

## *Sekvencijalna organizacija datoteke*

U datoteci sa sekvencijalnom organizacijom, slogovi su smešteni jedan za drugim u sekcesivne lokacije od početka memoriskog prostora dodeljenog datoteci. Fizička struktura datoteke sadrži informaciju o vezama između slogova logičke strukture podataka datoteke. Ta informacija je ugrađena u fizičku strukturu smeštanjem logički susednih slogova u fizički susedne lokacije. Bez uticaja na opštost, u daljem tekstu će se posmatrati samo slučaj kada su slogovi datoteke uređeni saglasno rastućim vrednostima ključa. Saglasno tome, slog sa najmanjom vrednošću ključa smešta se u prvu lokaciju memoriskog prostora dodeljenog datoteci. Slog sa narednom vrednošću ključa smešta se u drugu lokaciju itd. Slogovi se u datoteku smeštaju u blokovima od po  $f (\geq 1)$  slogova. Poželjno je da faktor blokiranja bude što veći.

U datoteci sa sekvencijalnom organizacijom realizovana je linearna logička struktura podataka smeštanjem sloga sa većom vrednošću ključa u lokaciju sa većom adresom. Zbog toga se ova organizacija naziva i "fizički sekvencijalnom". S druge strane, adresa lokacije, u koju je smešten slog  $S$ , nije funkcija vrednosti ključa  $k(S)$ , tako da pristup slogu  $S$  putem adrese lokacije, u opštem slučaju, nije moguć.

**Primer 9.1.** Na slici 9.1 prikazana je struktura jedne male sekvencijalne datoteke od  $N = 13$  slogova sa faktorom blokiranja  $f = 3$ . Datoteka sadrži slogove sa istim vrednostima ključa, kao i serijska datoteka sa slike 7.1. Indeks  $i$  u  $p(S_i)$  ( $i = 1, \dots, 13$ ) ukazuje na hronološki redosled smeštanja slogova u sekvencijalnu datoteku. U polje ključa prve slobodne lokacije upisan je specijalni znak \*, kao oznaka kraja datoteke. □

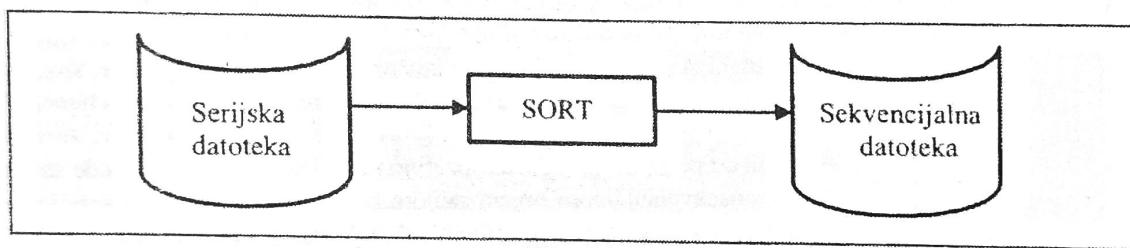
Korisnicima mainframe računarskih sistema je stajala na raspolaganju sekvencijalna metoda pristupa, koja im je pružala određeni broj usluga pri formirajući obradi datoteka sa sekvencijalnom organizacijom. Ove usluge opisane su u glavi 4. Savremeni operativni sistemi, kao što je Unix i objektno-orientisani jezici, kao što su C, C++ i Java, podržavaju samo sekvencijalni način pristupa, a ostavljaju korisnicima da naprave svoje sopstvene sekvencijalne metode pristupa.

$A_1$	<table border="1"> <tr><td>03</td><td><math>p(S_1)</math></td><td>07</td><td><math>p(S_2)</math></td><td>13</td><td><math>p(S_3)</math></td></tr> </table>	03	$p(S_1)$	07	$p(S_2)$	13	$p(S_3)$	$A_2$	<table border="1"> <tr><td>15</td><td><math>p(S_4)</math></td><td>19</td><td><math>p(S_5)</math></td><td>23</td><td><math>p(S_6)</math></td></tr> </table>	15	$p(S_4)$	19	$p(S_5)$	23	$p(S_6)$
03	$p(S_1)$	07	$p(S_2)$	13	$p(S_3)$										
15	$p(S_4)$	19	$p(S_5)$	23	$p(S_6)$										
$A_3$	<table border="1"> <tr><td>25</td><td><math>p(S_7)</math></td><td>27</td><td><math>p(S_8)</math></td><td>29</td><td><math>p(S_9)</math></td></tr> </table>	25	$p(S_7)$	27	$p(S_8)$	29	$p(S_9)$	$A_4$	<table border="1"> <tr><td>34</td><td><math>p(S_{10})</math></td><td>43</td><td><math>p(S_{11})</math></td><td>49</td><td><math>p(S_{12})</math></td></tr> </table>	34	$p(S_{10})$	43	$p(S_{11})$	49	$p(S_{12})$
25	$p(S_7)$	27	$p(S_8)$	29	$p(S_9)$										
34	$p(S_{10})$	43	$p(S_{11})$	49	$p(S_{12})$										
$A_5$	<table border="1"> <tr><td>64</td><td><math>p(S_{13})</math></td><td>*</td><td></td><td></td><td></td></tr> </table>	64	$p(S_{13})$	*											
64	$p(S_{13})$	*													

Slika 9.1.

## 9.1 FORMIRANJE SEKVENCIJALNE DATOTEKE

Datoteka sa sekvencijalnom organizacijom dobija se sortiranjem serijske datoteke saglasno rastućim vrednostima ključa. Na slici 9.2 prikazan je postupak formiranja sekvencijalne datoteke.



Slika 9.2.

