Ostvarivanje jednostavna memorijskoga datotečnoga sustava

Vinko, Cerovečki

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:354654

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-04-27**

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Vinko Cerovečki

OSTVARIVANJE JEDNOSTAVNA MEMORIJSKOGA DATOTEČNOGA SUSTAVA

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Vinko Cerovečki

Matični broj: 44087/15–R

Studij: Informacijski sustavi

OSTVARIVANJE JEDNOSTAVNA MEMORIJSKOGA DATOTEČNOGA SUSTAVA

ZAVRŠNI RAD

Mentor:

Luka Milić, mag. ing. comp.

Varaždin, rujan 2018.

Vinko Cerovečki

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Središnja tema ovog završnog rada je, kao što i sami naslov govori, izrada jednostavna memorijskoga datotečnoga sustava. U radu su najprije opisani najpoznatiji i najkorišteniji datotečni sustavi današnjice, a to su NTFS, ext4, FAT i HFS Plus datotečni sustav. Također, napravljena je kratka usporedba navedenih sustava prema ključnim parametrima kao što su maksimalna veličina datoteke, maksimalna veličina diska, te još neki. U sklopu rada napravljen je jednostavni memorijski datotečni sustav u C# programskom jeziku, te primjenski program kojim su testirane sve funkcionalnosti sustava. Nadalje, opisan je izrađeni datotečni sustav, te proces same izrade. Opis procesa izrade sustava obuhvaća sve od odabira tehnologije preko implementacije do izrade primjenskog programa i samog testiranja rada sustava.

Ključne riječi: datotečni sustav; datotečni podsustav; datoteka; disk; sektor;

Sadržaj

1. Uvod 1

2. Datotečni podsustav 3

2.1. NTFS 3

2.2. ext4 datotečni sustav 8

2.3. HFS Plus 12

2.4. Kratka usporedba NTFS, ext, ext2, ext3, ext4, HFS i HFS Plus datotečnih sustava 13

2.5. Zašto baš ovi datotečni sustavi? 14

3. Izrada jednostavna memorijskoga datotečnoga sustava 15

3.1. Datotečni opisnik (FileDescriptor) 16

3.2. Opisnik otvorene datoteke (FileHandle) 16

!Neočekivani kraj formule

3.4. Datotečni sustav (FileSystem) 17

3.5. FileSystemPersistor 18

3.6. UML dijagram razreda 19

3.7. Stvaranje datoteke 20

3.8. Stvaranje kazala 20

3.9. Otvaranje datoteke 20

3.10. Zatvaranje datoteke 20

3.11. Postavljanje položaja datotečnog pokazivača 20

3.12. Doznavanje položaja datotečnog pokazivača 21

3.13. Pisanje u datoteku 21

3.14. Čitanje iz datoteke 21

3.15. Preimenovanje datoteke 22

3.16. Preimenovanje kazala 22

3.17. Premještanje datoteke 22

3.18. Premještanje kazala 23
3.19. Kopiranje datoteke 23
3.20. Kopiranje kazala 23
3.21. Pisanje N bajtova u datoteku 23
3.22. Čitanje N bajtova iz datoteke 24
3.23. Pohranjivanje datotečnog sustava u datoteku 24
3.24. Učitavanje datotečnog sustava iz datoteke 25
3.25. Stvaranje novoga datotečnoga sustava 26
3.26. Brisanje datotečnog sustava 26
4. Organizacija memorije 27
4.1. Usporedba mojeg datotečnog sustava s opisanim datotečnim sustavima 28
5. Opis primjenskog programa 29
6. Zaključak 33
Popis literature 34
Popis slika 35
Popis tablica 36

1. Uvod

Datoteke su jedan od osnovnih pojmova s kojima se danas susreću svi ljudi koji se služe računalima. Datoteke se rabe u svim složenijim operacijskim sustavima za trajnu pohranu podataka.

Većina programa instaliranih na računalu za normalan rad osim instrukcija trebaju i određene podatke. Ti podaci se mogu pohranjivati u određenom dijelu zajedno sa samim programom te s programom činiti jednu cjelinu. No, takav način pohranjivanja podataka nije trajan, već su podaci pohranjeni tako dugo dok se program izvršava. Za trajno pohranjivanje podataka potrebne su dodatne zasebne cjeline koje se nazivaju datoteke.

Kao fizičko spremište podataka obično se rabe diskovi. S gledišta korisnika računala i programa koje korisnik rabi, datoteke su osnovne jedinice podataka pohranjene na disku. U datoteke se pohranjuju korisnički programi, audio zapisi, slike, tekst, programi operacijskih sustava…

Na računalima postoje posebni podsustavi operacijskih sustava čija je zadaća isključivo organizacija datoteka i upravljanje datotečnim sustavima. U sklopu svojih zadaća datotečni podsustavi omogućavaju rabljenje dostupnih datotečnih sustava. Danas postoji mnogo različitih datotečnih sustava, a najpoznatiji od njih su datotečni sustavi najpoznatijih operacijskih sustava Windows, te Linux. Windows operacijski sustavi rabe NTFS (eng. *New Technology File System*, Datotečni sustav nove tehnologije) i FAT (eng. *File Allocation Table*, Tablica razmještaja datoteka) datotečne sustave, dok Linux rabi ext (eng. *fourth extended file system*, Četvrta generacija proširenog datotečnog sustava) datotečne sustave. Postoji više inačica ext datotečnih sustava koje se u određenoj mjeri razlikuju po nekim obilježjima, pa tako razlikujemo ext, ext2, ext3, te ext4 datotečne sustave. Više riječi o suvremenim datotečnim sustavima će biti u kasnijim poglavljima.

Za svoj završni rad odabrao sam upravo ovu temu jer pokriva područje informatike koje nisam tako detaljno uspio obraditi tokom dosadašnjeg studiranja, a smatram da bih sa završenim preddiplomskim studijem informacijskih sustava trebao baratati tim znanjima.

Rad se sastoji od 6 poglavlja i pripadajućih podpoglavlja. Prvo poglavlje je trenutačno, uvod. U drugom poglavlju su opisani datotečni sustavi općenito i neki danas najkorišteniji i najpoznatiji datotečni sustavi. Treće poglavlje opisuje postupak izrade i izgled mog datotečnog sustava. U četvrtom poglavlju opisana je organizacija memorije u mom datotečnom sustavu, te je napisana kratka usporedba mog datotečnog sustava sa sustavima opisanim u drugom

poglavlju. Peto poglavlje obuhvaća opis primjenskog programa kojim je testiran rad sustava. I konačno, u šestom poglavlju je iznesen zaključak.

2. Datotečni podsustav

U ovom poglavlju opisat ću datotečne sustave i njihovu svrhu kao sastavnih dijelova operacijskih sustava, kako bi ostatak rada bio što jasniji.

Računala sama po sebi ne bi imala nekog smisla da ne postoje programi koji su instalirani na računala i koji se na računalima mogu pokretati i izvršavati. Većina današnjih programa kao izvor podataka za obradu i spremište za obrađene podatke rabi datoteke, a i sami programi su također pohranjeni u datotekama. Osim korisničkih podataka i programa, važni su i sustavski programi i podaci koji su također pohranjeni u datotekama, te rabe datoteke kao izvor i spremište podataka. S obzirom na to da operacijski sustavi upravljaju svim elementima računala, rad operacijskih sustava organiziran je na način da za svaki element sustava postoji različiti podsustav. Tako razlikujemo podsustave za upravljanje ulazno-izlaznim jedinicama, programima, sigurnošću, korisničko sučelje, spremnikom, te nama važne podsustave za upravljanje datotekama, datotečne podsustave.

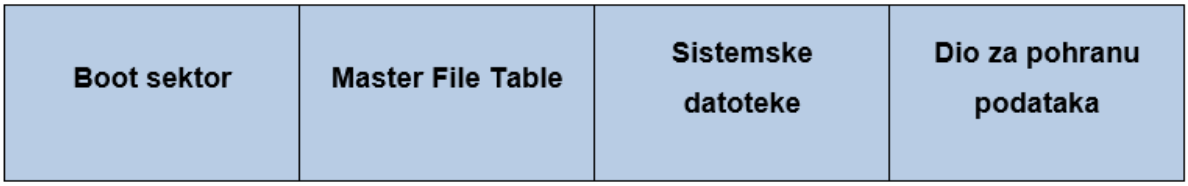
Sada kada znamo kolika je važnost datoteka na računalu, možemo shvatiti zašto su upravljanje i omogućavanje porabe datoteka na računalu zadaće zasebnog podsustava čije su to i jedine zadaće.

2.1. NTFS

Jedan od najpoznatijih i najrabljenijih datotečnih sustava današnjice je NTFS. NTFS je datotečni sustav razvijen u Microsoftu i to kao zadani datotečni sustav Windows NT operacijskih sustava. Prva verzija Windowsa koja je rabila NTFS bila je Windows NT 3.1, te ga ranije verzije Windowsa ne mogu rabiti. Sredinom osamdesetih godina prošlog stoljeća Microsoft i IBM (eng. *International Business Machines*) radili su na zajedničkom projektu stvaranja nove generacije grafičkih operacijskih sustava, a rezultat njihovog zajedničkog rada bio je OS/2 operacijski sustav koji je rabio HPFS (eng. *High Performance File System*, Datotečni sustav visokih performansi) datotečni sustav. Međutim, IBM i Microsoft nisu se slagali u mnogo ključnih pitanja, pa je OS/2 ostao IBM-ov projekt, a Microsoft se bacio na razvoj sustava Windows NT koji, kao što smo rekli, rabi NTFS datotečni sustav. No, to nije bio jedini uzrok nastajanja NTFS-a, već je jedan od uzroka bila i ovisnost Windows operacijskih sustava o postojećem FAT datotečnom sustavu koji je pak imao loše značajke za pohranu datoteka i upravljanje njima.

