SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 173

OSTVARENJE RASPOREĐIVANJA DRETVI

Berislav Hunjadi

Zagreb, lipanj 2011.

Zahvaljujem mentoru doc. dr. sc. Leonardu Jelenkoviću na predloženoj temi i pruženoj stručnoj pomoći i savjetima kod izrade rada.

Sadržaj

1. Uvod

Operacijski je sustav posrednik između sklopovlja i programa na nekom računalu. Većina današnjih računala ima operacijski sustav, a jedine su iznimke jednostavna sklopovlja poput mikrokontrolera i sličnih naprava koje se koriste za specifične primjene. Operacijski sustav dio je gotovo svih kompleksnijih računala ili sklopova kao što su stolna računala, poslužitelji, mobilni uređaji, itd. Funkcioniranje kompleksnih sustava gotovo je nezamislivo bez pripadajućeg operacijskog sustava jer on olakšava upotrebu, primjerice, vanjskih uređaja, korištenje višedretvenosti i upravljanja memorijom oko kojih ne mora brinuti sama aplikacija, već se za tu funkcionalnost koristi sučelje operacijskog sustava.

Suvremene jezgre operacijskih sustava poput Linuxa, BSD-a i OS X potekle su od Unixa, dok su Microsoft Windows proizašli iz MS-DOS-a koji je bio neovisan o Unixu. Iako se ti operacijski sustavi razlikuju prema filozofiji prema kojoj su nastali, zajednička im je osnovna funkcionalnost koju pružaju, a to je komunikacija s vanjskim jedinicama, upravljanje memorijom, raspoređivanje zadataka i mogućnost korištenja sinkronizacijskih mehanizama. Prvi su operacijski sustavi bili jednozadaćni i trebali su samo pružati sučelje preko kojeg se komuniciralo s vanjskim jedinicama. No, ubrzo je došla potreba za više zadataka u sustavu, a ostvarenje višezadaćnosti u samom programu nije bilo dovoljno efikasno i isplativo. Sklopovski je paralelizam već bio postignut uvođenjem protočne arhitekture, a programsko rješenje višezadaćnosti omogućilo je kreiranje proizvoljnog broja zadataka u sustavu i njihovog izvođenja na jednom ili više procesora. Višezadaćni su operacijski sustavi morali upravljati memorijom i raspoređivati dretve na temelju nekog algoritma raspoređivanja. Pojavom virtualne memorije mogućnosti višezadaćnih operacijskih sustava su se proširile, dok je sigurnost izvođenja programa povećana uvođenjem korisničkog i jezgrenog načina rada procesora.

Raspoređivač zadataka bitan je dio višezadaćnog sustava čiji je zadatak u suradnji s ostalim dijelovima sustava omogućiti izvođenje više različitih zadataka istovremeno ili, u slučaju da to nije moguće, tako da izvođenje prividno izgleda istovremeno. Raspoređivač odlučuje na temelju nekog algoritma raspoređivanja i daje dretvama ili određeni vremenski odsječak ili ih pušta da same prepuste procesorsko vrijeme ostalim dretvama. Vremenski odsječci moraju biti mali tako da rad više zadataka na jednom procesoru izgleda kao istovremen. Različiti algoritmi pogodni su za različite primjene pa tako, primjerice, novije

verzije Microsoft Windows operacijskih sustava koriste algoritam višerazinskog povratnog reda čekanja koji je dobra opcija za interaktivni rad budući da daje prednost kratkim zadacima i onim zadacima vezanim za ulazno-izlazne naredbe, a to je većina korisničkih zadataka. Rad takvog raspoređivača korisniku djeluje ugodno i sve se operacije obavljaju bez čekanja. S druge strane, Linux koristi potpuno pravedan raspoređivač koji nastoji biti što pravedniji prema svim zadacima u sustavu i dati im onoliko vremena koliko bi dobili u idealnom slučaju. Taj raspoređivač daje slabije rezultate na interaktivnoj razini, ali zato postiže odlične rezultate za pozadinske zadatke i čini ga pogodnim raspoređivačem za poslužitelje. Sustavi za rad u stvarnom vremenu pak koriste druge algoritme jer su ograničenja zadataka u takvim sustavima puno stroža nego u običnim sustavima.

Teoretski dio ovog rada prikazan je u poglavljima 2. do 4. u kojima se objašnjavaju osnovni pojmovi potrebni za razumijevanje raspoređivanja, nakon toga algoritmi raspoređivanja koji se koriste u operacijskim sustavima te u 4. poglavlju detaljniji teoretski prikaz potpuno pravednog raspoređivača. Ostvarenje praktičnog dijela objašnjeno je u 5. poglavlju, a rezultati ispitivanja prikazani su i objašnjeni u poglavlju 6.

2. Raspoređivač zadataka u operacijskom sustavu

Cilj raspoređivača zadataka u operacijskom sustavu je omogućiti istodobno izvođenje više zadataka na računalu gdje najčešće postoji mnogo više zadataka koji su spremni za izvođenje nego što postoji fizičkih procesora. Iz prethodne tvrdnje slijedi da se svi spremni zadaci u sustavu ne mogu izvoditi istovremeno već raspoređivač zadataka na temelju nekog algoritma raspoređivanja određuje koji će se zadatak sljedeći izvoditi i koliki će mu vremenski odsječak biti pridijeljen. Da bi svi zadaci u sustavu izgledali kao da se izvode istodobno, vremenski odsječci koji im se pridaju su kratki, reda veličine 10 ms ili maksimalno 100 ms. Pri raspoređivanju se u obzir mora uzeti prioritet nekog zadatka te se na temelju prioriteta mora donijeti odluka kada će taj zadatak dobiti procesorsko vrijeme i koliki će mu biti vremenski odsječak.

Sa stajališta jezgre operacijskog sustava zadatak je neka jedinica raspoređivanja, a to je najčešće dretva. Zbog toga se može reći da raspoređivač zadataka u operacijskom sustavu raspoređuje dretve koje pak predstavljaju zadatak ili dio zadatka.

2.1. Procesi i dretve

Proces je instanca računalnog programa koji se izvodi. Računalni program je samo pasivni skup strojnih instrukcija, dok proces predstavlja njihovo stvarno izvođenje. Svaki proces sadrži na početku jednu, primarnu, dretvu koja može stvoriti druge dretve koje će pripadati istom procesu [1]. Dretve su osnovne jedinice raspoređivanja u operacijskom sustavu te one čine dijelove procesa koji se mogu izvoditi na procesoru.

Proces se sastoji od preslike strojnog koda učitane iz izvršne datoteke, pripadajuće memorije u virtualnom adresnom prostoru, opisnika resursa koje je operacijski sustav dodijelio procesu, sigurnosnih obilježja, procesorskog stanja (konteksta), unikatnog identifikatora, varijabli okoline, klase prioriteta, minimalne i maksimalne veličine prostora rada i najmanje jedne dretve.

Dretve nekog procesa dijele njegovu memoriju ili preciznije, dijele instrukcije i kontekst tog procesa. S druge strane različiti procesi ne dijele zajedničke resurse, ali se njihova komunikacija može ostvariti nekim drugim mehanizmima za međuprocesnu komunikaciju operacijskog sustava (poruke, zajednička memorija i slično). Osim navedenih razlika različiti procesi imaju odvojene adresne prostore, dok dretve dijele isti adresni prostor.

Dretve sadrže upravljače iznimaka, prioritet raspoređivanja, lokalne podatke, unikatni identifikator i strukture koje će operacijski sustav koristiti pri promjeni konteksta. Proces također može sadržavati prioritet raspoređivanja pa se u tom slučaju prioritet dretve izvodi iz prioriteta same dretve, ali i prioriteta procesa kojem ta dretva pripada. Kontekst dretvi je manji od konteksta procesa i sastoji se od registara procesora koje koristi dretva (osnovni registri, registri koprocesora), stoga jezgre, bloka okruženja dretve i korisničkog stoga, a može sadržavati i sigurnosni kontekst kao i procesi.

Proces je, što se resursa tiče, najzahtjevniji dio jezgrinog raspoređivanja, dok su jezgrine dretve manje zahtjevne za raspoređivanje. Jezgrine dretve mogu biti prekinute ako ih operacijski sustav raspoređuje po prekidljivom algoritmu raspoređivanja. Osim jezgrinih dretvi postoje i korisničke dretve koje su implementirane u korisničkom prostoru. Jezgra operacijskog sustava nije svjesna da korisničke dretve postoje te se one moraju raspoređivati u korisničkom prostoru. Neke implementacije koriste nekoliko jezgrinih dretvi nad kojima se izvode korisničke dretve s ciljem boljeg iskorištenja višeprocesorskog sustava. Najčešće se kreira onoliko jezgrinih dretvi koliko postoji procesora u sustavu [2]. Prednost korisničkih dretvi je njihovo brzo kreiranje i relativno jednostavno upravljanje, ali veliki je nedostatak što mogu biti blokirane ako su sve njihove pripadajuće jezgrine dretve blokirane iako su u tom trenutku korisničke dretve spremne za izvođenje. Vlakna (*engl.* fibers) su najlakša jedinica raspoređivanja i raspoređuju se suradnički što njihovu implementaciju čini još jednostavnijom od jezgrinih i korisničkih dretvi. One se koriste za poboljšanje performansi u slučajevima za koje jezgrin raspoređivač nije optimalno podešen.

U nekom trenutku dretva može biti u jednom od sljedećih stanja: aktivno, pripravno, blokirano i pasivno. Aktivno stanje označava da se ta dretva trenutno izvodi na nekom od procesora. U pripravnom su stanju sve dretve koje su spremne na izvođenje (a ne izvode se) i to je početno stanje svake dretve, dok su u blokiranom stanju one dretve koje čekaju na neki resurs ili događaj. U pasivnom stanju su sve one dretve koje su završile, ali se njihovi resursi još nisu oslobodili. Memorijski prostor koji zauzima pasivna dretva može biti oslobođen nakon što resursi te dretve više nisu potrebni niti jednoj drugoj dretvi u sustavu i u tom trenutku pasivna dretva prestaje postojati unutar tog sustava. U aktivno stanje dretva ulazi kada je raspoređivač zadataka odabere na temelju nekog algoritma iz strukture pripravnih dretvi. Dretva može iskoristiti cijeli vremenski odsječak i u tom se slučaju stavlja natrag u red pripravnih. Ako se dretva blokira na nekom resursu (npr. čekanje na semaforu) ili događaju (npr. može čekati na završetak druge dretve,

spavati i slično), ona iz aktivnog ide u blokirano stanje. Kada se resurs oslobodi ili dogodi neki događaj, blokiranu dretvu operacijski sustav oslobađa iz reda čekanja i stavlja u strukturu pripravnih dretvi.

2.2. Višezadaćni rad

Operacijski sustavi koji podržavaju višezadaćni rad (*engl.* multitasking) raspoređuju slobodno procesorsko vrijeme između pripravnih dretvi. Raspoređivač daje određeni vremenski odsječak dretvi koju je izabrao kao sljedeću ili pušta dretvu da se izvodi sve do kraja ili do trenutka kada ne postane blokirana. Raspoređivač najčešće koristi male vremenske odsječke tako da korisnik dobiva dojam paralelnog rada više zadataka. Posljedica višezadaćnog rada je promjena konteksta koja uključuje pohranjivanje sadržaja svih registara procesora u opisnik dretve i obnavljanje stanja procesora kojeg koriste i druge dretve u sustavu.

Raspoređivač zadataka može raditi na dva osnovna načina – raspoređivanjem po prioritetu i raspoređivanjem podjelom vremena (gdje prioritet također može biti faktor u odluci o raspoređivanju).

Operacijski sustavi mogu imati različite strategije raspoređivanja: multiprogramski sustavi, sustavi s raspodjelom vremena (*engl.* time-sharing, multitasking) i sustavi u realnom vremenu (*engl.* real time systems).

