru (koji zaobilazi VFS i INET servere).

Za TCP test prenosa je korišćena poznata test aplikacija *iperf* [Pillai2013], dok je za test direktnog mrežnog protoka (direktna komunikacija sa drajverom) korišćen za tu svrhu napravljen posaban sistemski server - LNTAP. Implementacija ovog servera je detaljnije opisana u dodatku A.

Sva merenja su obavljena na tri različite konfiguracije operativnog sistema:

- *MINIX* 3 sistem (podrazumevana instalacija)
- *MINIX* 3 sistem sa agregacionim drajverom
- *Linux* kao sistem za poređenje

Linux konfiguracija je uzeta kao kontrolni sistem, jer je Linux jezgro jedan od najznačajnijih predstavnika familije sistema sa monolitnim jezgrom i njegove mrežne performanse su dovedene skoro do maksimuma za tehnologije prenosa koje ovde razmatramo. Zato su sve konfiguracije testirane kao komunikacioni partner u odnosu na Linux sistem.

U prethodnom poglavlju smo videli da agregacioni drajver može da podatke između INET servera i drajvera prenosi uz dodatno kopiranje (direktne memorijske dozvole) ili bez kopiranja (indirektne memorijske dozvole). Zbog toga su TCP testovi izvršeni za oba navedena slučaja.

Verzije operativnih sistema korišćene u testiranju su MINIX 3.3.0 i Linux 3.4.52. Nisu vršena nikakva izmene podrazumevanih podešavanja performansi. Verzija *iperf* test programa korišćenog u testovima je 2.0.5.

Svi testovi su izvršeni na dve brzine mrežnog interfejsa:

- 100Mbit/s
- 1000Mbit/s

Svako pojedinačno merenje performansi je rađeno na sledeći način:

- svako merenje je ponovljeno 7 puta
- najveća i najmanja dobijena vrednost je odbačena
- za preostalih 5 vrednosti je nađena srednja vrednost i standardna devijacija

- dobijeni rezultati su prikazani u obliku stubastog diagrama sa oznakama odstupanja (engl. *error bar*).

### 6.2.2 Testiranje performansi

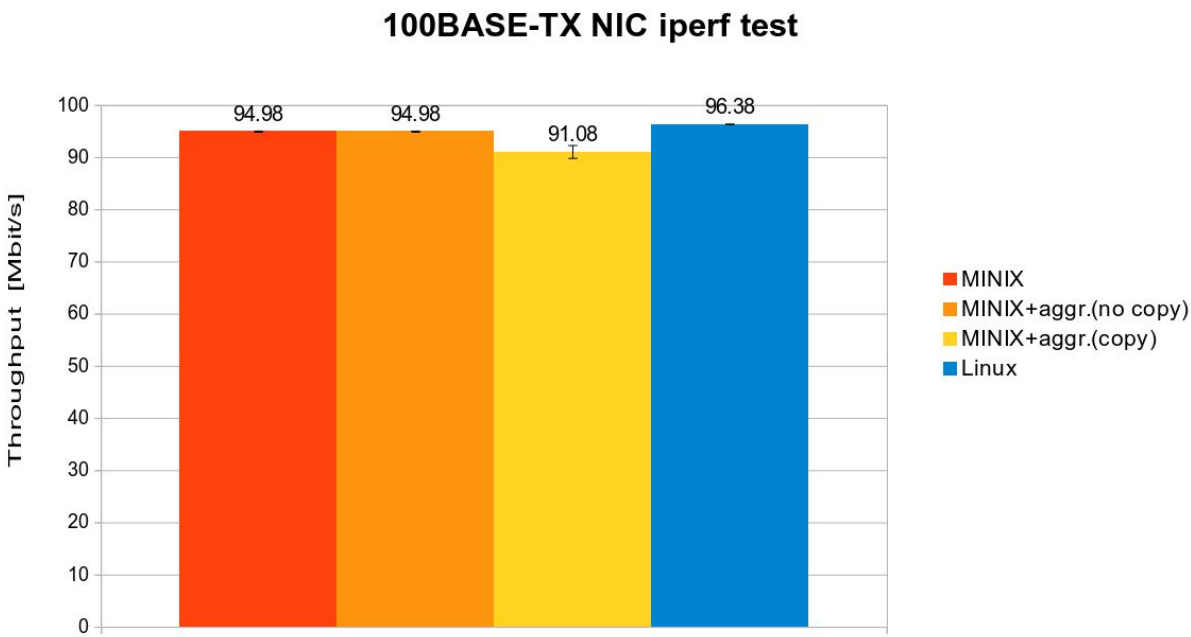
Sledi opis izvršenih testova sa prikazom rezultata:

#### 6.2.2.1 Test perormansi 1: iperf TCP test na 100Mbit/s

Za ovo testiranje oprema je priključena u 100BASE-TX mrežne kartice.

