Univerzitet u Kragujevcu
Prirodno - matematički fakultet
Institut za matematiku i informatiku

S E M I N A R S K I R A D

Tema: Implementacija fajl sistema

Predmet: Operativni sistemi 2

Student:
Nada Krivčević 69/11

Mentor:
Doc. dr Miloš Ivanović

Kragujevac, septembar 2016. godine

S A D R Ž A J:

1. Uvod

Fajl sistem predstavlja način organizovanja zapisivanja podataka na disk, samim tim omogućava njihovo lakše pronalaženje. Pored toga što poboljšavaju pronalaženje podataka, fajl sistemi su zaduženi i za kalkulaciju praznog prostora, pamćenje direktorijuma (foldera) i njihovih imena, kao i praćenje gde je svaki od fajlova fizički postavljen na disku. Osnovna jednici su fajlovi koji su organizovani u direktorijume.

Najjednostavnija implemtacija fajl sistema je VSFS (Very Simple File System). Ovaj sistem predstavlja pojednostavljenu verziju UNIX sistema i kao takav služi da se uvedu neki od osnovih načina čuvanja podataka na disku koji su razumljivi operativnom sistemu i metode pristupa koje se mogu naći u danasnjim sistemima. Fajl sistem je softver. Za njegovo bolje funkcionisanje nije potrebno dodavanje novih hardverskih komponenata, mada treba obratiti pažnju na karakteristike uređaja, kako bi bili sigurni da će sistem raditi bolje.

Postoje velike razlike u fajl sistemima i to pre svega zbog velike fleksibilnosti prilikom njegove izrade. Svi fajl sistemi imaju različite strukture podataka kao i određene prednosti i mane u odnosu na ostale sisteme. Pa tako implementacija fajl sistema varira od sistema do sistema. Neki od operativnih sistema prepoznaju samo jedan fajl sistem, a neki više njih.

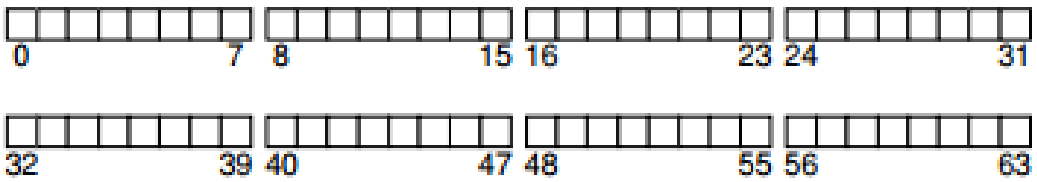
DOS, Windows 3.xx, 95, OS/2,	*FAT*
Windows 95SE, 98, 98SE, Me	*FAT, FAT32*
Windows NT, 2000, XP	*FAT16, FAT32, NTFS*
OS/2, starije verzije NT-a	*HPFS*
Netware serveri	*NetWare fajl sistem*
Linux	*Linux Ext2, Linux Swap*

Ono što je bitno da se razume kada pričamo o fajl sistemu je na koji način se podaci i metapodaci čuvaju na disku, šta se dešava kada prispupimo nekom fajlu i koje metode pristupa se koriste za čitanje i pisanje fajlova.

Fajl sistem ukazuje na dva različita aspekta. Prvi je struktura podataka u fajl sistemu, odnosno način na koji se čuvaju podaci i metapodaci na disku od strane fajl sistema. VSFS radi sa jednostavnim strukturama podataka kao što su nizovi i neki objekti, dok sa druge strane složeni fajl sistemi koriste komplikovanije strukture kao što su stabla. Drugi aspekt fajl sistema predstavljaju pristupne metode kao što su open(), read(), write() i njima slične metode nad strukturama. Takođe, koje strukture su pročitane tokom izvršenja nekog određenog sistemskog poziva; Koje su napisane; koliko efikasno su svi ovi koraci obavljeni.

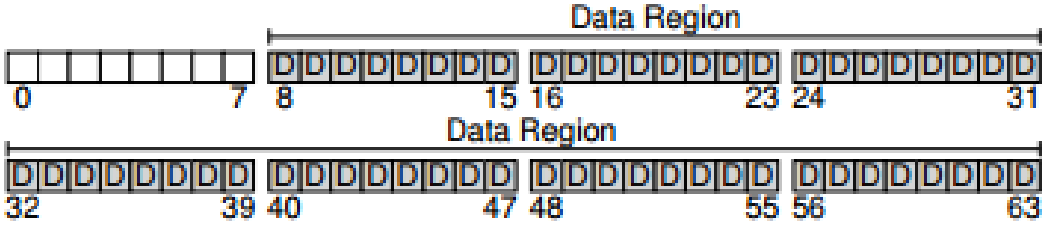
2. Organizacija fajl sistema

Kako bi pravilno organizovali fajl sistem, prva stvar koju je potrebno uraditi je podela diska na blokove. Jednostavni fajl sistemi koriste blokove fiksne veličine, gde je najčešće korišćena veličina 4KB. Particija diska sastoji se iz niza blokova fiksne veličine koji su numerisani od 0 do (N-1), gde N predstavlja veličinu particije na disku. Kako bi predstavili jedan jednostavan fajl sistem, uzećemo kao primer mali disk sa 64 bloka (Slika 1.).