2.2.1. Promjena konteksta

Promjena konteksta (*engl.* context switch) jedan je od ključnih elemenata u operacijskom sustavu. Ono što pripada u kontekst ovisi o procesoru (tj. registrima koje sadrži) i operacijskom sustavu. Operacija promjene konteksta je računalno zahtjevna i vrlo česta pa se vrijeme promjene konteksta nastoji minimizirati, stoga se dosta pažnje posvećuje optimiranju korištenja promjene konteksta. Optimiranje promjene konteksta svodi se na izbjegavanje nepotrebnih promjena konteksta kad je to moguće. Promjena konteksta dretve zahtijeva manje vremena od promjene konteksta procesa.

Pseudokod promjene konteksta:

```
promjena_konteksta() {
        Stavi registre na stog;
        Spremi pokazivače na kod i podatke;
        Spremi pokazivač stoga;
```

```
        Uzmi sljedeći proces za izvršavanje;

        Uzmi pokazivač stoga tog procesa;
        Uzmi pokazivače na kod i podatke;
        Uzmi registre sa stoga;
    }
```

Algoritam raspoređivanja definira sljedeći proces koji će se izvršiti (funkcija `Uzmi sljedeći proces za izvršavanje` u pseudokodu). Stanje procesora mora se pohraniti da bi neka dretva mogla kasnije nastaviti svoje izvođenje točno s onoga mjesta i u onom kontekstu (stanju procesora) na kojem je bila prekinuta neovisno o svim ostalim dretvama koje su u toku izvođenja promijenile stanje procesora. Prije pokretanja nove dretve njezino se stanje obnavlja nakon čega se ta dretva može početi izvoditi.

Procesni informacijski blok (*engl.* Process Control Block, PCB) je struktura u kojoj se pohranjuju informacije o kontekstu. Strukturi je moguće pristupiti samo iz jezgre operacijskog sustava i ona sadrži sve potrebne informacije za upravljanje određenim procesom. Zbog važnosti tih informacija ta struktura mora biti zaštićena od pristupa korisnika. U nekim operacijskim sustavima se PCB smješta na početak stoga procesa u jezgri jer je to prikladna i zaštićena lokacija. PCB struktura je ovisna o implementaciji, ali se u njoj najčešće nalaze, direktno ili indirektno:

- identifikator procesa,
- vrijednosti registara i programskog brojača,
- adresni prostor procesa,
- prioritet,
- informacije o ulazu i izlazu (npr. lista otvorenih datoteka),
- pokazivač na sljedeću PCB strukturu (tj. pokazivač na sljedeći proces koji se treba izvršiti).

Za vrijeme promjene konteksta, jezgra mora prekinuti izvođenje trenutne dretve te kopirati vrijednosti registara i ostale podatke u PCB, pročitati vrijednosti iz PCB strukture nove dretve te je pokrenuti.

Promjena konteksta se događa u sljedećim situacijama:

- promjena aktivne dretve u višezadaćnom sustavu koji koristi prekidljivi raspoređivač,

- obrada prekida prilikom čega jezgra obrađuje prekid u kontekstu prekinutog procesa (u jezgrinom ili korisničkom načinu rada) i mora spremiti dovoljno informacija da bi se nakon obrade mogao nastaviti prekinuti proces,
- izmjena korisničkog i jezgrinog načina rada.

Promjena konteksta može biti implementirana sklopovski i programski. Procesor Intel 80386 i sve sljedeće arhitekture imaju sklopovsku podršku za promjenu konteksta, ali operacijski sustavi Microsoft Windows i Linux ne koriste tu mogućnost. Programska promjena konteksta može se koristiti na svim procesorima i omogućava pohranjivanje i obnovu samo onog stanja koje je potrebno u toku izvođenja (npr. ako neka dretva ne koristi MMX registre, nije ih potrebno spremati u kontekst). Osnovni kontekst čine pokazivač stoga, pokazivač instrukcija, statusni registar, registri opće namjene i registri podatkovnih segmenata. Ostale registre se ne mora spremati ako se ne koriste. Primjerice, procesor može generirati iznimku kada aplikacija prvi puta koristi MMX ili SSE instrukciju što operacijski sustav može iskoristiti tako da sljedeći put u kontekst dretve koja koristi te instrukcije stavi i pripadajuće registre. Većina novih operacijskih sustava ne koristi segmentaciju pa nije potrebno spremati registre segmenata pri promjeni konteksta. Sklopovska promjena konteksta sprema i te registre, a kada procesor učitava nove registre segmenata, mora uraditi sve provjere pristupa i dozvola koje usporavaju postupak promjene konteksta. Zbog toga se sklopovske promjene konteksta napuštaju pa i noviji 64 bitni procesori u 64 bitnom načinu rada (*engl.* long mode) ne pružaju mogućnost sklopovske promjene konteksta [3].

2.2.2. Multiprogramski sustavi

Na početku razvoja računala procesorsko je vrijeme bilo skupo, a periferni su uređaji bili veoma spori. Kada je program trebao pristupiti vanjskom uređaju, procesor je trebao čekati na taj uređaj [4]. To je bilo veoma neefikasno. Prvi pokušaji stvaranja multiprogramskih sustava bili su 1960-ih godina. Osnovna je ideja bila pružiti mogućnost izvođenja ostalim procesima u sustavu kada je aktivni proces postao blokiran na nekom vanjskom uređaju. U tom je trenutku sustav spremio kontekst aktivnog procesa i pokrenuo sljedeći proces koji je bio spreman za izvođenje. Glavni je nedostatak ovog principa da neki proces ne mora komunicirati s okolinom pa ništa ne garantira da će se ostali procesi izvršiti jer taj proces nikad neće biti u stanju čekanja na vanjski uređaj.

Multiprogramski sustavi počeli su se povećano koristiti nakon pojave virtualne memorije što je omogućilo korištenje fizičke memorije i resursa operacijskih sustava kao

da su svi istovremeni programi međusobno nepostojeći i nevidljivi. Program se u kontekstu multiprogramskih sustava može shvatiti kao skup zadataka. Svaki zadatak je relativno mali skup instrukcija koji logički čine jedan korak prema izvršenju nekog posla ili programa. Zadatak često završava zahtjevom za micanjem podataka pa je to pogodno vrijeme da se nekom drugom programu predaju sustavski resursi. U multiprogramskim se sustavima paralelno izvođenje zadataka postiže kada operacijski sustav prepozna priliku da može prekinuti jedan program koji je trenutno između zadataka i u tom trenutku preda kontrolu drugom programu. Mogućnost sustava da dijeli svoje resurse podjednako ili prema određenim prioritetima ovisi o dizajnu programa kojima sustav upravlja i kako često oni mogu biti prekinuti. Dakle, ne može se bilo koji skup programa raspoređivati u multiprogramskom sustavu pa ispravnost sustava ovisi o skupu programa koji se trenutno nalaze u sustavu. Taj problem rješavaju višezadaćni sustavi.

2.2.3. Sustavi s raspodjelom vremena (višezadaćni sustavi)

Nakon što je korištenje računala uznapredovalo na interaktivnu razinu, multiprogramski pristup više nije bio dovoljno kvalitetno rješenje za novonastale zahtjeve. Sa stajališta operacijskog sustava, korisnicima je potrebno omogućiti rad sa sustavom koji svakom korisniku izgleda kao da se njegovi programi jedini izvode u cijelom sustavu, neovisno o broju drugih korisnika i programa u sustavu. Takav je rad omogućio princip u kojem se vremenski odsječci pridaju svakom programu. Razlika između multiprogramskih sustava i sustava s raspodjelom vremena je da potonji mogu prekinuti programe u izvođenju, a ne samo prilikom blokiranja na nekom vanjskom uređaju. Pritom se pod pojmom program smatra neka jedinica izvođenja, što je u modernim operacijskim sustavima najčešće dretva.

Multiprogramski sustavi žele maksimalno iskoristiti resurse i pritom ne brinu o kojim drugim kriterijima (npr. odziv, propusnost). S druge strane, višezadaćni sustavi se bolje ponašaju po svim kriterijima osim iskorištenja resursa zato jer češće mijenjaju aktivne zadatke, a time i kontekst. Kao primjer se mogu uzeti dva zadatka koja koriste isključivo procesor (nema ulazno-izlaznih operacija), prvi ima vrijeme izvođenja od 30 min, a drugi samo 1 min. Prvi dolazi u sustav u početno vrijeme sustava (0 min), a drugi za 5 min. U multiprogramskom će sustavu drugi zadatak čekati dok se prvi ne izvede do kraja kako bi se on mogao početi izvoditi, a to je do 30. minute. U ovom je slučaju procesor maksimalno iskorišten za koristan rad, ali je drugi zadatak proveo čak 25 minuta u redu čekanja. Višezadaćni sustav može već u 5. minuti dati drugom zadatku procesorsko

vrijeme te ih paralelno izvoditi dajući svakom zadatku određeni vremenski odsječak tako dugo dok se jedan ne izvede do kraja. Korisno iskorištenje procesora je manje u odnosu na multiprogramske sustave zbog učestalih promjena konteksta, ali je zato odziv puno bolji u ovom slučaju.

Sustavi s raspodjelom vremena mogu se ostvariti na dva osnovna načina: suradnički (*engl.* cooperative) i prekidljivi (*engl.* preemptive). Kod suradničkog načina rada, svaka dretva odlučuje u kojem će trenutku predati procesorsko vrijeme nekoj drugoj dretvi. Prednost ovog načina rada u odnosu na prekidljivi je da nema promjena konteksta budući da je točno poznato stanje dretve nakon što je predala procesorsko vrijeme drugim dretvama. Veliki nedostatak ovog načina rada je da jedan loše dizajniran program može usporiti cijeli sustav ili, u najgorem slučaju, blokirati čitav sustav. Iako je vrlo teško dizajnirati i implementirati suradnički sustav, on ipak nalazi neke primjene (npr. svemirski program) jer pravilno dizajniran takav sustav omogućava visoku pouzdanost i deterministčku kontrolu kompleksnih događaja.

Prekidljivi način rada omogućava sustavu da s većom pouzdanošću garantira da će neki proces dobiti dio procesorskog vremena. Također je moguće dodjeljivanje prioriteta procesima u slučaju bitnih odsječaka koda koji se moraju brzo izvoditi ili kod obrade ulaznih podataka. Prekidljivi višezadaćni rad podržavala je i prva verzija UNIX-a iz 1969. godine. U bilo kojem trenutku procesi mogu biti podijeljeni u dvije skupine, na procese koji čekaju na ulaz ili izlaz i one koji u potpunosti koriste procesor. U primitivnim sustavima programi su najčešće prozivali ili koristili radno čekanje prilikom čekanja na neki ulaz ili događaj. Sustav to vrijeme nije radio ništa korisno, iako su se mogli na njemu izvoditi drugi programi koji trebaju procesor. Pojava prekida i prekidljivog rada omogućila je bolje iskorištenje takvih sustava. Prekid je davao informaciju o nekom događaju te više nije bilo potrebno koristiti radno čekanje ili prozivanje pa je procesor mogao u to vrijeme biti iskorišten za neki koristan rad. Promjena konteksta karakteristična za ovaj način rada jedini je nedostatak naspram ostalih načina raspoređivanja.

2.2.4. Sustavi za rad u stvarnom vremenu

Višezadaćni je rad veoma bitan u sustavima za rad u stvarnom vremenu. U takvim sustavima postoji mnogo nepovezanih vanjskih aktivnosti koje kontrolira jedan procesor. Kako bi se osiguralo da glavne aktivnosti dobivaju veće odsječke vremena, spajaju se u sustav hijerarhijskih prekida i prioriteta.

Ukupna točnost operacija u sustavima za rad u stvarnom vremenu ne ovisi samo o logičkoj točnosti, već i o vremenu u kojem se operacija odvijala. Svaki zadatak u takvim sustavima ima rok završetka, a prema posljedicama koje nastupaju ako se promaši rok završetka razlikuju se: tvrdi, čvrsti i meki sustavi [5]. Ako se neki zadatak ne izvrši do svoga roka krajnjeg završetka, u tvrdim sustavima to označava pad sustava, dok je u čvrstim sustavima iskoristivost rezultata jednaka nuli ako je dobiven nakon roka završetka, a rijetka kašnjenja se dozvoljavaju. U mekim se sustavima korisnost rezultata smanjuje s većim kašnjenjem.