a) *Linux* kontrolno testiranje.
   I na jednom i na drugom računaru je instaliran Linux. Na jednom od računara je pokrenut iperf server, dok je na drugom pokrenut iperf klijent, oba u podrazumevanim konfiguracijama.
b) Instaliran je *MINIX*, pa je *iperf* testiranje ponovljeno, takođe sa podrazumevanim konfiguracijama
c) U *MINIX* konfiguraciju iz prethodnog slučaja je ubačen agregacioni drajver i podešen da podatke između INET servera i NIC drajvera prenosi bez dodatnog kopiranja (pomoću indirektne memorijske dozvole)
d) U konfiguraciji iz prethodnog slučaja agregacioni drajver je podešen da podatke prenosi tako što ih prethodno kopira u svoj bafer (koristeći direktne memorijske dozvole)

Dobijeni rezultat je prikazan na slici 6.3. Vidi se da testovi daju prilično ujednačen rezultat. Nešto slabiji protok se dobija na *MINIX*-u u slučaju kada se podaci dodatno kopiraju prilikom prenosa.



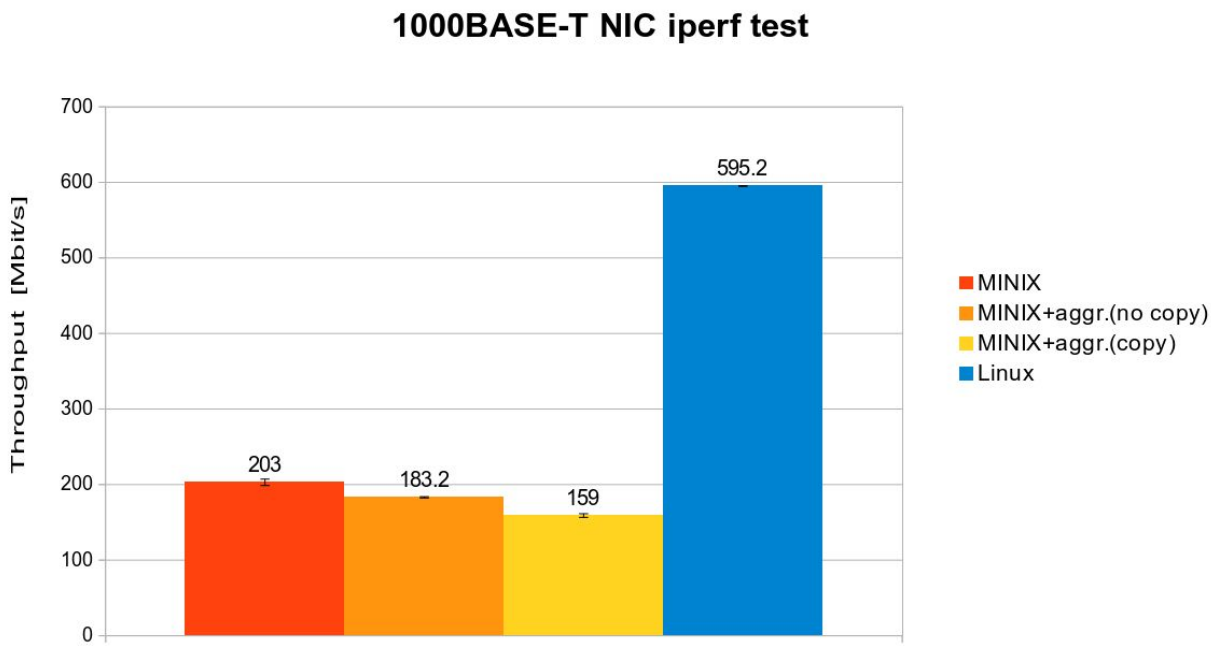

Slika 6.3: rezultati merenja iperf TCP testa na 100Mbps

#### 6.2.2.2 Test performansi 2: iperf TCP test na 1000Mbit/s

Prilikom ovog testiranja su korišćene 1000BASE-T mrežne kartice, a sam postupak je bio sličan kao i u prethodnom testu.

a) *Linux* kontrolno testiranje.
b) *MINIX* konfiguracija
c) *MINIX* konfiguracija sa instaliranim agregacionim drajverom bez dodatnog kopiranja
d) *MINIX* konfiguracija sa instaliranim agregacionim drajverom uz dodatno kopiranje

Rezultati ovog testiranja su dati na slici 6.4. Ovde se može videti da rezultati *Linux*-a daleko prednjače u odnosu na ostale. Takođe se vidi da je sada veća razlika između rezultata čiste *MINIX* konfiguracije, *MINIX* konfiguracije sa agregacijom bez kopiranja i *MINIX* agregacije sa kopiranjem podataka.



