Prirodno-matematički fakultet Kragujevac
Institut za matematiku i informatiku

Seminarski rad iz predmeta Operativni sistemi 2
Tema: **FSCK i vođenje dnevnika**

Student:
Veljko Jelenić 65/11

Profesor:
dr Miloš Ivanović

Kragujevac, oktobar 2016. godine

Sadržaj

1. Uvod

Strukture podataka sistema datoteka moraju biti istrajne, drugim rečima, moraju "preživeti" nepredviđene situacije, kao što su nestanak struje ili pad sistema.

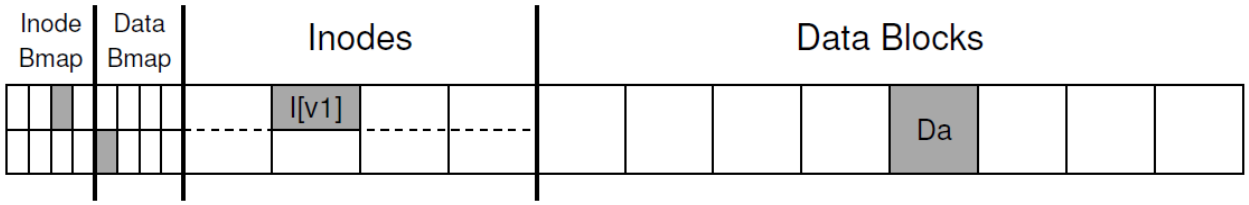
Na primer, šta će se desiti ako nestane struja u toku ažuriranja struktura podataka na disku, ili dođe do pada sistema? Svakako, ovo je situacija koja može da nam se desi, koja može dovesti do gubitka korisničkih podataka, a to ne bismo želeli. Ovaj problem je poznat kao "**crash-consistency problem**".

Na jednosnavnom primeru ćemo objasniti ovaj problem. Zamislimo dve strukture podataka koje se nalaze na disku, struktura A i struktura B. Želimo da ažuriramo ove dve strukture kako bismo kompletirali neku operaciju, jedan od zahteva za ažuriranje će biti prvi opsluživan. Ako dođe do nepredviđene situacije nakon jednog kompletiranog zahteva, struktura podataka će ostati u nekonzistentnom stanju.

U nastavku ćemo razmotriti metode za rešavanje ovog problema. Prvo ćemo govoriti o **fsck (file system checker)** pristupu, koji koriste stariji sistemi datoteka. Potom ćemo objasniti i pristup koji brže oporavlja sistem, poznat kao vođenje dnevnika (**journaling**).

2. Pisanje na disk

Sada ćemo pogledati malo detaljniji primer, kako bismo preciznije objasnili naše istraživanje. Koristićemo jednostavne strukture sistema datoteka više o sistemima datoteka možete pronaći u [2]. Reč je o ažuriranju struktura na disku. Pretpostavićemo da je to dodavanje podatka u već postojeći fajl. Dodavanje se postiže otvaranjem fajla, pozivanjem metode *lseek()* kako bi se pokazivač u fajlu pomerio na kraj, a zatim se prosleđuje blok podataka koji se upisuje i na kraju se zatvara fajl. Vidimo da naš primer (slika 1) poseduje bitmapu indeksnih čvorova (sa 8 bitova, jedan po indeksnom čvoru), bitmapu podataka (8 bitova, jedan po bloku podataka), indeksne čvorove (ukupno 8, numerisani su od 0 do 7, podeljeni su u 4 bloka), i na kraju blokove podataka (ukupno 8, numerisani su od 0 do 7).



Slika 1. Jednostavna struktura sistema datoteka

Alociran je jedan indeksni čvor (I[V1]) i jedan blok podataka (Da), indeksni čvor je markiran u bitmapi indeksnih čvorova, a blok podataka u bitmapi blokova podataka. Oznaka V1 kod indeksnog čvora znači da je ovo prva verzija ovog indeksnog čvora, koji će uskoro biti ažuriran. Prvo ćemo pogledati pojednostavljeni primer unutrašnjosti indeksnog čvora (naravno, u realnosti indeksni čvorovi imaju mnogo više polja) koji je alociran:

owner *: remzi*

permissions *: read-write*

size *: 1*

pointer *: 4*

pointer *: null*

pointer *: null*

pointer *: null*

Veličina fajla je 1 (ima jedan alociran blok), prvi pokazivač pokazuje na blok 4 (na blok Da), ostali pokazivači su setovani na *null*

Kada dodajemo u fajl, potrebno je ažurirati na disku tri strukture: indeksni čvor, novi blok podataka Db, i moramo markirati novo polje u bitmapi blokova podataka. Dakle, tri bloka moramo upisati na disk. Ažurirani indeksni čvor sada izgleda ovako:

owner *: remzi*

permissions *: read-write*

size *: 2*

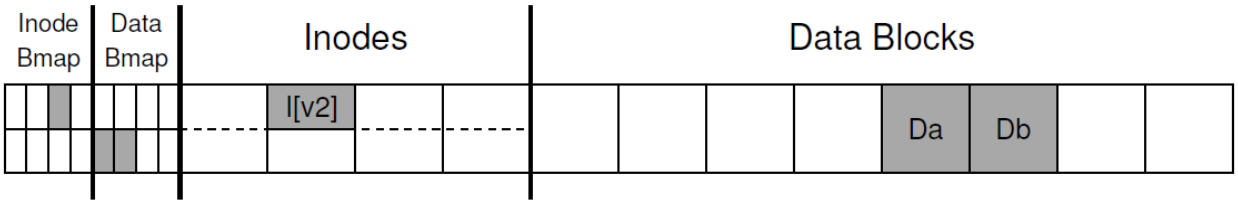
pointer *: 4*

pointer *: 5*

pointer *: null*

pointer *: null*

Ažurirana bitmapa podataka (B[V2]) sada izgleda ovako: 00001100. Na kraju, imamo i nov blok podataka Db, a disk će sada izgledati kao na slici (slika 2).