U sustavima za rad u stvarnom vremenu najčešće se koristi raspoređivanje po prioritetu bez dodjeljivanja vremenskih odsječaka. Uz to, koristi se algoritam raspoređivanja prema krajnjim trenucima završetka (*engl.* Earliest Deadline First) koji uzima zadatak koji ima najbliže krajnje vrijeme završetka i algoritam prilagodljivog dodjeljivanja procesorskog vremena (*engl.* Adaptive Partition Scheduler) koji se koristi u sustavima koji koriste zadatke u stvarnom vremenu i ostale zadatke.

2.3. Vrste raspoređivača zadataka

U operacijskim sustavima mogu postojati tri vrste raspoređivača: dugoročni, srednjoročni i kratkoročni. Imena su im određena učestalost korištenja [6].

2.3.1. Dugoročni raspoređivač

Dugoročni raspoređivač odlučuje koji će procesi biti propušteni u red čekanja spremnih procesa, tj. kojima će se dopustiti izvođenje, a koji bi trebali izaći iz sustava. Primjerice, kada se želi izvršiti neki program, raspoređivač može dopustiti da proces postane dio trenutačno izvršavajućih procesa ili ga može odgoditi. Dugoročni raspoređivač odlučuje o stupnju multiprogramabilnosti unutar višezadaćnog sustava i drži se određenih pravila o mogućnosti sustava da obradi neki zadatak ili o odabiru zadatka ako je više različitih zadataka istovremeno ušlo u sustav. Ovim raspoređivačem se nastoji postići kompromis između stupnja multiprogramabilnosti i propusnosti što je vrlo bitno u interaktivnim sustavima. Kada je u sustavu veći broj zadataka, oni dobivaju manje procesorskog vremena, a troškovi operacijskog sustava se povećavaju (učestale promjene stranica – *engl.* page faults, page thrashing). No, u sustavu mora uvijek biti dovoljan broj zadataka koji nisu blokirani na ulazno-izlaznim operacijama tako da se procesor može opteretiti korisnim poslom. Metode odlučivanja koje se koriste u ovom raspoređivaču su raspoređivanje po redu prispijeća i raspoređivanje po prioritetu. Za raspoređivanje po redu

prispijeća vrijedi da zadatke pušta u sustav sve dok nije dosegnut prethodno definiran maksimum zadataka u sustavu nakon čega odbacuje zadatke ili ih stavlja u red čekanja za kasniju obradu. Prioriteti u ovom raspoređivaču imaju drukčije značenje nego u kratkoročnom raspoređivanju, ovdje prioritet utječe na odabir sljedećeg zadatka koji će ući u sustav, dok u kratkoročnom raspoređivaču utječe na odluku o sljedećem pripravnom procesu koji će se izvoditi.

Dugoročni raspoređivač odlučuje hoće li više ili manje zadataka biti istovremeno izvršavano te kako upravljati procesima koji intenzivno rade neke ulazno-izlazne operacije i zadacima koji intenzivno koriste procesor. U modernim operacijskim sustavima ovaj se raspoređivač koristi da bi zadaci u stvarnom vremenu dobili dovoljno procesorskog vremena da ispune svoje rokove završetka.

2.3.2. Srednjoročni raspoređivač

Srednjoročni raspoređivač prisutan je u svim sustavima koji imaju virtualnu memoriju. Zadatak mu je privremeno premještanje procesa iz glavne memorije u sekundarnu (npr. čvrsti disk) ili obrnuto [6]. Budući da je zadaća tog raspoređivača upravljanje memorijom, najčešće je implementiran kao dio podsustava koji upravlja memorijom u operacijskom sustavu. Interakcija srednjoročnog i kratkoročnog raspoređivača je vrlo bitna za performanse sustava jer je sekundarna memorija veoma spora i uzrokuje veliki pad performansi sustava ako se krivi zadaci premještaju iz glavne memorije. Ovaj raspoređivač utječe na blokirane zadatke. Najčešće kriteriji prema kojima srednjoročni raspoređivač premješta zadatke u sekundarnu memoriju su:

- proteklo vrijeme od zadnjeg izvršavanja,
- prioritet,
- količina memorije koju proces zauzima u operacijskom sustavu,
- broj promašenih stranica.

Kasnije te zadatke srednjoročni raspoređivač može natrag staviti u glavnu memoriju pa čak i ako oni ostaju blokirani, npr. kad se memorija oslobodi.

2.3.3. Kratkoročni raspoređivač

Kratkoročni raspoređivač odlučuje koji će procesi u redu čekanja biti izvršeni, tj. kojima će biti dodijeljeno procesorsko vrijeme nakon prekida sata, prekida ulazno-izlaznog uređaja, sustavskog poziva ili nekog drugog oblika signala. Kratkoročni raspoređivač

odlučuje mnogo češće od ostalih, najmanje jedanput u vremenskom odsječku koji je jako kratak. Ovaj raspoređivač može biti suradnički ili prekidljivi. Kada se spominje pojam raspoređivač na razini jezgre operacijskog sustava, misli se upravo na kratkoročni raspoređivač. Kratkoročni raspoređivač odlučuje na temelju implementiranog algoritma raspoređivanja, a algoritam određuje tip operacijskog sustava – multifunkcionalni sustav (raspodjela vremena), sustav za rad u stvarnom vremenu itd. Dizajn kratkoročnog rasprođivača je kritičan dio u dizajnu cijelog operacijskog sustava jer on direktno utječe na performanse sustava sa stajališta korisnika. Primjerice, uz slične troškove raspoređivanja moguće je postići bitno različite vrijednosti propusnosti i odziva dvaju algoritama raspoređivanja. Sa stajališta sustava bitno je samo da iskoristivost procesora bude visoka dok korisniku u interaktivnom sustavu mnogo više znači odziv kako ne bi morao dugo čekati na obradu ulaza koje je zadao nekom zadatku.

Osnovne karakteristike koje svaki kratkoročni raspoređivač nastoji zadovoljiti su brze promjene zadataka, efikasna izvedba dijeljenih podatkovnih struktura (rješenje je da se one nalaze u memoriji) i dodjeljivanje pravednih vremenskih udjela svim zadacima u sustavu [7]. Algoritam raspoređivanja definira što je pravedno.

3. Algoritmi raspoređivanja

Algoritmi raspoređivanja metode su pomoću kojih se dretvama, procesima ili tokovima podataka dodjeljuju resursi sustava kao što su procesorsko vrijeme ili propusnost. Algoritmi raspoređivanja koriste se, osim u operacijskim sustavima, i u usmjerivačima pri upravljanju prometom paketa, u čvrstim diskovima pri ulazno-izlaznim operacijama, pisačima itd. Glavna zadaća algoritama je minimiziranje izgladnjivanja resursa da bi se osigurala pravednost između strana koje koriste neke resurse. Potreba za tim algoritmima proizlazi iz zahtjeva za višezadaćnim radom, kojeg većina modernih sustava podržava, i multipleksiranjem.

U literaturi se često spominje da se raspoređuju procesi, iako to nije sasvim precizna tvrdnja. Naime, u računarskoj znanosti proces je zapravo okolina koja omogućuje izvođenje nekim jedinicama izvođenja (npr. dretvama), ali sam proces nije nešto što se može izvoditi. Tvrdnja da se raspoređuju procesi je točna ako pojam proces shvaćamo na višoj razini – kao skup procedura koje iz nekakvih ulaza daju željene izlaze. U ovom će se radu umjesto pojma proces koristiti pojam zadatak, kao što se koristi i u jezgri Linuxa za svaku jedinicu izvođenja. U smislu računarske znanosti taj pojam preciznije opisuje funkciju raspoređivača zadataka i jednoznačno je određen.

Najpoznatiji i najčešće korišteni algoritmi su višerazinski povratni red čekanja, potpuno pravedan raspoređivač i O(1) raspoređivač. U ovom će poglavlju biti opisani neki teoretski važni raspoređivači te najčešće korišteni raspoređivači u današnjim operacijskim sustavima, dok će u sljedećem poglavlju detaljnije biti opisan potpuno pravedan raspoređivač.

3.1. Implementacije raspoređivača u različitim operacijskim sustavima

Rane verzije MS-DOS i Microsoft Windows operacijskih sustava nisu bile višezadaćne. Windows 3.1x je koristio suradnički raspoređivač pa nije prekidao procese u izvođenju, a Windows 95 bio je prvi Microsoftov operacijski sustav koji je koristio prekidljivi raspoređivač, ali su se 16 bitne aplikacije pokretale bez prekidanja. Microsoftovi operacijski sustavi bazirani na Windows NT (nakon Windows 2000) koristili su višerazinski povratni red čekanja (prekidljivi algoritam). U tom raspoređivaču postoje 32 razine prioriteta, od kojih su prvih 16 normalni prioriteti, a sljedećih 16 prioriteti realnog vremena. Korisnik može koristiti šest prioriteta. Jezgra može promijeniti razinu

prioriteta ovisno o ulazno-izlaznim naredbama, korištenju procesora i procjeni interaktivnosti zadatka. Razina prioriteta podiže se interaktivnim zadacima i onima koji su vezani za ulazno-izlazne naredbe dok procesima vezanim za procesor smanjuje prioritet da bi se postigao što bolji odziv interaktivnih aplikacija. Windows Vista ima malo modificiran raspoređivač koji koristi registar koji broji cikluse procesora i prati koliko je procesorskih taktova određena dretva dobila za razliku od prijašnje implementacije raspoređivača koji je koristio prekid satnog mehanizma.

Mac OS 9 koristi suradnički raspoređivač u kojem jedan proces kontrolira više suradničkih dretvi. Za dodjeljivanje procesorskog vremena procesima jezgra koristi algoritam kružnog posluživanja. Svaki proces ima svoju kopiju upravljača dretvi koji dodjeljuje procesorsko vrijeme dretvama. Jezgra zatim koristi prekidljivi algoritam koji omogućava da svi zadaci dobe procesorsko vrijeme. Mac OS X koristi višerazinski povratni red čekanja.

Linux od verzije jezgre 2.6 koristi višerazinski povratni red čekanja sa prioritetima od 0 do 140. Prioriteti od 0 do 99 su rezervirani za zadatke u realnom vremenu, a ostali se koriste za normalne zadatke. Za zadatke u realnom vremenu sustav dodjeljuje odsječke od oko 200 ms, a za normalne zadatke 10 ms. Raspoređivač gleda prvo procese s najvišim prioritetom pa kad završi njihov odsječak stavlja ih u red isteklih procesa. Kad u redu aktivnih više nema procesa, red isteklih se proglasi aktivnim. Od verzije 2.6 do verzije 2.6.23 koristi se O(1) raspoređivač, a od verzije 2.6.23 koristi se potpuno pravedan raspoređivač koji koristi crveno-crna stabla umjesto redova čekanja.

Solaris također koristi višerazinski povratni red čekanja, ali s prioritetima od 0 do 169. Za razliku od Linuxa zadatku koji je završio dodijeli se novi prioritet i tada se stavlja natrag u red čekanja.

FreeBSD koristi višerazinski povratni red čekanja sa prioritetima 0-255. Prioriteti 0-63 su rezervirani za prekide, 64-127 za gornju polovicu jezgre, 128-159 za korisničke dretve u realnom vremenu, 160-223 za korisničke dretve koje dijele resurse s ostalima i 224-255 za korisničke dretve niskog prioriteta.

3.2. Kriteriji raspoređivanja

Kriteriji raspoređivanja donose informaciju o performansama pojedinih algoritama raspoređivanja. Na temelju kriterija raspoređivanja i poznavanja sustava mogu se izabrati karakteristike bitne za taj sustav i odlučiti koji će se algoritam raspoređivanja primijeniti.