Iako je nudio mnogo bolje značajke od dotadašnjeg FAT datotečnog sustava, NTFS je imao i nekih nepotrebnih značajki zbog kojih su u Microsoftu i dalje razvijane posebne verzije Windowsa koje su još uvijek podržavale FAT datotečni sustav. Neke od tih značajki su mogućnost kreiranja datoteka većih od 4 GB, te upravljanje diskovima velikih kapaciteta, dok su osobna računala imala diskove manje od 1 GB.[1]

NTFS u radu rabi B stabla za indeksiranje podataka, što u velikoj većini slučajeva osigurava bržu pretragu podataka. Strukturu NTFS-a čine boot sektor i MFT (eng. *Master File Table*, Glavna tablica datoteka) koji su odmah na početku, a tek nakon njih slijede ostale sistemske datoteke i dio za pohranu podataka. Slika 1. prikazuje strukturu diska kod NTFS datotečnog sustava prethodno opisanu.[1]



Slika 1. Struktura NTFS-a

Prilikom NTFS formatiranja prvih 16 sektora se alocira za datoteku u kojoj se nalaze metapodaci za podizanje sustava. Prvi sektor je zapravo „bootstrap“ kod, a sljedećih 15 sektora čini program za učitavanje, odnosno IPL (Initial Program Loader). U svrhu osiguranja pouzdanosti datotečnog sustava, u zadnjem sektoru NTFS particije se nalazi kopija boot sektora.

Tablica 1. prikazuje strukturu boot sektora.[1]

Tablica 1. NTFS Boot sektor

Byte Offset	Filed Length	Field Name
0x00	3 bytes	Jump Instruction
0x03	LONGLONG	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	End of Sector Marker

Svaka datoteka na disku u MFT tablici predstavljena je kao jedan zapis. Prvih 16 zapisa MFT-a rezervirani su za posebne informacije. Prvi zapis MFT tablice opisuje samu tablicu, a slijedi ga MFT zrcalni zapis. MFT zrcalni zapis rabi se ako je prvi zapis MFT tablice oštećen. Naime, ako je prvi zapis MFT tablice oštećen, NTFS čita drugi zapis kako bi pronašao MFT zrcalni zapis čiji prvi zapis je jednak prvom zapisu MFT tablice. Lokacije podatkovnih sektora MTF i MFT zrcalne datoteke su zapisane u boot sektoru. MFT za svaku datoteku dodjeljuje određenu količinu memorijskog prostora. U dodijeljeni prostor se zapisuju atributi datoteke, a male datoteke ili kazala (najčešće 512B ili manje) se mogu i čitavi nalaziti u zapisu MFT tablice. Takav način rada čini pristup datotekama izrazito brzim, za razliku od FAT datotečnog sustava. *Slika 2.* prikazuje primjer u kojem je mala datoteka zapisana u jednom zapisu MFT tablice.[1]

Slika 2. MTF zapis male datoteke ili kazala

Metapodaci su podaci koji zapravo opisuju svojstva nekih drugih podataka. U NTFS se metapodaci automatski generiraju i pohranjuju na početku NTFS particije. MTF tablica je po svojim obilježjima vrlo slična relacijskim bazama podataka. Svako kreiranje datoteke ili kazala na računalu rezultira između ostalog i unosom vrijednosti atributa u MTF tablicu. Maksimalna količina podataka koja se može pohraniti u MFT određena je veličinom klastera. Kao što smo već rekli, male datoteke se sa svrhom uštede prostora na disku spremaju direktno u MTF tablicu. [1]

Tablica 2. prikazuje prvih 16 zapisa MFT tablice.

Tablica 2. Prvih 16 zapisa u MTF tablici

Datoteka	Naziv datoteke	Redni broj MTF zapisa	Opis
$Mft	Master File Table	0	$Mft datoteka pohranjuje zapis svih datoteka i kazala u sustavu.
$MftMirr	MFT zrcalni zapis	1	Zrcalni zapis MFT-a koji se rabi ako je prvi sektor MFT-a oštećen.
$LogFile	Log File	2	Pohranjuje informacije koje se rabe za ubrzavanje oporavka.

$Volume	Volume	3	Pohranjuje informacije o NTFS particiji (naziv, verzija…).
$AttrDef	Attribute Deffinitions	4	Sadrži imena atributa.
.	Root file name indeks	5	Korijensko kazalo.
$Bitmap	Cluster bitmap	6	Pohranjuje podatke o neporabljenim klasterima.
$Boot	Boot sector.	7	Pohranjuje adresu BIOS Paramter Block-a, koji pak služi za postavljanje particije.
$BadClus	Bad cluster file	8	Pohranjuje klastere particije koji imaju grešku.
$Secure	Security file	9	Pohranjuje sigurnosne deskriptore za sve datoteke u volumenu.
--	--	12-15	Sektori rezervirani za novije verzije.

Unutar MFT-a postoje stalni i nestalni, odnosno dodatni atributi. Vrijednosti stalnih atributa pohranjene su u MFT zapisu, a vrijednosti nestalnih atributa pohranjuju se u dodatnom zapisu u MFT-u ili čak izvan MFT-a. Do pojave nestalnih atributa dolazi jer s većim podacima dolazi do potrebe za pohranjivanjem više atributa, pa datotečni sustav neke atribute izbacuje iz zapisa i oni postaju nestalni.

Tablica 3. sadrži popis atributa koji se pohranjuju u MFT.

Tablica 3. Popis i opis atributa koji se pohranjuju u MFT

Atribut	Opis
Standard Information	Sadrži informacije poput vremenske oznake i brojač poveznica.
Attribute List	Sadrži lokacije svih zapisa atributa koji nisu u MFT (nestalni atributi).
File Name	U skraćenom obliku sadrži 8 znakova za ime i 3 za ekstenziju, a u produljenom do 255 Unicode znakova.
Security Descriptor	Sadrži podatke o vlasniku datoteke, te pravima pristupa.
Data	Sadrži podatke o datoteci. Data atributa može biti više.
Object ID	Jedinstveni identifikator datoteke u particiji. Nemaju ga sve datoteke.
Logged Utility Stream	Sličan kao i tok podataka.
Reparse Point	Rabi se kod montiranih diskova.

Index Root	Rabi se za implementaciju kazala i ostalih indeksa.
Index Allocation	Rabi se za implementaciju kazala i ostalih indeksa.
Bitmap	Rabi se za implementaciju kazala i ostalih indeksa.
Volume Information	Pohranjuje verziju particije.
Volume Name	Pohranjuje ime particije.

Prva verzija NTFS-a imala je implementirano 5 osnovnih dozvola. Novije verzije NTFS-a imaju mnogo dodatnih razina dozvola, ali početnih 5 je zadržano.

Tablica 4. prikazuje popis osnovnih dozvola koje je imala prva verzija NTFS-a.

Tablica 4. Osnovne dozvole u NTFS-u [2]

Dozvola	Opis
Full Control	Korisniku je dozvoljeno uređivanje, dodavanje, brisanje i premještanje datoteka i kazala, te promijene svojstava.
Modify	Korisniku je dozvoljeno dodavanje, brisanje, pregledavanje i uređivanje datoteka, te njihovih svojstava.
Read & Execute	Korisniku je dozvoljeno pregledavanje i pokretanje programskih datoteka.
Read	Korisniku je dozvoljeno pregledavanje sadržaja datoteka, te svojstava datoteka.
Write	Korisniku je dozvoljeno pisanje sadržaja u datoteke.

Osim osnovnih dozvola dostupnih od prve verzije NTFS datotečnog sustava, kasnije su implementirane i dodatne dozvole prikazane u tablici ispod.

Tablica 5. Popis dodatnih dozvola u NTFS-u [2]

Dozvola	Opis
Traverse Folder/Execute File	Korisniku je dozvoljeno pregledavanje različitih kazala, podkazala i datoteka u njima.
List Folder/Read Data	Korisniku je dozvoljeno pregledavanje liste sadržaja nekog kazala i datoteka.
Read Attributes	Korisniku je dozvoljeno pregledavanje atributa datoteka ili kazala.
Read Extended Attributes	Korisniku je dozvoljeno pregledavanje proširenih atributa datoteka ili kazala.
Create Files/Write Data	Korisniku je dozvoljeno stvaranje datoteka unutar točno određenog kazala, te mu je dozvoljeno mijenjanje i pisanje preko postojećeg teksta u točno određenoj datoteci.
Write Attributes	Korisniku je dozvoljeno mijenjanje atributa datoteka ili kazala.
Write Extended Attributes	Korisniku je dozvoljeno mijenjanje proširenih atributa datoteka ili kazala.
Delete	Korisniku je dozvoljeno brisanje datoteka ili kazala.

Read Permissions	Korisniku je dozvoljeno pregledavanje dozvola kao što su Full Control, Read i Write nad određenom datotekom ili kazalom.
Change Permissions	Korisniku je dozvoljeno mijenjanje dozvola nad datotekama ili kazalima.
Take Ownership	Korisniku je dozvoljeno preuzimanje vlasništva nad određenom datotekom ili kazalom.

Svi datotečni sustavi koje rabe Windows operacijski sustavi, pa tako i NTFS organiziraju disk na način da ga podijele na klastere. Klasteri predstavljaju najmanje jedinice diska na koje se može pohraniti datoteka. Kod pohranjivanja datoteke na disk može doći do slučaja da neki klaster ostane djelomično neporabljen, ali se taj neporabljeni dio svejedno ne može više rabiti za pohranu. Prema nekim podacima, veličina neporabljenog prostora na današnjim tvrdim diskovima može se izračunati prema formuli (1).

$$\frac{veličina\ klastera}{2 * broj\ datoteka}. \quad (1)$$

Kod verzije 4.0 i novijih verzija Windows NT operacijskih sustava koji rabe NTFS datotečni sustav, zadana maksimalna veličina klastera je 4 kilobajta i to zato jer kod klastera veće veličine kompresija datoteka ne bi bila moguća. Korisnik ipak može promijeniti tu postavku. Slijedeća tablica prikazuje neke verzije Windows operacijskih sustava i veličine klastera kod tih verzija.

