SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1413

Raspoređivanje dretvi za operacijske sustave s blagim vremenskim ograničenjima

Mario Iličić

Zagreb, lipanj 2010.

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1413

Raspoređivanje dretvi za operacijske sustave s blagim vremenskim ograničenjima

Mario Iličić

Zagreb, lipanj 2010.

Sadržaj

1. Uvod

Svaki računalni sustav ima za zadaću izvršavanje određenih programa, i to mu je sigurno jedan od glavnih razloga postojanja. Postoje razne vrste programa, s raznim funkcijama i ciljevima. Svaki program, bilo korisnički bilo sustavski, sastoji se od barem jedne dretve. Redoslijed izvođenja dretvi određuju određeni parametri dretvi. Neke dretve imaju veći prioritet pa imaju prednost nad onim manjeg prioriteta. Osim prednosti u izvođenju, prioritet može određivati i druge prednosti, kao npr. ako je veći prioritet, dretve dobiva veći dio procesorskog vremena za izvođenje.

U sustavu praktički u svakom trenu postoji po nekoliko dretvi koje čekaju za izvršavanje. Koja će postati aktivna, ovisi o operacijskom sustavu i algoritmu korištenom za raspoređivanje. Postoje razne kombinacije operacijskih sustava i pripadnih algoritama za raspoređivanje. U ovom radu će se detaljno proučiti i simulirati Potpuno pravedan raspoređivač (eng. *Completely Fair Scheduler*, skraćeno CFS) koji se koristi u Linux operacijskom sustavu. Na primjeru je pokazano kako on radi. Za simulaciju CFS-a koriste se crveno-crna stabla, struktura podataka koja se koristi u realnoj implementaciji CFS-a.

Na početku su objašnjeni raspoređivač dretvi i raspoređivanje općenito. Slijedi pregled algoritama za raspoređivanje koje koriste suvremeni operacijski sustavi. Također, objašnjavaju se kriteriji koji se uzimaju u obzir prilikom raspoređivanja. Zatim su prikazani CFS i crveno-crna stabla koja on koristi. Na kraju je prikazana simulacija CFS-a sa primjerom.

2. Raspoređivanje – vrste raspoređivača

Suvremeni operacijski sustavi za zadaću imaju izvući optimalne performanse iz osnovnih hardverskih resursa. To se postiže uglavnom virtualizacijom dva glavna hardverska resursa: centralne procesorske jedinice (skraćeno CPU) i memorije. Također, moderni operacijski sustavi su višezadaćni sustavi jer pružaju okruženje za obavljanje više zadataka koje daje svakom zadatku njegov vlastiti virtualni CPU. Zadatak općenito nije svjestan činjenice da to ne mora biti isključivo korištenje CPU-a.

Davanjem vlastitog virtualnog memorijskog adresnog prostora svakom zadatku koji se onda preslikava u stvarnu memoriju sustava, virtualizira se memorija. Također, zadatak nije svjestan činjenice da se njegove virtualne memorijske adrese možda neće preslikati u iste fizičke adrese u stvarnoj memoriji. CPU se virtualizira tako da se „podijeli“ između više zadataka – tj. svaki zadatak za izvršavanje dobije mali dio CPU-a u regularnom intervalu. Odabiranje zadatka zove se raspoređivanje, a algoritam koji se koristi za odabir jednog zadatka na vrijeme od više dostupnih zove se raspoređivač. Raspoređivač je jedna od najvažnijih komponenti kod bilo kojeg operacijskog sustava. Implementirati algoritam za raspoređivanje je teško zbog nekoliko razloga. Prvo, prihvatljivi algoritam mora dodijeliti vrijeme CPU-a tako da zadaci većeg prioriteta dobiju prednost nad zadacima nižeg prioriteta. U isto vrijeme, raspoređivač mora zaštiti procese nižeg prioriteta od gladovanja. Drugim riječima, procesima nižeg prioriteta mora biti omogućeno izvođenje, bez obzira koliko se procesa višeg prioriteta natječe za vrijeme CPU-a. Raspoređivači također moraju biti pažljivi, tako da se procesi pojavljuju za izvođenje istovremeno, bez prevelikog utjecaja na propusnost sustava.

Razlikujemo RT operacijske sustave i one koji to nisu, odnosno RT dretve i 'normalne' dretve. Kod RT sustava, redoslijed izvođenja dretvi određuje prioritet. Dretva većeg prioriteta istiskuje dretvu nižeg prioriteta. Kod 'normalnih' dretvi raspoređivanje je malo drugačije. Postoji prioritet, ali ima drugačiju funkciju. Određuje koliki dio procesorskog vremena će dretva dobiti u odnosu na druge dretve.

Ovaj rad se koncentrira na ne-RT dretve, dakle na 'normalne' dretve. Ali prije toga, kratak pregled operacijskih sustava sa pripadajućim raspoređivačima (tablica 1).

Tablica 1 Operacijski sustavi sa pripadajućim raspoređivačem

Operacijski sustavi	Algoritam za raspoređivanje
Windows 3.1x	Kooperativni raspoređivač
Windows 95, 98, ME	Samo za 32-bitne operacije
Windows NT, XP, Vista	Višerazinski redovi s povratkom (MFQ)
Mac OS (do inačice 9)	Kooperativni raspoređivač
Mac OS X	*Mach (kernel)*
Linux (do inačice2.5)	Višerazinski redovi s povratkom (MFQ)
Linux (2.5 – 2.6.23)	0(1) raspoređivač
Linux (od inačice 2.6.23)	Potpuno pošteno raspoređivanje (CFS)
Solaris, FreeBSD, NetBSD	Višerazinski redovi s povratkom (MFQ)

2.1 Pregled operacijskih sustava sa pripadajućim raspoređivačima

Windows

Inačice windows operacijskih sustava koriste različite raspoređivače. Tako rane verzije MS-DOS-a uopće nemaju raspoređivač. Inačice Windows 3.1x imaju kooperativni raspoređivač, ne-istiskujući (eng. *non-preemptive*), drugim riječima ne prekida program. Inačica 95 uvodi rudimentaran istiskujući raspoređivač, ali ipak, 16-bitne aplikacije ne mogu se prekinuti u izvođenju. Windows NT koristi višerazinske redove s povratkom (eng. *multilevel feedback queue*, skraćeno MFQ), dok Vista koristi modifikaciju istog.

Mac OS

Mac OS 9 koristi kooperativni raspoređivač, gdje jedan proces upravlja višestrukim dretvama. Jezgra operacijskog sustava procesima upravlja algoritmom za kružno posluživanje (eng. *Round-robin*). Mac OS X koristi *Mach* jezgru (*Mach* je

mikrojezgra za operacijski sustav razvijena za istraživačke operacijske sustave, prvenstveno distribuirano i paralelno računanje). Svaka dretva je povezana sa vlastitim odvojenim procesom.

Linux

Od verzije 2.5 Linux je koristio MFQ s prioritetnim razinama od 0 do 140. Verzije od 2.6 do 2.6.23 koriste O(1) raspoređivač. Inačica 2.6.23 zamijenila ga je CFS raspoređivačem.

FreeBSD, NetBSD, Solaris

Ovi operacijski sustavi koriste MFQ.