Slika 1: Particije na disku

Kada razmišljamo šta nam je potrebno da sačuvamo na blokove prvo što nam pada na pamet su korisnički podaci. Većinu prostora na svakom disku zauzimaju upravo oni. Deo diska na kome čuvamo korisničke podatke nazivamo korisnički region i on uglavnom zauzima poslednje blokove na disku (Slika 2.).

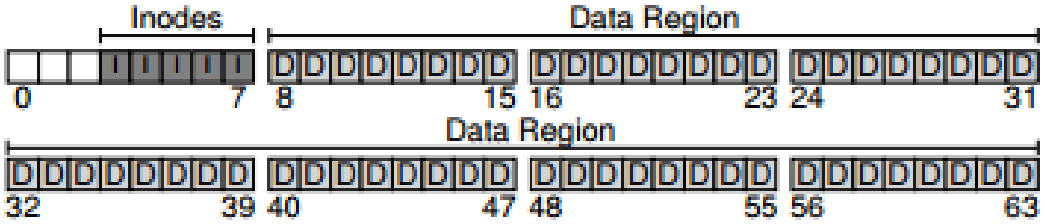


Slika 2: Korisnički podaci na disku

Fajl sistem mora da ima informacije vezane za svaki fajl. Ta informacija je ključni deo metapodataka i prati stvari kao što su blokovi podataka u korisničkom regionu koji predstavljaju jedan fajl, veličina fajla, vlasnik i prava pristupa, vreme promena, lokacija i slične informacije. Za čuvanje ove informacije, fajl sistem koristi strukturu indeksnih čvorova (i-node). Prostor na disku u kome čuvamo podatke o indeksnim čvorovima nazivamo indeksnom tabelom (Slika 3.).

Pretpostavićemo da se za indeksne čvorove koristi pet blokova. Indeksni čvorovi nisu uvek veliki (128byte ili 256byte), pa tako ako uzmemo da je veličina jednog čvora 256bytes, zaključujemo da jedan blok od 4KB može da sadrži 16 čvorova. Tako na ovom primeru u indeksnoj tabeli možemo smestiti 80 indeksnih čvorova. Ovaj broj predstavlja maksimalni broj fajlova koji se mogu naći na ovom fajl sistemu koji se sastoji od 64 bloka na particiji. Međutim,

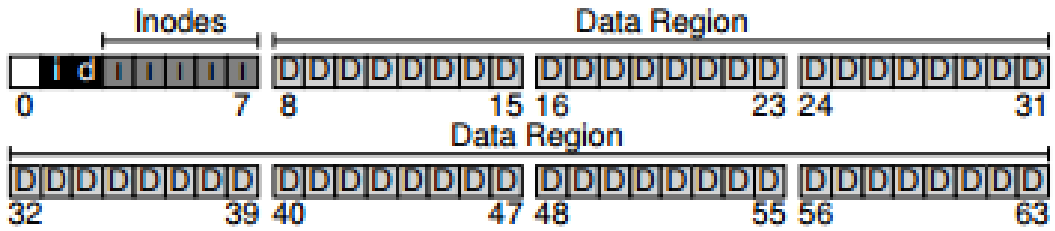
u zavisnosti od veličine diska, fajl sistem bi mogao da izdvoji veći prostor za indeksnu tabelu i samim tim čuva veći broj fajlova na disku.



Slika 3: Indeksna tabela

Gledajući ovaj primer, trenutno imamo blokove podataka(D) i indeksne čvorove(I), ali nedostaje još komponenata. Jedna od osnovnih zadataka svakog fajl sistema jeste da prati da li su indeksni čvorovi i blokovi podataka slobodni.

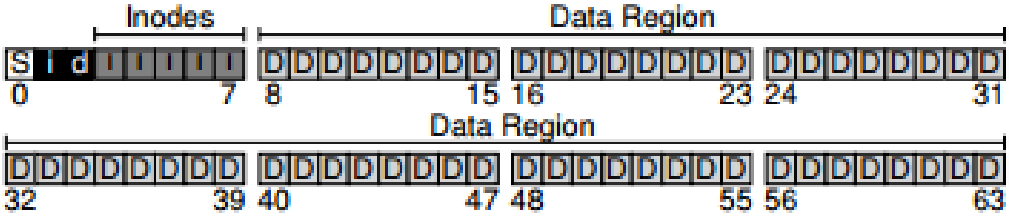
Za praćenje slobodnog prostora na disku postoji mnogo različitih struktura. Na primer, možemo koristiti listu pokazivača za prvi slobodni blok, koji pak pokazuje na pravi naredni slobodan blok i tako dalje. Još se može koristiti i jednostavna i veoma popularna bitmapa i to jedna za korisnički region, a druga za indeksnu tabelu. Bitmapa predstavlja jednostavnu strukturu u kojoj bit 0 označava da je blok slobodan, dok bit 1 oznava zauzetost. Na slici 4. je prikazan novi raspored fajl sistema sa dodatom bitmapom indeksne tabele i bitmapom korisničkog regiona.