Tablica 6. Zadane veličine klastera kod NTFS-a

Veličina particije	Windows NT 3.51	Windows NT 4.0	Windows XP	Windows Vista	Windows 7
7 – 512 MB	512 bajtova	4 KB	4 KB	4 KB	4 KB
512 MB – 1 GB	1 KB	4 KB	4 KB	4 KB	4 KB
1 – 2 GB	2 KB	4 KB	4 KB	4 KB	4 KB
2 GB – 2 TB	4 KB	4 KB	4 KB	4 KB	4 KB
2 – 16 TB	-	-	4 KB	4 KB	4 KB
16 – 32 TB	-	-	8 KB	8 KB	8 KB
32 – 64 TB	-	-	16 KB	16 KB	16 KB
64 – 128 TB	-	-	32 KB	32 KB	32 KB
128 – 256 TB	-	-	64 KB	64 KB	64 KB
> 256 TB	-	-	-	-	-

2.2. ext4 datotečni sustav

ext je prvi datotečni sustav dizajniran posebno za Linux jezgru. ext je nastao 1992. godine, a napravio ga je francuski softver inženjer Remy Card. ext je nastao kako bi se uklonili

nedostaci dotadašnjeg MINIX datotečnog sustava. Jedna od osnovnih prednosti ext datotečnog sustava nad MINIX-om je mogućnost adresiranja do 2 GB memorijskog prostora, što kod MINIX-a nije bilo moguće.[3]

Međutim, ext zbog svojih nedostataka, od kojih je najveći to što je postojala samo jedna vremenska oznaka po datoteci, nije dugo čekao nasljednika. Već godinu dana nakon nastajanja ext datotečnog sustava, Remy Card dizajnirao je ext2 kao zamjenu za ext koja je pokrivala dotadašnje nedostatke. ext2 podržavao je datoteke veličina do 2 TB, pa je tako veličina datoteke ovisila samo o veličini sektora i arhitekturi računala. Također, najveća moguća veličina particije ovisila je samo o veličini sektora i arhitekturi računala. ext2 bilo je moguće rabiti čak i s MacOS (eng Macintosh Operating System, Mekintoš operacijski sustav) i Windows operacijskim sustavima. No, i kod ext2 datotečnog sustava bili su prisutni tipični nedostaci datotečnih sustava iz devedesetih godina prošlog stoljeća. Naime, kod većine tadašnjih datotečnih sustava dolazilo je do velikih problema ukoliko bi se sustav srušio ili izgubio napajanje u trenutku u kojem su se podaci zapisivali na disk. Tako bi u slučaju iznenadnog prestanka napajanja električnom energijom dok se zapisuju podaci na disk dolazilo do gubitka ili oštećenja dijelova datoteka koje su se u tom trenutku spremale na disk. Također, veliki nedostatak koji je opisivao gotovo sve tadašnje datotečne sustave bio je gubitak performansi zbog fragmentacije. No, bez obzira na spomenute nedostatke, ext2 se u pojedinim slučajevima rabi i danas, najčešće kod formatiranja USB (eng Universal Serial Bus, Univerzalna serijska sabirnica) memorija. [3]

Na ext2 datotečnom sustavu radio je Stephen Tweedie, te je 1998. godine najavio da je napravio značajna poboljšanja. 3 godine kasnije, 2001. godine, nastao je ext3 datotečni sustav. Nedostatak gubitka i oštećenja podataka u slučaju iznenadnog prestanka napajanja kod ext2 datotečnog sustava, u ext3 datotečnom sustavu riješen je uvođenjem dnevnika. Naime, većina datotečnih sustava nastalih u kasnim devedesetim godinama prošlog stoljeća, između ostalih i NTFS, imali su sustav vođenja dnevnika transakcija u koji se bilježilo sve što se radilo sa, po datotečni sustav važnim, podatkovnim strukturama. Ako je operacija zapisivanja na disk uspješno završila, podaci iz dnevnika se samo prenose u datotečni sustav. Ako pak transakcija podataka nije bila uspješna, kod ponovnog podizanja sustava obrađuje se nedovršena transakcija. Kod takvog načina rada podaci čije zapisivanje nije uspješno dovršeno još uvijek mogu biti izgubljeni, ali sami datotečni sustav ostaje neoštećen. [3]

Kod ext3 datotečnog sustava postoje 3 načina vođenja dnevnika transakcija, *journal*, *ordered* i *writeback* način. [3]

Journal je način kod kojeg postoji najmanji rizik. Metapodaci i sadržaj datoteke najprije se zapisuju u dnevnik i tek kada završi njihovo zapisivanje u dnevnik, dodaju se u datotečni

sustav čime se osigurava konzistentnost datoteke koja se zapisuje, te konzistentnost samog datotečnog sustava. Međutim, nedostatak ovog načina zapisivanja na disk je mogućnost da dođe do značajnog smanjivanja performansi računala zbog porabe računalnih resursa kod zapisivanja na disk. [3]

Redoslijedni (*ordered*) način rada, koji je najčešće rabljen u Linux operacijskim sustavima manje rabi resurse računala, ali kod njega postoji veći rizik od gubitka podataka. Spomenuti način funkcionira tako da se najprije u dnevnik zapisuju metapodaci, a podaci se zapisuju direktno u datotečni sustav. Tek kada su podaci uspješno zapisani u datotečni sustav dodaju im se i metapodaci. Takav način rada u slučaju prestanka napajanja osigurava da metapodaci ostanu pohranjeni u dnevniku, te datotečni sustav može „očistiti“ oštećene podatke. Kod takvog načina rada prilikom pada sustava može doći do oštećenja datoteke koja se u trenutku pada sustava zapisuje na disk, ali sami datotečni sustav je siguran isto kao i datoteke s kojima se u trenutku pada ništa ne radi. [3]

Treći način vođenja dnevnika, *writeback*, način je koji nosi najveći rizik. U tom načinu rada ne postoji redoslijed zapisivanja metapodataka i podataka. Naime, u svrhu poboljšanja performansi operacijskog sustava metapodaci i podaci zapisuju se u redoslijedu koji je najbolji za što bolje performanse operacijskog sustava. Time se mogu postići značajna poboljšanja u performansama računala, ali postoji puno veći rizik od gubitka ili oštećenja podataka. Datotečni sustav sam po sebi je još uvijek siguran, ali datoteka koja se zapisuje u trenutku pada sustava može biti u potpunosti oštećena ili podaci mogu biti nepovratno izgubljeni. [3]

Kao što se može vidjeti u gore opisanim načinima vođenja dnevnika transakcija kod ext3 datotečnog sustava uklonjena je bilo kakva mogućnost oštećenja samog datotečnog sustava do kojeg bi u ext2 datotečnom sustavu došlo prilikom pada sustava ili iznenadnog prestanka napajanja. Također, ovisno o načinu koji se primjenjuje moguće je da čak ne dođe do oštećenja podataka koji se zapisuju na disk u trenutku pada sustava, ali važno je naglasiti da kod pojedinih načina vođenja dnevnika može doći do značajnijih pogoršanja u performansama računala. [3]

Konačno, 2006. godine Theodore Ts'o, jedan od razvojnih programera etx3 datotečnog sustava najavio je dolazak ext4 datotečnog sustava. Prva stabilna verzija ext4 datotečnog sustava dolazi dvije godine kasnije zajedno s jezgrom 2.6.28. [3]

S obzirom na to da dolazi kao nadogradnja za ext3 datotečni sustav, ext4 je osmišljen tako da bude kompatibilan koliko je god to moguće sa ext3 datotečnim sustavom. ext4 je napravljen tako da ne samo da omogućava ext3 datotečnom sustavu nadogradnju na ext4, već i omogućava ext4 datotečnom sustavu da automatski ugrađuje ext3 datotečne sustave u

ext3 modu i time se uklanja potreba za odvojenim održavanjem tih dvaju datotečnih sustava. [3]

Dok je ext3 datotečni sustav rabio 32 – bitno adresiranje što je omogućavalo pohranu datoteka do 2 TiB, ext4 rabi 48 – bitno adresiranje koje omogućava pohranu datoteka veličine do 16 TiB. [3]

Značajna poboljšanja koja donosi ext4 tiču se alokacije blokova za pohranu. Tim poboljšanjima dolazi do značajnih poboljšanja performansi kod čitanja i pisanja na disk. Naime, kod datoteka s velikim brojem blokova evidentiranje svakog bloka nije efikasan način. ext4 donosi rješenje za taj problem u vidu rabljenja *extent*-ova. *Extent* je skup uzastopnih fizičkih blokova u koje se mogu zapisivati podaci, pa je tako nepotrebno „pamtiti“ sve blokove na koje je neka datoteka spremljena, već se ona sprema na uzastopne blokove. [3]

Kod alokacije memorijskog prostora za pohranjivanje datoteke većina datotečnih sustava mora alociranje blokove popuniti nulama, ali ext4 rabi `fallocate()` sistemski poziv. `fallocate()` je posebni Linux sistemski poziv koji osigurava memorijski prostor za spremanje datoteka. `fallocate()` alocira prostor na disku i to prema specificiranom ofsetu i duljini. Ako dođe do toga da zbroj ofseta i duljine bude veći od veličine datoteke, veličina datoteke se mijenja. Blokovi obuhvaćeni ofsetom i duljinom koji ne pohranjuju nikakve podatke inicijaliziraju se i postavljaju na 0. Nakon uspješnog poziva svim zapisima koji se naknadno dodaju osiguran je memorijski prostor i ne mogu „propasti“ zbog nedovoljno prostora na disku. Dok ext3 datotečni sustav alocira memorijski prostor na način da automatski dodjeljuje blokove bez obzira jesu li svi podaci pristigli u predmemoriju, ext4 rabi odgođenu alokaciju kojom se omogućava da alokacija prostora počne tek kad su podaci spremni za zapisivanje. Time se omogućuje bolje izdvajanje blokova za pohranu datoteka, što smanjuje fragmentaciju, a samim time poboljšava performanse računala. [3]

Theodore Ts'o ext4 datotečni sustav opisuje kao tehnologiju koja je premostila nedostatke ext3 datotečnog sustava, ali se još uvijek oslanja na staru tehnologiju. Također, prema njegovom mišljenju, ext4 može zamijeniti tek datotečni sustav nove generacije. [3]

Tablica 7. prikazuje usporedbu ext datotečnog sustava i svih njegovih nasljednika.

Tablica 7. Usporedba ext datotečnih sustava

	ext	ext2	ext3	ext4
Maksimalna veličina particije	2 GiB	32 TiB	32 TiB	1 EiB

Maksimalna duljina naziva datoteke	255	255	255	255
Vođenje dnevnika transakcija	NE	NE	DA	DA
Maksimalna veličina datoteke	2 GiB	2 TiB	2 TiB	16 TiB

Kao što smo naveli u opisu samih sustava gore, u tablici možemo vidjeti da se maksimalna veličina prostora koji se mogao adresirati povećavala svakom novom generacijom ext datotečnog sustava. Također, maksimalna veličina datoteke se isto povećavala, dok je maksimalna duljina naziva datoteka već kod ext datotečnog sustava bila 255 znakova. ext i ext2 datotečni sustavi nisu imali implementirano vođenje dnevnika transakcija podataka na disk i zbog toga su se kod njih javljali problemi ukoliko bi u trenutku pohranjivanja datoteke na disk došlo do pada sistema ili prestanka napajanja, dok je kod ext3 i ext4 datotečnih sustava taj problem riješen uvođenjem dnevnika. [5]

2.3. HFS Plus

HFS Plus (eng. *Hierarhical File System Plus*, Hijerarhijski datotečni sustav plus) je datotečni sustav razvijen u kompaniji Apple. Dolazi uz Mac OS 8.1 i novije verzije, te Mac OS X operacijske sustave. HFS+ nastao je 1998. godine kao zamjena za HSF (eng. *Hierarhical File System*, Hijerarhijski datotečni sustav), dotadašnji primarni datotečni sustav Macintosh računala i iPod playera za reprodukciju audio datoteka.