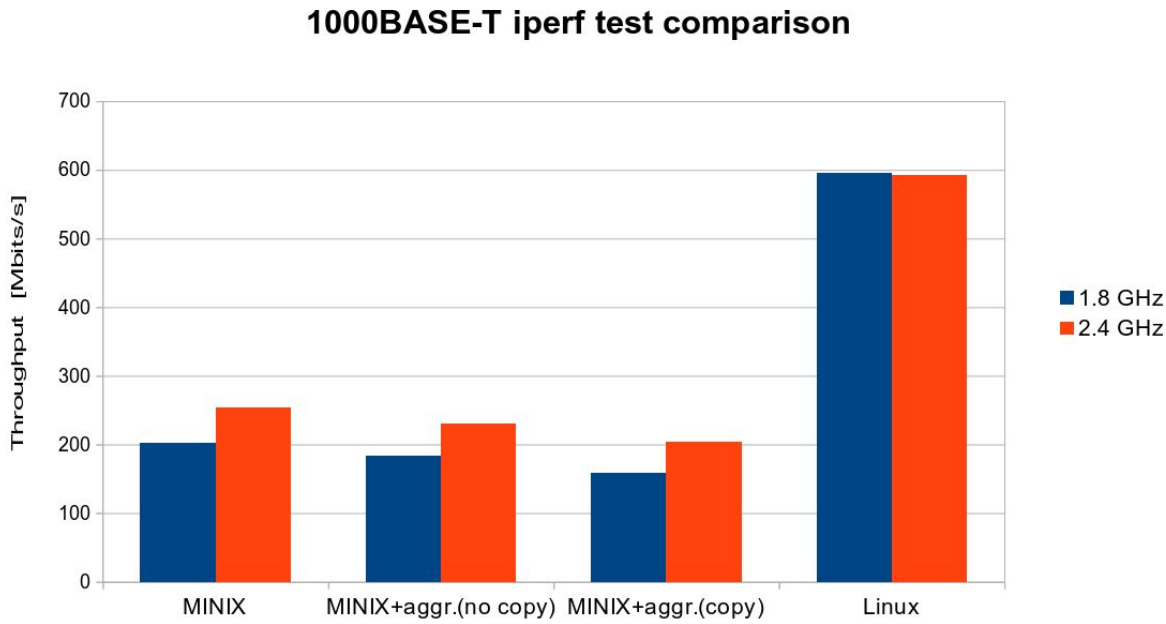

Slika 6.4: rezultati merenja iperf TCP testa na 1000Mbit/s

#### 6.2.2.3 Test performansi 2-2: iperf TCP test na 1000Mbit/s sa bržim procesorom

Pretpostavka je da su rezultati *MINIX*-a znatno lošiji zbog njegove mikrokernel arhitekture, odnosno zbog toga što je procesor znatno više zauzet opsluživanjem komunikacije između komponenata multiserverskog mikrokernel sistema.

Za proveru ove tvrdnju ponovljen je kompletan test 2 ali sa većom radnom frekvencijom procesora (procesor je “overklokovan”). Uporedni rezultati oba eksperimenta se mogu videti na slici 6.5. Uočljivo je da se rezultati *MINIX*-a popravljaju u svim slučajevima kada se frekvencija rada procesora poveća. Sa povećanjem brzine centralnog procesora se smanjuje razlika između performansi *MINIX* 3 sistema i *Linux* sistema, što ukazuje na poznatu činjenicu da *MINIX* 3 (kao i ostali sistemi sa mikrojezgrom) više opterećuje procesor od

sistema sa monolitnim jezgrom.Ovo je indikator da procesorska snaga ograničava *MINIX* 3 i da sa porastom procesorske snage raste i upotrebljivost mikrokernel arhitekture.



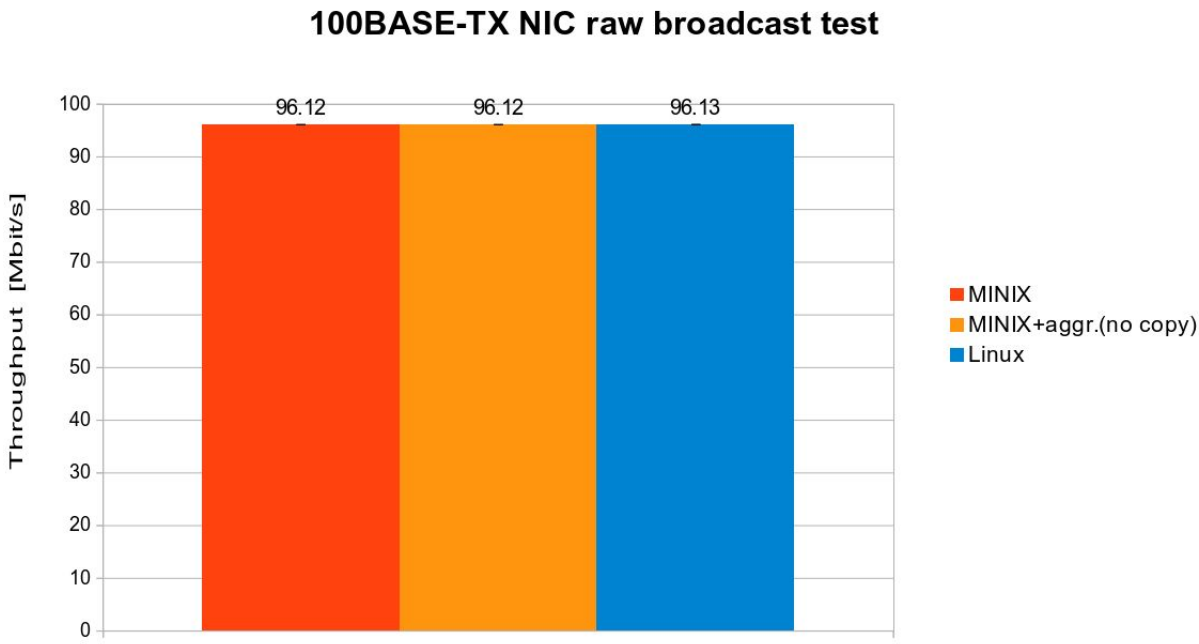

Slika 6.5: Uporedni rezultati merenja iperf TCP testa na 1000Mbit/s na različitim frekvencijama rada procesora.