Slika 4: Bitmape indeksnih čvorova i korisničkih podataka

Blok od 4Kb je previše za bitmapu, pošto se ona može čuvati na mnogo manjem prostoru. U ovom slučaju, jedna bitmapa može da prati 32 hiljade objekata, a opet mi imamo samo 80 indeksnih čvorova i 56 blokova sa podacima. Pošto u ovom fajl sistemu ima prostora, mi čuvamo bitmape u dva bloka radi jednostavnosti.

Ostao je još jedan slobodan blok na disku, prvi (0-ti). Njega zovemo superblok i označavamo ga sa S. On sadrži informacije o svim fajlovima u sistemu, uključujući na primer broj blokva na disku, koliko indeksnih čvorova postoji, koliko blokva sa podacima, gde počinje indeksna tablela, gde su prvi podaci na bloku, maksimalna veličina fajla i slično. Uglavnom obuhvata i takozvani „magic number“ koji predstavlja tip datoteke. Tako prilikom startovanja operativnog sistema, prvo se učitava superblok, inicijalizuju se neki parametri i onda dodaje ostatak fajl sistema. Sistem će tako znati tačno gde se šta nalazi na disku.

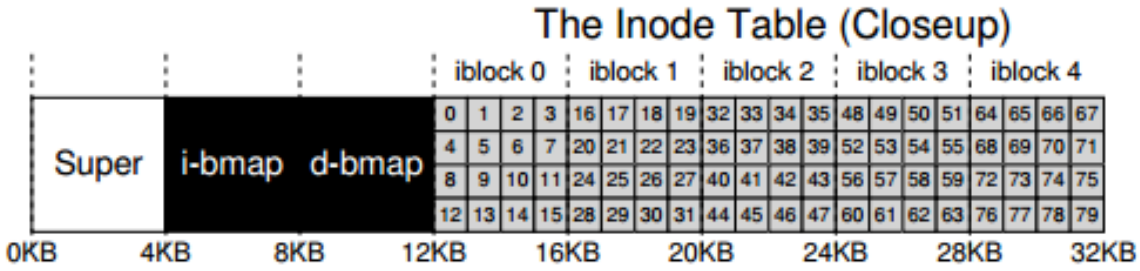


Slika 5: Superblok

3. Indeksni čvor (i-node)

Jedan od najvažnijih struktura na fajl sistemu su indeksni čvorovi. Indeksni čvor je upravljačka struktura koja sadrži ključne informacije koje su potrebne operativnom sistemu za određenu datoteku. Ime je nastalo u UNIX i eventualno ranijim sistemima, tako što su se u nizu čuvale vrednosti, a te vrednsoti bi omogućavale da se pomoću njih nađe čvor.

*Svaki indeksni čvor predstavnja broj, koji se zove indeksni broj (i-number). U VSFS kada se setuje vrednsot čvoru, možemo direktono izračunati gde se na disku odgovarajući čvor nalazi. Na primer, uzmimo indeksnu tabelu sa slike 6., dakle imamo 20KB za indeksnu tabelu, odnosno 5*4KB, i pretpostavimo da svaki indeksni čvor iznosi 256 bytes. Takođe ćemo pretpostaviti da indeksna tabela počinje na 12KB, tj. superblok počinje od 0KB, indeksna bitmapa od 4KB, bitmapa podataka od 8KB i nakon toga sledi indeksna tabela.*



Slika 6: Zaglavlje fajl sistema

*U ovom primeru želimo da pročitamo indeksni čvor 32. Fajl sistem će prvo izračunati deo u indeksnoj tabeli (32 * sizeOf (inode) ili 8192byte). Nakon toga dobijenu vrednost iz indeksne tabele dodajemo na početnu vrednost same tabele, i na taj način izračunavamo adresu željenog bloka. Bitno je znati da disk nije adresiran po byte-ovima, već da se sastoji od velikog broja blokava, koji uglavnom iznose 512 byte. Kako bismo našli blok koji sadrži indeksni čvor 32, fajl sistem mora da pročita sektor (20 * 1024/512 = 40), kako bi došao do željenog bloka. Generalno, adresa sektora može se izračunati na sledeći način:*

blok = (indeksni broj * veličina indesne tabele) / veličinom bloka

sektor = ((blok * veličina bloka) + početna adresa indeksne tabele) / veličinom sektora

Unutar svakog indeksnog čvora nalaze se sve potrebne informacije vezane za fajl poput tipa (fajl, direktorijum i sl.), veličine, broja blokova koje zauzima, vlasnika, ko može da mu pristupa i na koji način, vremenske informacije (datum kreiranje, poslednje promene, poslednji pristup i sl.). Takođe sadrži i informaciju o tome gde se njegovi blokovi podataka nalaze na

disku. Svaka takva informacija predstavlja metapodatak, tj. svaka informacije unutar fajl sistema koja nije obična korisnička informacija definiše se kao metapodatak.

Jedna od najvažnijih odluka u dizajnu indeksnih čvorova je kako označiti gde se blokovi nalaze. Jedan od načina bio bi da ima jedan ili više različitih pokazivača unutar indeksnog čvora, tako da svaki pokazivač pokazuje na jedan blok diska koji pripada fajlu.