Slika 2. Jednostavna struktura sistema datoteka posle dodavanja bloka Db

Da bi kompletirao transakciju, sistem datoteka mora izvršiti tri različita upisivanja na disk, jedno za indeksni čvor, jedno za bitmapu (B[V2]) i jedno za blok podataka (Db). Ova upisivanja se ne dešavaju momentalno, moguće je da novi podaci čekaju u glavnoj memoriji neko vreme pre upisa. Kada sistem datoteka odluči da ih upisuje, neophodno je da pošalje zahtev za pisanje disku. Ako dođe do pada sistema nakon jednog ili dva ažuriranja, dolazimo do problema jer nismo uspeli da izvršimo sva tri ažuriranja i sistem ne smemo ostaviti u takvom stanju.

3. Mogući scenariji kod pada sistema

Kako bismo bolje razumeli problem, pogledaćemo primere nepoželjnih scenarija nakon pada sistema. Prvo ćemo opisati situacije u kojima je samo jedno ažuriranje uspelo.

- **Samo je blok podataka (Db) upisan na disk.** U ovom slučaju, blok podataka je upisan na disk, ali ne postoji indeksni čvor koji pokazuje na taj blok i bitmapa je zadržala prethodno stanje, odnosno ne ukazuje na to da je blok alociran. Dakle, ovo je isto kao da ništa nismo upisali. Ovo nije problem iz perspektive sistema datoteka, što se tiče konzistentnosti, ali je problem iz perspektive korisnika, jer je na ovaj način izgubio podatke.
- **Samo je ažuriranje indeksnog čvora (I[V2]) upisano na disk.** U ovom slučaju indeksni čvor pokazuje na adresu (5) koja se nalazi na disku, gde bi trebalo da bude upisan blok podataka Db, ali još uvek nije. Ukoliko čitamo podatke sa te adrese, dobićemo otpad (stari sadržaj sa te adrese).

  Ovako dobijamo novi problem, koji se zove **nekonzistentnost sistema datoteka.** Bitmapa nam govori da na adresi 5 nije alociran blok podataka, a indeksni čvor govori suprotno. Ovaj nesporazum je problem nekonzistentnosti struktura podataka sistema datoteka, i on mora biti rešen.

- **Samo je ažuriranje bit mape (B[V2]) upisano na disk.** U ovom slučaju, bit mapa ukazuje da je blok podataka 5 alociran, ali ne postoji indeksni čvor koji pokazuje na taj blok podataka. Sistem datoteka je ponovo u nekonzistentnom stanju. Ukoliko ovo ostane nerazrešeno, javlja se problem koji je poznat kao **space leak**, sistem datoteka nikada neće upotrebiti blok 5.

Takođe, imamo još tri moguća scenarija, a to su situacije kada su dva od tri ažuriranja uspešno upisana na disk.

- **Indeksni čvor (I[V2]) i bit mapa (B[V2]) su upisani na disk, ali blok podataka (Db) nije.** U ovoj situaciji, metapodaci su konzistentni, indeksni čvor pokazuje na blok podataka 5, a bit mapa ukazuje da je taj prostor alociran. Iz perspektive metapodataka sve izgleda dobro, ali na lokaciji 5 se ponovo nalazi otpad.
- **Indeksni čvor (I[V2]) i blok podataka (Db) su upisani na disk, ali bit mapa nije ažurirana.** Ovde imamo indeksni čvor koji pokazuje na ispravne podatke na disku, ali imamo i nekonzistentno stanje između indeksnog čvora i bitmape.
- **Bit mapa (B[V2]) i blok podataka (Db) su upisani, ali indeksni čvor (I[V2]) nije.** U ovom slučaju imamo nekonzistentno stanje između indeksnog čvora i bitmape podaka. Iako je blok podataka upisan na disk i bit mapa ukazuje na zauzetost, mi ne znamo kojoj datoteci pripada, jer nemamo indeksni čvor koji pokazuje na datoteku.

Objasnili smo neke probleme usled pada sistema – nekonzistentno stanje, **space leak**, čitanje otpada. U nastavku ćemo govoriti o rešavanju ovih problema.

4. Rešenje 1: FSCK

FSCK je alat operativnog sistema *UNIX* koji služi za pronalaženje nekonzistentnog stanja i rešavanje tog problema. **FSCK** radi u dva režima. Obično radi u ne-interaktivnom režimu od strane operativnog sistema. U ovom režimu će biti načinjene samo one promene za koje je operativni sistem siguran da su uvek ispravne. Ukoliko se desi neočekivano nekonzistentno stanje, **fsck** će vratiti nulu kao izlazni status. Nakon toga, operator pokreće **fskc** interaktivno. Kada je u ovom režimu, za svaki problem su predložene akcije za rešavanje. Operator mora da odluči koju akciju da upotrebi.