#### 6.2.2.4 Test performansi 3: Direktni test emisije (“*broadcast*”) na 100Mbit/s

Da bi se izmerile sirove performanse agregacionog drajvera, bez uticaja viših slojeva (kao što su VFS i INET), napravljen je sistemski proces (server) pod nazivom LNTAP (videti dodatak A), koji direktno komunicira sa drajverom. Ovo je bilo neophodno jer na *MINIX*-u jedino sistemski proces može da direktno komunicira sa drajverima. Server na komandu počinje da šalje maksimalnom brzinom pakete maksimalne veličine, koje klijentska mašina broji i meri protok. U merenju protoka prilikom sirove emisije (engl. *raw broadcast throughput*) korišćene su konfiguracije:

a) *Linux* kontrolno testiranje,
b) *MINIX* osnovna konfiguracija i
c) *MINIX* konfiguracija sa instaliranim agregacionim drajverom bez dodatnog kopiranja.

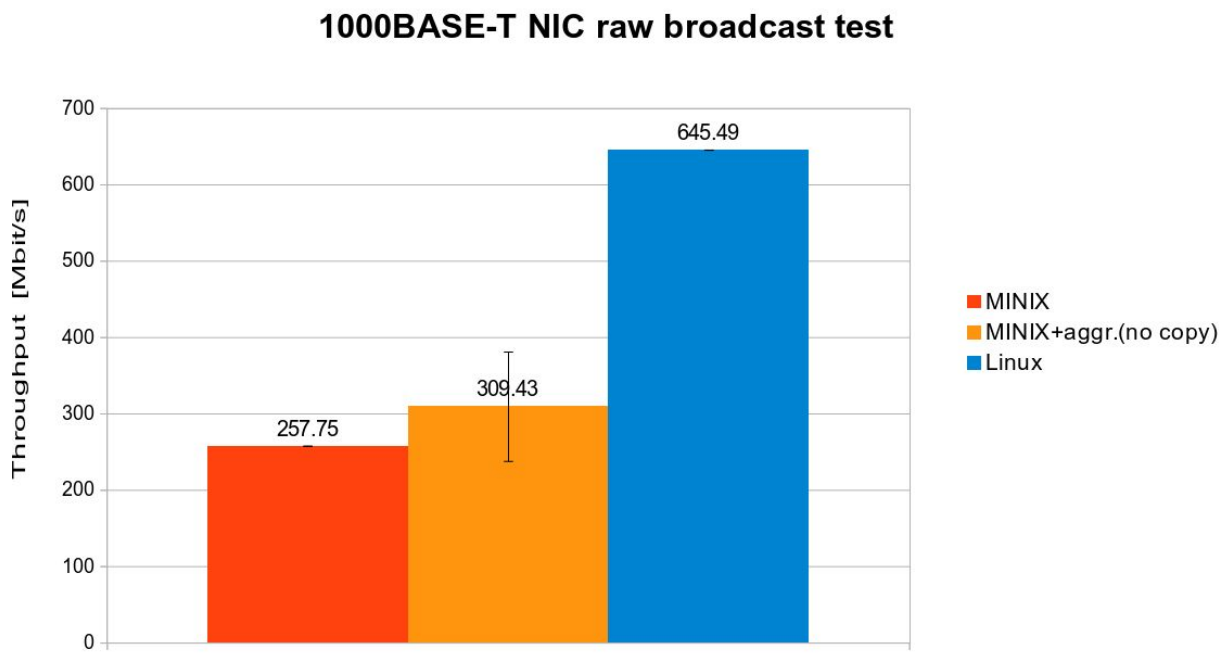
Na slici 6.6 se vidi da je maksimalni protok, izmeren na ovaj način, praktično identičan u sve tri konfiguracije.




Slika 6.6: rezultati merenja sirovog broadcast testa na 100Mbps

### 6.2.2.5 Test performansi 4: Direktni test emisije na 1000Mbit/s

Prethodni test je takođe izvršen i pri brzini od 1000Mbit/s, gde je došla do izražaja razlika u performansama između *Linux* i *MINIX* operativnih sistema. Ono što je posebno zanimljivo u rezultatima ovog testa (slika 6.7) je da je *MINIX* sa agregacionim drajverom postigao bolje rezultate nego osnovni *MINIX*. Odstupanja (engl. *error bar*) prikazana na grafiku prikazuju da dobijeni rezultat nije stabilan, odnosno da dosta varira. Moguće objašnjenje ove pojave je da primena agregacionog drajvera unosi vremensko kašnjenje koji verovatno bolje odgovara sinhronizmu komunikacije mrežnog drajvera sa NIC-om ili se bolje se uklapa u tajminge magistrale na koju je NIC povezan.