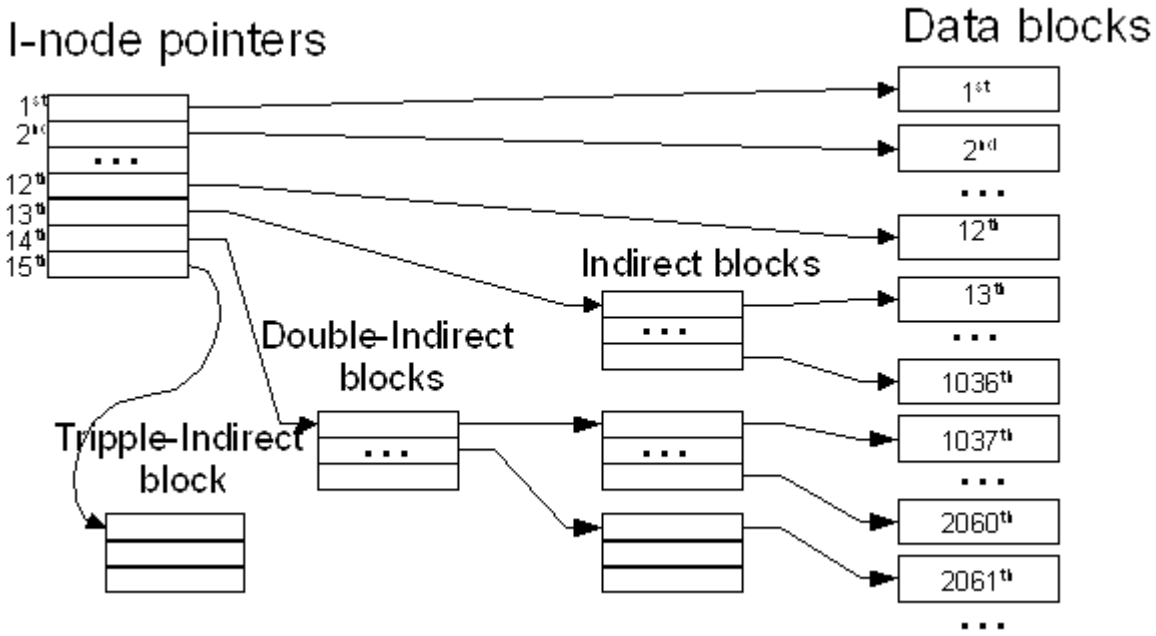
4. Višestruki indeksi (multi-level index)

Kako bi podržali veće fajlove, dizajneri fajl sistema morali su da uvedu različite strukture unutar indeksnih čvorova. Jedna od ideja je uvođenje specijalnih pokazivača koji su poznati kao indirektni pokazivači (indirect pointer). Umesto da pokazuju na blok koji sadrži podatke, indirektni pokazivači, usmereni su ka bloku pokazivača, a svaki od njih pokazuje na korisničke podatke.

*Indeksni čvor može da ima fiksan broj direktnih pokazivača i jedan indirektan pokazivač. U tom slučaju, ako je fajl dovoljno veliki, pridružuje se indeksni čvor i slot indeksnog čvora sa indirektinim pokazivačima je spreman da ukaže na njega. Kao primer uzećemo da imamo 12 direktnih pokazivača, dakle možemo smestiti fajl veličine 4144KB odnosno (12 + 1024) * 4KB. U opštem obliku:*

Veličina fajla = (broj direktnih pokazivača + ostali pokazivači) * veličinom bloka

Uz ovakav pristup vrlo lako se čuvaju i veći fajlovi. Kako bismo to uradili dodajemo još jedan indirektan blok pokazivača – dvostruki indirektni pokazivači. Ovaj blok sadrži pokazivač koji sadrži pokazivač koji vodi ka jednostrukom indirektnom bloku, od kojih svaki sadrži pokazivač koji pokazuje na blok podataka. Dvostruki indirektni blok podržava fajlove veće od 4GB. U slučaju da je potrebno čuvati fajlove veće od 4GB, tu je trostruki indirektni pokazivač.



Slika 7: Indirektni indeksni čvorovi

Ovo neuravnoteženo drvo poznato je kao „multi-level index“ pristup pokazivanja na blok podatka. Kada imamo 12 direktnih pokazivača, jedan indirektan i jedan dvostruki indirektan

blok, pritom pretpostavićemo da je blok veličine 4KB i da su pokazivači veličine 4byte, ova struktura može da primi fajl veličine 16GB ($12 + 256 + 256^2 + 256^3$).

Drugi pristup jeste da se koristi extents umesto pokazivača. Extents predstavlja pokazivač diska+dužina fajla (u blokovima). Pa tako umesto da se traži pokazivač za svaki blok fajla, potreban je samo pokazivač i dužina kako bi se odredila tačna lokacija fajla na disku. Ovaj pristup je ograničen pošto se može javiti problem oko pronalaženja slobodnog prostora prilikom raspoređivanja fajla.

Kada uporedimo ova dva pristupa zaključujemo da je pristup zasnovan na indirektinim pokazivačima najfleksibilniji, ali koristi i veliku količinu metapodataka po fajlu. Extents pristup je manje fleksibilan, ali više kompaktan i radi jako dobro kada ima dovoljno slobodnog prostora na disku.