## 9.2 TRAŽENJE U SEKVENCIJALNOJ DATOTECI

U sekvencijalnoj datoteci se vrši traženje logički narednog sloga ili traženje slučajno odabranog sloga. Ako je datoteka smeštena na medijum eksternog memorijskog uređaja, tada traženje slučajno odabranog sloga linearnom metodom traženja nema praktičnog smisla, zbog velikog broja pristupa. Binarna metoda traženja bi se mogla koristiti, ali se, zbog potrebe višekratnog pristupanja srednjem bloku intervala traženja, koji može biti dosta udaljen od tekućeg, i ona retko koristi. Ako obrada podataka nalaže traženje slučajno odabranog sloga, tada se datoteka organizuje kao rasuta ili indeksna, a ne kao sekvencijalna. Traženje slučajno odabranog sloga u sekvencijalnoj datoteci ima praktičnog smisla jedino ako se ona nalazi u operativnoj memoriji. Tu malu sekvencijalnu datoteku može predstavljati neka linearna struktura nad skupom slogova, ili blok neke druge datoteke (na primer indeks - sekvencijalne). U daljem tekstu obraden je samo postupak traženja logički narednog sloga u sekvencijalnoj datoteci, smeštenoj

na medijum eksternog memorijskog uredaja. Traženje slučajno odabranog sloga u linearu strukturi, smeštenoj u operativnu memoriju, obrađeno je u Glavi 2.

Traženje logički narednog sloga u sekvencijalnoj datoteci vrši se linearom metodom traženja. Kada je reč o datoteci na eksternoj memoriji, počevši od prvog, fizički susedni bloovi se učitavaju u operativnu memoriju, a u centralnoj jedinici se vrši upoređivanje argumenta traženja i vrednosti ključa sukcesivnih slogova. Ovo upoređivanje se vrši dok se traženi slog ne pronađe, ili dok argument traženja ne postane manji od vrednosti ključa sloga, ili dok se ne dođe do kraja datoteke. Traženje novog, logički narednog sloga, započinje od sloga na koji se prethodno traženje zaustavilo. Taj slog se naziva tekućim sloganom datoteke. Algoritam linearog traženja logički narednog sloga u sekvencijalnoj datoteci, prikazan je na slici 5.1 u glavi 5.

Broj pristupa  $R$  i pri uspešnom i pri neuspešnom traženju logički narednog sloga sekvencijalnoj datoteci, uzima celobrojne vrednosti iz intervala  $[0, B - i]$ , odnosno

$$0 \leq R \leq B - i,$$

gde je  $B$  broj blokova datoteke, a  $i$  redni broj tekućeg bloka datoteke u odnosu na početak datoteke.

Broj uporedenja  $U$  argumenta traženja i vrednosti ključa slogova i pri uspešnom i pri neuspešnom traženju logički narednog sloga u sekvencijalnoj datoteci, uzima celobrojne vrednosti iz intervala  $[I, N - i + 1]$ , odnosno

$$I \leq U \leq N - i + 1,$$

gde je  $N$  broj slogova datoteke, a  $i$  redni broj tekućeg sloga u logičkoj strukturi podataka datoteke. Redni broj tekućeg sloga u logičkoj strukturi podataka datoteke predstavlja broj zadataka veći od dužine puta između prvog (najmanji čvor) i tekućeg sloga.

### 9.3 OBRADA SEKVENCIJALNE DATOTEKE

Datoteka sa sekvencijalnom organizacijom može se obradivati i u režimu redosledne i u režimu direktnе obrade. O direktnoj obradi sekvencijalne datoteke ima smisla govoriti jedino ako je datoteka mala, tako da može da se smestiti u operativnu memoriju. Tu malu datoteku može predstavljati niz slogova, neka tabela, ili blok indeksne datoteke. Performanse direktnе obrade velikih sekvencijalnih datoteka, smeštenih na eksterne memorijske uredaje, malo se razlikuju od performansi obrade serijske datoteke, tako da tom pitanju neće biti posvećena dalja pažnja.

Sekvencijalna datoteka se može koristiti kao vodeća i u režimu redosledne i u režimu direktnе obrade. Kada se datoteka koristi kao vodeća, iz nje se počevši od prvog, sukcesivno učitavaju fizički susedni blokovi. Ukupan broj pristupa vodećoj sekvencijalnoj datoteci, potreban da se pročita svih  $N$  slogova iznosi  $R = B$ , gde je

$$B = \left\lceil \frac{N + 1}{f} \right\rceil,$$

Broju slogova  $N$  se dodaje 1, pri izračunavanju broja blokova, zbog oznake kraja datoteke.

## → REDOSLEDNA OBRADA SEKVENCIJALNE DATOTEKE

U svakom koraku redosledne obrade, vodeća datoteka generiše logički naredne vrednosti ključa za traženje u obradivanoj sekvencijalnoj datoteci. Traženje logički narednog sloga vrši se metodom linearног traženja. Na slici 9.3 prikazan je pseudokod algoritma redosledne obrade sekvencijalne datoteke smeštene na medijum eksternog memoriskog uređaja. Algoritam sa slike 9.3 poziva algoritam linearног traženja prikazan na slici 6.5 i fiktivni proces obrada\_slogova, čiji zadaci zavise od konkretnе primene.

### ALGORITAM REDOSLEDNE OBRADE SEKVENCIJALNE DATOTEKE

**PROCES** redosledna\_obrađa\_sekv\_dat(U( $D_v, D_o, D_i$ ), I(), UI())

(\*)

U  $D_v$  je oznaka vodeće datoteke, čiji slog  $S_v$  sadrži ključ sloga  $S_o$  obradivane sekvencijalne datoteke  $D_o$ .  $D_i$  je oznaka izlazne datoteke sa sloganom  $S_i$ .