2.2 'Normalne' dretve – non-Real-Time

Kao što je već rečeno, raspoređivanje 'normalnih' dretvi je malo drugačije nego raspoređivanje RT dretvi. Kod RT dretvi prioritet određuje koja dretva će se prije izvršiti. Kod 'normalnih' dretvi prioritet ima drugu ulogu. On određuje koliki dio procesorskog vremena će dretva dobiti u odnosu na druge dretve. Na primjer, ako imamo dvije dretve, jedna prioriteta 5, druga 10 (manji broj označava veći prioritet), i pustimo ih da rade neko vrijeme, procesor bi trebao dretvi prioriteta 5 dati više vremena nego dretvi prioriteta 10.

Kod raspoređivanja 'normalnih' dretvi velik značaj ima procesorsko vrijeme, odnosno podjela istog, kao i različite metode određivanja trajanja vremenskog intervala dodijeljenog nekoj dretvi, i/ili frekvencija dodjele intervala dretvi. Osim tih parametara, mogu se dodatno koristiti i neki drugi, kao na primjer dosadašnja potrošnja procesorskog vremena. Na temelju dosadašnje potrošnje procesorskog vremena, određuje se da li se dretvi smanjuje ili povećava prioritet, a to pak određuje da li će dretva dobiti više ili manje procesorskog vremena. Zbog toga se kod raspoređivanja 'normalnih' dretvi, često se koristi metoda *multilevel feedback queue* (skraćeno MFQ) koja duljim poslovima smanjuje prioritet, a samim time i dio procesorskog vremena koji posao treba dobiti, i obrnuto, [6].

Opis rada metode MFQ:

➔ Dolaskom u sustav,dretva se postavlja na kraj reda pripravnih dretvi s najvećim prioritetom

➔ kada se dretva počne izvršavati, tada:

⇨ ako se do kraja izvrši, ona napušta sustav

⇨ ako dodijeljeno vrijeme istekne, dretva se prekida i smješta na kraj reda pripravnih dretvi idućeg manjeg prioriteta

⇨ ako se pri povratku u pripravno stanje dretva blokira, vraća se na kraj reda u isti red iz kojeg je otišla pri blokiranju

⇨ ako je dretva UI tipa, i blokira se na UI operaciji, pri povratku u pripravno stanje postavlja se u red pripravnih dretvi idućeg, za jedan većeg, prioriteta od reda iz kojeg je prethodno otišla u blokirano stanje

Dvije najraširenije arhitekture danas su Win32 i Linux. Zbog toga ćemo pogledati raspoređivanje kod njih.

2.3 Raspoređivanje u Win32 arhitekturi

Jedna od najraširenijih arhitektura danas je Win32 arhitektura. U njoj dretve imaju osnovni i dinamički prioritet. Osnovni prioritet računa se korištenjem prioritetne klase procesa i prioritetne razine dretve, dok se dinamički prioritet odnosno dinamičko povećanje prioriteta radi se samo za dretve prioriteta manjeg od 16 (samo za ne-RT dretve, 'normalne' dretve). Prioritet se povećava dretvama koje rijetko koriste procesor, dok smanjivanje dobivaju one koje puno koriste procesor, [6].

2.4 Raspoređivanje u Linuxu

Kao i većina modernih operacijskih sustava, Linux je višezadaćni operacijski sustav, i ima raspoređivač. Za RT raspoređivanje koristi algoritme za kružno posluživanje (eng. *Round-Robin*) i posluživanje po redu prispijeća (eng. *First In First Out*, skraćeno FIFO) i prioritete 0-99. Za raspoređivanje 'normalnih' dretvi koriste se raspoređivač *CFS* i prioriteti 100-140.

3. Kriteriji za raspoređivanje

Danas postoje različiti algoritmi raspoređivanja, svaki sa svojim prednostima i nedostacima. Da bismo odabrali najoptimalniji algoritam za raspoređivanje u određenoj situaciji, moramo uzeti u obzir različita svojstva pojedinog algoritma. Proučavajući karakteristike raspoređivača možemo napraviti razliku među algoritmima i odabrati najbolji. Različiti su kriteriji usporedbe algoritama za raspoređivanje. Najznačajniji kriteriji su:

- **Iskorištenost procesora (eng. *CPU utilization*)** - postići maksimalnu zaposlenost procesora

- **Propusnost (eng. *Throughput*)** – broj izvršenih procesa u jedinici vremena. Za duge poslove to može biti npr. jedan proces po satu,a za kratke može biti 10 procesa po sekundi.

- **Vremenski ciklus (eng. *Turnaround time*)** – vrijeme potrebno za dovršavanje posla. Uključuje vrijeme čekanja u memoriji, čekanja u redu pripravnih, vrijeme izvršavanja, i vrijeme obavljanja U/I operacije.

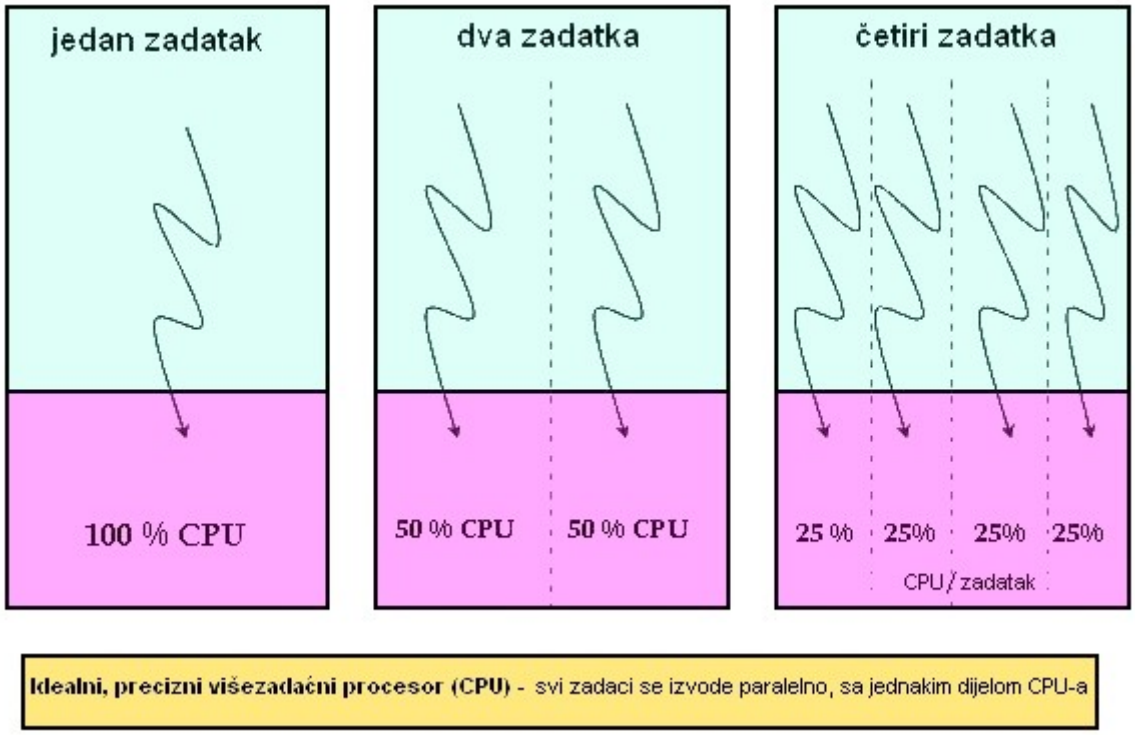
- **Vrijeme čekanja (eng. *Waiting time*)** – zbroj perioda potrošenih na čekanje u redu pripravnih. Raspoređivač nema efekta na dio vremena za vrijeme kojeg se proces izvršava ili obavlja U/I operaciju. On ima efekta samo na vrijeme potrošeno čekajući u radu pripravnih.