Slika 6.7: rezultati merenja sirovog broadcast testa na 1000Mbps

## 6.3 Literatura

[Black2009] Black, Rex (August 2009). Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. Hoboken, NJ: Wiley. ISBN 0-470-40415-9.

[Pillai2013] S. Pillai, IPERF: How to test network Speed,Performance,Bandwidth, 2013. http://www.slashroot.in/iperf-how-test-network-speedperformancebandwidth

[Postel1981a] Postel, J. (ed.), "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.

[Postel1981b] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[SIGST2001] BCS SIGIST (British Computer Society Specialist Interest Group in Software Testing): Standard for Software Component Testing, Working Draft 3.4, 27. April 2001. http://www.testingstandards.co.uk/Component%20Testing.pdf

# 7. Zaključak

Osnovni cilj teze je bio pronalaženje načina da se modularni mikrokernel operativni sistem *MINIX* 3 proširi dodavanjem agregacije mrežnih veza radi poboljšanja oslonjivosti (kroz poboljšanje dostupnosti) mrežnog podsistema, što je u skladu sa ciljem nastanka ovog operativnog sistema. Oslonjivost *MINIX* 3 mrežne konekcije je povećana jer je podržana tolerancija mrežnog podsistema na poremećaj, kao što je npr. prekid mrežnog kabela.

Zahvaljujući modularnosti *MINIX* 3 operativnog sistema, dodavanje agregacije mrežnih veza je napravljeno bez izmena izvornog koda bilo koje od njegovih komponenata. Ugrađeni agregacioni drajver je strukturno pozicioniran između mrežnog modula (INET servera) i mrežnih drajvera. Ovaj novi agregacioni drajver se učitava prilikom pokretanja sistema, tako da je konfiguracioni fajl za mrežu jedini deo sistema koji je morao da bude izmenjen.
Testirane su mrežne performanse *MINIX* 3 sistema sa i bez ovog agregacionog drajvera, da bi se utvrdio njegov uticaj na rad *MINIX* 3 sistema. Utvrđeno je da razlika između fukcionisanja *MINIX* 3 sistema sa i bez agregacionog drajvera zavisi od opterećenja sistema. Tako testovi sa 100MBit/s mrežom pokazuju minimalnu razliku između mrežnog protoka *MINIX* 3 sistema u odnosu na kontrolni *Linux* sistem, dok je razlika znatno veća prilikom testova sa 1000MBit/s mrežom. Sa povećanjem brzine centralnog procesora ta razlika se smanjuje, što ukazuje na poznatu činjenicu da *MINIX* 3 (kao i ostali sistemi sa mikrojezgrom) više opterećuje procesor od sistema sa monolitnim jezgrom.

Agregacioni modul koji je implementiran tokom istraživanja omogućava dve vrste agregacije:
- aktivni - pomoćni i
- kružno prosleđivanje (engl. “*round-robin*”).

Iako oba režima mogu da obezbede otpornost na otkaze, režim aktivni-pomoćni se najčešće koristi u ovu svrhu.

Dalji rad bi mogao da obuhvati proširenje implementacije radi obuhvatanja i drugih metoda agregacije mrežnih veza, kao i ispitivanje njihove efikasnosti. Predmet daljeg rada bi mogao da bude dizajn i implementacija modula koji podržava neku drugu vrstu mrežne komunikacije, kao što je recimo 802.1Q VLAN.
U daljem radu bi se mogle unaprediti osobine mrežnih drajvera *MINIX* 3 operativnog sistema radi bolje podrške agregaciji mrežnih veza. Takođe, Iskustvo implementacija agregacionog drajvera na MINIX3 operativnom sistemu bi moglo da doprinese implementaciji ove i sličnih tehnika na drugim operativnim sistemima sa mikrojezgrom gde ovakve osobine još ne postoje.

# Dodatak A: Implementacija modula za testiranje mrežnog interfejsa

Modul LNTAP je pseudo-mrežni server koji služi kao podloga za testiranje drajvera mrežnog interfejsa na *MINIX* 3 operativnom sistemu. On komunicira direktno sa mrežnim drajverom i omogućava direktno slanje paketa. Razlog za nastanak ovog modula je potreba za testiranjem agregacionog drajvera tokom njegovog razvoja.