Mnogi istaraživači koji proučavaju fajl sisteme i kako se oni koriste su otkrili neke „istine" koje opstaju i važe već decenijama unazad. Jedna od njih je da su fajlovi mali, pa baš zato ovo neuravnoteženo drvo često biva izabrano kao najbolje rešenje, za čuvanje fajlova. U njemu imamo 12 direktnih pokazivača koji pokazuju na 48KB podataka, a ukoliko se koriste nešto veći fajlovi dodaje se odgovarajući indirektni blok.

4.1. Povezana lista (Linked – based)

Još jedan jednostavan pristup u dizajnu indeksa je povezana lista. Unutar indeksnog čvora, umesto da postoji više pokazivača, potreban je samo jedan koji bi upućivao na prvi blok fajla. Kako bismo kontrolisali veći fajl, dodajemo još jedan pokazivač na kraj tog bloka i tako dalje.

Ovaj način se nije baš pokazao kao uspešan. Na primer, ako imamo čitanje poslednjeg bloka iz fajla ili ako je potrebno nasumično da pristupimo nekom bloku, moramo proći gotovo celu listu. Kako bi se poboljšao ovaj pristup, neki sistemi će zadržati tabelu u memoriji o informacijama linka, umesto da čuvaju sledeći pokazivač, sa samim blokom podataka. Tabela je označena nazivom bloka podataka D, sadržaj unosa je D-ov sledeći pokazivač, tj. naziv sledećeg bloka u fajlu koji prati D. Vrednost 0 se tako nalazi u tabli i uglavnom označava kraj fajla ili neki drugi marker koji označava slobodan blok. Ako imamo ovakvu tabelu dobijamo šemu koja omogućava uspešan pristup nasumičnim fajlovima i to tako što će prvo da skenira tabelu kako bi našla željeni blok, a onda i da mu pristupi direktno.

Ovakva šema predstavlja klasičan stari Window-s fajl sistem baziran na jednostavnoj šemi raspoređivanja koja je zasnovana na vezama. Razlikuje se od UNIX fajl sistema, tako što ne postoje indeksne tabele, već se u direktorijumu čuvaju metapodaci i odnose se direktno na prvi blok fajla, što sprečava stvaranje stabilnih veza.

5. Organizacija fajla

Direktorijumi imaju veoma jednostavnu organizaciju, kako u VSFS tako i u mnogim drugim sistemima. U suštini direktorijum sadrži listu parova (ime, broj indeksnog čvora), pa tako za svaki fajl ili direktorijum u datom direktorijumu, postoji niz i broj u blokovima podataka tog direktorijuma. Svaki naziv ima i svoju dužinu (pretpostavimo da postoje imena različitih veličina).

Na primer, pretpostavimo da direktorijum dir (indeksni broj 5) ima tri fajla (foo, bar i foobar) i da su njihovi brojevi 12, 13 i 24, onda podaci na disku izgledaju ovako:

inum	reclen	strlen	name
5	4	2	.
2	4	3	..
12	4	4	foo
13	4	4	bar
24	8	7	foobar

Slika 8: Direktorijum „dir“

U ovom primeru se vidi da svaki unos ima indeksni broj, dužinu snimka (svi bajtovi koji se odnose na ime + slobodni prostor ako postoji), dužina niza (stvarna dužina imena) i samo ime unosa. Svaki direktorijum ima dva dodatna unosa, . („dot“) i .. („dot-dot“). Diretorijum „dot“ je trenutni direktorijum, u ovom slučaju dir, dok je „dot-dot“ povezan direktorijum, u ovom slučaju to je root.

Brisanje fajla može da ostavi prazan prostor u sred direktorijuma i trebalo bi da postoji neki način da se i to obeleži. Upravo brisanje je razlog zašto se koristi dužina snimka. Novi snimak tako može da iskoristi stari, kako bi sačuvao veći unos i da ima dodatni prostor unutar njega.

Često fajli sistem tretira direktorijum kao posebnu vrstu fajla. On ima indeksni čvor u tabeli (označen je kao direktorijum umesto regularnog fajla). Direktorijum ima blokove podataka na koje pokazuje indeksni čvor. Ovi blokovi podataka nalaze se u regionu blokova. Treba takođe napomenuti da ova prosta linearna lista unosa nije jedini način da se sačuva takva informacija. Neki sistemi čuvaju direktorijume u obliku B-stabla i na taj način brže dolaze do željenog fajla nego što je to slučaj sa prostim listama.

6. Upravljanje slobodnim prostorom na disku

Fajl sitem mora pratiti koji indeksni čvorovi i blokovi su slobodni, a koji nisu, tako kada smešta novi fajl ili direktorijum može naći prostor za njega. Takvo upravljanje slobodnim prostorom važno je za sve fajl sisteme. U VSFS imamo dve jednostavne bitmape koje vode računa o slobodnom prostoru na disku.