Najčešće spominjani kriteriji raspoređivanja su:

1. Iskoristivost procesora – procesor mora biti što više zaposlen u obavljanju različitih zadataka. Nastoji se izbjeći radno čekanje ili prozivanje. Bitno je napomenuti da raspoređivanje u sustavu za rad u stvarnom vremenu ne mora nužno davati najbolje rezultate u ovom kriteriju jer u takvom sustavu je važnije držati se krajnjih rokova zadataka i njihovih prioriteta, a ne isključivo optimalne iskoristivosti procesora.
2. Propusnost – broj zadataka koji se izvršio u nekom vremenskom periodu.
3. Vrijeme zadržavanja zadatka u sustavu – vrijeme koje je prošlo od stavljanja zadatka u sustav pa sve do njegovog završetka.
4. Vrijeme čekanja – sveukupno vrijeme koje je zadatak čekao u sustavu.
5. Vrijeme odziva – vrijeme od stavljanja zadatka u red pripravnih pa do prvog odziva, tj. prvog vremenskog odsječka pridanog tom zadatku. U interaktivnim je sustavima ovo bolja mjera odziva od vremena zadržavanja zadatka u sustavu budući da zadaci često proizvode nekakav izlaz vrlo rano u svom izvođenju.
6. Zadovoljavanje vremenskih ograničenja zadataka – sposobnost operacijskog sustava (odnosno njegovog raspoređivača) da poštuje predefinirane krajnje trenutke završetka svih zadataka u sustavu.
7. Predvidljivost – sposobnost sustava da osigura da neki zadatak bude izveden u određenom vremenskom intervalu, i/ili osigura određeno vrijeme odziva neovisno o opterećenju sustava.
8. Pravednost – niti jedan zadatak u sustavu ne bi smio biti izgladnjivan čekajući na procesorsko vrijeme, a procesorsko bi vrijeme trebalo pravedno rasporediti (pravednost određuje algoritam raspoređivanja) te bi svi zadaci koji su ekvivalentni po određenom kriteriju (npr. po prioritetu ako algoritam tako razlikuje zadatke) trebali biti jednako tretirani od strane raspoređivača [8, 9].

Cilj algoritma raspoređivanja je minimizirati vrijeme odziva, vrijeme čekanja i vrijeme u sustavu, dok se propusnost i iskoristivost procesora nastoje maksimizirati. Nastoji se postići što veća predvidljivost i pravednost, ali ne pod preveliku cijenu troškova operacijskog sustava. Neki kriteriji su međusobno u konfliktu, a cilj je raspoređivača naći prikladan kompromis između svih kriterija.

Postoje još i neki specifični kriteriji koji se koriste za evaluaciju nekih sustava (npr. superračunala). Primjerice, ukupna iskoristivost procesora koristi se za procjenu iskorištenja i isplativosti skupocjenog sklopovlja, a pokazuje koliko je vremena procesor bio aktivan (neovisno o tome jesu li se izvodili korisnički zadaci ili neke procedure jezgre).

3.3. Raspoređivanje po redu prispijeća

Raspoređivanje po redu prispijeća (*engl.* First In First Out Scheduling, FIFO) jedan je od najjednostavnijih algoritama raspoređivanja. Zadaci se smještaju u jedan red po onom redoslijedu u kojem su došli u sustav. Dobre strane ovog raspoređivanja su rijetke promjene konteksta koje se događaju samo prilikom završetka zadatka ili trenutka kada zadatak prelazi u blokirano stanje. Također, nije potrebna nikakva reorganizacija reda čekanja zadataka za vrijeme izvođenja. Zbog navedenih je karakteristika trošak raspoređivanja minimalan. Propusnost je prilično slaba budući da dugi procesi dugo koriste procesor iz čega slijedi da propusnost ovisi o vremenima izvođenja procesa koji su u sustavu. Vremena čekanja, odziva i zadržavanja u sustavu mogu biti vrlo visoka zbog istih razloga. Budući da nema prioriteta, sustav s FIFO raspoređivačem ima probleme u zadovoljavanju roka završetka pojedinog zadatka. Izgladnjivanje je moguće u slučaju da u sustavu postoje zadaci koji nemaju kraja.

3.4. Prednost kraćim poslovima

Prednost kraćim poslovima (*engl.* Shortest Job First) je algoritam raspoređivanja koji uzima zadatak na čekanju koji ima najkraće vrijeme izvođenja [10]. Velika prednost ove metode je njegova jednostavnost i maksimizacija propusnosti zadataka u smislu broja zadataka koji se izvrše u nekom vremenskom periodu. Problem koji donosi ovaj algoritam je mogućnost izgladnjivanja dugih zadataka u slučaju konstatnog priljeva kratkih zadataka. Ova metoda se efikasno može koristiti kod zadataka čije se izvođenje temelji na čekanju na nekakav ulaz i izvršavanje naredbi povezanih s tim ulazom. Ako se izvođenje naredbi smatra nezavisnim procesom, prema njegovom prošlom ponašanju može se procijeniti vrijeme izvršavanja. Ovaj je algoritam suradnički, a njegova prekidljiva izvedba je kriterij najkraćeg preostalog vremena. Suradnički način povećava iskoristivost procesora, ali daje slabiji odziv. Problem oba algoritma je određivanje vremena izvođenja zadataka za koje u većini slučajeva ne postoje točni podaci. Zbog toga se koristi procjena trajanja izvođenja na temelju prijašnjih poznatih vremena izvođenja, obično preko eksponencijalne funkcije raspada [11]. Ova metoda raspoređivanja koristi se samo u specijaliziranim okruženjima jer zahtijeva točne procjene vremena izvođenja svih pripravnih zadataka u sustavu.

3.5. Kriterij najkraćeg preostalog vremena

Kriterij najkraćeg preostalog vremena (*engl.* Shortest Remaining Time) je algoritam raspoređivanja koji se bazira na algoritmu najkraći poslovi najprije. Značajka ovog algoritma je da je prekidljiv. Pokreće se zadatak koji ima još najmanje vremena do kraja izvršenja. Budući da je zadatak koji se trenutno izvršava upravo onaj koji ima najkraće vrijeme do završetka, zadaci će se izvršavati dok se ne izvrše ili dok se novi zadatak ne pojavi u sustavu. Kada se pojavi novi zadatak, potrebno je samo provjeriti ima li kraće vrijeme do završetka od trenutno aktivnog. Kao i kod prethodnog algoritma problem izgladnjivanja dugih procesa je prisutan i u ovom algoritmu raspoređivanja.

3.6. Prekidljivo raspoređivanje s fiksnim prioritetima

U sustavima za rad u stvarnom vremenu često se koristi prekidljivo raspoređivanje s fiksnim prioritetima. Algoritam osigurava da u bilo kojem trenutku od svih pripravnih zadataka u sustavu bude aktivan zadatak s najvećim prioritetom ili jedan od zadataka ako postoji više zadataka s istim najvećim prioritetom. Svaki zadatak dobiva svoj fiksni prioritet, a raspoređivač grupira pripravne zadatke prema njihovom prioritetu. Zadatak nižeg prioriteta je prekinut ako u sustav uđe zadatak višeg prioriteta. Ovaj algoritam ima podjednaku propusnost kao i FIFO raspoređivanje. Vrijeme čekanja i odziv ovisni su o prioritetu zadatka pa tako zadaci s višim prioritetom imaju kraće vrijeme čekanja i manji odziv. Poštivanje krajnjih rokova izvršenja je izvedivo tako da se procesima sa kraćim krajnjim rokovima izvršenja daje veći prioritet. Kod ovog algoritma moguće je izgladnjivanje procesa nižeg prioriteta ako su u sustavu uvijek prisutni procesi višeg priroteta.

3.7. Kružno raspoređivanje

Raspoređivač pridaje fiksni interval svim zadacima u sustavu i ne razlikuje prioritete. Što je manji interval koji se daje zadacima to su troškovi raspoređivanja veći. Postoji jedan red čekanja i svaki se novi zadatak stavlja na kraj tog reda. Ako se zadatak nije izvršio unutar dobivenom fiksnog intervala, on se prekida i stavlja na začelje reda pripravnih zadataka isto kao i svaki novi zadatak. Iz toga slijedi da je ovaj algoritam izveden iz FIFO raspoređivanja, a razlikuju se po tome što je kružno raspoređivanje prekidljivo. Propusnost kraćih zadataka je veća nego kod FIFO raspoređivanja, dok za dulje zadatke vrijedi da je propusnost veća nego kod algoritma najkraći poslovi najprije iz

čega slijedi da je propusnost balansirana između FIFO i najkraći poslovi najprije raspoređivanja. Prosječno vrijeme odziva je vrlo brzo, a odziv ne ovisi o duljini zadataka u sustavu, već samo o broju zadataka. Bitna posljedica nepostojanja prioriteta je da se ne može dogoditi izgladnjivanje. Kružno raspoređivanje nije prikladno u slučaju da su veličine poslova vrlo različite. Posao koji se sastoji od mnogo velikih zadataka će dobiti prednost pred ostalim zadacima.

3.8. Višerazinski povratni red čekanja

Višerazinski povratni red čekanja je algoritam koji se često koristi u slučajevima kada je moguće zadatke podijeliti u više grupa. Na primjer, jedna takva podjela je na interaktivne i pozadinske zadatke. Ta dva tipa zadatka imaju različite zahtjeve za brzinom odziva.

Cilj ovog algoritma je dati prednost kratkim zadacima, zadacima koji su vezani za ulazno-izlazne naredbe te brzo ustanoviti prirodu zadatka i dodijeliti mu vrijeme sukladno tome. Algoritam koristi više FIFO redova čekanja i temelji se na sljedećim postavkama:

1. Novi se zadatak pozicionira na kraj najvišeg FIFO reda.
2. Nakon nekog vremena proces dospije na početak reda te mu se dodijeli procesorsko vrijeme.
3. Ako je proces gotov, izlazi iz sustava.
4. Ako proces svojevoljno prepusti kontrolu, izlazi iz reda čekanja, a tek kad opet postane spreman, ulazi u sustav u isti red čekanja ili čak i red više.
5. Ako proces iskoristi cijeli vremenski odsječak, stavlja ga se na kraj reda nižeg prioriteta.
6. Postupak se ponavlja sve dok proces ne završi izvođenje ili dosegne red osnovne razine, tj. početno pridijeljenog prioriteta [12].

Na osnovnoj razini zadaci cirkuliraju na temelju algoritma kružnog posluživanja sve dok nisu gotovi. Iz algoritma proizlazi da se preferiraju kratki zadaci budući da se svi zadaci inicijalno smještaju u najviši FIFO red, a tek ako iskoriste cijeli vremenski odsječak, postepeno im se smanjuje dinamički prioritet. Na taj način se postiže vrlo brz odziv za kratke zadatke kao i za zadatke vezane za ulazno-izlazne uređaje jer se prioritet ne smanjuje u slučaju da je zadatak blokiran, već se čak može i povećati. Pozadinski zadaci nakon nekog vremena dosežu osnovnu razinu i cirkuliraju po principu kružnog posluživanja te ne smetaju drugim interaktivnim zadacima. Ovaj algoritam omogućava dobar odziv te veliku propusnost.

3.9. O(1) raspoređivač

O(1) raspoređivač dodjeljuje zadacima vremenski odsječak unutar nekog konstantnog vremena neovisno o broju zadataka u sustavu. Osnovna je ideja ovog algoritma smanjenje resursa koje koristi sam raspoređivač. Naziv O(1) ne znači nužno da je riječ o najbržem algoritmu, već da postoji gornja granica koja određuje maksimalno vrijeme koje je raspoređivaču potrebno za donošenje odluke o raspoređivanju. Dvije ključne strukture podataka omogućavaju O(1) raspoređivaču da donosi odluke neovisno o broju procesa: redovi pripravnih zadataka i polja prioriteta. Redovi pripravnih zadataka su temelj na kojem je izgrađen cijeli algoritam raspoređivanja i sadrže informaciju o zadacima pridijeljenim pojedinim procesorima. Svaki procesor ima vlastiti red spremnih zadataka. Svaki taj red ima dva polja prioriteta, aktivno i isteklo polje. Svi zadaci pridijeljeni nekom procesoru počinju u aktivnom polju, a nakon što iskoriste svoj vremenski odsječak idu u isteklo polje. Pritom se računa novi vremenski odsječak kojeg će sljedeći put taj zadatak dobiti. Kada u aktivnom polju nema više zadataka, ono se zamijeni s isteklim poljem i postupak se ponavlja. Svako polje prioriteta je polje povezanih lista koje predstavljaju prioritete. Iz aktivnog polja O(1) raspoređivač uzima uvijek zadatak s najvećim prioritetom.