- **Vrijeme odgovora (eng. *Response time*)** – vrijeme od podnošenja zahtjeva do prvog odgovora procesa.

Poželjno je povećati iskorištenost procesora i propusnost, a smanjiti ciklusno vrijeme, vrijeme čekanja i vrijeme odgovora. U mnogim slučajevima, možemo optimizirati prosječne vrijednosti. Ipak, u nekim okolnostima poželjno je optimizirati maksimalne ili minimalne vrijednosti, a ne prosječne, [2].

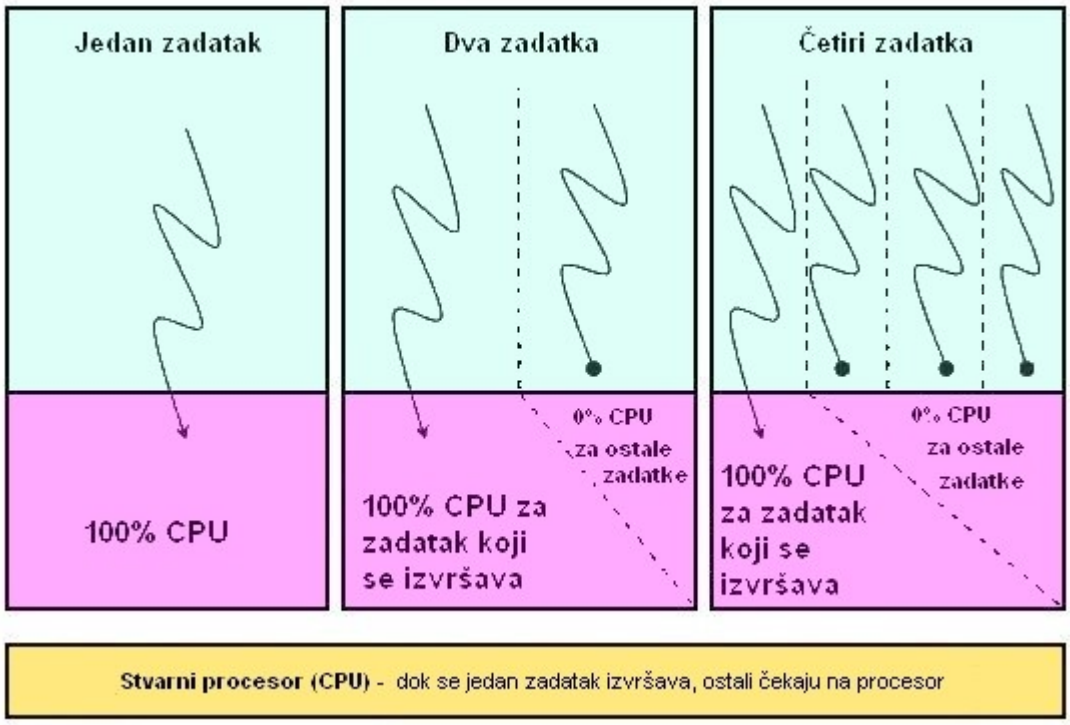
4. Completely Fair Scheduler – CFS

Completely Fair Scheduler (potpuno pravedan raspoređivač) je ime za raspoređivač zadataka na Linuxu. Implementirao ga je Ingo Molnar. „CFS u osnovi modelira idealni, precizni, višezadaćni CPU na stvarnom hardveru“ (Molnar, 2007). To bi “u prijevodu“ značilo da je to hardverski CPU koji izvršava više procesa istovremeno (paralelno), dajući svakom procesu jednak dio snage procesora (Slika 1). Dakle, ako se samo jedan proces izvršava, on će dobiti 100% snage procesora, ako su dva procesa, izvršavat će se u paraleli svaki sa 50% snage procesora. Ako se pak izvršavaju četiri procesa, svaki će dobiti 25% snage procesora i, također, izvršavat će se paralelno. Dakle, takav CPU bi bio pravedan prema svim procesima u sustavu.



Slika 1 Idealni, precizni višezadaćni procesor

Očito je da takav idealni CPU ne postoji, ali CFS pokušava softverski oponašati takav procesor. U stvarnosti procesoru samo jedan zadatak može biti dodijeljen. Zbog toga, svi ostali zadaci čekaju za vrijeme dok se taj jedan izvodi. Drugim riječima, zadatak koji se trenutno izvršava dobiva 100% procesorske snage, ostali dobivaju 0% (Slika 2). To očito nije pravedno.



Slika 2 Realni procesor

Kako mu samo ime govori, CFS ukloniti nepoštenu podjelu procesorskog vremena među zadacima u sustava. On pokušava održati poštenu podjelu CPU-a koja bi bila dostupna svakom procesu u sustavu. Pa onda pokreće “pošteni sat“ na stvarnoj CPU brzini. Stopa porasta poštenog sata se računa dijeljenjem graničnog vremena (u nano-sekundama) ukupnim brojem procesa koji čekaju. Dobivena vrijednost je iznos CPU vremena na koji svaki proces ima pravo.

Dok proces čeka na CPU, raspoređivač prati iznos vremena koji bi on iskoristio na idealnom procesoru. Ovo vrijeme čekanja se koristi za rangiranje procesa za raspoređivanje i određivanje iznosa vremena dopuštenog procesu za izvršavanje. Raspoređivač odabire proces s najvećim vremenom čekanja i dodjeljuje ga procesoru. Kada se proces izvršava, smanjuje mu se vrijeme čekanja, dok se vremena drugih zadataka koji čekaju povećavaju. To u osnovi znači da će se nakon nekog vremena pojaviti proces koji ima najveće vrijeme čekanja, a trenutni zadatak će se istisnuti. Koristeći te principe, CFS pokušava biti pošten prema svim zadacima i uvijek pokušava imati sustav sa vremenom čekanja jednakim nula za svaki proces – svaki proces ima jednak dio CPU-a (kao što to radi idealni, precizni, višezadaćni CPU).Prije CFS-a Linux je koristio raspoređivač O(1) koji se temeljio na aktivnim redovima. Za razliku od njega, njegov nasljednik CFS bazira se na crveno-crnim stablima, [5].