## A.1 Implementacija

LNTAP je pisan u formi znakovnog drajvera (koristi `chardriver` frejmvork) jer je predviđeno da se pisanjem na virtuelni znakovni uređaja upravlja radom LNTAP servera.
Sastoji se iz C fajla `lntap.c` koji sadrži:

- main() funkcija koja inicijalizuje SEF (engl. *System Event Framework*) pozivom f-je `sef_local_startup()`, a zatim se izvršava glavnu petlja,odnosno petlja događaja (engl. *event loop*) u kojoj prima i reaguje na poruke
- Funkcije za SEF inicijalizaciju

SEF inicijalizaciju omogućuju funkcije:

- `sef_local_startup()` registruje *callback* funkcije:
  - `sef_cb_init()` za sve vrste inicijalizacije
  - omogućava Live Update (LU) i registruje funkciju `sef_cb_lu_state_save()` za čuvanje stanja prilikom LU

sef_cb_init():

- ukoliko je potrebno, inicijalizuje F2 tipku za aktiviranje testa
- poziva `nic_init_all()` koja inicijalizuje tabele dozvola kao i tabele uređaja
- ispituje tip inicijalizacije (FRESH, LU ili RESTART) i ispisuje poruku o tome
  - ukoliko je LU, vraća se snimljeno stanje i otkazuje se oglašavanje drajvera
- pretplaćuje se na DS događaje `"drv\\.net\\..*"`
- oglašava se drajver pomoću `chardriver_announce()`

### A.1.1 Inicijalizacija parametara za slanje paketa

Drajveru se šalje DL_CONF poruka kako bi se saznala MAC adresa interfejsa. Ova adresa se upisuje u svaki paket koji se šalje.

### A.1.2 Inicijalizacija dozvola

funkcija nic_init_all():

- inicializuje dozvolu `local_txgrant`
- inicijalizuje grantove za sve članove io vektora `local_tx_iovec`

- Inicijalizuje sve članove vektora `devices[]` koji predstavljaju strukture za opis mrežnih drajvera sa kojima se komunicira:

### A.1.3 Petlja za prijem IPC poruka (petlja događaja)

Prijem IPC poruka se vrši unutar beskonačne petlje. IPC poruka se prima pomoću funkcije `sef_receive_status()`. Ukoliko je poruka od VFS-a, reč je o operaciji sa `/dev/lntap` fajlom i poziva se funkcija `chardriver_process()` za obradu. Ako je u pitanju notifikacija, proverava se izvor:

- ukoliko je izvor CLOCK, radi se o tajmeru
- za izvor DS, obrada se vrši f-jom `ds_event()`
- za izvor PM, ne postoji obrada
- za izvor TTY, radi se o pritisku na F2 tipku za aktiviranje *broadcast* testa pozivom funkcuje `bcast_test()`
- inače se radi o nepoznatom izvoru notifikacije

Ako je u pitanju poruka (a nije od VFS-a), onda se radi o poruci od drajvera koja se obrađuje funkcijom `driver_request()`

#### Obrada DS notifikacije

Poziv funkcije `ds_event()` vrši obradu notifikacije od strane DS-a. Obrada se vrši tako što se sa `ds_check()` redom čitaju ključevi, a zatim uzimaju u32 vrednosti za te ključeve. Proverava se da li ključ ima prefiks karakterističan za mrežne drajvere "`drv.net`." i da li je u32 vrednost DS_DRIVER_UP tj. 1. IPC identitet mrežnog drajvera za svaki pojedinačan NIC se pamti kao element vektora u `devices[].drv_ep` promenljivoj.

#### Obrada odgovora drajvera

Funkcija `driver_request()` servisira odgovore drajvera tako što proverava tip poruke:

- DL_CONF_REPLY
- DL_TASK_REPLY
- DL_STAT_REPLY

Trenutno se samo ispisuju poruke o tipu odgovora.

## A.2 Upotreba

LNTAP se učitava na sledeći način:

```
service up ${DIR}/lntap -dev /dev/lntap -config ${DIR}/system.conf
-script ${PWD}/down
```

Kao argument je moguće navesti sa kojim mrežnim drajverom će biti vršena komunikacija, odnosno testiranje. Ako se prilikom startovanja LNTAP servera kao argument navede opcija "`-netdriver=imedrajvera_#instanca`", LNTAP će se vezati samo za navedenu instancu navedenog mrežnog drajvera. U suprotnom će se vezati za prvi mrežni drajver koji nađe.

# A.3 Testiranje mrežnog drajvera

Pod testiranjem mrežnog drajvera se u ovom slučaju podrazumeva slanje maksimalnog broja paketa emisije (engl. *broadcast*) kako bi se opteretili mrežni drajver i interfejs i kako mogao da se izmeri maksimalan mrežni protok.

## A.3.1 Slanje probnog broadcast paketa

Funkcija `bcast_test()` radi sledeće:

- Puni niz `test_pkt` sa 1500 znakova 0xff (zameniti sa ARP paketom?)
- postavlja veličinu prvog iov segmenta
- kreira direktnu dozvolu za čitanje prvog iov segmenta - `test_pkt`
- kreira dozvolu za čitanje `iovec`
- kopira `tx_iovec` u lokalni `copy_tx_iovec`
- kreira indirektnu dozvolu za prvi segment `local_tx_iovec`-a od dozvole koja je bila u `copy_tx_iovec`
- kopira veličinu `copy_tx_iovec`
- kreira direktni grant za `local_tx_iovec`
- funkcijom `asynsend()` šalje DL_WRITEV_S poruku drajveru sa dozvolom za `local_tx_iovec` sa jednim iov segmentom

Slanje ovih paketa se aktivira pritiskom na taster F2 i ponavlja se sve dok funkcionisanje procesa ne bude prekinuto.