\*)

**POČETAK PROCESA** redosledna\_obrađa\_sekv\_dat

OTVORI( $D_v, \text{ČIT}, stat$ ) OTVORI( $D_o, \text{ČIT}, stat$ ) OTVORI( $D_i, PIS, stat$ )

ČITAJ\_SEQ( $D_o, S_o, stat$ )

RADI obrada DOK JE ( $stat = 0$ )

ČITAJ\_SEQ( $D_v, S_v, stat$ )

AKO JE  $stat = 0$  TADA

POZOVI linearno\_traženje\_u\_seq( $D_o, k(S_o), int, stat, S_o$ )

AKO JE  $int = 0$  TADA

POZOVI obrada\_slogova( $S_v, S_o, S_i$ )

PIŠI\_SEQ( $D_i, S_i, stat$ )

INAČE

(\*neuspšeno traženje u datoteci  $D_o$ \*)

KRAJ AKO

INAČE

(\*kraj datoteke  $D_v$  ili neuspšeno čitanje\*)

KRAJ AKO

KRAJ RADI obrada

KRAJ PROCESA redosledna\_obrađa\_sekv\_dat

Slika 9.3.

Pri redoslednoj obradi sekvencijalne datoteke svaki blok se učitava samo jedanput u operativnu memoriju. Ako se sekvencijalna datoteka obrađuje uz pomoć vodeće datoteke, koja sadrži  $N_v$  slogova ( $1 \leq N_v$ ), uključujući i slog sa najvećom vrednošću ključa obradivane sekvencijalne datoteke, ukupan broj pristupa iznosi

$$R_{uk} = B,$$

a srednji broj pristupa  $\bar{R}$  datoteci, potreban za uspešno ili neuspšeno traženje logički narednog sloga, iznosi

$$(9.1) \quad \bar{R} = \frac{B}{N_v}.$$

Na osnovu formule (9.1) može se zaključiti da je, pri datom broju slogova  $N$ , redosledna obrada sekvenčijalne datoteke to efikasnija što je faktor blokiranja obradivane sekvenčijalne datoteke veći i što je veći broj slogova vedeće datoteke.

S obzirom na opisani postupak traženja slogova u sekvenčijalnoj datoteci pri redoslednoj obradi, ukupan broj upoređenja argumenta traženja i vrednosti ključa slogova iznosi  $U \geq N + N_v$ , a srednji

$$(9.2) \quad \bar{U} \geq \frac{N}{N_v} + 1.$$

Svaki slog obradivane datoteke na kojem se zaustavilo uspešno ili neuspešno traženje učestvuje najmanje dva puta u upoređivanju vrednosti argumenta traženja i ključa<sup>4)</sup>.

**Primer 9.2.** Ako vodeća datoteka generiše sledeći niz vrednosti ključa (03, 06, 13, 19, 25, 29, 49, 55, 64) za traženje pri obradi sekvenčijalne datoteke sa slike 9.1, tada ukupni broj pristupa datoteci, pri redoslednoj obradi, iznosi  $R_{uk} = 5$ , a  $U_{uk} = 22$ .

Očekivani broj pristupa i pri uspešnom i pri neuspešnom traženju logički narednog sloga, iznosi  $\bar{R} \approx 0,56$ , a očekivani broj upoređenja iznosi  $\bar{U} \approx 2,44$ .  $\square$

**Primer 9.3.** Broj pristupa  $R$  sekvenčijalnoj datoteci od  $N = 10\ 000$  slogova sa faktorom blokiranja  $f = 10$ , pri redoslednoj obradi uz pomoć vodeće datoteke od  $N_v = 9\ 300$  slogova, iznosi  $R_{uk} = 1\ 001$ , tako da srednji broj pristupa pri traženju logički narednog sloga iznosi  $\bar{R} \approx 0,11$ . U slučaju da vodeća datoteka sadrži samo  $N_v = 10$  slogova, srednji broj pristupa bi iznosio  $\bar{R} = 100,1$ .

Pri redoslednoj obradi sekvenčijalne datoteke, metoda pristupa učitava fizički susedne blokove datoteke u operativnu memoriju. Neka je srednje vreme pristupa susednom bloku i vreme njegovog prenosa u operativnu memoriju  $\bar{t} = 10 \text{ msek}$ . Tada vreme  $\bar{T} = R_{uk}\bar{t}$  potrebno za prenos svih blokova datoteke u operativnu memoriju, radi traženja  $N_v = 8\ 400$  slogova u redoslednoj obradi, iznosi  $\bar{T} = 10 \text{ sek}$ .  $\square$

## PROGNOSTIČKO KORIŠĆENJE BAFERA

Činjenica da se, pri redoslednoj obradi sekvenčijalne datoteke, u svakom narednom koraku traži slog memorisan u fizički narednoj lokaciji, u odnosu na tekuću, dozvoljava, u monoprogramskom režimu obrade, efikasnu primenu prognostičkog korišćenja bafera. Za razmenu podataka sa sekvenčijalnom datotekom, u operativnoj memoriji se predviđe dva ili više bafera. Na početku izvršavanja programa, metoda pristupa pročita prvi blok datoteke, smesti ga u prvi bafer, a program počne da ga obrađuje. Paralelno sa obradom prvog bloka prenosi se drugi blok u drugi bafer. Čim program završi obradu sadržaja prvog bafera, metoda pristupa inicira prenošenje narednog bloka iz datoteke u prvi bafer. Na taj način se obrada i čitanje sadržaja datoteke preklapaju, što skraćuje realno vreme izvršavanja programa. Ista tehnika se primenjuje i pri prenošenju slogova, odnosno blokova, iz operativne na eksternu memoriju.

Za obradu sekvenčijalne datoteke uz primenu dva bafera karakteristična su dva slučaja:

<sup>4)</sup> Više od dva puta pri ažuriranju.