Osnovni pregled onoga što **fsck** radi:

- **Provera superblok:** Sistem datoteka je opisan pomoću superbloka. Superblok poseduje osnovne informacije o sistemu datoteka, kao što su: broj blokova na disku, koliko ima indeksnih čvorova, maksimalna veličina datoteke i slično. Zbog toga što superblok sadrži važne podatke, sistem datoteka ih kopira radi sigurnosti. Informacije koje su povezane sa superblokom su najpodložnije nekonzistentnosti, zato što se menjaju sa svakom promenom blokova ili indeksnih čvorova u sistemu datoteka. Kod superbloka se proverava veličina sistema datoteka, broj indeksnih čvorova, broj slobodnih blokova i broj slobodnih indeksnih čvorova. Veličina sistema datoteka mora biti veća od broja blokova koji su alocirani. Iako ne postoji način da zaista proveri ove veličine, pošto su statistički određene, **fsck** može da proveri da li su veličine razumne. Ukoliko se pojavi neka sumnja, sistem ili administrator mogu odlučiti da upotrebe kopiju superbloka.

- **Provera slobodnih blokova: FSCK** proverava da li su svi blokovi koji su markirani kao slobodni zaista slobodni. **Fsck** proverava da li je broj slobodnih blokova plus broj blokova za koje indeksni čvorovi tvrde da su zauzeti jednak ukupnom broju blokova.

  Informacije koje se nalaze u superbloku sadrže ukupan broj slobodnih blokova u sistemu datoteka. **Fsck** poredi taj broj sa brojem slobodnih blokova koje je pronašao u sistemu datoteka. Ako nisu isti brojevi, menja pogrešan broj sa stvarnim brojem slobodnih blokova.

  Takođe, informacije koje se nalaze u superbloku imaju podatak o broju slobodnih indeksnih čvorova u sistemu datoteka. Isti postupak se radi kao i za broj slobodnih blokova.

Ukoliko dođe do nesaglasnosti između indeksnih čvorova i bitmapa, veruje se indeksnim čvorovima.

- **Provera stanja indeksnih čvorova:** Svaki indeksni čvor se proverava. Proveravaju se sekvencijalno. Proverava se da nije stanje nekog čvora nekonzistentno, što uključuje provere formata i tipa, dupliranih blokova, pogrešnih blokova i veličine indeksnog čvora.

  Svaki indeksni čvor sadrži režim koji opisuje njgov tip i stanje. Indeksni čvor može da ima sledeća stanja: ***unallocated***, ***allocated***, i ni nealocirano ni alocirano. Ovo poslednje stanje govori da indeksni čvor nije dobro formatiran. Jednini mogući način za *fsck* je da obriše ovaj indeksni čvor.

- **Linkovi indeksnih čvorova:** Svaki indeksni čvor ima ukupan broj linkova.
  Broj linkova predstavlja broj različitih direktorijuma koji sadrže referencu na fajl. Kako bi verifikovao broj linkova, ***fsck*** prolazi kroz celo stablo direktorijuma, počevši od korena stabla i pravi sopstveni broj linkova za svaki fajl i direktorijum u sistemu datoteka. Ukoliko se novi broj razlikuje od broja koji se nalazi u indeksnom čvoru, broj u indeksnom čvoru se mora ispraviti. Ukoliko se otkrije neki alocirani indeksni čvor, ali nijedan direktorijum nema referencu na njega, pomera se u direktorijum *lost+found*.

- **Duplikati:** ***fsck*** proverava da li postoje duplikati pokazivača. Recimo, situacije gde dva različita indeksna čvora pokazuju na isti blok. U ovim situacijama, ukoliko je jedan indeksni čvor očigledno loš, može da se obriše. Alternativno, blok na koji pokazuju može biti kopiran, tako da oba indeksna čvora imaju svoj blok.

- **Pogrešni blokovi:** Provera pogrešnih blokova se vrši tako što se prolazi kroz listu svih pokazivača. Pokazivač se smatra neispravnim ako pokazuje na nešto van dometa.
  Primer: Pokazuje na adresu koja ukazuje na blok koji je veći od veličine particije. U ovom slučaju se obriše pokazivač.

- **Provera direktorijuma:** ***Fsck*** ne razume sadržaj korisničkih fajlova. Direktorijumi sadrže formatirane informacije od strane sistema datoteka. ***Fsck*** proverava sadržaj svakog direktorijuma, uveravajući se da su na početku unutrašnjosti direktorijuma „.“ i „..“.

FSCK i slični pristupi imaju veliki problem: previše su spori. Ukoliko je disk veliki, skeniranje celog diska radi pronalaženja svih alociranih blokova i čitanje celog stabla može da potraje. Previše je skupo skeniranje celog diska da bi se rešili problemi koji su nastali. Možemo da uporedimo sa sledećom situacijom: zamislite da vam ispadnu ključevi na pod u spavaćoj sobi, a onda započnete pretragu ključeva tako što počevši od podruma pretražimo svaku prostoriju. Može, ali je gubljenje vremena. ***FSCK*** je detaljnije opisan u [3].

5. Rešenje 2: Vođenje dnevnika (Journaling)

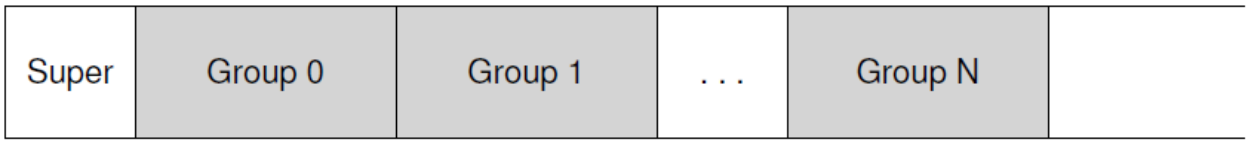
Verovatno, najbolji ideja za rešavanje problema konzistentnosti je ideja koja se koristi u sistemu za upravljanje bazama podataka. Ta ideja poznata kao ***write-ahead logging*** izmišljena je za rešavanje ovakvih problema. U sistemima datoteka, ***write-ahead logging*** nazivamo ***journaling***, odnosno vođenje dnevnika. U nastavku ćemo govoriti o vođenju dnevnika, koje je opisano u [1].