# A.4 Linux verzija programa

Verzija programa iste funkcionalnosti je napravljena i za *Linux* pošto je bilo potrebno da se istovetan test napravi i na ovoj platformi radi poređenja performansi. Ova verzija programa je napravljena kao običan korisnički proces koji se pokreće kao i svaki drugi korisnički program. Program komunicira direktno sa drajverom, što se postiže otvaranjem *Linux* mrežnog soketa tipa SOCK_RAW (koji za to i služi). Ovo zahteva da korisnik koji pokreće ovaj program ima privilegije korisnika *root*.
Program se poziva iz komandne linije na sledeći način:

```
# ./netbtest eth0
```

Ovo će da uzrokuje slanje na interfejs eth0 paketa (istovetnih kao LNTAP) dok proces ne bude prekinut. Ovakav korisnički program nije napravljen za *MINIX* 3 jer bi u tom slučaju u vreme slanja ulazilo i vreme komunikacije između komponenti *MINIX* 3 operativnog sistema.

# Dodatak B: Pokretanje *MINIX* 3 operativnog sistema na PC računaru

Kada se x86 PC računar uključi, firmver u ROM-u (BIOS) će pokušati da u memoriju učita prvi sektor sa blok-uređaja za startovanje sistema (engl. *boot device*), kao što su na primer flopi ili hard disk ili CD-ROM, i da ga pokrene. Ako se radi o particionisanom mediju, kao što je hard disk, prvi sektor koji se učitava, naziva se MBR (engl. *Master Boot Record*), a njegov zadatak je da učita prvi sektor sistemske particije (engl. *partition boot record*).
*MINIX* 3 je sadrži izvestan broj programa prebačenih sa *NetBSD* projekta [NetBSD]. U slučaju aktuelne verzije *MINIX*-a (3.3.0), za podizanje sistema se podrazumevano koristi "*NetBSD bootloader*", program za učitavanje i pokretanje operativnih sistema koji zadovoljavaju *Multiboot* specifikaciju [FSF2009] u koje spada i *MINIX*. Prema *Multiboot* protokolu, kernel i moduli se učitavaju na za to određena mesto u memoriji, u skladu sa konfiguracionim podešavanjima sistema i sa unesenim komandama.

| **Naziv** | **Izvorni kod** | **Komentar** |
|---|---|---|
| *Kernel* | `src/minix/kernel` | sadrži SYS i CLOCK taskove |
| *Process Manager server* | `src/minix/servers/pm` | |
| *Virtual File System server* | `src/minix/servers/vfs` | |
| *Resurrection Service* | `src/minix/servers/rs` | |
| *Memory storage driver* | `src/minix/drivers/storage/memory` | sadrži sliku početnog fajl sistema na `/dev/imgrd` |
| *Scheduler server* | `src/minix/servers/sched` | izdvojen iz kernela od verzije 3.1.7 |
| *Terminal driver* | `src/minix/drivers/tty/tty` | |
| *Data Store Server* | `src/minix/servers/ds` | |
| *Minix File System server* | `src/minix/fs/mfs` | za fajl sistem na `/dev/imgrd` |
| *Virtual Memory manager* | `src/minix/servers/vm` | |
| *Pipe File System server* | `src/minix/fs/pfs` | |
| *Init process* | `src/sbin/init` | |

Slika B.1: Tabela sadržaja memorijske slike MINIX 3 operativnog sistema (jezgro i moduli)

Memorijska slika početnog MINIX3 sistema se učitava sa diska u memoriju i pokreće radi dalje inicijalizacije sistema. Sastoji se od jezgra i nekoliko modula, odnosno sistemskih programa, neophodnih za početno funkcionisanje sistema i nastavka inicijalizacije. Sadržaj memorijske slike je dat u tabeli (slika B.1).

Putanja do kernel imidža i modula se nalazi u fajlu `/boot.cfg`. Kernel imidž sa modulima se prema konvenciji nalazi u /boot direktorijumu ili nekom od njegovih poddirektorijuma. Program za startovanje zato mora da bude u stanju da čita dati fajl sistem (u ovom slučaju se radi o *MINIX*-ovom MFS fajl sistemu).