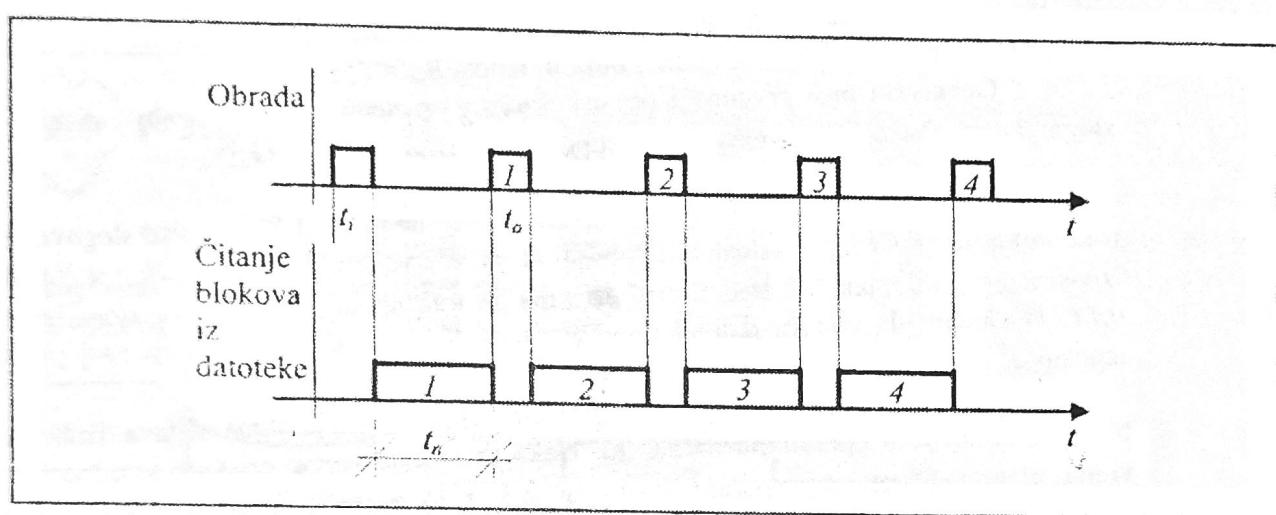
- kada je  $t_o \geq t_n$ , gde je  $t_o$  vreme obrade bloka i iniciranja čitanja narednog bloka, a  $t_n$  vreme prenosa bloka između eksterne i operativne memorije i
- kada je  $t_o < t_n$ .

U prvom slučaju, rad centralne jedinice teče kontinualno. U drugom slučaju, koji je mnogo karakterističniji za poslovnu obradu podataka nego prvi, kontinualno se čitaju blokovi iz datoteke.

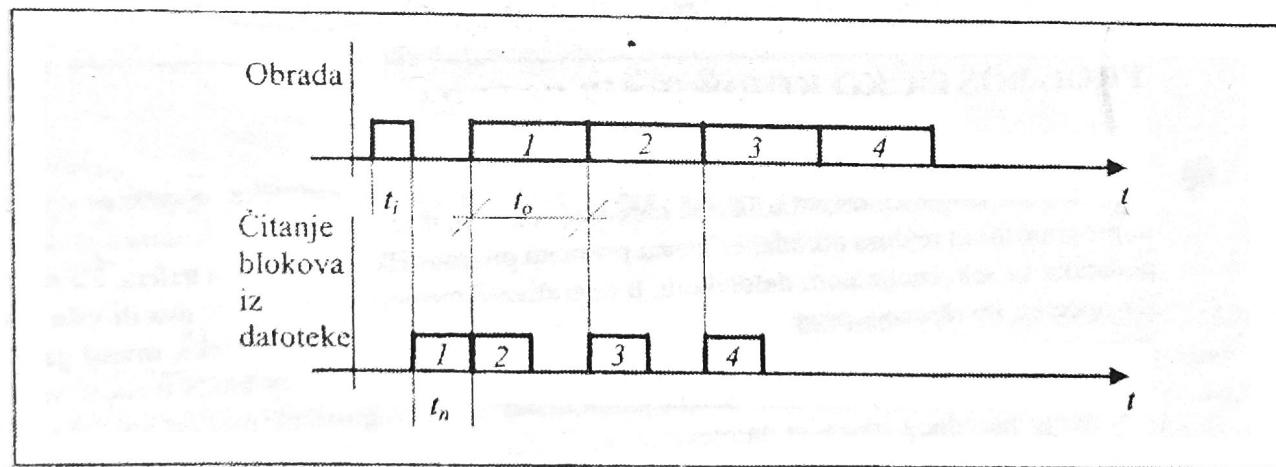
Na slici 9.4 prikazan je vremenski dijagram odvijanja aktivnosti obrade sekvencijalne datoteke uz pomoć samo jednog bafera. Aktivnosti čitanja blokova i njihove obrade odvijaju se strogo naizmenično. Ukupno vreme  $T_u$  obrade datoteke iznosi

$$T_u = t_i + B(t_n + t_o),$$

gde je  $t_i$  vreme iniciranja čitanja prvog bloka, a  $B$  broj blokova u datoteci.



Slika 9.4.