Postoji mnogo načina za upravljanje slobodnim prostorom. Bitmape su samo jedan od njih. Neki rani fajl sistemi koriste liste, gde jedan pokazivač iz superbloloka pokazuje na prvi slobodan blok. Unutar tog bloka pokazivač na sledeći slobodan blok i na taj nacin se formira lista slobodnih blokova. Kada je potreban blok, koristi se glava liste i u skladu sa tim ažurira lista. Savremeni fajl sistemi koriste drugačiju strukturu podataka. Na primer, neki oblik B-stabla kako bi prikazali kompaktnu sliku delova diska koji su slobodni.

Kada napravimo datoteku moramo izdvojiti indeksni čvor za taj fajl. Sistem datoteka će na taj način pretražiti bitmapu indeksnih čvorova i pronaći koji je slobodan i njega dodeliti fajlu. Fajl sistem označava indeksni čvor koji se koristi sa 1 i na kraju ažurira bitmapu sa tačnim informacijama. Slično se dešava i kada se dodaju podaci u blok.

Neki drugi faktori takođe mogu doći do izražaja prilikom određivanja blokova podataka na datoteci. Neki Linux fajl sistem, kao što je ext2 ili ext3, traži niz slobodnih blokova kako bi novi fajl bio kreiran i smesten u blokove. Pronalaze se takvi blokovi, a zatim se obrađuju za novokreirani fajl, čime sistem garantuje da će delovi datoteke biti susedni i samim tim će performance biti poboljšane.

7. Pristupanje fajlu

Sada kada imamo neku predstavu o tome kako su datoteke i direktorijume sačuvani na disku, možemo da pratimo aktivnosti tokom procesa čitanja ili pisanja fajla. Razumevanje ovoga je jako bitna stvar kako bi razumeli fajl sistem. Kako bi lakse objasnili proces pristupanja fajlu, pretpostavićemo da fajl sistem već postoji, što znači da je superblok u memoriji, a indeksni čvorovi i direktorijumi na disku.

7.1. Čitanje fajla sa diska

U ovom primeru ćemo jednostavno otvoriti fajl, pročitati ga i nakon toga zatvoriti (foo/bar). Pretpostavićemo da fajl ima samo 4KB što znači da je smešten u jednom bloku. Kada izvršimo naredbu open("/foo/bar", O_RDONLY) fajl sistem prvo mora da nađe indeksni čvor za bar datoteku i da dobije neke osnovne informacije o fajlu (dozvole, velicinu I sl). Kako bismo to uradili fajl sistem mora da bude u stanju da pronađe indeksni čvor, ali sve što sada ima je puna putanja. Tako da fajl sistem mora proći kroz putanju i na taj način naći željeni indeksni čvor.

Sve putanje počinju u korenu sistema (root) u osnovnom direktorijumu koji je označen sa /. Dakle, prva stvar koju fajl sistem radi je čitanje indeksnog čvora iz root fajla. Da bismo pronašli indeksni čvor moramo znati indeksni broj (i-number). Obično nalazimo indeksini broj fajla ili direktorijuma u njegovom roditeljskom fajlu (koren nema roditelja po definiciji). Zato indeksni broj root-a mora biti poznat; Fajl sistem mora da zna šta je to kada postavlja fajl sistem. U većini UNIX sistema indeksni broj korena je 2. Znači, proces koji je započeo otvaranje fajla čita sadržaj iz bloka čvora 2.

Kada se indeksni čvor pročita, fajl sistem može pogledati unutar njega i pronaći pokazivač na blok podataka koji sadrži root direktorijum. Fajl sistem će pomoću pokazivača čitati kroz fajl ili dikretoriju u ovom slučaju tražeći foo. Čitajući jedan po jedan deo bloka podataka, naćiće foo. Kada ga pronađe, uzeće negov indeksni broj (recimo 44) koji mu je potreban kako bi nastavio putanju.

Sledeći korak je da se rekurzivno prolazi kroz putanju sve dok se ne pronađe željeni čvor. U ovom primeru, fajl sistem čita sadržaj bloka foo i njegove podatke, i tako nailazi na bar. Završni korak otvaranja je čitanje bar čvora iz memorije. Fajl sistem onda radi konačnu proveru, raspoređuje deskriptor kako bi otvorio fajl tabelu (tablela koja čuva informacije o fajlovima) i vraca je korisniku. Kada je jednom otvori, program će zatim izdati sistemski poziv

za čitanje iz datoteke. Prvo se čita prvi blok fajla (offset = 0) , zajedno sa indeksnim čvorom koji će pronaći lokaciju tog bloka. Takođe može ažurirati indeks sa novim "vremenom korišćenja". Čitanje će dalje ažurirati otvorenu fajl tabelu u memoriji za deskriptor, ažuriranjem datoteke offset će se uvećati i preći na čitanje drugog bloka fajla.

U jednom trenutku fajl će biti zatvoren, dok će podaci iz fajl tabele biti izbrisani nešto kasnije kada više ne budu bili potrebni. Čitav ovaj proces prikazan je na slici 9. Bitno je napomenuti da bitmapa služi samo prilikom smeštanja direktorijuma na disk, za sve ostalo koristi se indesni čvorovi i broj, kao i direktorijum, jer oni imaju sve što je potrebno.