# B.1 Inicijalizacija sistema

Kada se *MINIX* 3 pokrene, prvo se inicijalizuje kernel sa svojim taskovima (CLOCK i SYS) koji se svi izvršavaju u privilegovanom režimu rada procesora. Zatim se inicijalizuju procesi u neprivilegovanom režimu. Na *MINIX*-u postoje dve osnovne klase procesa: sistemski i korisnički. Sistemski procesi imaju posebnu strukturu i ovlašćenja iako se izvršavaju u neprivilegovanom režimu. U sistemske procese spadaju drajveri i serveri dok su korisnički procesi svi ostali. Server RS (engl. *Resurrection Service*) postaje roditelj (engl. *parent*) svih sistemskih procesa, dok će jedini korisnički proces u memorijskoj slici sistema, "*init*", biti predak svih kasnije nastalih korisničkih procesa.
MINIX koristi "`init`" sistem inicijalizacije u stilu BSD-a (engl. *BSD-style init*) [Mewburn2001], što znači da se inicijalizacija obavlja tako što "`init`" proces pokreće inicijalizacioni skript "`/etc/rc`". Ovo se dešava tako što se mali fajl sistem, sadržan unutar "*memory*" drajvera, proglasi za korenski direktorijum ( "/" ), a zatim "*init*" proces pokretanjem komandnog interpretera "*sh*" počinje interpretiranje "*/etc/rc*" skripta.
Pomenuti fajlovi, zajedno sa ostalima neophodnim za pokretanje sistema, nalaze se na malom fajl sistemu ramdiska. Sam ramdisk sadrži nekoliko neophodnih drajvera, komandi i konfiguracionih fajlova, kako bi inicijalizacija mogla da se izvrši.

## B.1.1 Inicijalizacija fajl sistema

Skript "*/etc/rc*" proverava stanje sistema i sistemske particije na disku i vezuje je na korenski ("/") direktorijum. Zatim postavlja vremensku zonu, mapiranje kodnog rasporeda tastature i postavlja vreme sa hardverskog časovnika realnog vremena. Posle toga se proverava i vezuje "*/usr*" particija.

## B.1.2 Multikorisnička inicijalizacija

Skript "*/etc/rc*" poziva skript "`/usr/etc/rc`", koji prazni "`/tmp`" i "`/usr/tmp`" direktorijume, resetuje odnosno obrće logove (pozivom skripta /usr/etc/daily), pokreće "`update`" i "`cron`" demone i inicijalizuje mrežu pokretanjem mrežnih drajvera definisanih u fajlu /etc/inet.conf, kao i mrežni server (trenutno je podržano pozivanje "*inet*" ili "*lwip*" servera, u zavisnosti od parametara kernela). Ukoliko postoji skript "`/etc/rc.net`" za statičku konfiguraciju mrežnih adresa, on se poziva, a ako ovaj skript ne postoji, pokreće se "`dhcpd`" demon.
Na kraju "`/usr/etc/rc`" pokreće sve skriptove koji eventualno postoje u direktorijumu "`/usr/local/etc/rc.d/`". Pojedini programski paketi u taj direktorijum stavljaju skriptove čije im je izvršavanje potrebno prilikom podizanja sistema.

Po završetku se kontrola vraća "`/etc/rc`" skriptu, koji zatim pokreće skript "`/usr/local/etc/rc`" ukoliko postoji, kako bi bila izvršena i host-zavisna inicijalizacija ako je potrebno.

### B.1.3 Inicijalizacija terminala

Kada završi sa izvršavanjem rc skript(ov)a, proces "*`init`*" čita fajl "`/etc/ttys`" (vidi izvorni kod "*init*"-a) i pokreće program "getty" za svaku terminalsku liniju, kako bi korisnicima bio omogućeno prijavljivanje (engl. *login*). Program "`getty`" učitava korisničko ime, pa pokreće program "`login`" koji proverava lozinku i ako je sve u redu, pokreće komandni interpreter (engl. *shell*), čije su ime i putanja nevedeni u fajlu "*`/etc/passwd`*".

### B.1.4 Komandni interpreter (*shell*)

Ukoliko je prijavljivanje bilo uspešno, "`login`" proces izvršava (sistemski poziv "`exec`") školjku tj. komandni interpreter. Na MINIX-u se u ovu svrhu podrazumevano koristi *Almquist shell* odnosno "*`ash`*" [Mascheck2014] pre svega zbog njegove male velične i brzine. Pored interakcije sa korisnikom, komandni interpreter se koristi i za izvršavanje svih sistemskih skriptova prilikom podizanja sistema. Kao i ostali komandni interpreteri, kada je potrebno da se izvrši neki eksterni program, "*`ash`*" stvara proces-dete koji zatim izvrši "`exec`" sistemski poziv. Po završetku datog procesa, "`ash`" kao roditeljski proces preuzima njegov celobrojni izlazni kod (engl. *exit code*).

Prilikom pokretanja od strane "`login`"-a, "*`ash`*" počinje svoj rad tako što izvršava skriptove "`/etc/profile`" i "`$HOME/.profile`", koji se koriste da podese sistemske i korisničke podrazumevane vrednosti pojedinih sistemskih parametara. Na *MINIX*-u podrazumevani "`$HOME/.profile`" skript takođe poziva i "`$HOME/.ashrc`" skript, ukoliko postoji.

## B.2 Literatura

[FSF2009] Free Software Foundation, Multiboot Specification version 0.6.96, 2009. http://www.gnu.org/software/grub/manual/multiboot/multiboot.html

[Mascheck2014] S. Mascheck, Ash (Almquist Shell) Variants, 2014. http://www.in-ulm.de/~mascheck/various/ash/

[Mewburn2001] Luke Mewburn. 2001. The Design and Implementation of the NetBSD rc.d System. In Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, Clem Cole (Ed.). USENIX Association, Berkeley, CA, USA, 69-79.

[NetBSD] The NetBSD Project, https://www.netbsd.org/