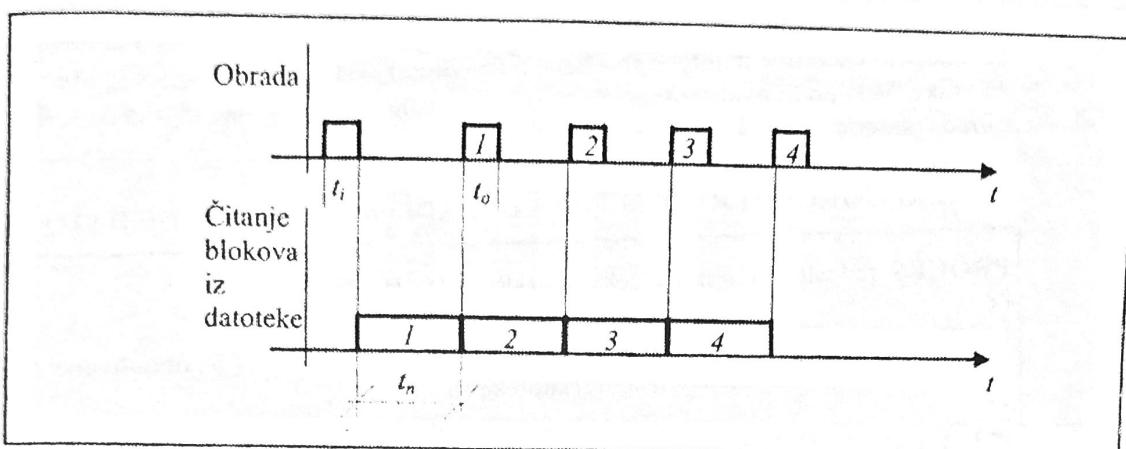
Slika 9.5.

Na slici 9.5 prikazan je vremenski dijagram odvijanja aktivnosti obrade sekvencijalne datoteke uz pomoć dva bafera, u slučaju kada je  $t_o > t_n$ . Aktivnosti čitanja i obrade blokova se preklapaju, tako da je

$$T_u = t_i + t_n + Bt_o.$$

Na slici 9.6 prikazani su efekti prognostičkog korišćenja bafera kada je  $t_o < t_n$ . Tada je

$$T_u = t_i + t_o + Bt_n.$$



Slika 9.6.

Sprovedena analiza ukazuje da, sa tačke gledišta skraćenja realnog vremena monoprogramske obrade programa, nema smisla predvideti više od dva bafera za jednu sekvencijalnu datoteku. Međutim, ovaj zaključak važi samo za slučaj kada je  $t_o < t_n$  ili se datoteka nalazi na magnetnoj traci. Ako je  $t_p < t_o$ , a datoteka se nalazi na disku, pri analizi se mora uzeti u obzir i vreme rotacije disk paketa. Čitanje bloka sa diska se može započeti tek kada početak bloka dođe pod glavu za čitanje i upis, bez obzira što je bafer prazan. Zbog toga je pogodno, u cilju obezbeđenja kontinualne obrade, rezervisati veći broj bafera za jednu datoteku, tako da se oni pune jedan za drugim, kada se jedanput izvrši pozicioniranje glava na prvi od blokova.

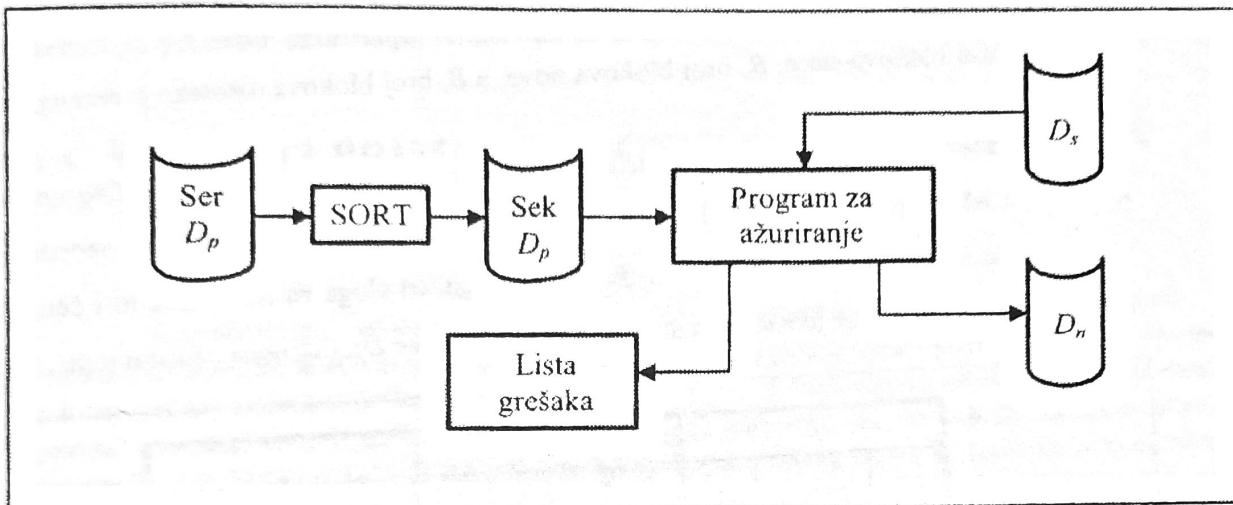
## 9.4 AŽURIRANJE DATOTEKE SA SEKVENCIJALNOM ORGANIZACIJOM

Ažuriranje predstavlja jedan od najozbiljnijih problema, koji se javlja pri korišćenju sekvencijalno organizovanih datoteka. Naime, da bi se novi slog upisao u sekvencijalnu datoteku, potrebno je prvo pronaći tzv. logičko mesto upisa novog sloga. Logičko mesto upisa predstavlja slog sa prvom većom vrednošću ključa od novog sloga. Pošto novi slog treba da se upiše baš u lokaciju sloga sa prvom većom vrednošću ključa, bilo bi potrebno pomeriti za jednu lokaciju udesno sve slogove sa vrednostima ključa većim od vrednosti ključa novog sloga. Brisanje postojećeg sloga iz datoteke zahteva njegovo prethodno pronalaženje. Nakon toga, bilo bi potrebno pomeriti za jednu lokaciju uлево sve slogove sa većom vrednošću ključa, jer se brisanje vrši fizički. Izmena sadržaja nekog sloga u datoteci zahteva njegovo prethodno pronalaženje, a to pak znači, kao i u prethodnim slučajevima, traženje slučajno odabranog sloga.