U implementaciji raspoređivača u jezgri Linuxa postoji 40 statičkih prioriteta (od -20 do 19) koji se često nazivaju i razine dobrote. Inicijalno je svakom zadatku pridijeljena dobrota od 0, ali se ta vrijednost može mijenjati sustavskim pozivom. Raspoređivač nikad ne mijenja statički prioritet, to je mehanizam kojim korisnik može utjecati na raspoređivač i raspoređivač će to uzeti u obzir. O(1) raspoređivač daje prednost zadacima vezanim uz ulazno-izlazne operacije, a kažnjava zadatke vezane za procesor kao i algoritam višerazinskog povratnog reda čekanja. Prilagođen prioritet je dinamički prioritet i određuje se na temelju tipa zadatka. Raspoređivač može progurati interaktivne zadatke povećanjem prioriteta do 5 jedinica i isto tako kazniti radne zadatke smanjenjem prioriteta do 5 jedinica [13]. Heuristika za određivanje prirode zadatka se temelji na računanju vremena koje neki zadatak provodi u stanju spavanja u odnosu na vrijeme koje se izvodi. Pretpostavlja se da zadatak spava kada je blokiran jer čeka na ulaz ili izlaz. Priroda zadatka može biti i drugačija, pa heuristika ne daje binaran rezultat, već je rezultat unutar određenog raspona vrijednosti koje predstavljaju stupanj interaktivnosti zadatka. Raspoređivač daje veći dinamički prioritet zadacima koji imaju veću vrijednost akumuliranog spavanja. Zadaci koji dugo spavaju pa zatim dugo koriste procesor ne smiju dobiti interaktivni bonus.

Navedena heuristika daleka je od idealne te se često događa da raspoređivač pogriješi u procjeni prirode zadatka i upravo to je najveći nedostatak ovog raspoređivača. Zbog tog nedostatka ubrzo je u jezgri Linuxa zamijenjen potpuno pravednim raspoređivačem.

4. Potpuno pravedan raspoređivač

Potpuno pravedan raspoređivač koristi se u Linuxu od verzije jezgre 2.6.23. pa nadalje. Autor algoritma je Ingo Molnar. Ideja za algoritam potaknuta je *Staircase Deadline* raspoređivačem kojeg je izradio Con Kolivas. Nakon odluke da će se u jezgri Linuxa koristiti potpuno pravedan raspoređivač mnogo se raspravljalo je li njegov autor Ingo Molnar preuzeo kompletnu ideju od Cona Kolivasa. Svi važniji ljudi u Linux zajednici su stali na stranu Inga Molnara pa sukob nije dugo trajao. Ingo Molnar priznaje da je preuzeo osnovnu ideju te je zatim doradio [14].

Prema riječima autora, potpuno pravedan raspoređivač modelira "idealan i precizan višezadaćni" procesor. Idealan i precizan višezadaćni procesor je sklopovski procesor koji može izvoditi više procesa istovremeno (tj. paralelno) i daje svakom procesu jednaku količinu procesorske snage [15]. Primjerice, u slučaju kada je u sustavu jedan proces, on dobiva 100% snage procesora, a u slučaju kada se u sustavu nalaze dva procesa svaki dobiva 50% snage i izvode se paralelno. Dakle, takav je procesor pravedan prema svim zadacima u sustavu, no u stvarnosti nepostojeći. Potpuno pravedan raspoređivač pokušava programski emulirati takav procesor.

4.1. Način rada

Realan procesor može istovremeno izvoditi samo jedan proces, a to znači da u tom trenutku taj proces dobiva 100% procesorske snage, dok svi ostali dobivaju 0%. Očito je da to nije pravedno. Kako bi ispravio tu nepravednost u sustavu, potpuno pravedan raspoređivač vodi evidenciju o pravednom udjelu procesorskog vremena koje bi bilo raspoloživo svakom procesu u sustavu. Algoritam koristi pravedan sat koji je izveden iz frekvencije procesora pomnožene određenim koeficijentom, faktorom povećanja koji ovisi o broju procesa u sustavu. Rezultirajuća je vrijednost ukupna količina vremena procesora na koju ima pravo u nekom trenutku pojedini proces. Preciznost računanja vremena je u nanosekundama.

Kada proces čeka na procesor, raspoređivač računa količinu vremena koju bi on dobio za to vrijeme na idealnom procesoru. To vrijeme čekanja je za svaki zadatak pohranjeno u varijabli `wait_runtime` koja se koristi za za rangiranje procesa za raspoređivanje i da bi se odredila duljina vremenskog odsječka koji će biti dodijeljen procesu prije nego bude prekinut [16]. Sljedeći proces koje je na redu je onaj s najvećom

potrebom za procesorom, a to je upravo onaj koji najdulje čeka, tj. ima najveću vrijednost u varijabli `wait_runtime`. Vrijeme čekanja tog procesa se u tom trenutku smanjuje za razliku vremena koju bi ionako dobio na idealnom procesoru i vremena koje će se izvoditi. Kada se proces izvodi, on koristi 100% procesora te ga samim tim "previše“ koristi, tj. nepravedan je prema ostalim zadacima u sustavu koji u tom trenutku ne dobivaju procesorsko vrijeme. To znači da će nakon nekog vremena neki drugi zadatak imati najveće vrijeme čekanja i trenutni će zadatak biti prekinut. U teoriji, da bi se izbjeglo gubljenje procesorskog vremena na zamjene konteksta i odluke o raspoređivanju, aktivnom procesu se daje malo više vremena tako da mu vrijeme čekanja može biti dovoljno manje od sljedećeg na redu. Na taj način potpuno pravedan raspoređivač nastoji biti pravedan prema svakom procesu i nastoji postići da svi procesi u sustavu imaju vrijeme čekanja jednako nuli (kako i imaju na idealnom procesoru).

4.2. Implementacija u jezgri Linuxa

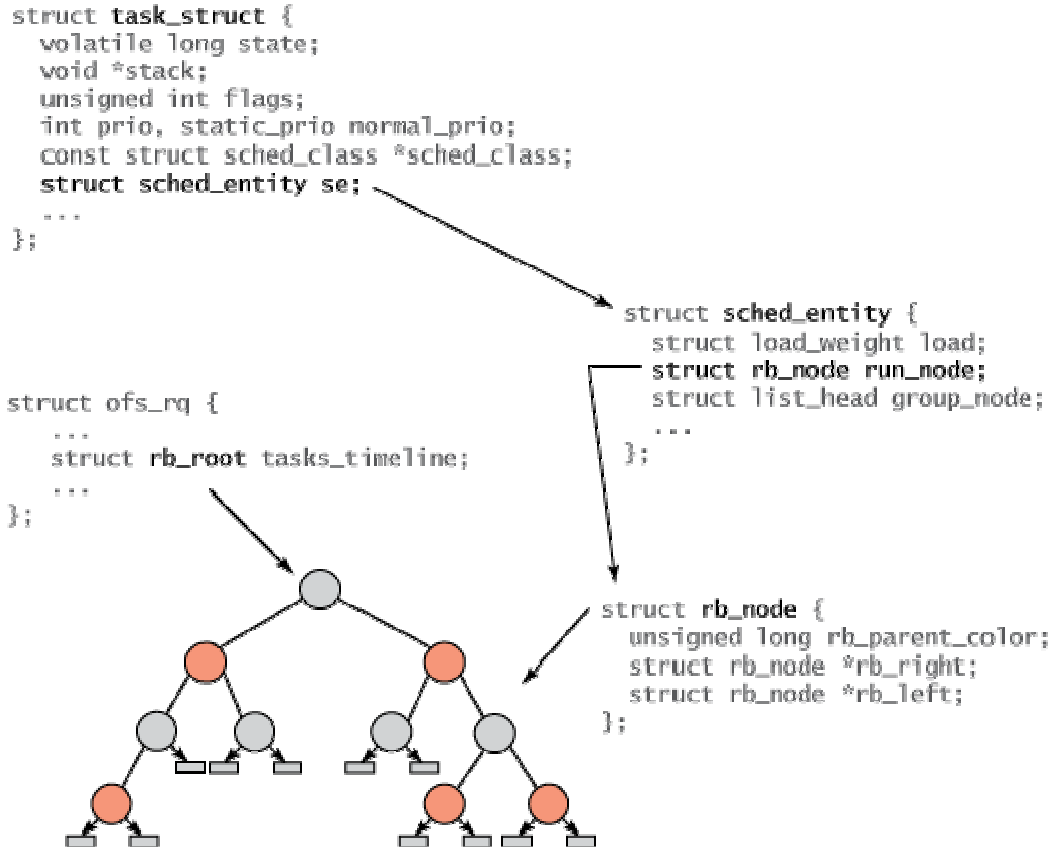
Kao struktura podataka u kojoj se nalaze zadaci iz sustava više se ne koristi lista nego crveno-crno stablo jer ima neke korisne karakteristike zbog kojih je ta struktura pogodna za implementaciju. Ona su samobalansirajuća što znači da niti jedan put u stablu neće biti više od dva puta duži od bilo kojeg drugog puta. Drugo vrlo važno svojstvo je da se sve važne operacije izvode u *O(log n)* složenosti što povećava brzinu i efikasnost.

Varijabla u koju se sprema vrijednost pravednog sata zove se `fair_clock` i zajednička je za sve procese. Povećava se u svakom vremenskom trenutku za određeni udio vremena tako da predstavlja idealno vrijeme za pojedini zadatak u sustavu. Npr. ako imamo N zadataka u sustavu, `fair_clock` se povećava *1/N* puta sporije nego stvarno vrijeme. Crveno-crno stablo sortira se prema ključu `fair_clock - wait_runtime`. Raspoređivač iz tog stabla izabire najljeviji čvor koji predstavlja zadatak s najvećom vrijednosti čekanja, odnosno zadatak s najmanjom vrijednošću u varijabli `wait_runtime`. Kako vrijeme odmiče, novi zadaci se stavljaju u stablo sve više prema desno (jer im je vrijeme čekanja inicijalno jednako nuli). Na taj način je omogućeno da svaki zadatak jednom postane najljeviji. Novom zadatku u sustavu vrijednost ključa treba biti jednaka `fair_clock - wait_runtime` vrijednosti. No, kako novi zadatak još uopće nije čekao na procesor, njegovo je vrijeme čekanja jednako 0 pa je ključ s kojim on ulazi u stablo zapravo jednak vrijednosti u `fair_clock` varijabli u trenutku njegovog ulaska u sustav.

Potpuno pravedan raspoređivač ne koristi prioritete direktno, nego koristi težinske

zadatke. Svakom se zadataku pridaje težina bazirana na statičkom prioritetu. Vremenski odsječak koji se dodjeljuje zadatku ovisi o njegovoj težini i dobiva se kao umnožak osnovnog vremenskog odsječka i koeficijenta koji ovisi o težini zadatka. Logičko vrijeme izvođenja nekog zadatka jednako je kvocijentu stvarnog dobivenog vremena i tog koeficijenta. U slučaju da je zadatak iskoristio cijeli vremenski odsječak njegovo će se logičko vrijeme izvođenja povećati upravo za osnovni vremenski odsječak. U općenitijem slučaju, kad će zadatak biti prekinut ili će se blokirati na nekom resursu, logičko će se vrijeme morati izračunati iz stvarno dobivenog fizičkog vremena i koeficijenta. Iz navedenog slijedi da će za veći koeficijent logičko vrijeme zadatka biti manje nego za manji koeficijent. Zbog toga se zadatku s manjom težinom daje manji stvarni vremenski odsječak nego zadatku s većom težinom za isto logičko vrijeme. Na navedenim postavkama temelji se pravednost ovog algoritma raspoređivanja.