Ideja je sledeća: pre upisivanja novih podataka na disk, prvo zapisati malu napomenu na nekoj lokaciji na disku, koja opisuje šta želimo da uradimo. Pisanje napomene je ***write ahead*** deo, i upisujemo je u strukturu koja je organizovana kao ***log***.

Pisanjem napomene je zagarantovano da ukoliko dođe do pada sistema u toku ažuriranja, možemo se vratiti, pročitati šta smo želeli da uradimo i pokušati ponovo. Tako ćemo znati šta tačno treba popraviti nakon pada sistema, ne moramo skenirati ceo disk. Pisanje napomena zahteva malo više posla, ali zato smanjuje vreme oporavka.

Opisaćemo kako ***Linux ext3*** uključuje vođenje dnevnika. Disk je podeljen na grupe blokova i svaka grupa blokova ima indeksni čvor i bitmapu podataka kao i indeksne čvorove i blokove podataka. Nova struktura je upravo dnevnik, koji zauzima mali prostor. Dnevnik nekada može biti smešten i na drugi uređaj.

Sistem datoteka ext2 bez dnevnika izgleda ovako:



Slika 3. Sistem datoteka bez dnevnika

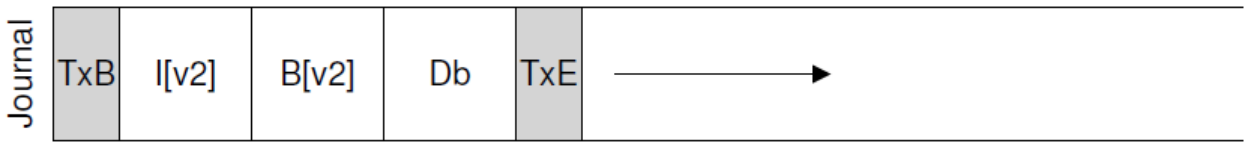
Sistem datoteka ext3 sa dnevnikom izgleda ovako:

Slika 4. Sistem datoteka sa dnevnikom

Razlika je u postojanju dnevnika i načinu na koji se koristi.

5.1. Vođenje dnevnika

Pogledajmo jednostavan primer, kako bismo razumeli na koji način vođenje dnevnika funkcioniše. Zamislimo da imamo ažuriranje kao na početku, gde je potrebno upisati na disk indeksni čvor (I[V2]), bitmapu (B[V2]), i blok podataka (Db). Pre njihovog upisivanja na finalnu lokaciju, prvo ćemo ih upisati u log fajl, odnosno denvnik. Dnevnik će ovako izgledati kao na slici 5:



Slika 5. Dnevnik sa upisanim podacima i metapodacima

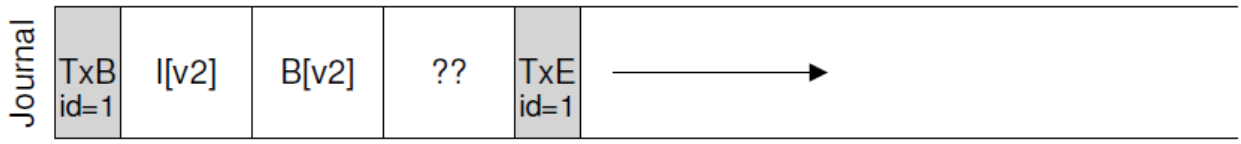
Kao što vidite, upisano je 5 blokova. Početak transakcije (TxB) govori o ovom ažuriranju, informacije o ažuriranju koje treba da se upišu na disk (recimo, finalne adrese blokova I[V2], B[V2], i Db), poseduje i identifikator transakcije **TID.** Blok TxE predstavlja kraj transakcije i takođe sadrži **TID**. Ovo se naziva i ***physical logging*** jer stavljamo stvarni fizički sadržaj u dnevnik.

Kada je ova transakcija bezbedno smeštena na disk, spremni smo da podatke upišemo umesto starih struktura. Ovaj proces se naziva ***checkpointing.*** Kako bismo izvršili ***checkpoint*** sistema datoteka, prosleđujemo I[V2], N[V2], i Db njihovim konačnim lokacijama na disku i upisujemo ih. Ukoliko se ova upisivanja uspešno kompletiraju, ***checkpoint*** je uspešan.

Ovo su koraci operacije:

1. **Upisivanje u dnevnik*:*** Upisati transakciju, uključujući blok za početak transakcije, sve podatke i metapodatke za ažuriranje i blok koji označava kraj transakcije, u dnevnik. Sačekati kompletiranje ovih upisivanja.
2. ***Checkpoint*** Upisati podatke i metapodatke iz transakcije na njihove konačne lokacije na disku.