U odnosu na HSF, HSF+ donosi mogućnost alokacije većeg broja blokova na disku, te pohranu datoteka manje minimalne veličine nego što je to bio slučaj kod HSF datotečnog sustava, te se tako optimizira smještajni kapacitet velikih tvrdih diskova. HSF datotečni sustav rabio je 16 – bitnu tablicu razmještaja datoteka, a to je uključivalo ozbiljna ograničenja koja su se ticala broja alociranih sektora. Naime, HFS ne podržava više od 65536 sektora. To ograničenje prije nije predstavljalo neki problem. No, s pojavom diskova većeg kapaciteta, to ograničenje došlo je do izražaja. Uzmemo li za primjer datoteku veličine 1 bajta, na disku veličine 1 GB, ta datoteka zauzimala bi 16 KB prostora. Opisani problem je jedan od važnijih nedostataka zbog kojeg je došlo do potrebe za efikasnijim i boljim datotečnim sustavom.

HFS Plus između ostalog omogućuje i upotrebu do 255 znakova za nazive datoteka i kazala što kod HFS-a nije bilo moguće. Za razliku od HFF-a, HFS Plus rabi 32 – bitnu

datotečnu tablicu, čime se doskočilo problemu ograničenog maksimalnog broja sektora. Za pohranjivanje metapodataka particije HSF Plus rabi binarna stabla. U verziji HSF Plus iz 2002. godine postojala je mogućnost da korisnik sam isključi automatsko vođenje dnevnika transakcija. Isključivanjem vođenja tog dnevnika koji je opisan u poglavlju o ext datotečnim sustavima mogu se značajno poboljšati performanse računala. Međutim, isključivanjem te opcije javlja se rizik od oštećenja ili potpunog gubitka podataka prilikom čijeg spremanja dođe do iznenadnog prestanka napajanja ili pada sustava. Stoga sve verzija HSF Plusa od 2003. godine nadalje imaju zadanu opciju vođenja dnevnika transakcija. [4, str. 318]

Tablica 8. prikazuje usporedbu HFS i HFS+ datotečnih sustava.

Tablica 8. Usporedba HFS i HFS Plus datotečnih sustava

	HFS	HFS Plus
Maksimalna veličina particije	2 TiB	~ 8 EiB
Maksimalna duljina naziva datoteke	31	255 UTF 16 znakova
Vođenje dnevnika transakcija	NE	DA
Maksimalna veličina datoteke	2 GiB	~ 8 EiB

2.4. Kratka usporedba NTFS, ext, ext2, ext3, ext4, HFS i HFS Plus datotečnih sustava

Slijedeća tablica prikazuje usporedbu svih datotečnih sustava koje sam kroz prethodnih nekoliko odlomaka opisao. Pri uspoređivanju tih datotečnih sustavu treba svakako uzeti u obzir činjenicu da svi ti datotečni sustavi nisu nastali u isto vrijeme. Tako neki datotečni sustavi imaju nedostataka u odnosu na ostale jer u vrijeme njihova nastanka tehnologija nije bila na istom stupnju razvoja, a i nisu bili otkriveni neki osnovni problemi datotečnih sustava koji su se s daljnjim razvojem otklanjali. Najbolji primjer takvog problema je oštećenje ili gubitak podataka ili čak oštećenje datotečnog sustava prilikom pada sustava u trenutku zapisivanja datoteke na disk.

Tablica 9. Usporedba NTFS, ext, ext2, ext3, ext4, HFS i HFS+ datotečnih sustava

	NTFS	ext	ext2	ext3	ext4	HFS	HFS Plus
Maksimalna veličina particije	16 EiB	2 GiB	32 TiB	32 TiB	1 EiB	2 TiB	8 EiB
Maksimalna duljina naziva datoteke	255	255	255	255	255	31	255
Vođenje dnevnika transakcija	DA	NE	NE	DA	DA	NE	DA
Maksimalna veličina datoteke	16 EiB	2 GiB	2 TiB	2 TiB	16 TiB	2 GiB	~ 8 EiB

2.5. Zašto baš ovi datotečni sustavi?

U prethodnim podpoglavljima opisao sam NTFS, ext, HFS i HFS Plus datotečne sustave. Razlog odabira NTFS i ext datotečne sustave je u činjenici da su ti datotečni sustavi jedni od najpoznatijih i najkorištenijih datotečnih sustava. HFS i HFS Plus datotečni sustavi mi nisu bili toliko poznati. No, s obzirom na to da se koriste u Mac operacijskim sustavima, odlučio sam se baš za njih tako da mogu napraviti usporedbu što više datotečnih sustava koji se koriste u različitim operacijskim sustavima. Svakako, jedan od razloga za odabir HFS i HFS Plus datotečnih je upravo taj što mi nisu bili poznati, pa sam tako imao prilike u njih proučiti.

3. Izrada jednostavna memorijskoga datotečnoga sustava

Glavni zadatak ovog rada je, kao što i sam naslov govori, ostvarenje jednostavna memorijskoga datotečnoga sustava. U ovom poglavlju rada opisat ću na koji način sam radio i napravio jednostavni datotečni sustav.

Nakon što sam proučio NTFS, ext, ext2, ext3, ext4, HFS i HFS Plus datotečne sustave, te dobio sliku o tome što datotečni podsustav zapravo predstavlja i koje su njegove glavne uloge, krenuo sam u izradu vlastitog datotečnog sustava. Čitajući na internetu o datotečnim sustavima općenito, te o datotečnim sustavima koje sam opisao u prethodnom poglavlju naučio sam mnogo toga uporabljivog. Međutim, u početku mi je jako teško bilo shvatiti na koji način bi se to moglo napraviti u memoriji alociranoj za izvođenje programa. Nakon dosta potrošenog vremena na osmišljanje načina na koji bih organizirao sami „disk“ došao sam na ideju da to napravim pomoću vezane liste u kojoj bi svaki element bio jedan sektor tog „diska“. Pojam diska ovdje namjerno stavljam pod navodnike jer se on ne odnosi na tvrdi disk računala već na dio memorije koji je alociran za izvođenje mog sustava i u kojem će biti moj datotečni sustav. Ideja s vezanim listama činila mi se jednostavnom za implementaciju, ali me mentor podsjetio na činjenicu da ću u daljnjem razvoju sustava morati implementirati i pohranjivanje tog sustava u datoteku kako bi se on kasnije mogao ponovno učitati i kako ne bi dolazilo do gubitka svih podataka u datotečnom sustavu nakon gašenja sustava. Pohranjivanje sustava u datoteku ne bi bio nikakav problem. Problem kojeg bi uzrokovalo moje rješenje s vezanim listama u C++ programskom jeziku vezan je uz pokazivače. Naime, prilikom ponovnog učitavanja datotečnog sustava iz datoteke sustav bi učitao listu sektora, ali pokazivači koje bi učitao ne bi mu ništa predstavljali jer bi pokazivali na memorijske lokacije alocirane Nakon proučavanja nastalog problema, došao sam do ideje da sustav napravim u C# programskom jeziku na način na koji sam to zamislio. Naime, liste u C# programskom jeziku napravljene su uz pomoć indeksa, dok su u C++ napravljene uz pomoć pokazivača.

Prema dogovoru s mentorom za prvu verziju sustava bilo je dovoljno da radi samo „u memoriji“, odnosno da „disk“ postoji tako dugo dok se program izvršava, a kod završetka izvršavanja nije trebalo biti trajnog pohranjivanja već se kod idućeg pokretanja sustava kreirao novi datotečni sustav. Prva verzija prema dogovoru je trebala uključivati sljedeće:

- Otvaranje datoteke
- Zatvaranje datoteke

- Doznavanje datotečnih pokazivača za čitanje/pisanje
- Postavljanje datotečnih pokazivača za čitanje/pisanje
- Stvaranje datoteke/kazala
- Brisanje datoteke/kazala
- Premještanje datoteke/kazala
- Kopiranje datoteke/ kazala
- Preimenovanje datoteke/ kazala
- Pisanje N bajtova u datoteku
- Čitanje N bajtova iz datoteke
- Stvaranje novoga praznoga datotečnoga sustava
- Brisanje datotečnoga sustava iz memorije

Kasnije sam još trebao implementirati mogućnosti spremanja datotečnog sustava u datoteku na disk računala, te učitavanje datotečnog sustava iz datoteke. Sve navedene mogućnosti sam implementirao, a u sljedećih nekoliko poglavlja ću ih opisati zajedno sa strukturama.

3.1. Datotečni opisnik (`FileDescriptor`)

Datotečni opisnici sadrže slijedeće informacije o datotekama:

- `Path` – string, sadrži putanju datoteke
- `Readable`– `bool`, sadrži informaciju o mogućnosti čitanja iz datoteke
- `Writeable` – `bool`, sadrži informaciju o mogućnosti pisanja u datoteku

U programskom rješenju, razred se naziva `FileDescriptor`.

3.2. Opisnik otvorene datoteke (`FileHandle`)

Opisnik otvorene datoteke sadrži slijedeće informacije o datoteci:

- `currentSector` – `int`, sadrži informaciju o sektoru diska koji je na redu za čitanje/pisanje

- `currentByte` – `int`, sadrži informaciju o bajtu koji je na redu za čitanje/pisanje
- `WritingFilePointer` – `int`, sadrži informaciju o bajtu u datoteci koji je na redu za pisanje
- `ReadingFilePointer` – `int`, sadrži informaciju o bajtu u datoteci koji je na redu za čitanje
- `fileDescriptor` – `FileDescriptor`, sadrži opisnik datoteke
- `fileSystem` – `FileSystem`, sadrži informacije o datotečnom sustavu

Opisnik otvorene datoteke u programskom rješenju ima naziv `FileHandle`.