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
open(bar)			**read**			**read**				
				read						
							read			
					read					
read()										
								read		
					write					
read()										
					read					
									read	
					write					
read()										
					read					
										read
					write					

Slika 9: Postupak čitanja fajla sa diska

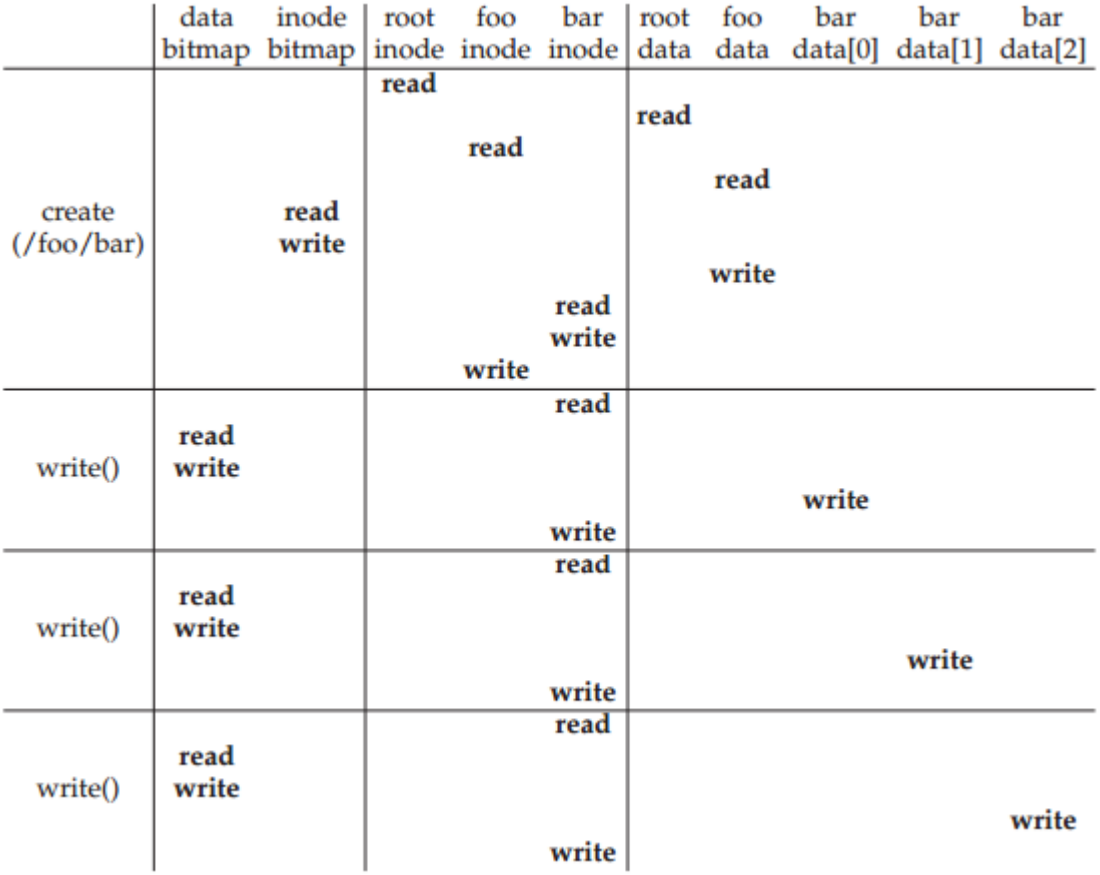
7.2. Pisanje u fajl na disku

Pisanje u fajl je sličan proces procesu čitanja. Prvo fajl mora biti otvoren i aplikacija mora izdati naredbu write() kako bi se ažurirao fajl sa novim sadržajem. Nakon toga fajl se zatvara.

Za razliku od čitanja, pisanje u fajl sadrži još jedan korak to je alokacija bloka za upis. Kada upisujemo u nov fajl, svaki upis mora da odredi koji blok će se alocirati za taj fajl, uz to ažurira i neke druge strukture. Tako se svaki upis sastoji od pet ulazno-izlaznih operacija: čitanje i upisivanje u bitmamu, čitanje i upisivanje u indeksni čvor i na kraju upis u sam blok.

Količina komunikacija je još gora kad se uzme u obzir jednostavna operacija kao što je kreiranje fajla. Da bi se fajl kreirao fajl sistem ne samo da mora da alocira indeksni čvor nego i prostor u okviru diriktorijuma koji će sadržati novi fajl. Ukupna količina ulazno-izlaznih komunikacije je jako velika: jedno čitanje bitmape (nađi slobodan čvor), jedno pisanje u bitmapu (obeleži čvor), pisanje u novi indeksni čvor (inicijalizacija), upisivanje podataka u

direktorijum i jedno čitanje i pisanje u indeksni čvor direkrorijuma kako bi se ažurirao. Ako veličina direktorijuma treba da se povećava da bi primila nov zapis, dodatne ulazno-izlazne operacije će biti potrebne, a sve to da bi se kreirao fajl.