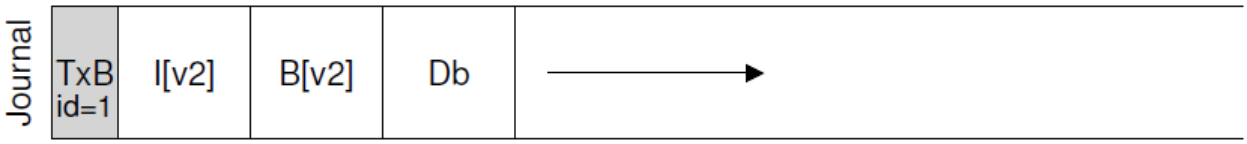
Stvari mogu da pođu po zlu ukoliko dođe do pada sistema u trenutku upisivanja u dnevnik. Pokušavamo da upišemo set blokova u transakciju (TxB, I[V2], B[V2], Db, TxE) na disku. Jednostavan način bi bio da upisujemo jedan po jedan blok, čekajući kompletiranje svakog bloka. Ali ovo je suviše sporo. Bilo bi dobro kada bismo moglil da prosledimo svih pet blokova odjednom i da budu odjednom upisani. Ovakav način nije bezbedan iz više razloga: veliki upis, disk može interno da rasporedi i kompletira male delove velikog upisa u bilo kom redosledu. U našem primeru moguće je da disk napravi raspored upisa tako što će da podeli upis na dva dela: na primer, prvo da upiše TxB, I[V2], B[V2] i TxE, a potom Db. Problem može da nastane u slučaju pada sistema između dva upisa. Onda bi na disk bilo upisano sledeće:



Slika 6. Dnevnik posle pada sistema u toku upisivanja

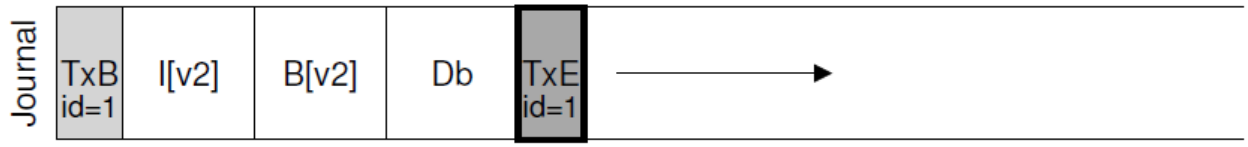
Ovo je problem jer transakcija izgleda ispravno, ali ipak nije jer fali jedan podatak. Sistem datoteka ne može znati da taj blok nedostaje.

Kako bismo izbegli ovaj problem, sistem datoteka prosleđuje transakciju u dva koraka. Prvi, upisuje sve blokove u dnevnik, izuzev TxE bloka, tako što upisuje sve blokove odjednom. Kada se ova upisivanja kompletiraju, dnevnik izgleda ovako:



Slika 7. Prvi korak upisivanja u dnevnik

Kada se ovi upisi kompletiraju, sistem datoteka prosleđuje upis bloka koji označava kraj transakcije (TxE). Dnevnik sada izgleda ovako:



Slika 8. Upisan blok za komitovanje transakcije (TxE)

Važan aspect ovog procesa je garantovana atomičnost od strane diska. Ispostavlja se da disk garantuje da će se upis veličine 512 bajtova izvršiti ili neće uopšte.

Trenutni protokol za ažuriranje sistema datoteka:

1. **Upis u dnevnik:** Upisati sadržaj transakcije u log fajl. Sačekati da ovaj upis bude kompletiran.
2. **Komitovanje dnevnika:** Upisati blok za komitovanje transakcije (TxE) u log fajl. Sačekati da upis bude kompletiran.
3. ***Checkpoint:*** Upisati sadržaj na konačne lokacije.

5.2. Oporavak

Sada ćemo objasniti kako sistem datoteka može da iskoristi sadržaj dnevnika za oporavak od pada. Ukoliko se pad sistema desi pre upisa transakcije u log fajl, onda je oporavak lak: ažuriranje se preskače. Ukoliko se pad sistema desi nakon upisa transakcije u log fajl, ali pre nego što se ***checkpoint*** kompletira, onda se oporavak vrši na sledeći način. Kada se sistem učita, proces oporavka sistema datoteka će skenirati log fajl i potražiti transakcije koje su komitovane na disk, ove transakcije se redom ponavljaju, tako što sistem datoteka pokušava blokove podataka u transakciji ponovo upisati na njihove konačne lokacije. Ovo je jedan od najjednostavnijih načina i naziva se ***redo logging.*** Oporavkom komitovanih transakcija iz dnevnika, sistem datoteka obezbeđuje da su strukture podataka na disku konzistentne. Vidimo da će u najgorem slučaju doći do ponavljanja nekih ažuriranja, u toku oporavka. Ali to nije problem jer se oporavak ne dešava često.

5.3. Doziranje ažuriranja iz log fajla

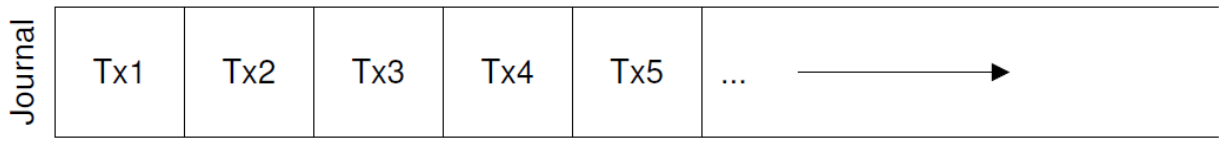
Zamislimo da u istom direktorijumu kreiramo dva fajla, *file1* i *file2*. Kod kreiranja fajla, potrebno je ažurirati određene strukture diska. Ažuriranja koja moraju da se izvrše su sledeća: ažuriranje bitmape indeksnih čvorova, potrebno je kreirati novi indeksni čvor za taj fajl, ažuriranje bloka podataka direktroijuma koji sadrži kreirani fajl, kao i indeksni čvor koji pripada direktorijumu (koji sada ima novo vreme promena). Kod sistema koji poseduju dnevnike, komitujemo sve ove informacije u dnevnik za svaki od fajlova; pošto su fajlovi u istom direktorijumu, i pretpostavljajući da im se indeksni čvorovi nalaze u istom bloku indeksnih čvorova, može se desiti da upisujemo u iste blokove svaki put.