3.3. Sektor (`Sector`)

Razred `Sector` sadrži sljedeće informacije o pojedinom sektoru diska:

- `Id` – `int`, identifikacijski broj sektora
- `Bytes` – `byte`, polje bajtova u koje se zapisuju podaci
- `Size` – `int`, količina iskorištenih bajtova određenog sektora

U programskom rješenju razred se naziva `Sector`.

3.4. Datotečni sustav (`FileSystem`)

Razred `FileSystem` sadrži informacije o datotečnom sustavu, te su u njoj implementirane metode za upravljanje datotečnim sustavom. Razred sadrži sljedeće informacije:

- `numberOfSectors` – `int`, sadrži informaciju o broju sektora na disku
- `sectorSize` – `int`, sadrži informaciju o veličini sektora (veličina je izražena u broju bajtova)
- `Files` – `Dictionary<string, Filedescriptor>`, lista datoteka tipa `Dictionary` pohranjenih na disku, ključ je putanja datoteke, a vrijednost predstavlja opisnik datoteke

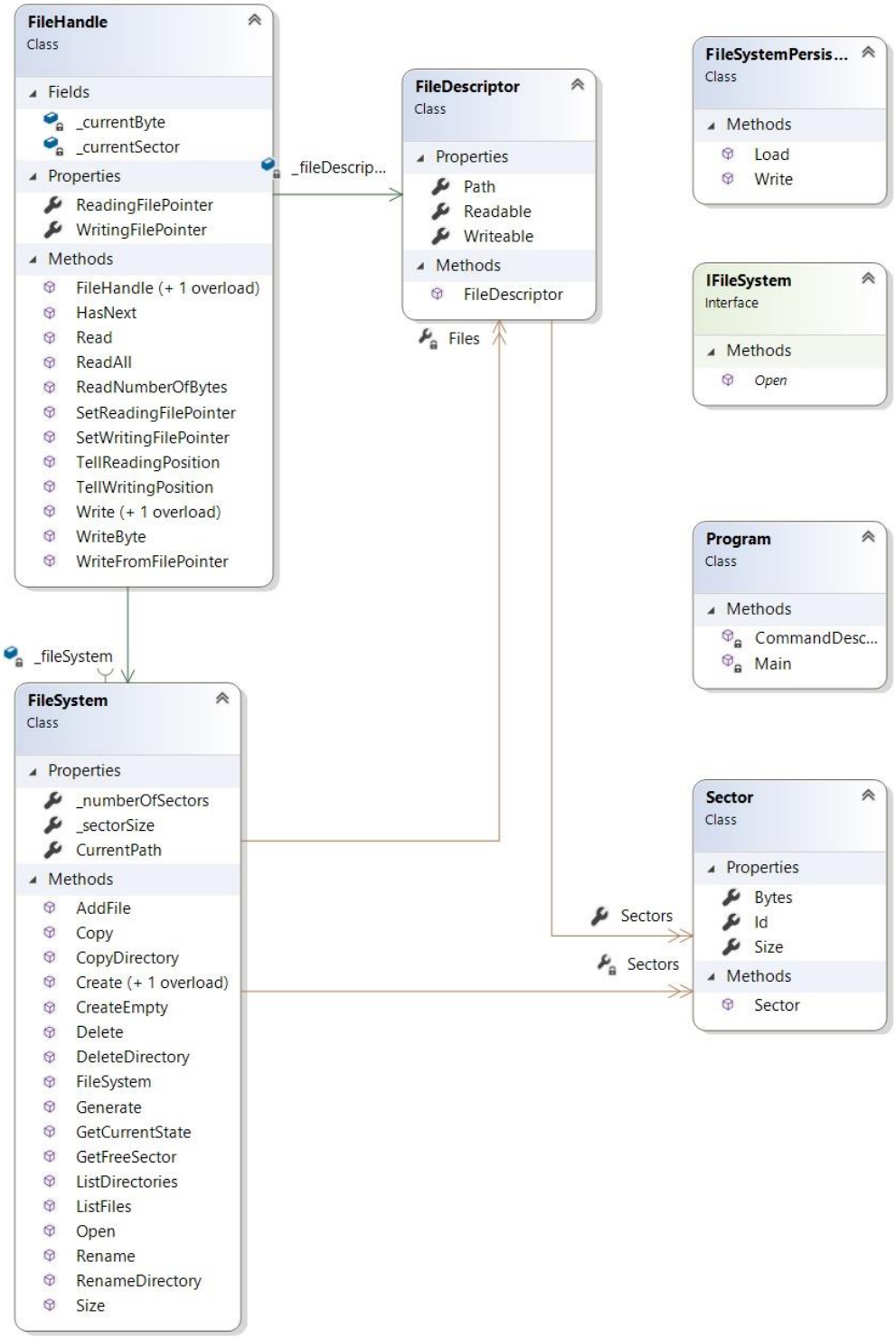
- `Sectors` – `Dictionary<int, Sector>`, lista sektora tipa `Dictionary` koji se nalaze na disku, ključ predstavlja identifikacijski broj sektora, a vrijednost predstavlja objekt razreda `Sector`

3.5. `FileSystemPersistor`

`FileSystemPersistor` je razred koja nema nikakvih atributa, već samo dvije metode i to jednu za spremanje datotečnog sustava u datoteku, a drugu za učitavanje datotečnog sustava iz datoteke.

3.6. UML dijagram razreda

Slika 3. prikazuje UML dijagram razreda generiran u Visual Studiu.



Slika 3. UML dijagram razreda

3.7. Stvaranje datoteke

Provjerava se postoji li na zadanoj putanji već neka druga datoteka. Ako ne postoji, kreira se datotečni opisnik za datoteku, te inicijalizira. Prilikom inicijalizacije opisnika datoteke postavljaju se `isReadable`, `isWriteable` i `Path` atributi. Također, alocira se novi sektor i on se dodaje u listu sektora kreirane datoteke.

Datoteka se stvara metodom `Create` razreda `FileSystem`. Ulazni parametri su putanja datoteke tipa string, te atributi koji opisuju je li datoteku moguće čitati ili u nju pisati koji su tipa `bool`. Metoda `Create` vraća datotečni opisnik kreirane datoteke.

3.8. Stvaranje kazala

U mojem rješenju, stvaranje kazala je vrlo slično stvaranju datoteka. Jedina razlika je u vrijednosti atributa `Path` opisnika datoteke, odnosno objekta razreda `FileDescriptor`. Naime, zadnji znak u putanji kazala je „\“, dok je kod datoteke onemogućeno korištenje tog znaka u nazivu putanje. Iz ovog proizlazi da su i kazala zapravo određena vrsta datoteke.

3.9. Otvaranje datoteke

Ako je zadana ispravna putanja do datoteke, odnosno ako na zadanoj putanji zaista postoji datoteka, stvara se i vraća opisnik otvorene datoteke (`FileHandle`). Ako nije zadana putanja ili pak je putanja nepostojeća, javlja se greška.

Otvaranje datoteke obavlja se metodom `Open` razreda `FileSystem`. Ulazni parametar metode je putanja datoteke tipa string, a izlazni parametar je objekt razreda `FileHandle` koji predstavlja datotečni opisnik otvorene datoteke.

3.10. Zatvaranje datoteke

Opisnik otvorene datoteke se poništava, odnosno postavlja se na `NULL` vrijednost.

3.11. Postavljanje položaja datotečnog pokazivača

Kod postavljanja datotečnog pokazivača za čitanje ili pisanje, provjerava se premašuje li zadani pokazivač veličinu datoteke. Ako premašuje, javlja se greška, a ako ne premašuje, postavlja se pokazivač za čitanje, odnosno pisanje na zadani bajt datoteke.

Ulazni parametar je brojčana vrijednost koja predstavlja buduću vrijednost datotečnog pokazivača za čitanje ili pisanje.

3.12. Doznavanje položaja datotečnog pokazivača

Kod doznavanja položaja datotečnog pokazivača za čitanje ili pisanje, dohvaća se atribut `ReadingFilePointer`, odnosno atribut `WritingFilePointer` opisnika otvorene datoteke, odnosno objekta razreda `FileHandle`.

3.13. Pisanje u datoteku

Pisanje u datoteku obavljaju dvije metode iz razreda `FileHandle`. Kod pokušaja pisanja u datoteku, pokreće se `for` petlja koja se vrti od 0 do broja bajtova koji se pokušavaju zapisati u datoteku. Za svaki korak for petlje, poziva se druga metoda u kojoj se najprije provjerava stanje atributa `Writeable` datotečnog opisnika datoteke u koju se pokušava pisati. Zatim, ukoliko je stanje atributa `true`, dohvaća se posljednji sektor datoteke, te se provjerava je li taj sektor popunjen ili ima još slobodnih bajtova. Ako sektor nije popunjen, novi bajt se zapisuje na poziciju određenu atributom `Size`, te se vrijednost atributa `Size` povećava za 1. Podsjetimo se, atribut `Size` sadrži broj iskorištenih bajtova određenog sektora. Ukoliko je sektor popunjen, alocira se novi sektor i pridružuje datoteci, te se bajt upisuje na nultu poziciju sektora. Ukoliko je stanje atributa `Writeable` datotečnog opisnika datoteke `false`, tada se javlja greška da u datoteku nije moguće pisati.

Ulazni parametar je polje bajtova koje predstavlja sadržaj koji se treba zapisati u datoteku.

3.14. Čitanje iz datoteke

Čitanje iz datoteke obavlja se također sa dvije metode iz razreda `FileHandle`. Kod pokušaja čitanja točno određenog broja bajtova iz datoteke, pokreće se `while` petlja koja se vrti tako dugo dok ima bajtova za čitati i dok nije pročitan zadani broj bajtova. U svakom koraku `while` petlje pokreće se metoda koja, ako je datoteku moguće čitati, čita određeni bajt iz određenog sektora. Zatim, ako se želi još čitati i pročitani bajt nije posljednji u sektoru povećava se vrijednost atributa `_currentByte`, a ako je pročitani bajt posljednji u sektoru i postoji još sektora koji se mogu čitati, tada se vrijednost atributa `_curentSector` povećava za jedan, a vrijednost atributa `_currentByte` se postavlja na 0.

Metoda za pokretanje čitanja datoteke nema ulaznih parametara, a izlazni parametar je polje pročitanih bajtova.

3.15. Preimenovanje datoteke

Preimenovanje datoteke obavlja se metodom `Rename` iz razreda `FileSystem`. Prvo se provjerava postoji li uopće datoteka pod nazivom koji se želi promijeniti. Zatim, ako datoteka sa zadanim starim nazivom postoji, mijenja se stari naziv datoteke u željeni novi. Ako pak ne postoji datoteka sa zadanim starim nazivom, vraća se `false`.