5. Podatkovna struktura – crveno-crna stabla

Crveno-crno stablo je tip samobalansirajućeg stabla za pretraživanje, struktura podataka korištena u računarskoj znanosti. Izvornu strukturu je izumio Rudolf Bayer 1972. godine nazvavši je „simetrično B-stablo“, a današnji naziv potječe od Leonidasa J. Guibasa i Roberta Sedgewicka iz 1978. godine. Kompleksno je, ali daje dobra vremena izvođenja za najlošije slučajeve za svoje operacije i efikasno je u praksi: može tražiti, dodavati i brisati u vremenu O(log *n*), gdje je *n* cijeli broj elemenata u stablu (Tablica 2). Kao što se vidi u tablici 2, vremenska kompleksnost za prosječni i najgori slučaj je jednaka, [4]. Iako jako jednostavno, crveno-crno stablo je binarno stablo za pretraživanje koje dodaje i uklanja elemente inteligentno, kako bi se osiguralo da je stablo poprilično uravnoteženo.

Tablica 2 Vremenska kompleksnost crveno-crnog stabla

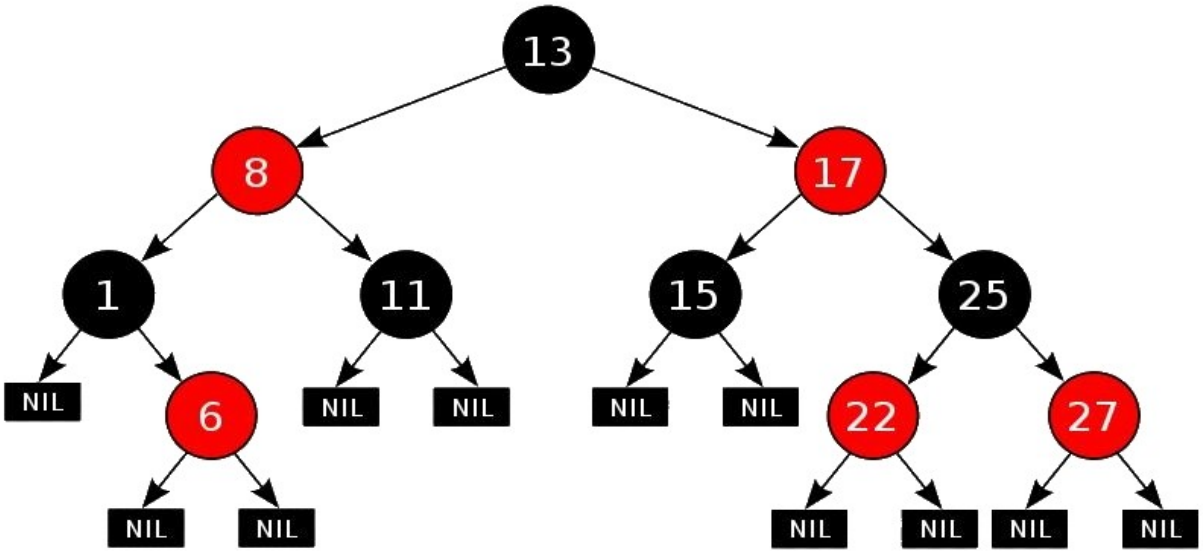
Crveno-crno stablo		
Tip	Stablo	
Vremenska kompleksnost u velikoj O notaciji:		
	Prosječno	Najgori slučaj
Prostor	O(n)	O(n)
Traženje	O(log n)	O(log n)
Dodavanje	O(log n)	O(log n)
Brisanje	O(log n)	O(log n)

Crveno-crno stablo je specijalni tip binarnog stabla. Koristi se u računalnoj znanosti za organiziranje dijelova istih tipova podataka, kao što su fragmenti teksta ili brojevi. Čvorovi listovi crveno-crnog stabla ne sadrže podatke. Ti listovi ne trebaju biti eksplicitno u memoriji računala jer *null* pokazivač na dijete može kodirati činjenicu da je to dijete list, ali ako su listovi čvorovi, to pojednostavljuje mnoge algoritme za operacije na crveno-crnom stablu. Da se uštedi memorija, jedan čvor stražar može obavljati ulogu svih listova čvorova; sve reference iz unutarnjih čvorova na listove dalje se povezuju sa čvorom stražarom.

Crveno-crna stabla, kao sva binarna stabla, dopuštaju efikasno *in-order* obilaženje članova – lijevo-korijen-desno. Vrijeme traženja rezultira iz obilaženja od korijena do listova, i zato balansirano stablo, koje ima najmanju moguću visinu, rezultira vremenom traženja O(log*n*), [4].

5.1 Svojstva crveno-crnih stabala

Crveno-crno stablo je binarno stablo za pretraživanje gdje svaki čvor ima atribut *boju*, čija je vrijednost ili *crveno* ili *crno* (Slika 3, [4]).



Slika 3 Primjer crveno-crnog stabla

Osim običnih zahtjeva koja imaju binarna stabla, crveno-crna stabla moraju zadovoljiti i sljedeće njima svojstvene zahtjeve:

1. Čvor je ili crven ili crn.
2. Korijen je crn.
3. Svi listovi su crni.
4. Oba djeteta crvenog čvora su crna.
5. Svaki jednostavni put od odabranog čvora do bilo kojeg lista potomka sadrži jednak broj crnih čvorova.

Iz svih navedenih ograničenja proizlazi ključno svojstvo crveno-crnih stabala: najduži put od korijena do bilo kojeg lista nije više od dvaput duži nego što je dug najkraći put od korijena do bilo kojeg lista u tom stablu. Zbog toga je to stablo grubo

balansirano. Operacije kao što su dodavanje, brisanje i traženje vrijednosti u najgorim su slučajevima vremenski proporcionalna visini stabla. Ta granica omogućuje crveno-crnim stablima da budu efikasniji u najgorim slučajevima, u odnosu na obična binarna stabla. Da bi se vidjelo zašto ta svojstva garantiraju to, dovoljno je napomenuti da put ne može imati dva crvena čvora u nizu, zbog svojstva 4. Najkraći mogući put ima sve crne čvorove, a najduži mogući put alternira između crvenih i crnih čvorova. Budući da svi putovi imaju jednak broj crnih čvorova, zbog svojstva 5, to pokazuje da nema puta koji je više od dvaput duži od bilo kojeg drugog puta.

Također, moguće je da čvor ima samo jedno dijete, i da čvor-list sadrži podatke. Prikaz crveno-crnog stabla u tom obliku mijenja nekoliko svojstava i komplicira algoritme. Zbog tog razloga, koriste se "null listovi", koji ne sadrže podatke i samo služe da pokažu gdje stablo završava, kako je pokazano u prethodnom primjeru. Ti čvorovi se često izostavljaju u slikama, rezultirajući stablo koje je čini se kontradiktorno gornjim načelima, ali u stvari nije. Posljedica toga je da svi unutarnji čvorovi imaju dva djeteta, iako jedno ili oba djeteta mogu biti null čvorovi. Neko tumači crveno-crna stabla kao binarna stabla za pretraživanje čiji su rubovi, umjesto čvorova, obojani u crveno ili crno, ali to ne stvara razliku. Boja čvora u jednom stablu odgovara boji ruba u drugom koji povezuje čvor sa njegovim roditeljem, osim što je čvor uvijek crn a odgovarajući rub ne postoji, [4].

6. Programska simulacija CFS raspoređivača

Ranije je opisana struktura podataka koju koristi CFS raspoređivač – crveno-crna stabla. Prije toga, opisan je princip po kojem on radi. U ovom poglavlju se opisuje jedno programsko ostvarenje, odnosno simulacija tog raspoređivača. Programski kod je pisan u *Visual Studiu 2008*, programskim jezikom C. U privitku se nalaze upute za simulaciju raspoređivača. Na priloženom CD-u nalazi se cijeli napisani kod sa komentarima.