Pojedini sistemi datoteka (Linux ext3) rešavaju ovaj problem tako što ne komituju ažuriranja jedno po jedno, već se baferuju sva ažuriranja u jednu globalnu transakciju. U primeru koji smo naveli, kada kreiramo dva fajla, sva potrebna ažuriranja će biti u jednoj transakciji. Kada je konačno vreme da se upišu promene na disk, ova transakcija se komituje. Tako, baferovanjem ažuriranja, sistem datoteka izbegava nepotreban saobraćaj pri pisanju na disk.

5.4. Ograničenja dnevnika

Kod ažuriranja, sistem datoteka baferuje ažuriranja u memoriji i zadržava ih neko vreme. Kada dođe vreme za upisivanje na disk, sistem datoteka prvo pažljivo upisuje podatke transakcije u dnevnik. Nakon što kompletira transakciju, sistem datoteka upisuje blokove na njihove konačne lokacije na disku.

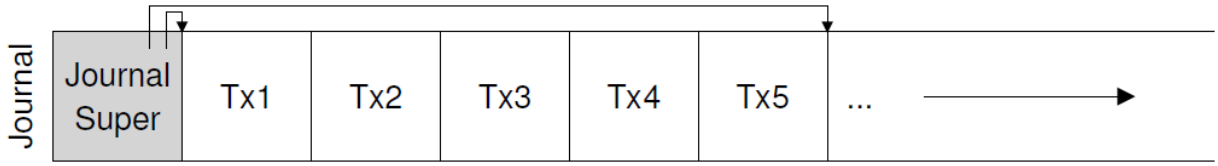
Međutim, prostor u dnevniku je ograničen. Pri upisivanju transakcija, dnevnik se može napuniti.



Slika 9. Više transakcija u dnevniku

Ukoliko se dnevnik napuni, javljaju se dva problema. Prvi problem je jednostavniji, ali i manje kritičan: što je veći log fajl, oporavak je duži, pošto proces oporavka mora da prođe kroz sve transakcije datim redosledom i da ih ponovi kako bi sistem bio oporavljen. Drugi problem je: kada se dnevnik popuni, nijedna transakcija više ne može biti komitovana na disk, ovo čini sistem datoteka beskorisnim.

Kako bi rešio ove probleme, sistem datoteka tretira dnevnik kao kružnu strukturu podataka, koristeći je iznova. Ovaj način je poznat i kao kružni dnevnik. Kako bi ovo funkcionisalo, sistem datoteka mora da preduzme neke akcije nakon upisivanja na konačne lokacije. Odnosno, nakon upisivanja transakcije na konačnu lokaciju, sistem datoteka treba da oslobodi prostor koji je ta transakcija zauzimala u dnevniku, na taj način omogućava ponovno korišćenje tog prostora. Jedan od načina da se ispuni ovaj cilj je sledeći: moguće je u super bloku dnevnika markirati najnoviju i najstariju neupisanu transakciju; preostali prostor je slobodan.



Slika 10. Transakcije koje nisu upisane na konačne lokacije

U super bloku dnevnika postoji dovoljno informacija o tome koje transakcije još uvek nisu upisane na konačnu lokaciju. Na taj način skraćuje vreme potrebno za oporavak i omogućava korišćenje dnevnika na kružni način.

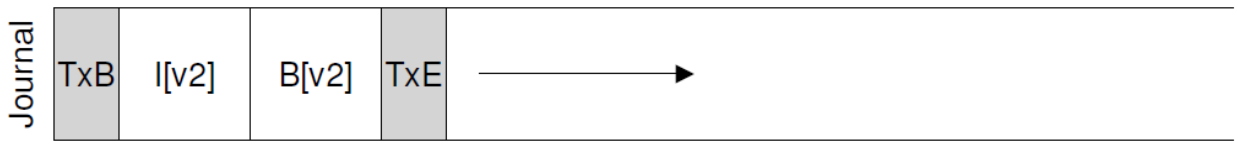
Tako smo dodali još jedan korak u naš osnovni protokol:

1. **Upisivanje u dnevnik**: Upisivanje sadržaja transakcije (TxB i sadržaj za ažuriranje) u dnevnik. Čeka se na kompletiranje ovok upisivanja.
2. **Komitovanje dnevnika**: Upisati blok za komitovanje (TxE) u dnevnik; sačekati da upisivanje bude kompletirano; Transakcija je sada komitovana.
3. **Checkpoint**: Upisati transakcije na njihove konačne lokacije.
4. **Oslobađanje prostora**: Osloboditi prostor koji je transakcija zauzimala u dnevniku.

Međutim, još uvek postoji problem: svaki blok podataka upisujemo dva puta na disk, što je prilično skupo.

5.5. Dnevnik metapodataka

Iako je sada brži oporavak, normalne operacije sistema datoteka su sada sporije. Za svako upisivanje na disk, prvo upisujemo u dnevnik, ovo duplira saobraćaj upisivanja. Zbog prilično skupe cene duplog upisivanja na disk, nastalo je nekoliko različitih načina za ubrzavanje performansi. Do sada smo govorili o vođenju dnevnika podataka, ali jednostavniji oblik vođenja dnevnika je dnevnik metapodataka. Jako je sličan, samo što se korisnički podaci ne upisuju u dnevnik. Ovako izgleda takav dnevnik:



Slika 11. Dnevnik metapodataka

Blok podataka Db sada nije upisan u dnevnik, pošto većinu U/I saobraćaja na disku čine korisnički podaci, izbegavanje duplog pisanja korisničkih podataka donosi prilično ubrzanje. Ali, dolazimo do pitanja: kada treba upisivati podatke na disk?