Ažuriranje sekvencijalne datoteke se može vršiti bilo u režimu direktne, bilo u režimu redosledne obrade. Međutim, ažuriranje u režimu direktne obrade zahteva, u proseku, pomeraњe polovine broja slogova za jednu lokaciju udesno ili uлево pri upisu ili brisanju svakog pojedincog sloga.

dinačnog sloga. Zato se primjenjuje samo u slučajevima kada je kompletna datoteka smeštena u operativnu memoriju. Tada se za traženje slučajno odabranog sloga koristi neka od odgovarajućih metoda traženja. Ovi postupci su obrađeni u glavi 2.

Ažuriranje datoteke čiji broj slogova je toliki da se ne može kompletno smestiti u operativnu memoriju, izvodi se nakon određenih intervala vremena, uz primenu posebnog postupka, u režimu redosledne obrade. Postupak je šematski prikazan na slici 9.7.



Slika 9.7.

Program za ažuriranje učitava slogove stare datoteke  $D_s$  i datoteke promena  $D_n$ , a na osnovu njihovog sadržaja generiše novu (ažuriranu) datoteku  $D_n$ . Preduslov za realizaciju ovog postupka ažuriranja je da su datoteke  $D_s$  i  $D_n$  uređene po istom sortnom pojmu. U daljem tekstu se pretpostavlja da je datoteka  $D_p$  uređena saglasno neopadajućim, a datoteka  $D_s$  saglasno rastućim vrednostima ključa  $k(S_s)$ . Osnovna logika odvijanja programa za ažuriranje sastoji se u: učitavanju slogova  $S_s$  i  $S_p$  datoteka  $D_s$  i  $D_p$ , upoređivanju vrednosti njihovih ključeva, generisanju slogova  $S_n$  (na osnovu sadržaja slogova  $S_s$  i  $S_p$ ) i u smeštaju slogova  $S_n$  u datoteku  $D_n$ . Pri tome, ili ključ  $k(S_p)$  sloga  $S_p$ , ili ključ  $k(S_s)$  sloga  $S_s$  predstavlja argument traženja, redom, u datotekama  $D_s$  ili  $D_p$ . Slogovi sve tri datoteke imaju, u principu, isti format. Jedino je format sloga  $S_p$  proširen poljem  $o(S_p)$ , gde  $o(S_p)$  uzima vrednosti iz skupa  $\{n, m, b\}$ . Ovo polje se može nazvati "oznakom sloga". Oznaka sloga govori da li je reč o novom slogu ( $n$ ), podacima za modifikaciju ( $m$ ), ili slogu koji se briše ( $b$ ). (Slog datoteke  $D_p$ , koji govori da određeni slog datoteke  $D_s$  treba da se briše, obično sadrži samo vrednost ključa sloga  $S_s$  i vrednost oznake, dok su mu ostala polja prazna. Prazna polja su potrebna da bi datoteka  $D_p$  sadržala slogove, odnosno blokove, fiksne dužine. Takođe se pretpostavlja da datoteka  $D_p$  ne sadrži dva sloga sa oznakom  $n$ , koji imaju istu vrednost ključa  $k(S_s)$ ). Na slici 9.7, *Lista grešaka* predstavlja datoteku, u koju se upisuju oni slogovi datoteke promena, koji nalažu:

- ponovni upis već postojećeg sloga u datoteku  $D_n$  i
- brisanje ili modifikaciju nepostojećeg sloga datoteke  $D_s$ .

Na slici 9.8 prikazan je pseudokod algoritma za ažuriranje sekvencijalne datoteke. Ovaj algoritam poziva proces obrada sloga, prikazan na slici 9.9.

## AŽURIRANJE SEKVENCIJALNE DATOTEKE U REDOSLEDNOJ OBRADI

**PROCES** redosledno\_ažuriranje\_sek( $U(D_s, D_p, D_n)$ ,  $I(stat)$ ,  $UI()$ )

**POČETAK PROCESA** redosledno\_ažuriranje\_sek

POSTAVI  $stat \leftarrow 0$

OTVORI( $D_s$ , SEQ, ČIT, statp)

OTVORI( $D_p$ , SEQ, ČIT, stats)

OTVORI( $D_n$ , SEQ, PIS, statn)

ČITAJ\_SEQ( $D_p, S_p, statp$ )

ČITAJ\_SEQ( $D_s, S_s, stats$ )

$D_s$  - stanac

$D_p$  - prom

$D_n$  - morske

RADI ažuriranje DOK NE BUDE ( $(stat = 1) \wedge (statp = 1)$ )

AKO JE  $k(S_s) = k(S_p)$  TADA

POSTAVI  $S_n \leftarrow S_s$

POSTAVI  $stn \leftarrow 1$

(\*stn je status sloga  $S_n$ \*)

POZOVI obrada\_sloga(stn,  $D_p, S_p, statp, D_n$ )

ČITAJ\_SEQ( $D_s, S_s, stats$ )

→ continue

INAČE

AKO JE  $k(S_s) < k(S_p)$  TADA

POSTAVI  $S_n \leftarrow S_s$

PIŠI\_SEQ( $D_n, S_n, statn$ )

ČITAJ\_SEQ( $D_s, S_s, stats$ )

} upis

INAČE

POSTAVI  $stn \leftarrow 0$

POZOVI obrada\_sloga(stn,  $D_p, S_p, statp, D_n$ )

} upis

KRAJ AKO

KRAJ AKO

AKO JE  $stat = 1$  TADA

POSTAVI  $k(S_p) \leftarrow *$

INAČE

KRAJ AKO

AKO JE  $statp = 1$  TADA

POSTAVI  $k(S_p) \leftarrow *$

INAČE

KRAJ AKO

KRAJ RADI ažuriranje

ZATVORI( $D_p$ )

ZATVORI( $D_s$ )

ZATVORI( $D_n$ )

KRAJ PROCESA redosledno\_ažuriranje\_seq

Slika 9.8.