U jezgri Linuxa svi su zadaci prezentirani strukturom `task_struct`. Ta struktura u potpunosti opisuje neki zadatak i sadrži trenutačno stanje zadatka, stog, zastavice, informaciju o prioritetu (statičkom i dinamičkom) itd. Budući da nije sve zadatke moguće pokrenuti, kreirana je nova struktura `sched_entity` koja prati informacije o raspoređivanju. Na slici 1. mogu se vidjeti međusobne ovisnosti pojedinih struktura.



Slika 1. Međusobne zavisnosti pojedinih struktura koje se koriste u raspoređivanju [17]

Korijenu stabla možemo pristupiti preko `cfs_rq` strukture. Svakom čvoru može se pristupiti preko `rb_node` strukture koja sadrži samo pokazivače na djecu i boju roditelja. U `sched_entity` strukturi je sadržan čvor `rb_node`, težina zadatka, varijabla `vruntime` itd. Varijabla `vruntime` jednaka je razlici `fair_clock – wait_runtime` i označava ukupno vrijeme izvođenja zadatka. Ta varijabla služi kao ključ u crveno-crnom stablu. Struktura `task_struct` je najviša u hijerarhiji i sadrži, uz ostale podatke, i `sched_entity` strukturu.

Raspoređivanje se vrši pomoću generičke funkcije `schedule()` koja prekida zadatak koji se trenutno izvodi. Zadatak koji se prekida vraća se u crveno-crno stablo funkcijom `put_prev_task()`. Nakon toga uzima se sljedeći zadatak koji će se izvoditi pozivom generičke funkcije `pick_next_task()` koja poziva raspoređivačevu funkciju `pick_next_task_fair()`. Ta funkcija uzima najljevije dijete iz crveno-crnog stabla i vraća pripadni `sched_entity`. Pozivom funkcije `task_of()` možemo konačno dobiti pokazivač na strukturu `task_struct` željenog zadatka.

U verziji jezgre 2.6.24 (odmah nakon verzije jezgre u kojoj se prvi puta koristi potpuno pravedan raspoređivač) je dodan novi koncept – grupno raspoređivanje. To je još jedan način na koji se uvodi pravednost u raspoređivanje. Na primjer, neki zadaci mogu tvoriti mnoge druge zadatke te na taj način jedan zadatak može kreiranjem novih zauzeti veći dio procesora. U takvim slučajevima je potrebno grupno raspoređivanje u kojem se svi zadaci ne tretiraju uniformno. Uzmimo npr. slučaj poslužitelja koji stvara mnogo zadataka da bi dolazne konekcije bile brže obrađene. Ostali zadaci u sustavu bit će potisnuti od strane zadatka poslužitelja koji je stvorio još mnogo novih zadataka. Zbog toga možemo pravednost pridijeliti grupi zadataka poslužitelja. Oni će se u tom slučaju tretirati kao jedan zadatak u sustavu, a izvođenje podzadataka ovisit će o njihovoj unutrašnjih hijerarhiji. Pravednost se može pridijeliti korisnicima, procesima i slično.

Zajedno s potpuno pravednim raspoređivačem predstavljena je i mogućnost raspoređivanja klasa. Svakom zadatku se pridijeli zadana klasa kojoj on pripada i na temelju te klase određuje se na koji će se način taj zadatak raspoređivati. Klasa raspoređivanja definira osnovni skup funkcija preko `sched_class` strukture. Svaki raspoređivač mora imati definirane funkcije za dodavanje zadatka, određivanje sljedećeg itd.

5. Ostvarenje potpuno pravednog raspoređivača u jednostavnom operacijskom sustavu

Osnovna zamisao ostvarenja raspoređivanja u ovom jednostavnom operacijskom sustavu je ostvarenje modularne jezgre koja može učitavati različite vanjske uređaje preko nekog pretpostavljenog zajedničkog sučelja koje moraju implementirati ti uređaji. Isti je princip korišten i kod implementacije raspoređivača. U sustavu može postojati proizvoljan broj različitih raspoređivača, a može se koristiti bilo koja kombinacija raspoređivača zadana preko `Makefilea`. Dakle, prije izgrađivanja slike sustava moguće je odrediti koji raspoređivači će se koristiti i moguće je podesiti neke njihove parametre, poput prioriteta.

U trenutnoj implementaciji postoje tri vrste raspoređivanja:

1. FIFO raspoređivanje po prioritetu
2. kružno raspoređivanje po prioritetu
3. potpuno pravedno raspoređivanje

Prva dva raspoređivača trebala bi se koristiti za raspoređivanje u realnom vremenu. Treći raspoređivač zadataka je potpuno pravedan raspoređivač i njegova namjena je raspoređivanje zadataka koji nisu u realnom vremenu.

U ovom će poglavlju na početku biti opisana ideja i implementacija raspoređivanja, a kasnije implementacija konkretnih raspoređivača.

5.1. Modularno ostvarenje i izgradnja raspoređivača

Raspoređivači zadani u `Makefileu` učitavaju se u jezgru. Funkcija koja se koristi za inicijalizaciju raspoređivača je `k_schedulers_init()` i definirana je u `sched.c` datoteci u jezgri. Ta funkcija kreira listu u koju će se smjestiti raspoređivači i zatim dodaje zadane raspoređivače u tu listu. Pretpostavka je da su u Makefile datoteci raspoređivači poredani po padajućem maksimalnom prioritetu, dakle očekuje se da će prvi raspoređivač u listi biti onaj koji ima najveći maksimalni prioritet, dok će zadnji raspoređivač u listi imati najmanji maksimalni prioritet. Nakon dodavanja u listu poziva se inicijalizacijska funkcija raspoređivača ako ona postoji. U toj funkciji se primjerice može rezervirati memorija za redove pripravnih dretvi, postaviti neke parametre i slično.

Svaki raspoređivač implementira sučelje raspoređivača definirano u datoteci `scheduler/sched.h` u jezgri. Struktura koja predstavlja svojstva i funkcije svakog od raspoređivača u sustavu nazvana je `sched_t` i definirana je na sljedeći način:

```
typedef struct _scheduler_t_ {
      int type;  /* SCHED_RR, SCHED_FIFO, SCHED_CFS */
      /* priority information */
      int prio_min;
      int prio_max;
      int prio_levels;
      int prio_default;

      /* scheduler interface */
      int (*init) (); /* initialization function */
      void (*move_to_ready) ( kthread_t *kthr );
      int (*remove_thread) ( kthread_t *kthr );
      int (*get_top_ready) ();
      void *(*schedule) ();
} scheduler_t;
```

Svojstvo `type` označava tip raspoređivača koji je definiran u `lib/sched.h`, a te definicije vidljive su i u korisničkom načinu rada. Ovo svojstvo jednoznačno određuje kojem raspoređivaču pripada struktura te se koristi u pretraživanju raspoređivača. Na primjer, kada se neka dretva treba ubaciti u red pripravnih dretvi, potrebno je prvo saznati s kojim raspoređivačem se ona raspoređuje da bismo mogli koristiti odgovarajuću funkciju stavljanja dretve u red pripravnih.

Sljedeća svojstva daju informaciju o prioritetima koji mogu biti dodijeljeni dretvama tog raspoređivača. Minimalni i maksimalni prioritet zadani su u `Makefileu` i njihove se vrijednosti ne provjeravaju jer se pretpostavlja da ih je korisnik točno unio.

Sučelje raspoređivača definirano je funkcijama `init()`, `move_to_ready()`, `remove_thread()`, `get_top_ready()` i `schedule()`. Funkcija `init()` koristi se na početku i služi za inicijalizaciju raspoređivača. Raspoređivač je ne mora nužno implementirati. Primjerice, inicijalizacijsku funkciju nije potrebno koristiti u slučaju kada raspoređivač koristi statičku alokaciju memorije i početno stanje mu je postavljeno već pri izgradnji slike sustava. Sve ostale funkcije raspoređivač mora implementirati. Za dodavanje dretve u strukturu pripravnih dretvi raspoređivača koristi se funkcija `move_to_ready()`, a za brisanje dretvi iz reda pripravnih koristi se funkcija `remove_thread()`. Funkcija `get_top_ready()` vraća prioritet dretve najvišeg prioriteta u redu pripravnih dretvi raspoređivača. Funkcijom `schedule()` raspoređivač vraća opisnik sljedeće aktive dretve iz reda pripravnih dretvi. Nakon toga jezgra operacijskog sustava obavi sve potrebne pripreme za pokretanje dretve te je pokrene.

Kada se neka dretva prekine od strane operacijskog sustava ili postane blokirana, raspoređivač zadataka traži sljedeću aktivnu dretvu. Ta je funkcionalnost implementirana u jezgrinoj funkciji `k_schedule_threads()` koja se nalazi u datoteci `thread.c` u jezgri operacijskog sustava. Budući da može postojati proizvoljan broj raspoređivača, potrebno je naći raspoređivač koji u redu pripravnih dretvi ima dretvu kandidata za sljedeću aktivnu dretvu. Za raspoređivače dretvi u realnom vremenu to je dretva najvećeg prioriteta ili jedna od dretvi ako postoji više dretvi s istim prioritetom. Kod potpuno pravednog raspoređivača umjesto da se uzima dretva najvećeg prioriteta i pridaje joj se neki vremenski odsječak, algoritam pridaje vremenske odsječke prema prioritetu, a dretve se odabiru prema tome koliko su dugo čekale u sustavu. Funkcija `k_get_ready_sched()` iz `sched.c` vraća raspoređivač koji ima pripravnu dretvu najvišeg prioriteta u cijelom sustavu. Funkcija počinje provjeravati listu raspoređivača od onog raspoređivača koji ima najviši maksimalni prioritet, a poredak raspoređivača prema maksimalnom prioritetu određen je u `Makefileu`. Ako niti jedan raspoređivač nema pripravnu dretvu, pokreće se neaktivna (*engl.* idle) dretva koja zaustavlja procesor. Funkcija `k_get_ready_sched()` ne vraća odmah sljedeću aktivnu dretvu, već pokazivač na raspoređivač zadataka koji ima sljedeću aktivnu dretvu. Na taj način možemo pozvati raspoređivačevu funkciju `schedule()` koja će vratiti opisnik dretve i izbrisati je iz reda svojih pripravnih dretvi.

U slučaju da se neka dretva želi izbrisati iz pripravnih potrebno prvo je potrebno saznati kojim se raspoređivačem ona raspoređuje, a to omogućuje funkcija `k_get_scheduler()` koja kao parametar prima tip raspoređivača.

Dosad opisana funkcionalnost mogla bi se proširiti dodavanjem funkcije koja bi mogla obrisati već postojeći raspoređivač. Primjerice, veći broj raspoređivača u sustavu zauzima više resursa, a ako postoji više raspoređivača istog tipa (u stvarnom vremenu ili konvencionalnih) moguće je da se u nekom trenutku koristi samo jedan raspoređivač. Budući da niti jedan raspoređivač nije idealan u općem slučaju, postojanje više različitih raspoređivača pokriva više specijalnih slučajeva, a ako se u određenom trenutku koristi samo jedan raspoređivač, druge se može izbaciti iz sustava te malo ubrzati sustav i osloboditi memoriju. Da bi takva implementacija imala smisla, redovi pripravnih dretvi ili neka druga struktura mora biti dinamički kreirana u inicijalizacijskoj funkciji, tako da se cijela struktura kasnije može osloboditi. U tom slučaju, brisanjem raspoređivača oslobađa se memorija koja je potrebna za strukturu pripravnih dretvi. Sučelje raspoređivača trebalo bi se proširiti funkcijom `free()`, a u `sched.c` bi bilo potrebno dodati `k_sched_remove()` funkciju.