Da li upisati Db na disk nakon upisivanja transakcije na konačnu lokaciju? Na žalost, ovaj pristup ima problem: sistem datoteka je konzistentan, ali indeksni čvor pokazuje na otpad. To je u slučaju da su I[V2] i B[V2] upisani, ali upisivanje Db-a nije uspelo.

Kako bi izbegli ovu situaciju, pojedini sistemi datoteka (Linux ext3) upisuju blokove podataka na disk pre metapodataka. Protokol izgleda ovako:

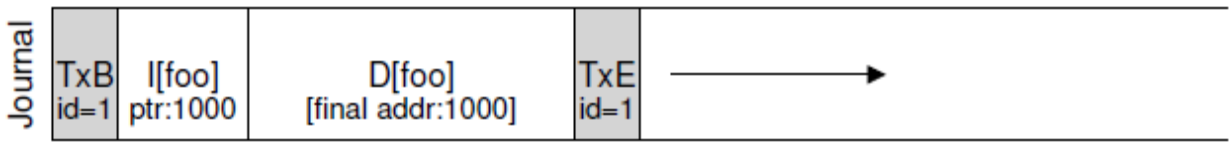
1. **Upis podataka**: Upisati korisničke podatke na konačnu lokaciju na disku; sačekati kompletiranje.
2. **Upis metapodataka u dnevnik**: Upisati početni blok i metapodatke u dnevnik; sačekati kompletiranje.
3. **Komitovanje dnevnika**: Upisati blok za komitovanje (TxE) u dnevnik; sačekati kompletiranje; transakcija je sada komitovana.
4. ***Checkpoint***: Upisati sadržaj dnevnika na konačne lokacije.
5. **Oslobađanje prostora**: Osloboditi prostor u dnevniku koji je zauzimala transakcija.

Upisivanjem korisničkih podataka prvo, sistem datoteka garantuje da pokazivač u indeksnom čvoru nikada neće pokazivati na otpad.

U mnogim sistemima, vođenje dnevnika metapodataka je mnogo popularnije u odnosu na vođenje dnevnika podataka. Na primer, Windows NTFS koristi ovakvo vođenje dnevnika. Linux ext3 daje mogućnost izbora.

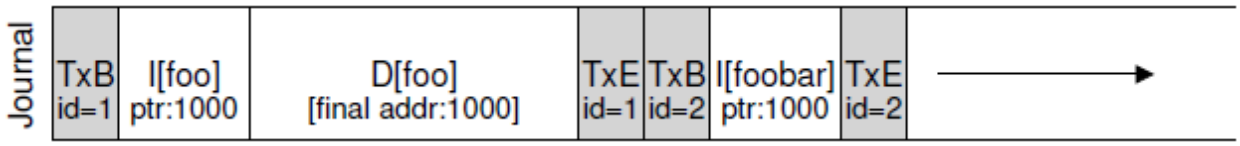
5.6. Ponovno korišćenje bloka

Pretpostavimo da koristimo dnevnik metapodataka. Recimo, imamo direktorijum *foo,* korisnik doda neki sadržaj u direktorijum *foo* (na primer kreira fajl) i tako je sadržaj direktorijuma *foo* upisan u dnevnik. Pretpostavićemo da je lokacija direktorijuma *foo* blok 1000. Dnevnik onda izgleda kao na slici 12.



Slika 12. Dnevnik sa direktorijumom foo

Sada, korisnik obriše direktorijum, kao i sve što se nalazilo u njemu, oslobađajući time blok 1000 za ponovno korišćenje. Potom, korisnik kreira novi fajl (*foobar*), koji zauzme lokaciju koju je pomenuti direktorijum zauzimao, lokaciju 1000. Indeksni čvor za fajl *foobar* je komitovan na disk, kao i podaci. Pošto koristimo dnevnik metapodataka, samo je indeksni čvor za *foobar* komitovan u dnevnik, blok podataka nije u dnevniku.



Slika 13. Dnevnik posle kreiranja fajla foobar

Zamislimo da sada dođe do pada sistema, dok se sve ove informacije i dalje nalaze u log fajlu. Tokom oporavka, proces oporavka ponavlja sve što se nalazi u dnevniku, pa i direktorijum *foo*. Stari direktorijum će u tom slučaju pregaziti podatke koji se nalaze na lokaciji 1000.