Ulazni parametri su stari naziv datoteke i novi naziv datoteke, a izlazni parametar je istina ili laž, ovisno o postojanju datoteke sa zadanim starim nazivom.

3.16. Preimenovanje kazala

Preimenovanje kazala je malo složeniji proces od preimenovanja datoteke. Sustav radi s apsolutnim putanjama, te svaki datotečni opisnik datoteke čuva informaciju o čitavoj putanji datoteke. Kako bi se promijenio naziv kazala, potrebno je promijeniti taj naziv u svim pod kazala i datotekama tog kazala. To funkcionira na način da se prolazi kroz sve datotečne opisnike čiji atribut `Path` sadržava staru putanju kazala koji se preimenuje, te premjestiti stari dio putanje novim.

Preimenovanje se obavlja metodom `RenameDirectory` (razred `FileSystem`). Ulazni parametri metode su stara putanja i nova putanja, a izlazni parametar je `true` ili `false`, ovisno o tome postoji li uopće kazalo sa zadanom starom putanjom ili ne postoji.

3.17. Premještanje datoteke

S obzirom na to da sam datotečni sustav trebao napraviti na način da radi s apsolutnim putanjama, te nisam trebao implementirati relativne putanje, premještanje datoteka u mom sustavu nije ništa drugo nego preimenovanje datoteke. S obzirom na to da je bilo dovoljno da sustav radi samo s apsolutnim putanjama, za premještanje datoteke je bilo dovoljno promijeniti vrijednost putanje.

3.18. Premještanje kazala

Premještanje kazala obavlja se na sličan način kao i premještanje datoteke. Kao što sam već nekoliko puta naveo, s obzirom na to da sustav treba raditi s apsolutnim putanjama, premještanje kazala napravljeno je preimenovanjem, jer se zapravo radi samo o promijeni naziva putanje kazala.

3.19. Kopiranje datoteke

Kopiranje datoteke obavlja metoda `Copy` razreda `FileSystem`. Najprije se otvara datoteka koja se želi kopirati, te se provjerava postoji li uopće datoteka sa zadanom putanjom. Ako se utvrdi da datoteka ne postoji, vraća se `false`. Inače, kreira se nova datoteka sa zadanom putanjom, te se sa `while` petljom prolazi kroz staru datoteku i bajt po bajt se kopira u novu datoteku. Na kraju se vraća `true`.

Ulazni parametri su putanja datoteke koja se kopira, te putanja nove datoteke. Izlazni parametri su `true` ili `false`, ovisno o postojanju datoteke koja se kopira.

3.20. Kopiranje kazala

Kopiranje kazala je nešto složeniji posao od kopiranja datoteke s obzirom na to da se kod kopiranje kazala trebaju kopirati sva kazala i datoteke koji su unutar tog kazala. Kod kopiranja kazala najprije se kreira prazna lista stavki koje treba kopirati. Zatim se prelazi po datotekama i u listu se spremaju datoteke ili kazala koji kao dio putanje imaju zadanu putanju izvorišta. Nakon prelaska kroz datoteke, prelazi se po listi, te se kreiraju nove datoteke koje imaju izmijenjeni dio putanje. Dio putanje koji je isti kao putanja izvorišta mijenja se zadanom putanjom odredišta.

Ulazni parametri metode za kopiranje kazala su putanja izvorišta i putanja odredišta, a izlazni parametri su `true` ili `false`.

3.21. Pisanje N bajtova u datoteku

Pisanje N bajtova u datoteku obavlja se na način da se najprije provjerava je li u datoteku uopće dozvoljeno pisati. Ako nije, vraća se `false`. Ako je pisanje dozvoljeno, bajt po bajt se zapisuju na način da se najprije izračunaju trenutni sektor i trenutni bajt sektora. Trenutni sektor računa se kao kvocijent datotečnog pokazivača za pisanje i veličine sektora.

Trenutni bajt sektora računa se kao cjelobrojni ostatak datotečnog pokazivača za pisanje i veličine sektora. Na taj način izračuna se u koji sektor i koji bajt tog sektora se zapisuje sljedeći bajt. Nakon što je izračunata pozicija, na nju se zapisuje bajt, te se datotečni pokazivač za pisanje povećava za 1.

Ulazni parametar je polje bajtova koje se treba zapisati u datoteku.

3.22. Čitanje N bajtova iz datoteke

Čitanje N bajtova iz datoteke je vrlo slično čitanju cijele datoteke opisanom u poglavlju 4.13. Jedina razlika je u tome što za čitanje cijele datoteke nije potrebno pratiti stanje datotečnog pokazivača za čitanje, jer se čita cijela datoteka od prvog do posljednjeg bajta, dok se kod čitanja N bajtova iz datoteke datoteka čita od datotečnog pokazivača za čitanje. Najprije se kreira lista tipa `byte` u koju će se dodavati bajtovi pročitani iz datoteke. Zatim se ulazi u `while` petlju čiji su uvjeti da ima još bajtova za pročitati i da je do sad pročitano manje od N bajtova. Ako su uvjeti zadovoljeni, čita se naredni bajt i dodaje u listu bajtova, te se vrijednost datotečnog pokazivača za čitanje povećava za 1. Kad se pročita N bajtova, izlazi se iz `while` petlje, te se vraća polje pročitanih bajtova.

Ulazni parametar je brojčana vrijednost koja predstavlja broj bajtova koji se žele pročitati, a izlazni parametar je polje pročitanih bajtova.

3.23. Pohranjivanje datotečnog sustava u datoteku

Za pohranjivanje datotečnog sustava u datoteku koristi se metoda razreda stvorenog isključivo za pohranjivanje i učitavanje datotečnog sustava u datoteku. Radi se o razredu `FileSystemPersistor` i njegovoj metodi `Write`. Najprije se poziva metoda `GetCurrentState`. To je metoda razreda `FileSystem` koja čitavi datotečni sustav smješta u polje bajtova i zatim to polje bajtova vraća. Najprije se u listu bajtova spremaju datotečni opisnici. Za svaki datotečni opisnik se radi sljedeće:

- U prvih 100 bajtova se sprema putanja datoteke
- U slijedeća dva bajta spremaju se informacije o mogućnosti čitanja, odnosno pisanja u datoteku
- Prolazi se kroz listu sektora datoteke i u listu bajtova se zapisuju identifikacijski brojevi sektora

Nakon prolaska kroz listu datotečnih opisnika, dodaje se onoliko praznih bajtova koliko je ostalo do 2 puna sektora nakon što su se spremili svi datotečni opisnici, jer su za datotečne opisnike rezervirana 2 sektora. Zatim se prolazi kroz sve sektore i oni se također dodaju u listu bajtova, te se nakon toga ta lista pretvara u polje i vraća. Metoda `Write` razreda `FileSystemPersistor` zatim pohranjuje dobiveno polje bajtova u datoteku odabranog imena.

Metoda kao ulazne parametre prima naziv datoteke u koju se pohranjuje datotečni sustav, te sami datotečni sustav koji se želi pohraniti u datoteku.

3.24. Učitavanje datotečnog sustava iz datoteke

Učitavanje datotečnog sustava iz datoteke, kao što je spomenuto u prošlom poglavlju, obavlja se metodom iz razreda `FileSystemPersistor` i to metodom `Load`. Metoda funkcionira na način da se najprije kreira polje bajtova u koje se učitaju svi bajtovi iz datoteke. Zatim se poziva metoda `Generate` razreda `FileSystem` koja prima broj i veličinu sektora, te zatim instancira i vraća objekt razreda `FileSystem`. Iz metode `Load` se zatim poziva opet metoda razreda `FileSystem`, ali se ovaj puta radi o metodi `Create`. Metoda `Create` poziva `while` petlju u čijem se svakom koraku događaju slijedeće aktivnosti:

1. Dohvaća se narednih 100 bajtova koji predstavljaju putanju datoteke
2. Provjerava se je li dohvaćeni string zaista putanja ili su pročitani svi datotečni opisnici
3. Ako su već svi datotečni opisnici pročitani, sa `break` se izlazi iz `while` petlje
4. Ako je dohvaćeni string zaista putanja datoteke, događa se sljedeće
5. Brojač se povećava za 100
6. Instancira se novi objekt razreda `FileDescriptor`, odnosno novi datotečni opisnik
7. Čitaju se sljedeći bajtovi polja u kojima se provjerava je li datoteku moguće čitati i pisati, te se ti podaci dodaju objektu
8. Dohvaća se veličina datoteke u bajtovima, te se iz toga računa broj sektora na kojima je datoteka zapisana
9. Pokreće se `for` petlja koja se vrti „broj sektora“ puta, a u svakom njezinom koraku se radi sljedeće

1. Čitaju se po 4 bajta koja govore o kojem sektoru se radi
2. U listu sektora se dodaje sektor s identifikacijskim brojem dobivenim u prethodnom koraku, te se u taj sektor dohvaća narednih „veličina sektora“ bajtova
3. Sektoru se pridružuje informacija o količini iskorištenih bajtova
4. Datotečnom opisniku se pridružuje sektor

10. Posljednjem pridruženom sektoru se ponovno preračunava količina iskorištenih bajtova kao cjelobrojni ostatak kod dijeljenja veličine datoteke sa veličinom sektora
11. Brojač se povećava za umnožak broja sektora trenutne datoteke sa 4

Po izlasku iz `while` petlje, vraća se objekt razreda `FileSystem`. Ulazni parametri metode `Load` su naziv datoteke, te broj i veličina sektora. Ulazni parametri u metodu `Create` koja se poziva iz metode `Load` su polje bajtova (datotečni sustav), broj sektora i veličina sektora datotečnog sustava.

3.25. Stvaranje novoga datotečnoga sustava

Stvaranje novog datotečnog sustava obavlja konstruktor razreda `FileSystem`. Prilikom stvaranja novog datotečnog sustava definira se broj sektora, veličina sektora, inicijaliziraju se prazna lista sektora i prazna lista datoteka, te se kreira korijenski kazalo. Naziv korijenskog kazala je zadan („C:\“), ali u daljnjem razvoju u planu je mogućnost odabira naziva prepustiti korisniku koji kreira datotečni sustav.

Ulazni parametri konstruktora su broj sektora i veličina sektora.