5.2. Ostvarenje FIFO i kružnog raspoređivača

FIFO i kružni raspoređivač u ovom sustavu namijenjeni su raspoređivanju dretvi u realnom vremenu. Za oba raspoređivača vrijedi da je prioritet prvi kriterij, a zatim FIFO odnosno kružno raspoređivanje. Za svaki prioritet postoji lista pripravnih dretvi koje pripadaju tom prioritetu, a liste su organizirane kao polje veličine broja prioriteta. Budući da oba raspoređivača koriste istu strukturu bitno je da korisnik u `Makefileu` pravilno rasporedi prioritete. U idealnom slučaju oba raspoređivača imaju isti raspon prioriteta što je i realan zahtjev budući da oba raspoređuju dretve u stvarnom vremenu. Druga krajnost je slučaj kada je minimalni prioritet bitnijeg (po prioritetu) raspoređivača veći od maksimalnog prioriteta manje bitnog raspoređivača jer u tom slučaju postoje neiskorištene liste prioriteta. Kreiranje liste implementirano je tako da se izračuna najveći prioritet oba raspoređivača te se od te vrijednosti oduzima najmanji prioriteti oba raspoređivača i na kraju dobivena razlika uveća za jedan pa se time dobiva raspon prioriteta oba raspoređivača. Nakon toga se zauzima memorija za potrebne liste.

Zajedničko sučelje koje koriste oba raspoređivača implementirano je u `scheduler/sched_rt.[ch]` dok se specifične funkcije svakog raspoređivača nalaze u `scheduler/sched_fifo.[ch]` za FIFO raspoređivač, odnosno u `scheduler/sched_rr.[ch]` za kružno raspoređivanje. Zajedničko sučelje ima definiranu funkciju za određivanje najvišeg prioriteta (`get_top_ready()`), za određivanje sljedeće aktivne dretve (`rt_schedule()`) i brisanje dretve iz reda pripravnih dretvi (`rt_remove()`). Prve dvije funkcije kao parametar primaju tip raspoređivača i njih pozivaju specifične funkcije raspoređivača.

FIFO raspoređivač ima svoju vlastitu inicijalizacijsku funkciju (`fifo_init()`), funkciju koja vraća prioritet dretve najvećeg prioriteta (`fifo_get_top_ready()`) i funkciju za raspoređivanje (`fifo_schedule()`). Zadnje dvije funkcije pozivaju već spomenute općenite funkcije za raspoređivače u stvarnom vremenu sa parametrom `SCHED_FIFO`.

Kružni raspoređivač iste je funkcionalnosti kao i FIFO raspoređivač, ali ima dodatne metode jer je prekidljivi algoritam. U njegovoj inicijalizacijskoj funkciji kreira se alarm kojim se ostvaruje upravljanje vremenom. Funkcija za brisanje dretve malo je modificirana s obzirom da se može izbrisati i aktivna dretva pa je u tom slučaju potrebno mijenjati stanje raspoređivača za razliku od FIFO raspoređivača koji nema stanja. Struktura u kojoj se pohranjuje stanje raspoređivača definirana je na sljedeći način:

```
typedef struct _rr_data_t_ {
      void *alarm_id;
      kthread_t *last_active;
      int time_on;
} rr_data_t;
```

U varijabli `alarm_id` pohranjen je identifikator alarma koji se koristi za postavljanje vremenskih odsječaka. Bitno je pratiti i zadnju dretvu koja je izabrana za aktivnu kako bi se kod prekida sata (aktivacija alarma) moglo odrediti da li je ta dretva već završila, da li je bila blokirana ili je iskoristila cijeli odsječak pa je sada treba prekinuti. U `time_on` varijabli je pohranjena informacija o stanju alarma.

Prilikom poziva funkcije za raspoređivanje potrebno je odrediti sljedeću aktivnu dretvu, podesiti alarm na određeni vremenski odsječak te osvježiti stanje raspoređivača. U slučaju da vrijeme raspoređivača teče i pozvana je funkcija za raspoređivanje (npr. jezgrin poziv ili aktivacija bilo kojeg alarma osim onog koji određuje vremenski odsječak) potrebno je vratiti zadnju aktivnu dretvu pa se ona zbog toga pamti u stanju raspoređivača. Dodatne funkcije služe za upravljanje vremenskim odsječcima – zaustavljanje i postavljanje alarma. Osim tih dviju funkcija, potrebna je i funkcija kojom se obrađuje prekid alarma. U njoj se zadnja aktivna dretva (ako nije obrisana) stavlja natrag u red pripravnih dretvi, te se poziva jezgrina funkcija za raspoređivanje. Duljina odsječka jednaka je za sve dretve i određena je s parametrima `RR_INTERVAL_NSEC` i `RR_INTERVAL_SEC`. Pretpostavljena duljina vremenskog odsječka je 100 ms. Povećanjem odsječka se smanjuju troškovi operacijskog sustava, ali i povećava vrijeme odziva. Najčešće korištene vrijednosti su između 10 ms i 100 ms.

5.3. Ostvarenje potpuno pravednog raspoređivača

Potpuno pravedan raspoređivač namijenjen je raspoređivanju "običnih" dretvi u sustavu. Implementacija je pojednostavljena verzija specifikacije potpuno pravednog raspoređivača u jezgri Linuxa. U jezgru je dodana i implementacija crveno-crnog stabla, strukture koju ovaj raspoređivač koristi za pohranu pripravnih dretvi. Implementacija se nalazi u datotekama `scheduler/rb_tree.[ch]`. U opisniku dretve proširuje se struktura `sched_t` koja je do sad sadržavala prioritet i tip raspoređivanja s varijablom `param` koja se koristi za pohranu pokazivača na čvor u crveno-crnom stablu. U ovaj pokazivač mogu i drugi raspoređivači spremati svoje interne podatke. Dretve raspoređivane FIFO ili kružnim raspoređivačem mogu identifikator reda u kojem se nalaze spremati u varijablu `queue` u

opisniku dretve jer je njima struktura pripravnih dretvi jednaka strukturi blokiranih dretvi, a to je lista. Potpuno pravedan raspoređivač koristi crveno-crno stablo pa je dretvama koje on raspoređuje potrebno dodatno mjesto za spremanje pokazivača na čvor.

Struktura koja se koristi za praćenje rada potpuno pravednog raspoređivača definirana je u `scheduler/cfs.h` na sljedeći način:

```
typedef struct _cfs_data_t_ {
	int time_on;
	kthread_t *last_active;
	time_t last_interval;
	time_t last_time;
	time_t real_clock;
	void *alarm_id;
	int thread_num;
	float total_weight;
	float last_coef;
	time_t fair_clock;
	int go_on;
} cfs_data_t;
```

Kao i kod kružnog raspoređivača, varijabla `last_active` se koristi za pohranu zadnje aktivne dretve, a `time_on` označava da je neka dretva aktivna i da postoji alarm koji će je prekinuti. Značenje i potreba za ostalim varijablama te konkretna implementacija ovog raspoređivača bit će objašnjena u nastavku teksta.

Svaka dretva, osim svog opisnika, ima i svoju strukturu u crveno-crnom stablu. Ta struktura sadrži vrijednosti potrebne za rad crveno-crnog stabla, ali i podatke o raspoređivanju potpuno pravednog raspoređivača. Struktura je definirana na sljedeći način:

```
typedef struct _rb_node_ {
	/* red black tree data */
	struct _rb_node_ *parent;
	struct _rb_node_ *left_child;
	struct _rb_node_ *right_child;
	unsigned char color;
	/* scheduler data */
	float weight;
	time_t time;
	kthread_t *kthr;
} rb_node;
```

Varijabla `weight` predstavlja težinu čvora i izvodi se direktno iz prioriteta. Pri samoj inicijalizaciji raspoređivača računaju se težine prioriteta i spremaju u dinamički alocirano polje težina kako bi se izbjeglo nepotrebno računanje težina čvorova i ubrzao rad raspoređivača. Težinske razlike prioriteta koje se koriste za izračun težina uzimaju se iz konstante `CFS_PRIO_DIFF` u `scheduler/cfs.h`. Varijabla `time` sadrži vrijednost logičkog vremena koje je dretva dobila za aktivno vrijeme provedeno u sustavu te je ekvivalentna varijabli `vruntime` iz teoretskog dijela o potpuno pravednom raspoređivaču. Zadnja varijabla u strukturi, `kthr`, pokazivač je na opisnik dretve koji je potreban za aktiviranje i brisanje dretve iz strukture.

Pravedan sat potrebno je ažurirati kada se izvršavaju dretve ovog raspoređivača. Pravedan se sat ažurira prema odsječku koji je dretva dobila, njenom prioritetu (tj. težini) i broju dretvi koje pripadaju potpuno pravednom raspoređivaču na način koji je opisan u poglavlju 4. Prekidanjem neke dretve pod vanjskim utjecajem ili istekom vremena njezinog odsječka vrijednost pravednog sata mora se obnoviti. Varijabla `fair_clock` sadrži vrijeme pravednog sata. Vrijeme koje se dodjeljuje dretvi ne ovisi samo o njezinom prioritetu kao kod kružnog raspoređivanja, već i o prioritetima svih ostalih dretvi u sustavu i zbog toga se vodi evidencija o ukupnoj težini svih čvorova u sustavu. Ukupna je težina pohranjena u varijabli `total_weight` koja se ažurira pri dodavanju nove dretve i izlasku neke dretve iz strukture pravednog raspoređivača [18]. Koeficijent kojim se množi osnovni vremenski odsječak koji se može dati dretvi jednak je kvocijentu težine sljedeće aktivne dretve i ukupne težine svih dretvi u sustavu. Na taj način dretve većeg prioriteta dobivaju veće odsječke, a osigurano je da odsječci ovise o količini dretvi u sustavu. U slučaju kada je u sustavu velik broj dretvi manji vremenski odsječci koji se daju dretvama povećavaju troškove raspoređivača, ali isto tako i povećavaju odziv te se postiže pravednije raspoređivanje – u bilo kojem vremenskom trenutku odstupanje od idealnog raspoređivača je manje nego kad dretve dobivaju fiksne odsječke ovisne o prioritetima. Osnovni vremenski odsječak definiran je u konstanti `CFS_INTERVAL_BASE` u `scheduler/cfs.h`. Također, tamo je definiran i najmanji vremenski interval koji se može dati dretvi, `CFS_INTERVAL_MIN`, i koristi se zbog neisplativosti davanja manjeg vremenskog odsječka nekoj dretvi. Nakon što je dretva prekinuta računa se njezino logičko vrijeme koje je jednako umnošku stvarno dobivenog vremena i koeficijenta prema kojem je izračunat vremenski odsječak (varijabla `last_coef`). Taj umnožak je jednak vrijednosti `CFS_INTERVAL_BASE` jer je vremenski odsječak dobiven kao kvocijent te dvije vrijednosti

samo u slučaju kad je dretva prekinuta od alarma potpuno pravednog raspoređivača što znači da je iskoristila cijeli odsječak. Pseudokod procedure koja obrađuje prekid alarma potpuno pravednog raspoređivača:

```
prekid() {
      ako postoji aktivna dretva i go_on == 0 {
            Osvježi logičko vrijeme dretve;
            Osvježi vrijednost pravednog sata;
            Stavi last_active u stablo;
            Postavi last_active na NULL;
            Postavi time_on na 0;
            Rasporedi dretve;
      }
}
```

Na kraju procedure poziva se funkcija `k_schedule_threads()` (u pseudokodu je to `Rasporedi dretve`) koja određuje sljedeću aktivnu dretvu.