Opisani postupak ažuriranja sekvencijalnih datoteka sa velikim brojem slogova zahteva pristupanje svakom bloku stare datoteke i generisanje jedne potpuno nove datoteke. Ove dve činjenice predstavljaju jedan od faktora koji utiče na dimenzionisanje dužine intervala vremena između dva ažuriranja. Dužina ovog intervala se određuje tako, da se tokom njega

nakupi toliki broj promena, koji bi opravdao pristupanje svim slogovima stare i generisanje nove datoteke. S druge strane, na dužinu intervala vremena između dva ažuriranja utiče i učestalost obrade. Datoteka se mora ažurirati pre svake obrade.

```

PROCES obrada_sloga(U(stn, Dp, Sp, statp, Dn), I(), UI())
(*stn = 0, ako k(Ss) > k(Sp), stn = 1, ako k(Ss) = k(Sp) *)
POČETAK PROCESA obrada_sloga
POSTAVI k(Sp_pret)  $\leftarrow$  k(Sp)
AKO JE statp = 0 TADA
RADI čitaj_Dp DOK NE BUDE (statp = 1)  $\vee$  (k(Sp_pret)  $\neq$  k(Sp))
AKO JE stn = 1 TADA
AKO JE o(Sp) = b TADA b nisauje
POSTAVI stn  $\leftarrow$  0 (* preskače se upis sloga Sn u datoteku Dn*)
INAČE
AKO JE o(Sp) = m TADA m modifikacija
Modifikuj Sn u skladu sa Sp
INAČE (*pokušaj upisa već postojećeg sloga*)
KRAJ AKO
KRAJ AKO
INAČE
AKO JE o(Sp) = n TADA
POSTAVI stn  $\leftarrow$  1
POSTAVI Sn  $\leftarrow$  k(Sp), p(Sp))
INAČE (*pokušaj brisanja ili modifikacije nepostojećeg sloga*)
KRAJ AKO
KRAJ AKO
POSTAVI k(Sp_pret)  $\leftarrow$  k(Sp)
ČITAJ_SEQ(Dp, Sp, statp)
KRAJ RAD čitaj_Dp
AKO JE stn = 1 TADA
PIŠI_SEQ(Dn, Sn, statn)
INAČE
KRAJ AKO
INAČE
KRAJ AKO
KRAJ PROCESA obrada_sloga

```

Slika 9.9.

Za ažuriranje sekvenčne datoteke datotekom promena, koja sadrži  $N_v^n$  novih slogova,  $N_v^b$  slogova za brisanje i  $N_v^m$  slogova za modifikaciju, odnosno

$$\lceil (N_v^n + N_v^b + N_v^m + 1)/f \rceil$$

blokova, potrebno je izvršiti  $\lceil (N + l)/f \rceil$  pristupa za čitanje stare datoteke i  $\lceil (N + N_v^n - N_v^b + l)/f \rceil$  pristupa za upis blokova u novu datoteku. Srednji broj pristupa, potreban za ažuriranje sekvencijalne datoteke jednim od  $N_v$  slogova datoteke promena, iznosi

$$\bar{R} = \frac{B_s + B_n + B_v}{N_v},$$

gde je  $B_s$  broj blokova stare,  $B_n$  broj blokova nove, a  $B_v$  broj blokova datoteke promena.

**Primer 9.4.** Na slici 9.10 je prikazana datoteka promena  $D_p$  sa  $N_v = 9$  slogova, putem koje treba ažurirati datoteku  $D_s$  sa slike 9.1. Datoteka  $D_p$  je uređena saglasno neopadajućim vrednostima ključa datoteke  $D_s$ . Sadrži dva nova sloga, tri sloga za modifikaciju i četiri sloga za brisanje. Na slici 9.11 je nacrtana struktura nove datoteke  $D_n$ , dobijena ažuriranjem.

$A_1$	<table border="1"> <tr><td>03</td><td><math>p(S_v^1)</math></td><td><math>b</math></td><td>06</td><td><math>p(S_v^2)</math></td><td><math>n</math></td><td>13</td><td><math>p(S_v^3)</math></td><td><math>m</math></td></tr> </table>	03	$p(S_v^1)$	$b$	06	$p(S_v^2)$	$n$	13	$p(S_v^3)$	$m$
03	$p(S_v^1)$	$b$	06	$p(S_v^2)$	$n$	13	$p(S_v^3)$	$m$		
$A_2$	<table border="1"> <tr><td>19</td><td><math>p(S_v^4)</math></td><td><math>m</math></td><td>25</td><td><math>p(S_v^5)</math></td><td><math>b</math></td><td>29</td><td><math>p(S_v^6)</math></td><td><math>b</math></td></tr> </table>	19	$p(S_v^4)$	$m$	25	$p(S_v^5)$	$b$	29	$p(S_v^6)$	$b$
19	$p(S_v^4)$	$m$	25	$p(S_v^5)$	$b$	29	$p(S_v^6)$	$b$		
$A_3$	<table border="1"> <tr><td>49</td><td><math>p(S_v^7)</math></td><td><math>m</math></td><td>55</td><td><math>p(S_v^8)</math></td><td><math>n</math></td><td>64</td><td><math>p(S_v^9)</math></td><td><math>b</math></td></tr> </table>	49	$p(S_v^7)$	$m$	55	$p(S_v^8)$	$n$	64	$p(S_v^9)$	$b$
49	$p(S_v^7)$	$m$	55	$p(S_v^8)$	$n$	64	$p(S_v^9)$	$b$		
$A_4$	<table border="1"> <tr><td>*</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	*								
*										

Slika 9.10.