6.1 Ideja

Ideja je da se u simulaciji što više približi rad stvarnog CFS raspoređivača, odnosno da se pokuša što preciznije simulirati rad istog. To bi značilo koristiti i slične, ako ne i iste, strukture podataka i algoritme. Tako će se upotrijebiti crveno-crno stablo i karakteristike realnog CFS-a. U simulaciji je potrebno i dretve reprezentirati odgovarajućim strukturama koje će imati neke karakteristike stvarnih dretvi i na taj način "povećati realnost" simulacije. I na kraju, treba pažljivo rukovati vremenima koja se nalaze u kontekstu realnih CFS-ova da bi simulacija bila što realnija.

6.2 Izvedba - opis koda

Programski kod se sastoji od nekoliko cjelina, svaka sa svojom svrhom. Kao i svaki program, tako i ovaj ima glavnu ili *main* funkciju. U njoj se pozivaju funkcije koje pripremaju okruženje za simulaciju CFS raspoređivača. Budući da CFS koristi crveno-crna stabla kao podatkovnu strukturu, i ovdje se na početku stvara takvo stablo koje se inicijalno sastoji samo od korijenskog čvora i "stražara" koji označava kraj stabla. Zatim slijedi stvaranje dretvi. Budući da je ovo simulacija, ni dretve nisu prave, nego su predstavljene određeno strukturom. Ta struktura je u biti čvor koji se dodaje u stablo. Zbog toga što je s jedne strane čvor, a s druge opisnik dretve, ta struktura sadrži parametre i jednog i drugog. Dakle, sadrži parametre koje ima jedan opisnik dretve, kao što su ID i prioritet dretve. Isto tako sadrži i parametre koje ima čvor u crveno-crnom stablu, kao što su boja čvora, pokazivač na lijevo dijete, pokazivač na desno dijete i pokazivač na čvor roditelj. Osim tih parametara, struktura sadrži i tri posebna parametra. Dva su vezana za raspoređivač, odnosno koriste se u radu raspoređivača. Preostali parametar koristi se prilikom povezivanja opisnika u listi postojećih dretvi. Struktura opisnika/čvora sa opisnim komentarima prikazana je kodom ispod (Kôd 1). Budući da struktura je ujedno i opisnik i čvor, u daljnjem tekstu radi jednostavnijeg zapisa koristi se naziv dretva.

Nakon što su dretve stvorene, slijedi njihovo pohranjivanje u ranije stvoreno stablo funkcijom `NapuniStablo`. Ta funkcija ne dodaje dretve u stablo eksplicitno, nego poziva funkciju koja to radi. To je funkcija `RB_INSERT`. Prilikom stavljanja dretvi u stablo, u obzir se uzima parametar dretve `dobrota`. Struktura crveno-crno stablo, kako je navedeno u jednom od prethodnih poglavlja, ima određena svojstva koja se striktno moraju čuvati u svakom trenutku. Zbog toga se prilikom svakog dodavanja dretve u stablo poziva funkcija koja popravlja svojstva ukoliko su narušena. To je funkcija `RB_INSERT_FIXUP`. U tom održavanju svojstava stabla, funkcija koristi druge potrebne funkcije.

```
typedef struct _Node {
	long ID;                    /* ID dretve                                */
	int prioritet_dretve;       /* prioritet                                */
	double dobrota;             /* trajanje dretve (u mikro sekundama)      */
	char NodeColor;             /* boja čvora                               */
	int efikasnost;             /* vjerojatnost da dretva bude prekinuta    */
	struct _Node *right;        /* kazaljka na desno dijete                 */
	struct _Node *left;         /* kazaljka na lijevo dijete                */
	struct _Node *parent;       /* kazaljka na roditelja                    */
	struct _Node *iduci_opisnik;/* kazaljka za poezivanje u listu postojece  */
	int t;
} Node;
```

Kôd 1 Struktura opisnika dretve/čvora

Dodavanjem svih dretvi i očuvanjem svojstava, pripremljena je okolina za CFS raspoređivač. Simulacija CFS-a počinje pozivom funkcije `rasporedjivac`. Kôd te funkcije nalazi se u daljnjem tekstu (Kôd 2).

Kao što je pojašnjeno u jednom od prethodnih poglavlja, CFS dodjeljuje procesor dretvi koja ima najveće vrijeme čekanja. U ovom programskom ostvarenju funkciju tog vremena obnaša parametar `dobrota`. CFS uzima dretvu najveće `dobrote` uklanjajući je istovremeno iz stabla. Nakon toga, dretva se prividno izvršava. Budući da dretve nisu “stvarne“ nego simulirane odnosno predstavljene određenom strukturom, u kodu je izvršavanje dretve reprezentirano funkcijom `Sleep`. Za razliku od *real-time* dretvi, prioritet kod 'normalnih' dretvi određuje koliko će dretva dobiti procesorskog vremena. Tako je i u ovoj simulaciji, vrijeme koje će se dretva izvršavati određeno je prioritetom dretve.

```
void rasporedjivac(){
	long T;
	FILE *dat;

	while( 1 ) {

		aktivna_D = uzmi_iducu_dretvu( stablo->root );

		RB_DELETE( aktivna_D );

		T = T0 * pow( 1.15, aktivna_D->prioritet_dretve );

		printf( "ID = %ld dobrota = %.0lf, T=%u, pri=%d\n\n",
			aktivna_D->ID, aktivna_D->dobrota, T,
			aktivna_D->prioritet_dretve );

		if ( rand() % 100 > aktivna_D->efikasnost ) {
			T = T / ( rand() % 4 + 2 );
			printf ( "Skracen rad = %u\n", T );
		}

		Sleep ( T );

		aktivna_D->t += T;

		aktivna_D->dobrota -= T / pow( 1.15, aktivna_D->prioritet_dretve );

		RB_INSERT( aktivna_D );
	}

}
```

Kôd 2 Simulirani CFS raspoređivač

Da bi se što više približilo stvarno okruženje gdje dretva može biti prekinuta raznim prekidima, uveden je mehanizam koji će to simulirati. Za potrebe tog mehanizma, dretva je dobila parametar nazvan `efikasnost`. On određuje vjerojatnost da dretva bude prekinuta, te na taj način određuje koliko je dretva "važna" (što je važnija manja je vjerojatnost da će biti prekinuta u radu):

```
if ( rand() % 100 > aktivna_D->efikasnost ) {
	T = T / ( rand() % 4 + 2 );
	printf ( "Skracen rad = %u\n", T );
}
```

Kôd 3 Mehanizam simuliranja prekida

Što se dretva više izvršava, `dobrota` joj se smanjuje. Nakon isteka dodijeljenog joj vremena, ona se ponovno dodaje u stablo, ali sada sa manjom `dobrotom`.

U jednom trenutku `dobrota` dretve koja se izvršava će se smanjiti ispod iznosa `dobrote` druge dretve koja čeka na izvršavanje. Budući da raspoređivač procesor dodjeljuje dretvi najveće `dobrote`, uzima dretvu čija je `dobrota` postala najveća. I tako CFS konstantno raspoređuje dretve.