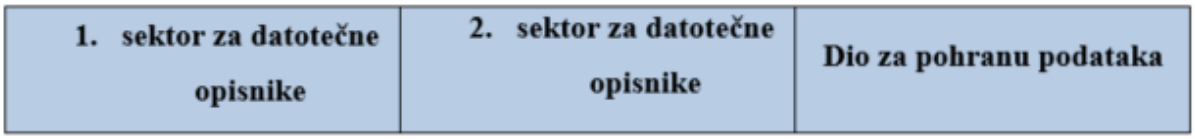
3.26. Brisanje datotečnog sustava

Kod brisanja datotečnog sustava ne događa se ništa posebno, osim što se objektu pridružuje `NULL` vrijednost.

4. Organizacija memorije

U ovom poglavlju ukratko je opisana organizacija memorije u mom datotečnom sustavu. S obzirom na to da se radi o *In-Memory* datotečnom sustavu, odnosno o sustavu koji se nalazi u dijelu radne memorije alociranom za izvođenje programa (datotečnog sustava), opisat ću njegov smještaj u datoteci.

Sustav je organiziran na način da su prva dva sektora rezervirana za pohranu datotečnih opisnika. *Slika 3.* ugrubo prikazuje organizaciju prostora na disku.



Slika 4. Organizacija prostora na disku

Datotečni opisnici su organizirani tako da prvih 100 bajtova zauzima putanja datoteke. U slijedeći bajt se zapisuje podatak o mogućnosti čitanja datoteke i pisanja u datoteku. Ako datoteku nije moguće čitati niti je u nju moguće pisati, u taj bajt zapisuje se 0. Ako je datoteku moguće samo čitati, u taj bajt zapisuje se 1. Ako je pak moguće samo pisati u datoteku, u taj bajt se zapisuje 2, a ako je moguće i čitati datoteku i pisati u nju, zapisuje se 3. Nakon podataka o mogućnosti čitanja i pisanja u datoteku, zapisuju se identifikacijski brojevi sektora na kojima je datoteka razmještena. Ako nakon pohranjivanja svih opisnika nisu popunjena prva dva sektora, oni se popunjavaju praznim bajtovima. *Slika 4.* prikazuje izgled jednog zapisa datotečnog opisnika u sektoru. Kao što je vidljivo na slici, prvih 100 bajtova zauzima putanja datoteke, zatim jedan bajt govori o mogućnosti čitanja i pisanja u datoteku, pa 4 bajta govore o veličini datoteke u broju bajtova i konačno dolaze redom identifikacijski brojevi sektora od kojih svaki zauzima po 4 bajta (`integer`).

100 bajtova	1 bajt	4 bajta	4 bajta	4 bajta	...
Putanja datoteke	Mogućnost čitanja/pisanja	Veličina datoteke (u broju bajtova)	Identifikacijski broj sektora	Identifikacijski broj sektora	...

Slika 5. Jedan zapis datotečnog opisnika na disku

Nakon što se zapišu svi datotečni opisnici bajtovi svih sektora redom se dodaju u polje bajtova.

Ovakva organizacija sustava mogla bi dovesti do situacije da se popune prva dva sektora opisnika i da za opisnike nema više mjesta, a taj problem do izražaja bi došao tek kod spremanja datotečnog sustava u datoteku. Taj dio u budućnosti planiram optimizirati na način da se broj sektora rezerviran za opisnike određuje uzimajući u obzir i broj datoteka koje se nalaze u datotečnom sustavu. Također, sustav nije osiguran od slučaja da dođe do oštećenja prva dva sektora u kojima su opisnici, što u budućnosti također planiram popraviti.

4.1. Usporedba mojeg datotečnog sustava s opisanim datotečnim sustavima

Za početak moram naglasiti da sam u izradu svog datotečnog sustava uložio puno truda, jer sam u tom području imao vrlo malo znanja što je i jedan od razloga zbog kojeg sam odabrao ovu temu. Teško je usporediti moj datotečni sustav kojeg sam radio mjesec dana sa datotečnim sustavima na kojima su godinama radili timovi inženjera, ali probat ću usporediti koliko je to moguće.

Uzmimo za početak veličinu diska. Kod opisanih datotečnih sustava najveća moguća veličina particije kretala se između 2 GiB i 16 EiB. Kod mojeg datotečnog sustava veličina particije određena je korisnički, ali najveća moguća veličina particije nikako ne može prelaziti veličinu RAM memorije alociranu za izvođenje programa.

Što se tiče maksimalne veličine datoteka, opisani datotečni sustavi podržavaju datoteke veličina od 2 GiB do 16 EiB, dok je kod mojeg datotečnog sustava veličina datoteke također ograničena veličinom memorije alocirane za izvođenje programa, odnosno popunjenošću prva dva sektora koji su alocirani za datotečne pokazivače. Naime, u datotečnim pokazivačima datoteka pohranjuju se identifikacijski brojevi sektora na koje je raspoređena

datoteka. Ako su prva dva sektora popunjena u potpunosti, neće se moći dodavati novi identifikacijski brojevi sektora, te se datoteka više neće moći povećavati. Veličina naziva datoteke kod opisanih sustava kreće se od 31 do 255 znakova. Kod mojeg sustava maksimalna veličina naziva datoteke određena je maksimalnom veličinom putanje i iznosi 100 znakova.

5. Opis primjenskog programa

U ovom poglavlju opisan je primjenski program napisan u svrhu testiranja funkcionalnosti datotečnog sustava. Kako bih provjerio funkcionira li sustav kako treba, napravio sam primjenski program putem kojeg se mogu pozivati funkcije datotečnog sustava, te se na taj način uvjeriti radi li sve kako treba.

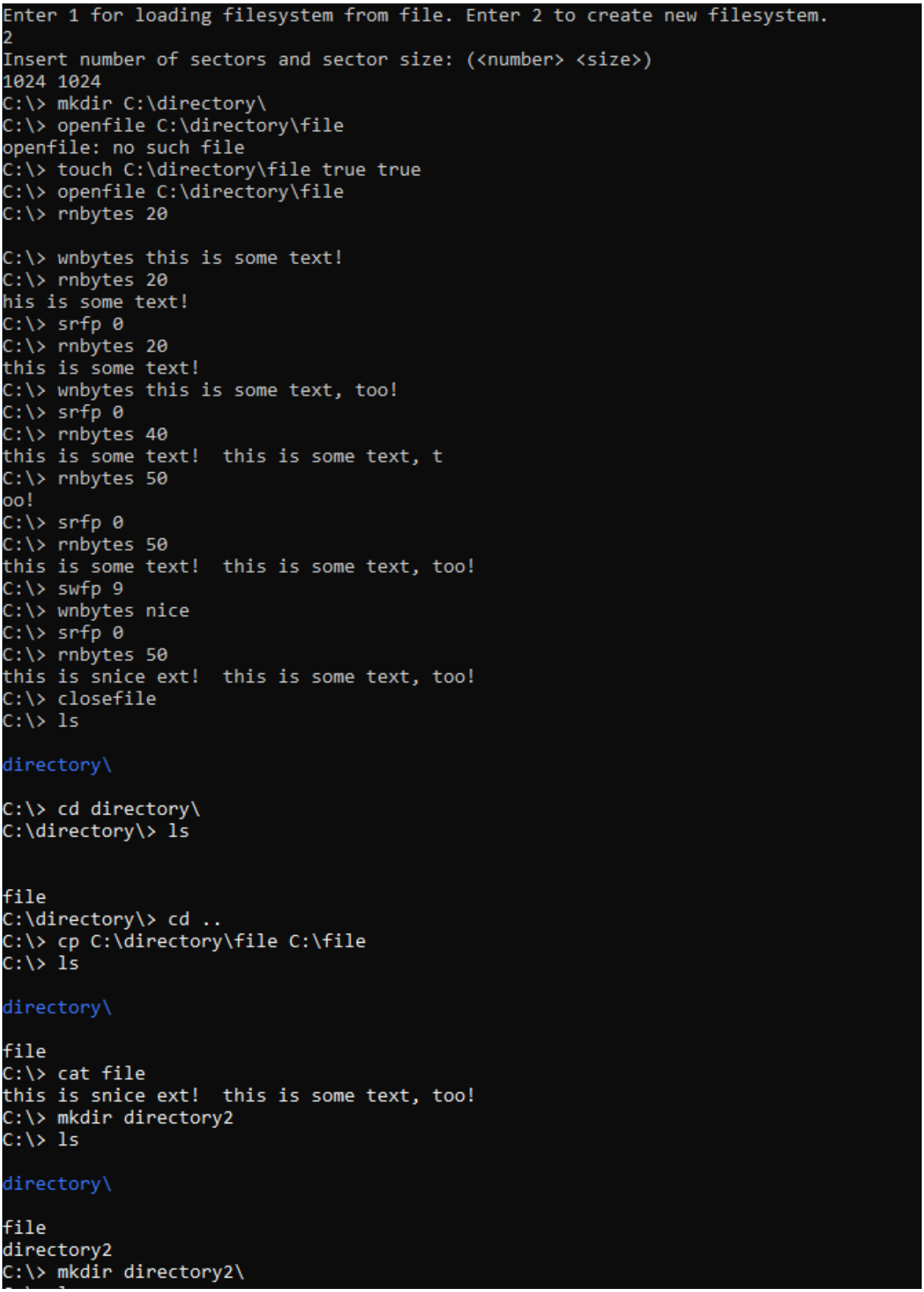
Primjenski program dobrim dijelom je napravljen po uzoru na UNIX okruženju s kakvim sam se imao prilike susresti na kolegijima Informatika 2, Operacijski sustavi 1 i Mreže računala 2. Svakako, moj sustav daleko je od UNIX sustava, ali su naredbe koje sam implementirao napravljene po uzoru na UNIX. Slijedeća tablica prikazuje popis naredbi koje sam implementirao, opis njihove zadaće, te primjer poziva naredbe.