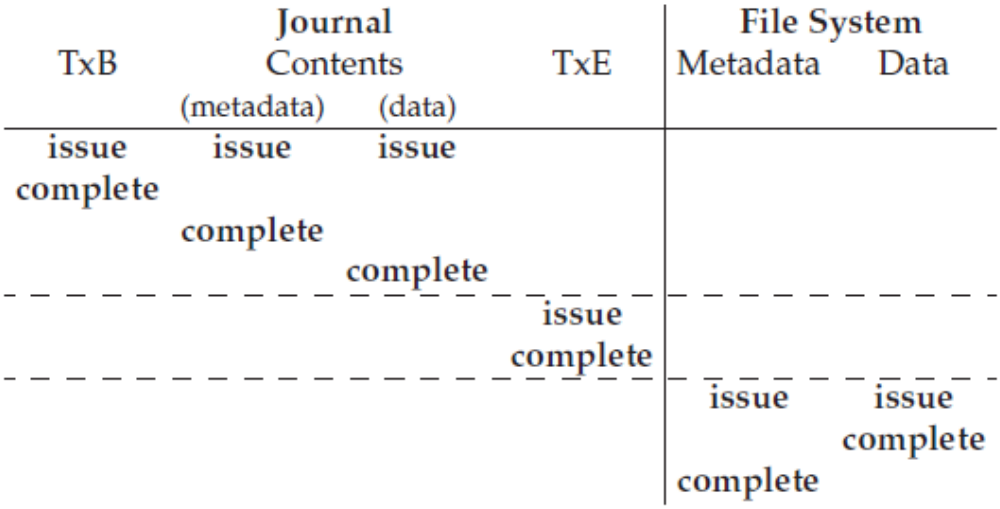
Jedno od rešenja jeste da se ne koristi ponovo isti blok, sve dok se podaci koji su obrisani sa tog bloka ne izbace iz dnevnika. Linux ext3 ovaj problem rešava tako što dodaje novi tip zapisa u dnevnik, a to je ***revoke*** zapis. Brisanje direktorijuma uzrokuje pisanje ***revoke*** zapisa u dnevniku. Prilikom oporavka sistema, prvo se pretražuju ***revoke*** zapisi. Takvi podaci nikada se ne ponavljaju u toku oporavka i na taj način se izbegava pomenuti problem.

5.7. Vođenje dnevnika - sumiranje

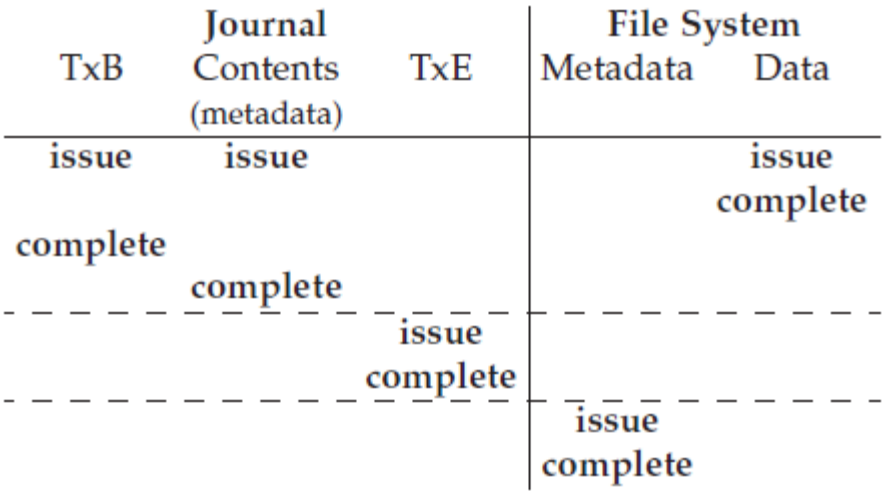
Pre nego što završimo izlaganje o vođenju dnevnika, rezimiraćemo protokole o kojima smo govorili.

Na slici 14 je protokol dnevnika podataka, a na slici 15 je protokol dnevnika podataka.

U oba slučaja vreme raste u smeru na dole i svaki red govori u kom vremenski logičnom intervalu bi neki upis treba proslediti ili izvršiti.



Slika 14. Protokol dnevnika podataka



Slika 15. Protokol dnevnika metapodataka

6. Rešenje 3: ostali pristupi

Opisali smo dva pristupa za održavanje konzistentnosti: lenji pristup koji se zasniva na **fsck** alatu, i popularniji pristup poznatiji kao vođenje dnevnika. Međutim, ovo nisu jedini pristupi. Kratko ćemo opisati neke. Pristup poznat kao **Soft Updates,** ovaj pristup osigurava uvek konzistentno stanje**.** Na primer, upisivanjem podataka na disk, pre upisivanja indeksnog čvora koji pokazuje na taj blok, osiguravamo da indeksni čvor nikada ne pokazuje na otpad.

Sledeći pristup se zove **copy-on-write**. Ova tehnika nikada ne upisuje preko već postojećih fajlova ili direktorijuma, umesto toga upisuje ažuriranja u prethodno nekorišćenu lokaciju na disku.

Pristup **backpointer-based consistency** kako bi održao konzistentnost, dodaje dodatni pokazivač u svaki blok u sistemu. Na primer, svaki blok podataka ima pokazivač na indeksni čvor kom pripada. Kako bi proverio konzistentnost, sistem datoteka proverava da li blok podataka na koji pokazuje indeksni čvor ima povratni pokazivač na indeksni čvor.

7. Zaključak

Upoznali smo se sa problemom nekonzistentnosti, i govorili o pristupima za njegovo rešavanje. Stariji pristupi koji su koristili **fsck** su radili, ali oporavak od pada sistema je bio jako spor za današnje prohteve. Danas, većina sistema datoteka koristi dnevnik. Upotreba dnevnika smanjuje vreme oporavka i upravo iz tog razloga većina sistema ga koristi. Videli smo da vođenje dnevnika može imati više raličitih oblika, najčešće se koristi vođenje dnevnika metapodataka, gde je smanjen saobraćaj upisa u dnevnik i obezbeđena je konzistentnost.

8. Literatura

[1] http://pages.cs.wisc.edu/~remzi/OSTEP/file-journaling.pdf
[2] http://pages.cs.wisc.edu/~remzi/OSTEP/file-implementation.pdf
[3] https://docs.freebsd.org/44doc/smm/03.fsck/paper.pdf
[4] http://minnie.tuhs.org/CompArch/Lectures/week11.html
[5] http://www.tldp.org/LDP/tlk/fs/filesystem.html
[6] https://wiki.archlinux.org/index.php/fsck