U kôdu se nalazi i ispis koji ovdje predstavlja jedini izlaz, i pomaže da se vidi da raspoređivanje funkcionira prema ideji.

6.3 Rezultat

Kada se pogledaju i prouče izlazi odnosno ispisi, vidi se da je ideja ostvarena jako dobro. Raspoređivač dretvi koja je najviše čekala dodjeljuje procesor, tj. izvršava se dretva koja je najviše čekala. Izvršavanjem se dretvi smanjuje `dobrota`. Također, ovisno o parametru *efikasnost*, simuliraju se i prekidi prema zamišljenom.

U idućem poglavlju se nalazi detaljno opisan primjera sa pripadajućim slikama.

7. Primjeri simulacije implementiranog raspoređivača

Prema uputama iz privitka, ručno unesemo 5 dretvi sa sljedećim parametrima:

Tablica 3 Parametri za primjer

	1	2	3	4	5
ID	**1**	**2**	**3**	**4**	**5**
PRIORITET	**4**	**3**	**9**	**7**	**1**
DOBROTA	**7000**	**3000**	**8000**	**4000**	**5000**
EFIKASNOST	**50**	**90**	**30**	**100**	**70**

Unosom tih dretvi, stablo se napunilo i raspoređivač je počeo svoj rad. Ispod se nalazi nekoliko prvih ispisa:

```
ID=3, time=3517, pri=9, dobr=8000, T=3517
ID=3, time=4396, pri=9, dobr=7000, T=879
ID=1, time=1749, pri=4, dobr=7000, T=1749
ID=3, time=6154, pri=9, dobr=6750, T=1758
ID=3, time=9671, pri=9, dobr=6251, T=3517
ID=1, time=2332, pri=4, dobr=6000, T=583
ID=1, time=2915, pri=4, dobr=5667, T=583
ID=1, time=3264, pri=4, dobr=5333, T=349
ID=3, time=10843, pri=9, dobr=5251, T=1172
ID=1, time=5013, pri=4, dobr=5134, T=1749
ID=5, time=1150, pri=1, dobr=5000, T=1150
ID=3, time=14360, pri=9, dobr=4918, T=3517
…
```

U ispisu `ID` označava ID dretve, `time` je ukupno vrijeme koje se dretva izvršavala, `pri` je prioritet dretve, a `T` je kvant procesorskog vremena dodijeljenog dretvi za izvršavanje.

Pomoću prioriteta možemo odrediti koliki najveći kvant procesorskog vremena pojedina dretva može dobiti. To računamo prema formuli:

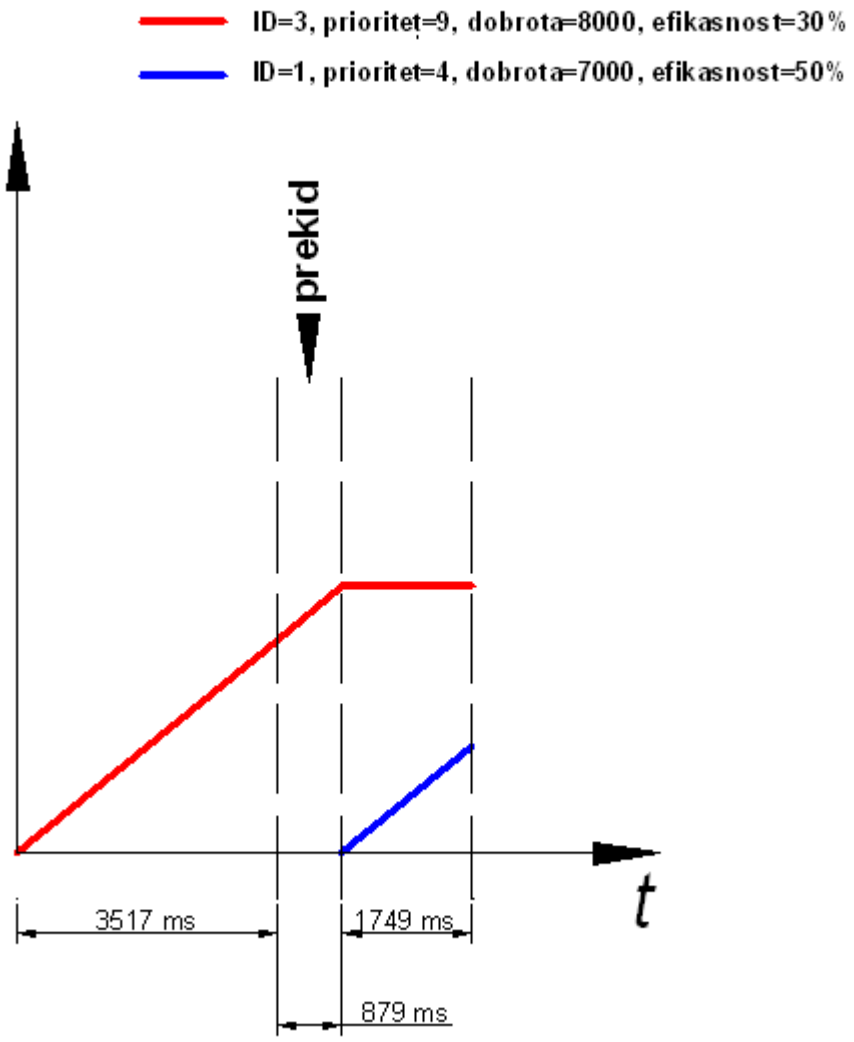
$$T = T_0 * 1.15^{prioritet}$$

gdje je T_0 = 1000.

Prema tome dobivamo:

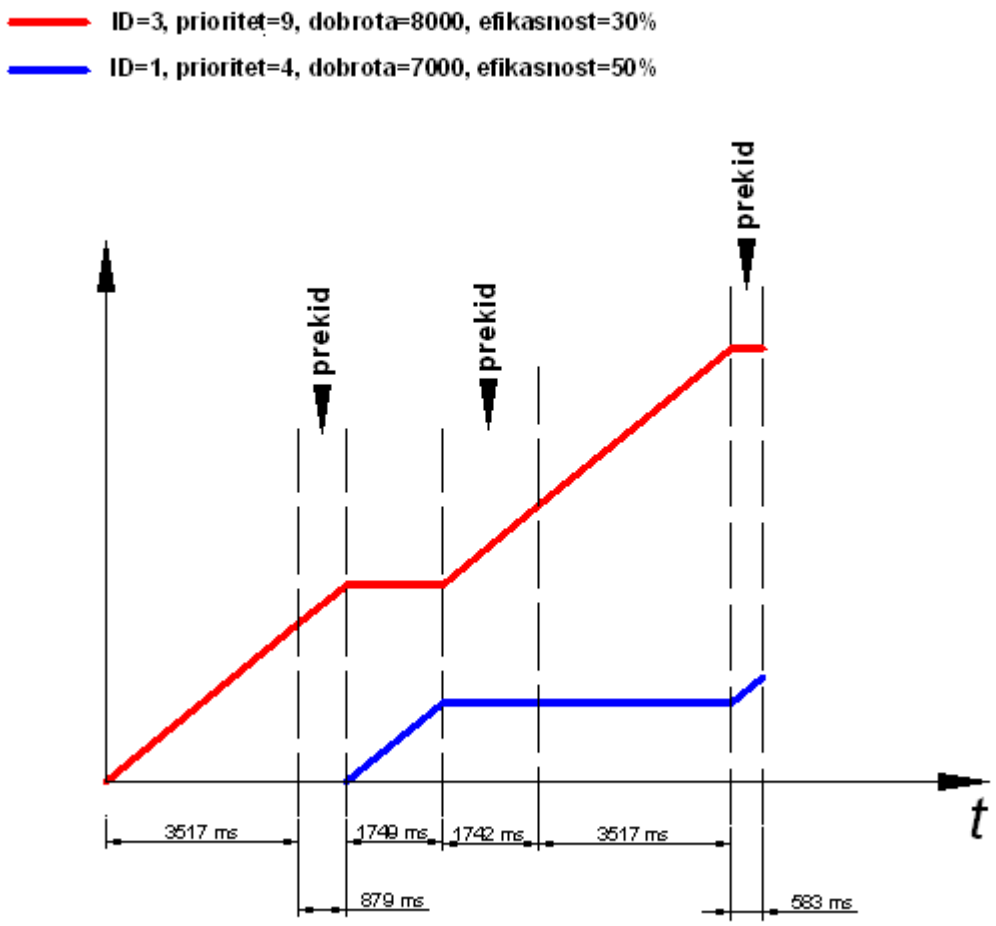
```
ID=1   T=1749 ms

ID=2   T=1520 ms

ID=3   T=3517 ms

ID=4   T=2660 ms

ID=5   T=1150 ms
```

Vidimo da se prvo krenula izvršavati dretva najveće `dobrote`, dretva 3. Budući da joj nakon prvog kvanta `dobrota` ostala najveća u stablu, dretva 3 se izvršava i u drugom koraku. U drugom koraku se dogodio prekid, pa se dretva izvršila samo 879 ms prije nego je prekinuta. Nakon drugog koraka `dobrota` dretve 3 nije više najveća u stablu. Najveća `dobrota` je `dobrota` dretve 1, i ona se počinje izvršavati. Ona se izvršava cijeli kvant koji joj je dodijeljen. Prva tri koraka prikazana su na idućoj slici (Slika 4).



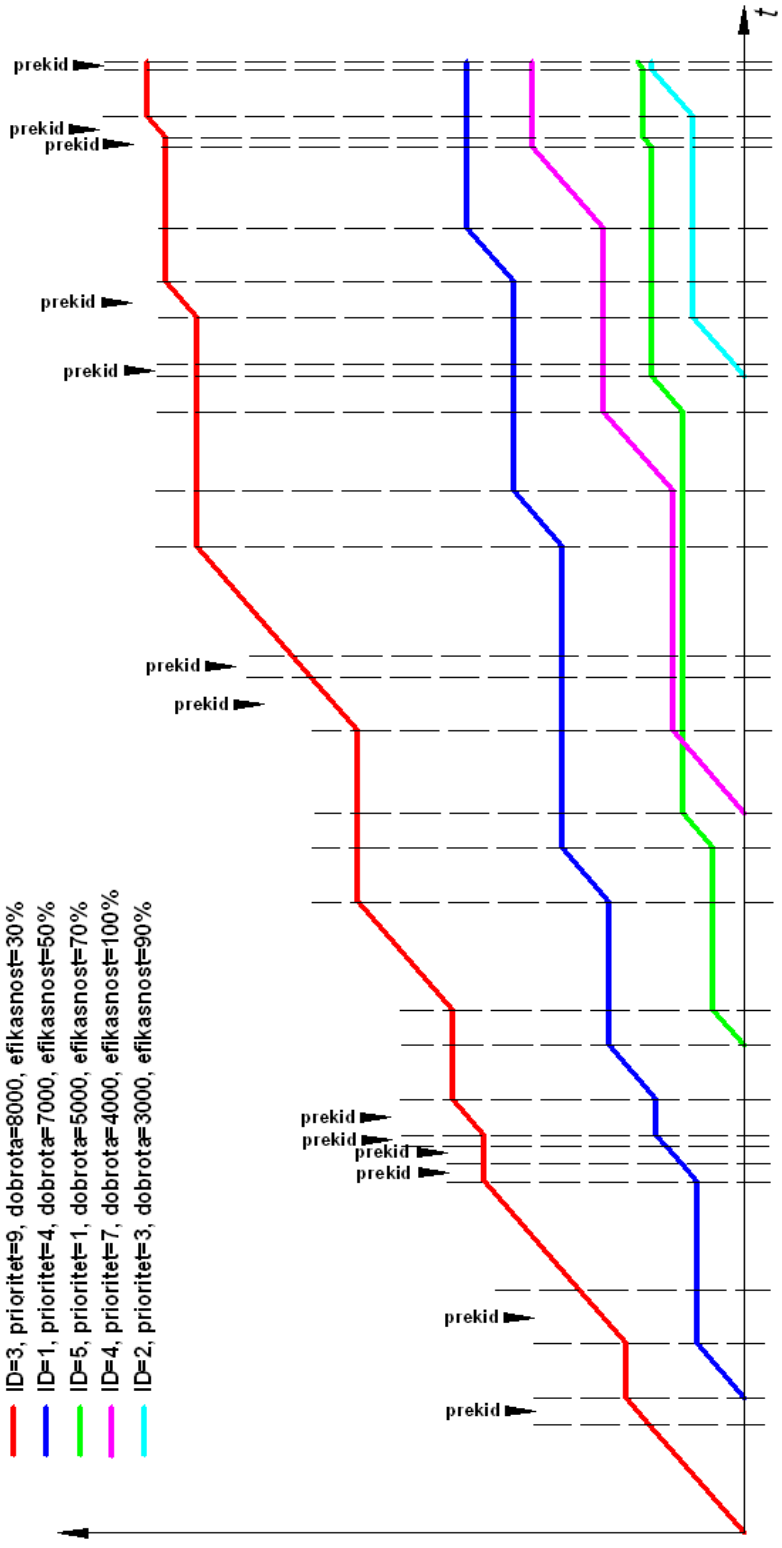
Slika 4 Prva tri koraka raspoređivanja dretvi

U iduća dva koraka ponovno se dretva 3 izvršava. Dok se ona izvodi, ostale dretve čekaju. Nakon što se izvodila dvaput zaredom, `dobrota` joj se smanjila ispod `dobrote` dretve 1, i dretva 1 je iduća dretva koja će se izvršavati (Slika 5). Kao što se vidi na slici, dretva 3 se u četvrtom koraku izvršavala manje nego što bi trebala. Drugim riječima, ponovno se dogodio prekid. To je slučaj i kod dretve 1.



Slika 5 Prvih šest koraka raspoređivanja u primjeru

Raspoređivanje se nastavlja na prema dosad opisanom postupku. Na slici 6 je prikazano stanje u jednom od kasnijih trenutaka raspoređivanja. Na priloženom CD-u nalazi se cijeli ispis prikazan tom slikom.