Tablica 10. Popis i opis naredbi

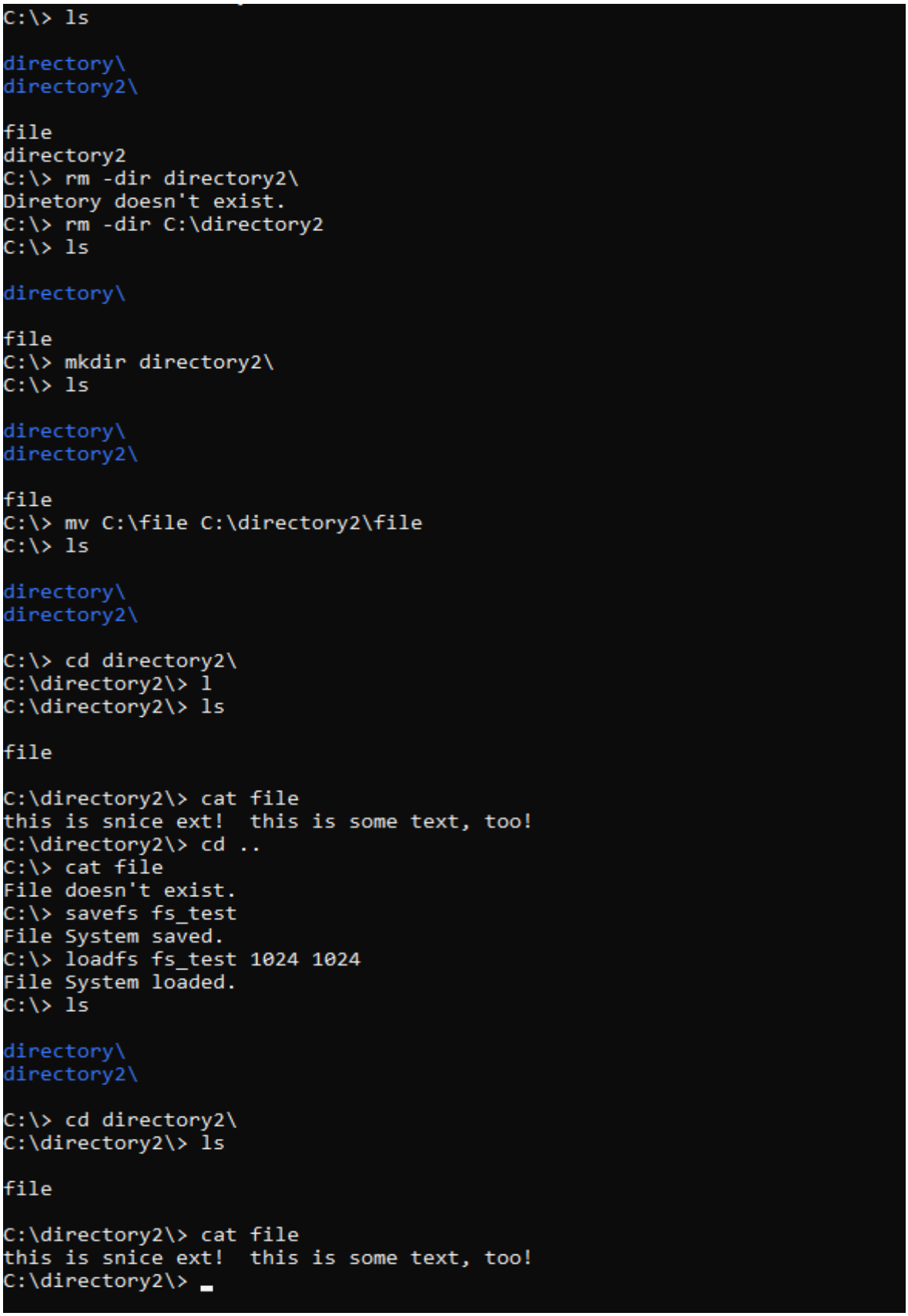
ZADATAK	NAREDBA	PRIMJER
ukrcavanje fs iz datoteke	loadfs {FILE PATH}	loadfs fs1.dat
iskrcavanje fs u datoteku	savefs {FILE PATH}	savefs fs1.dat
otvaranje datoteke	openfile {FILE PATH}	openfile C:\datoteka
zatvaranje datoteke	closefile	closefile
doznavanje pokazivača za čitanje	tellg	tellg
doznavanje pokazivača za pisanje	tellp	tellp
mijenjanje pokazivača za čitanje	srfp N	srfp 4
mijenjanje pokazivača za pisanje	swfp N	swfp 4
stvaranje datoteke	touch {FILE PATH} {READABLE} {WRITEABLE}	touch C:\datoteka true true
brisanje datoteke	rm {FILE PATH}	rm C:\datoteka
stvaranje direktorija	mkdir {DIRECTORY PATH}	mkdir C:\direktorij
brisanje direktorija	rm -dir {DIRECTORY PATH}	rm -dir C:\direktorij
premještanje datoteke	mv {FILE PATH {FILE PATH}	mv C:\direktorij\datoteka C:\direktorij2\datoteka
premještanje direktorija	mv -dir {DIRECTORY PATH} {DIRECTORY PATH}	mv -dir C:\direktorij\direktorij2 C:\direktorij2
kopiranje datoteke	cp {FILE PATH {FILE PATH}	cp C:\datoteka C:\datoteka_Copy
kopiranje direktorija	cp -dir {DIRECTORY PATH} {DIRECTORY PATH}	cp -dir C:\direktorij C:\direktorij_Copy

preimenovanje datoteke	mv {FILE PATH {FILE PATH}	mv C:\direktorij\datoteka C:\direktorij2\datoteka
preimenovanje direktorija	mv -dir {DIRECTORY PATH} {DIRECTORY PATH}	mv -dir C:\direktorij\direktorij2 C:\direktorij2
pisanje N bytea z datoteku	wnbytes {TEXT}	wnbytes text
čitanje N bytea iz datoteke	rnbytes {N}	rnbytes 5
stvaranje novoga praznoga fs	cfs {NUMBER OF SECTORS} {SECTOR SIZE}	cfs 1024 1024
brisanje fs iz memorije	dfs	dfs
pisanje u datoteku	write {FILE PATH}	write C:\datoteka
čitanje datoteke	cat {FILE PATH}	cat C:\datoteka
ispis popisa naredbi	help	help
ispis upute za korištenje naredbe	help {NAZIV NAREDBE}	help touch
naredba za izlaz	exit	exit
naredba za ispis sadržaja direktorija	ls	ls

Kao što je vidljivo u prethodnoj tablici, premještanje kazala ili datoteke i preimenovanje kazala ili datoteke radi se istom naredbom. Razlog tome je taj što su i preimenovanje i premještanje kazala ili datoteke zapravo samo promjena putanje. Nadalje, naredbe za pisanje N bajtova u datoteku, čitanje N bajtova iz datoteke, postavljanje datotečnih pokazivača i čitanje datotečnih pokazivača rade samo ako je prethodno otvorena datoteka, a inače vraćaju grešku. Kod pisanja naredbom `write` najprije se tom naredbom otvara datoteka za pisanje. Zatim se upisuje tekst, a kad se želi izaći iz datoteke i spremiti promijene, pritisne se Ctrl + Q kombinacija tipki. Naredba `write` zapravo funkcionira na sličan način kao naredbe `pico` ili `nano` u UNIX sustavu.



Slika 6. Prikaz rada sustava 1



Slika 7. Prikaz rada sustava 2

6. Zaključak

U ovom radu opisani su najvažniji datotečni sustavi današnjice, te su isti uspoređeni prema ključnim parametrima. Kroz njihove opise može se vidjeti koliko su se razvijali godinama i koliko je rada bilo potrebno da bismo danas imali datotečne sustave kakve imamo. Nakon proučavanja složenih datotečnih sustava, opisana je izrada jednostavnijeg datotečnog sustava. U izradi završnog rada koristio sam Visual Studio razvojno okruženje, C# programski jezik. Kao što je već u radu opisano, izbor samog alata bio je velik posao, jer su o tome odlučivali i neki tehnički parametri koji su imali utjecaja na kompleksnost implementacije pojedinih funkcionalnosti. Prilikom izrade datotečnog sustava, kao što sam i očekivao, naučio sam mnogo toga što tokom samog preddiplomskog studija nisam uspio. U daljnjem razvoju ovog sustava već sam predvidio mnoga poboljšanja koja bih mogao implementirati, te tako još više proširiti svoja znanja u tom području.

Zasad se datotečni sustav kojeg sam napravio može koristiti samo u eksperimentalne svrhe jer još dosta stvari koje suvremeni datotečni sustavi podržavaju u mojem još nije podržano. Dodatno sam počeo raditi na podržavanju relativnih putanja, te smatram da bi daljnjim razvojem svog datotečnog sustava mogao napraviti sustav koji bi bio primjenjiv u stvarnom korištenju.

Popis literature

[1] „NTFS — New Technology File System for Windows 10, 8, 7, Vista, XP, 2000, NT & Windows Servers 2016, 2012, 2008, 2003, 2000, NT“ na NTFS.com. Dostupno: http://www.ntfs.com/ntfs.htm [pristupano 30.08.2018.]

[2] M. Mullins, „Windows 101: Know the basics about NTFS permissions“, 2006. [Na interntetu]. Dostupno: https://www.techrepublic.com/article/windows-101-know-the-basics-about-ntfs-permissions/ [pristupano: 01.09.2018.]

[3] J. Salter, „Understanding Linux filesystems: ext4 and beyond“, 2018. [Na internetu]. Dostupno: https://opensource.com/article/18/4/ext4-filesystem [pristupljeno: 03.09.2018.]

[4] V. Damjanovski CCTV: Networking and Digital Technology, Elsevier Inc., 2005.

[5] "Comparison of file systems," na Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Comparison_of_file_systems [pristupano 03.09.2018.]

Popis slika

Slika 1. Struktura NTFS-a 4
Slika 2. MTF zapis male datoteke ili kazala 5
Slika 3. UML dijagram razreda 19
Slika 4. Organizacija prostora na disku 27
Slika 5. Jedan zapis datotečnog opisnika na disku 28
Slika 6. Prikaz rada sustava 1 31
Slika 7. Prikaz rada sustava 2 32

Popis tablica

Tablica 1. NTFS Boot sektor 4
Tablica 2. Prvih 16 zapisa u MTF tablici 5
Tablica 3. Popis i opis atributa koji se pohranjuju u MFT 6
Tablica 4. Osnovne dozvole u NTFS-u [2] 7
Tablica 5. Popis dodatnih dozvola u NTFS-u [2] 7
Tablica 6. Zadane veličine klastera kod NTFS-a 8
Tablica 7. Usporedba ext datotečnih sustava 11
Tablica 8. Usporedba HFS i HFS Plus datotečnih sustava 13
Tablica 9. Usporedba NTFS, ext, ext2, ext3, ext4, HFS i HFS+ datotečnih sustava 14
Tablica 10. Popis i opis naredbi 29