$A_1$	<table border="1"> <tr><td>06</td><td><math>p(S_n^1)</math></td><td>07</td><td><math>p(S_n^2)</math></td><td>13</td><td><math>p(S_n^3)</math></td></tr> </table>	06	$p(S_n^1)$	07	$p(S_n^2)$	13	$p(S_n^3)$	$A_2$	<table border="1"> <tr><td>15</td><td><math>p(S_n^4)</math></td><td>19</td><td><math>p(S_n^5)</math></td><td>23</td><td><math>p(S_n^6)</math></td></tr> </table>	15	$p(S_n^4)$	19	$p(S_n^5)$	23	$p(S_n^6)$
06	$p(S_n^1)$	07	$p(S_n^2)$	13	$p(S_n^3)$										
15	$p(S_n^4)$	19	$p(S_n^5)$	23	$p(S_n^6)$										
$A_3$	<table border="1"> <tr><td>27</td><td><math>p(S_n^7)</math></td><td>34</td><td><math>p(S_n^8)</math></td><td>43</td><td><math>p(S_n^9)</math></td></tr> </table>	27	$p(S_n^7)$	34	$p(S_n^8)$	43	$p(S_n^9)$	$A_4$	<table border="1"> <tr><td>49</td><td><math>p(S_n^{10})</math></td><td>55</td><td><math>p(S_n^{11})</math></td><td>*</td><td></td></tr> </table>	49	$p(S_n^{10})$	55	$p(S_n^{11})$	*	
27	$p(S_n^7)$	34	$p(S_n^8)$	43	$p(S_n^9)$										
49	$p(S_n^{10})$	55	$p(S_n^{11})$	*											

Slika 9.11.

**Primer 9.5.** Sekvencijalna datoteka od  $N = 10\ 000$  slogova sa faktorom blokiranja  $f = 10$  se ažurira datotekom promena od  $N_v = 9\ 300$  slogova sa faktorom blokiranja  $f_v = 10$ . Pri tome je  $N_v^a = 900$ ,  $N_v^b = 1\ 000$  i  $N_v^m = 7\ 400$ .

Za čitanje blokova datoteke promena potrebno je izvršiti  $R_{uk}^k = 931$  pristup eksternoj memoriji. Za čitanje blokova stare sekvencijalne datoteke potrebno je izvršiti  $R_{uk}^s = 1\ 001$  pristup. Za upis blokova nove datoteke potrebno je izvršiti  $R_{uk}^n = 991$  pristup. Ako je za jedan pristup potrebno 10 msek, tada ukupno vreme razmene podataka između operativne i eksternih memorija, pri ovom ažuriranju, iznosi oko 29 sek. □



## 9.5 OBLASTI PRIMENE I OCENA KARAKTERISTIKA SEKVENCIJALNE DATOTEKE

Sekvencijalna organizacija predstavlja najpogodniju organizaciju datoteke za redoslednu obradu. Kako se režim redosledne obrade u praksi često koristi, često se koristi i sekvencijalna organizacija datoteke. Svoju pogodnost za redoslednu obradu i popularnost u praksi, sekvencijalna organizacija datoteke duguje činjenici da su logički susedni slogovi memorisani u fizički susednim lokacijama. Učitavanjem jednog bloka iz datoteke, u operativnoj memoriji se dobija  $f$  slogova, koji sa velikom verovatnoćom učestvuju u  $f$  suksesivnih koraka obrade. Na taj način, srednji broj pristupa  $\bar{R}$  datoteci, potreban za pronalaženje logički narednog sloga, teži recipročnoj vrednosti faktora blokiranja  $f$ , kada broj slogova  $N_v$  vodeće datoteke teži broju slogova  $N$ -obradjivane sekvencijalne datoteke.

U dobre strane sekvencijalne organizacije datoteke mogu se ubrojati i ekonomično korišćenje memorijskog prostora i mogućnost korišćenja i magnetne trake i magnetnog diska kao medijuma.

U nedostatke se ubrajaju: nepogodnost za direktnu obradu, potreba sortiranja pri formiranju, relativno dugotrajan postupak ažuriranja nakon intervala vremena tokom kojeg sadržaj datoteke nije u saglasnosti sa stanjem klase entiteta, čije podatke datoteka sadrži.

## 9.6 OPERATIVNI SISTEM Unix I SEKVENCIJALNE DATOTEKE

Sekvencijalne datoteke imaju jednostavnu strukturu i njihovo korišćenje je lako. Operativni sistem prepoznaje te činjenice i podržava njihovo korišćenje putem niza naredbi svog komandnog jezika. Te komande prepostavljaju da je sekvencijalna datoteka tekstualna sa znacima za novu liniju (\n), kao separatorima sloga i blanko znacima, kao delimiterima polja. Za takve datoteke, Unix nudi naredbe, kao što su:

- cat i less (prikaz sadržaja datoteke na ekranu),
- wc (vraća broj linija, broj reči i broj karaktera u datoteci) i
- egrep (prikazuje one linije datoteke na ekranu, koje sadrže dati niz znakova).

Ove, kao i druge komande se mogu kombinovati, što rezultuje u ad hoc programima za obradu datoteka.

**Primer 9.6.** Neka je datoteka `student.data`, posmatrana u glavi 3 ove knjige realizovana korišćenjem blanko znakova, kao separatora polja i znakova za novu liniju, kao separatora slogova. Naredna komanda prikazuje na ekranu samo one slogove, kod kojih polje `broj_poena` sadrži vrednost 80.

```
% egrep '^[^ ]+ [^ ]+ 80' student.data
```

Komanda egrep filtrira slogove putem regularnog izraza '`^[^ ]+ [^ ]+ 80`', gde regularni izraz '`[^ ]+`' znači „bilo koji broj znakova, koji nije blanko znak”.

Naredna komanda vraća broj slogova, koji zadovoljavaju uslov `broj_poena` sadrži vrednost 80, broj polja i broj znakova u tim slogovima.

```
% egrep '^[^ ]+ [^ ]+ 80' student.data | wc -l
```