Slika 10: Kreiranje fajla na disk

Na slici 10, pokazano je kako se kreira fajl /foo/bar i tri bloka za upis u njega, kao i tok upisa svakog od tri bloka veličine 4KB. Čitanje i pisanje je grupisano u jedan sistemski poziv i neki grubi redosled kada će se oni izvršiti ide odozgo na dole gledajući sliku. Kao što vidimo potrebno je deset ulazno-izlaznih operacija da se prođe putanjom i da se konacno kreira fajl. Takođe možemo da vidimo da svaka alokacija upisa košta pet operacija, par za čitanje i ažuriranje čvora i bitmape, a onda i konačno upisivanje samih podataka.

8. Keširanje i zaštita

Kao što smo već pričali, čitanje datoteka ali i upisivanje u njih, može biti skupo izazivajući veliki broj ulazno-izlaznih operacija nad sporim diskom. Kako bi se otklonio veliki problem pri funkcionisanju, većina fajl sistema koristi sistemsku memoriju za keširanje važnih blokova.

Svako otvaranje fajla bez keširanja bi zahtevalo najmanje dva čitanja na svakom nivou u hijerarhiji direktorijuma (jedan za čitanje indeksnog čvora direktorijuma i najmanje jedan za čitanje njegovih podataka). Pa tako kada su u pitanju neki fajlovi koji imaju duge putanje, na primer /1/2/3/.../100/file.txt, samo za otvaranje fajla bilo bi potrebno stotine čitanja. Kako bi se rešili ovog problema, raniji sistemi su uveli keš fiksne veličine u koji su smeštali najčešće korišćene blokove.

Keš diska je bafer u glavnoj memoriji za sektore diska. Pa tako kada se napravi U/I zahtev za određeni sektor, prvo se proverava da li je taj sektor u kešu, ako jeste zahtev se zadovoljava preko keša. Ukoliko nije, traženi sektor se učitava u keš diska sa diska. Najčešći algoritam za zamenu u kešu je najmanje skoro korišćeni blok (LRU). Keš se sastoji od steka blokova, gde je najskorije referenciran blok na vrhu steka. Kada se blok koji je u kešu referencira on sa svoje pozicije prelazi na vrh steka, dok ukoliko traženi blok nije u kešu uklanja se blok koji je na dnu steka, a ulazni blok se stavlja na vrh.

Ako sada razmotrimo efekat keširanja prilikom upisa na disk primetićemo da pisanje saobraćaja mora da ide na disk kako bi postali trajni. Tako, keš ne služi kao ista vrsta filtera za pisanje saobraćaja kao za čitanje. Bafer za pisanje (kako se ponekad naziva) sigurno ima neke svoje prednosti pa tako neki upisi se mogu potpuno izbeći tako što se odlažu; Na primer, ako aplikacija kreira datoteku, a zatim je briše, odlaganje upisa se odražava na stvaranje datoteke na disku i izbegava ih u potpunosti. U ovom slučaju, lenjost (u pisanju blokova na disku) je vrlina.

Mnogi moderni operativni sistemi integrišu stranice virtuelnu memoriju i stranice fajl sistema u jedinstvenu keš stranicu [S00]. Na ovaj način, memorija može biti fleksibilnije premeštena preko virtuelne memorije i sistema datoteka, zavisno od toga kome je potrebno više memorije u datom trenutku.

Iz navedenih razloga, najsavremeniji sistemi datoteka štite upise u memoriji za bilo gde između pet i trideset sekundi, što predstavlja kompromis. Ako sistem padne pre nego što su ispravke prosleđene na disk, isprave se gube. Međutim, držeći upise duže u memoriji, performanse se mogu poboljšati reakcijom, rasporedom i izbegavanjem upisa. Neke aplikacije

(kao što su baze podataka) nemaju ovaj kompromis. Stoga, da bi se izbegao neočekivani gubitak podataka zbog upisa zaštite, oni jednostavno prisiljavaju upise na disk, pozivom fsinc ().

9. Zaključak

Jasno je da je problem upravljanja datotaka jedan od najzanimljivijih pitanja kada su u pitanju operativni sistemi, kao i uspešno upravljanje samim diskom. Upoznati smo sa razvojem mnogo novih fajl sistema, ali i dan danas mnogi imaju strukturu sličnu onoj o kojoj smo mi pričali.

Kao što smo već videli kako bi jedan fajl sistem funkcionisao mora da postoji informacija o svakom fajlu koja se čuva u indeksnim čvorovima (inode), a tu je i indeksni broj (inumber) koji je pokazivač na blok sa podacima. Ne treba zanemariti i neke druge strukture poput bitmapa, povezanih lista i mnogih drugih. Takođe bitna stvar su indirektne indeksi koji omogućavaju čuvanje velikih fajlova i po nekoliko GB. Jedna od najbitnijih stavi kada je implementacija fajl sistema u pitanju je da se disk mora tretirati kao disk, a ne kao RAM.

10. Literatura

- *http://pages.cs.wisc.edu/~remzi/OSTEP/file-implementation.pdf*
- *http://minnie.tuhs.org/CompArch/Lectures/week11.html*
- *http://www.cs.pomona.edu/~markk/filesystems/bsd.html*
- *http://www.tldp.org/LDP/tlk/fs/filesystem.html*
- *http://www.sk.rs/2010/11/sksc01.html*