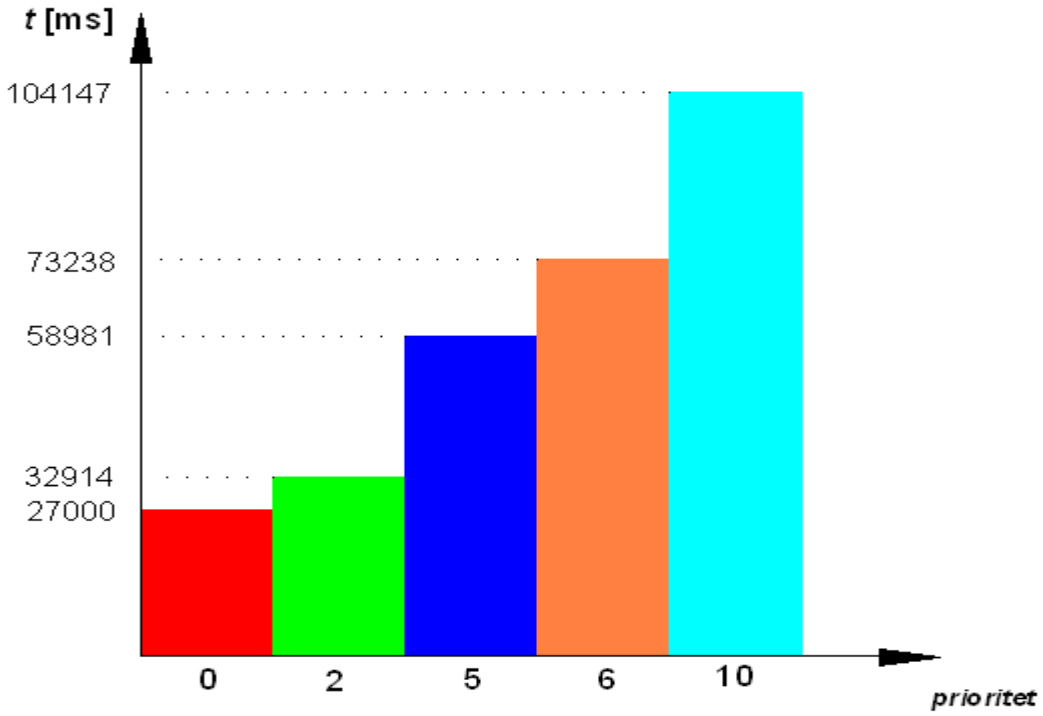


Slika 6 Raspoređivanje nakon većeg broja koraka

U nastavku se nalazi još jedan primjer, dugog izvođenja programa – oko 5 minuta. Primjer pokazuje kako su dretve s većim prioritetom dobile više procesorskog vremena od dretvi manjeg prioriteta. To je još jedan od pokazatelja da implementirana simulacija radi kako treba i kako je zamišljena, kao realni CFS. U tablici se nalaze podaci, a na slici grafički prikaz (Tablica 4, Slika 7). Ispis primjera priložen je na CD-u.

Tablica 4 Podaci za drugi primjer

ID	Prioritet	Dobiveno procesorsko vrijeme [ms]
1111	0	27000
2222	2	32914
3333	5	58981
4444	6	73238
5555	10	104147



Slika 7 Dobiveno procesorsko vrijeme u odnosu na prioritet

8. Zaključak

Pomoću *Visual Studia 2008* simuliran je CFS raspoređivač koji se danas koristi u Linux operacijskom sustavu. Za potrebe simulacije upoznali smo se sa načinom na koji on u stvarnosti radi, i strukturom podataka koju on koristi – crveno-crna stabla. Implementacija je pisana u jeziku C. Osim kriterija koje on koristi za raspoređivanje, obrađeni su i oni najčešći koji se uzimaju u obzir kod raspoređivanja. Na primjeru je pokazano da implementacija jako dobro simulira stvarni CFS.

9. Literatura

[1] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C. *Introduction To Algorithms. 2nd Edition*. USA: McGraw-Hill Book Company, 2001

[2] WIKIPEDIA , *Scheduling (computing)*, http://en.wikipedia.org/wiki/Scheduling_(computing), 06.06.2010.

[3] WIKIPEDIA, *Completely Fair Scheduler*, http://en.wikipedia.org/wiki/Completely_Fair_Scheduler, 18.05.2010.

[4] WIKIPEDIA, *Red-black tree*, http://en.wikipedia.org/wiki/Red-black_tree, 15.05.2010.

[5] LINUX JOURNAL, *Completely Fair Scheduler*, http://www.linuxjournal.com/magazine/completely-fair-scheduler?page=0,0, 06.06.2010.

[6] JELENKOVIĆ, L. Predavanja iz predmeta Sustavi za rad u stvarnom vremenu, 2009.

10. Sažetak

Jedan od ciljeva rada je bio implementacija jednog od algoritama za raspoređivanje u operacijskim sustavima s blagim vremenskim ograničenjem. Izabran je Potpuno Pravedan Raspoređivač. Da bi se to ostvarilo, prvo smo se upoznali sa principima i svojstvima takvog realnog raspoređivača i potrebne strukture podataka. Upotrebom crveno-crnih stabala i principa po kojima radi, implementiran je raspoređivač. Zbog upotrebe crveno-crnih stabala, vremenska složenost raspoređivača jednaka je vremenskoj složenosti crveno-crnih stabala. Prioritet i vrijeme čekanja su glavni parametri koji se uzimaju u obzir prilikom raspoređivanja. Poglavlje 6 pokazuje da simulacija nimalo ne odstupa od rada stvarnog raspoređivača.

Ključne riječi: *raspoređivanje, raspoređivač, potpuno pravedan raspoređivač, višerazinski redovi s povratkom, vrijeme čekanja, višezadaćnost, crveno-crna stabla*

11. Abstract

Title: Thread scheduling for operating systems with soft timings constrains

One of the aims of this study was an implementation of algorithms for scheduling in operating systems, with soft timing constrains. Elected Completely Fair Scheduler. To this end, we first learn about the principles and properties of such a real scheduler and the required data structures. Using red-black trees and the principles by which works, is implemented scheduler. Due to the use of red-black trees, the time complexity of the scheduler is equal to the time complexity of red-black trees. Priority and waiting time are the main parameters that are consider when scheduling. Chapter 6 shows that the simulation does not deviate at all from the work of a real scheduler.

Keywords: *scheduling, scheduler, completely fair scheduler, multilevel feedback queue, wait time, multitasking, red-black trees*

12. Privitak – tehnička dokumentacija

Upute za simuliranje

Pokretanjem kôda pojavljuje se ispis: `”Unesite broj dretvi: “.`

Unosimo broj između 1 i 10, jer trenutno se može unijeti maksimalno 10 različitih *dobrota*. Nakon toga pita nas da li želimo ručno ili automatski unijeti parametre:

`“Unosenje parametara dretve [Rucno (0), Automatski (1)]: “.`

Ako unesemo 1, parametri će se automatski odrediti. Ako pak unesemo 0, parametre unosimo ručno, uzimajući u obzir poruke odnosno ispise na ekranu koji se pojavljuju. Nakon toga počinje beskonačno raspoređivanje dretvi koje možemo prekinuti pritiskom na Ctrl+C.