U bilo kojem drugom slučaju potrebno je izračunati logičko vrijeme na temelju ukupnog vremena koje je dretva dobila do trenutka njezinog prekidanja. Logičko vrijeme dretve pribraja se ukupnom logičkom vremenu koje je dretva do tada dobila i koje je pohranjeno u varijabli `last_time`. Vrijednost pravednog sata povećava se za kvocijent vrijednosti logičkog vremena koje je dretva dobila i ukupnog broja dretvi u sustavu. Nakon toga moguće je dretvu vratiti natrag u crveno-crno stablo sa novom vrijednošću logičkog sata. U varijabli `last_interval` pamti se vrijednost vremenskog odsječka danog dretvi, a u varijabli `real_clock` vrijednost sata sustava neposredno prije aktiviranja dretve. Pomoću ta dva podatka može se odrediti točno vrijeme koje je dretva dobila ako je prekinuta zbog drugih razloga. Pri prekidanju dretve trenutno vrijeme sata oduzima se od varijable `real_clock`, a dobiveni rezultat je stvarni vremensku odsječak koji je dretva dobila. Budući da je ta dretva trebala dobiti još neko vrijeme (osim ako nije završila ili postala blokirana), računa se koliko vremena raspoređivač duguje dretvi kako bi joj to vrijeme mogao nadoknaditi kada će je sljedeći put moći rasporediti. Ako je dug manji od `CFS_INTERVAL_MIN`, smatra se kao da ga nema te se dretva stavlja u crveno-crno stablo. U slučaju većeg preostalog duga, postavlja se `go_on` i `last_interval` na vrijednost koju raspoređivač duguje toj dretvi, a dretva ostaje u `last_active` varijabli. Tijekom sljedećeg poziva `schedule()` funkcije, vraća se ta dretva i alarm se postavlja na vrijednost `last_interval`. Pseudokod funkcije kojom se zaustavlja potpuno pravedan raspoređivač:

```
zaustavi_raspoređivač() {
      ako postoji aktivna dretva {
            Izračunaj odsječak koji je dretva dobila;
            Osvježi logičko vrijeme dretve;
            Osvježi vrijednost pravednog sata;
            dug = Izračunaj dug prema dretvi;
            ako je dug >= CFS_INTERVAL_MIN {
                  Postavi go_on na 1;
                  Postavi last_interval na dug;
            }
            inače {
                  Stavi last_active u stablo;
                  Postavi last_active na NULL;
            }
            Postavi time_on na 0;
            Zaustavi alarm;
      }
}
```

Nova dretva u sustavu dobiva kao vrijednost svog logičkog sata trenutačnu vrijednost pravednog sata. U idealnom slučaju sve će dretve imati točno tu vrijednost logičkog sata pa time nova dretva nije niti u podređenom, niti u nadređenom položaju s obzirom na ostale dretve. U realnom slučaju logička vremena dretvi će biti vrlo blizu pravednog sata pa nova dretva dolazi otprilike u sredinu crveno-crnog stabla što se smatra prihvatljivim.

6. Ispitivanje rada raspoređivača

Ispitivanje rada potpuno pravednog raspređivača trebalo bi potvrditi teoretske postavke na temelju kojih je on ostvaren u ovom sustavu. Ispitivanja raspoređivača u kompleksnijim operacijskim sustavima temelje se na provjeravanju odziva, propusnosti i zadržavanja zadataka u sustavu. Po potrebi ispituju se i drugi kriteriji s time da se kod operacijskih sustava u kojima rade korisnici najviše ispituje brzina odziva jer je to osnova za ugodan rad na računalu. Raspoređivači na poslužiteljima trebali bi zadovoljavati druge kvalitete kao što su velika propusnost i kratko zadržavanje zadataka u sustavu. U ovom jednostavnom operacijskom sustavu nije se velika pažnja posvetila niti jednoj navedenoj karakteristici, već je implementirana osnovna verzija potpuno pravednog raspoređivača. To znači da se nikakve posebne prednosti ne daju novim dretvama u sustavu kako bi se povećao njihov odziv, niti se prati interaktivnost zadatka što bi u ovom slučaju značilo čekanje na ulaz. Zbog navedenih razloga ispitivat će se samo pravednost potpuno pravednog raspoređivača tako da se prate dobivena vremena potpuno pravednih dretvi u sustavu. Implementacije FIFO i kružnog raspoređivača nije potrebno dodatno ispitivati zbog njihove jednostavnosti, iako će oni biti posredno ispitani u drugom ispitivanju gdje će istovremeno u sustavu biti prisutne dretve svih triju raspoređivača.

6.1. Ispitivanje dobivenog procesorskog vremena pri raspoređivanju potpuno pravednih dretvi

Najjednostavnije ispitivanje pravednosti raspoređivača ostvareno je tako da se u sustavu kreira nekoliko dretvi različitih prioriteta i prati koliko su vremena te dretve dobile nakon zadanog vremenskog razdoblja. U ovom će ispitivanju biti prisutne isključivo dretve potpuno pravednog raspoređivača. Kod programa koji vrši ovo ispitivanje nalazi se u datoteci `programs/cfs_test/cfs_test.c`.

Razlika između dretvi kojima se prioritet razlikuje za jedan definirana je prije izgradnje slike operacijskog sustava. Ta je vrijednost postavljena inicijalno na vrijednost od 15% i ona označava da nakon što je dretva nižeg prioriteta dobila 100 sekundi procesorskog vremena, dretva višeg prirориteta morala je dobiti oko 115 sekundi procesorskog vremena u slučaju da su one bile pokrenute u isto vrijeme. Prilikom pokretanja ispitivanja prvo se kreiraju sve potpuno pravedne dretve i pridijele im se prioriteti. Svaka dretva obavlja vrlo jednostavan zadatak inkrementiranja vrijednosti svojih

brojača. To je jedan od načina evaluacije i usporedbe vremena koje su dretve dobile. Pseudokod funkcije svake dretve:

```
dretva_radi ( int id ) {
      for ( i[id] = 1; i[id] > 0; ++i[id] )
            for ( j[id] = 1; j[id] > 0; ++j[id] )
                  __asm__ __volatile__ ("":::"memory");
}
```

Varijable `i` i `j` globalna su polja tipa `uint32` (nepredznačeni 32 bitni cijeli broj). Ta se polja koriste da bi funkcija alarma koji se definira prije pokretanja dretvi mogla ispisati vrijednosti varijabli za sve dretve. Budući da funkcija dretvi ne radi ništa korisno, već samo inkrementira dvije vrijednosti, mora se koristiti naredba `__asm__ __volatile__ ("":::"memory");` koja sprečava prevoditelja da optimizira kod i izbaci ove dvije ugniježđene petlje iz izvršne datoteke. Ugniježđene su petlje korištene jer se unutarnja petlja prilično brzo izvede kada je u sustavu maleni broj dretvi. U rezultatima neće biti prikazane vrijednosti `i` i `j` za svaku dretvu već ukupan broj inkrementiranja koje je pojedina dretva napravila da bi rezultati bili jasniji.

U prvom su ispitivanju kreirane tri dretve s prioritetima 40, 35 i 30. Dakle, za razliku od 5 jedinica prioriteta i jediničnu razliku prioriteta od 15% dobiva se da se prioriteti 30 i 35 te 35 i 40 razlikuju za približno 100% prema udjelu procesorskog vremena koje dretve tih prioriteta trebaju dobiti. Ispitivanja će se izvršiti u programu VMWare koji može pokrenuti sliku nekog sustava unutar drugog operacijskog sustava. Odstupanja od teoretskih vrijednosti mogu biti posljedica nedovoljno preciznog rada vremenskog sklopa Intel 8253 kao i izvršavanja ispitivanja unutar virtualnog okruženja. U tablici 1. prikazani su rezultati ispitivanja. Vrijednosti brojača praćene su kroz vrijeme od 40 sekundi.

Tablica 1. Vrijednosti brojača za tri dretve u sustavu

Dretva [prioritet]	**Vrijednost nakon 5 sekundi [milijuna]**	**Vrijednost nakon 10 sekundi**	**Vrijednost nakon 20 sekundi**	**Vrijednost nakon 30 sekundi**	**Vrijednost nakon 40 sekundi**
Dretva 1 [40]	1438	2888	5795	8690	11497
Dretva 2 [35]	720	1443	2893	4336	5736
Dretva 3 [30]	373	745	1490	2228	2953

Dobiveni su rezultati blizu idealnih, a omjeri vrijednosti dretve 1 i dretve 2 su bliže idealnima nego omjeri tih dviju dretvi s dretvom 3. Omjeri vrijednosti brojača dretvi 1 i 2 su približno jednaki 1.99 (povećanje od 99%), a omjeri dretvi 2 i 3 kreću se oko 1.94 (povećanje od 94%).

Malo kompleksnije ispitivanje provedeno je na 6 dretvi s razlikama prioriteta od 3 jedinice. Prva dretva ima prioritet 40, a ostalima se smanjuje za 3 jedinice. Dretva s prioritetom za 3 jedinice većim od neke druge dretve trebala bi dobiti 52% više procesorskog vremena što znači da je željeni omjer brojača susjednih dretvi po prioritetu 1.52. U tablici 2. prikazani su rezultati ovog ispitivanja.

Tablica 2. Vrijednosti brojača za 6 dretvi u sustavu

Dretva [prioritet]	**Vrijednost nakon 5 sekundi [milijuni]**	**Vrijednost nakon 10 sekundi**	**Vrijednost nakon 20 sekundi**	**Vrijednost nakon 30 sekundi**	**Vrijednost nakon 40 sekundi**
Dretva 1 [40]	928	1864	3701	5507	7327
Dretva 2 [37]	614	1227	2450	3665	4859
Dretva 3 [34]	414	831	1660	2470	3266
Dretva 4 [31]	270	543	1085	1614	2138
Dretva 5 [28]	175	353	703	1048	1387
Dretva 6 [25]	124	245	486	728	966

U ovom se slučaju omjeri dretvi 1 i 2 kreću oko 1.51, a omjeri dretvi 2 i 3 oko 1.48. Omjeri dretvi 3 i 4 te dretvi 4 i 5 su 1.53, dok se za zadnje dvije dretve postiže omjer od 1.44 te jedino ta vrijednost pokazuje veće odstupanje od teoretskih vrijednosti. Ova nepravilnost može se objasniti velikom ukupnom težinom dretvi u sustavu pa dretva najmanjeg prioriteta u tom slučaju dobiva malen vremenski odsječak u odnosu na ostale te se u tom slučaju najviše manifestira nepreciznost sustavskog sata. U prilog ovoj tvrdnji ide i činjenica da je omjer brojača dretvi 1 i 5 jednak 5.28 što ne odstupa mnogo od idealne vrijednosti 5.35, dok je za dretve 1 i 6 omjer brojača 7.58 što predstavlja značajnije odstupanje od teoretske vrijednosti omjera koja je jednaka 8.14. Vidljivo je da je dretva manjeg prioriteta u ovom slučaju u prednosti naspram ostalih dretvi u sustavu. Jedno moguće rješenje ovog problema je korištenje preciznijeg vremenskog sklopa kao što je satni mehanizam visoke preciznosti (*engl.* High Precision Event Time) čija je maksimalna

frekvencija mnogo veća nego kod sklopa Intel 8253. Drugo moguće rješenje je promjena načina izračuna logičkog vremena neke dretve. Ukoliko je dretva dobila cijeli vremenski odsječak njezino se logičko vrijeme računa na temelju željene duljine vremenskog odječka, a ne na temelju stvarnog vremenskog odsječka koje je ona dobila, a koje može biti drukčije od željenog odsječka zbog nepreciznosti satnog mehanizma. Vrijednost trenutnog sata već se uzima pri ponovnom pokretanju dretve (tj. pridjeljivanju vremenskog odsječka) za slučaj da dretva bude prekinuta. Kada bi se uzela vrijednost sata i nakon što dretva iskoristi vremenski odsječak, bilo bi moguće izračunati stvarno vrijeme koje je ona dobila. Nedostatak takve implementacije je gubitak vremena zbog komunikacije sa satnim mehanizmom, a upitna je i isplativost takvog načina rada budući da se pravednost povećava samo za maleni udio, a troškovi operacijskog sustava povećavaju.

6.2. Ispitivanje rada dretvi svih raspoređivača u sustavu

Prethodno ispitivanje može se proširiti s dretvama svih raspoređivača i provjeriti da li dretve potpuno pravednog raspoređivača dobivaju ispravne vremenske odsječke i kada u sustavu postoje dretve u stvarnom vremenu. Kod ispitivanja nalazi se u datoteci `programs/sched_test/sched_test.c`.

Ispitivanje je zamišljeno tako da se kreira nekoliko FIFO dretvi istog prioriteta koje su konstantno u sustavu i kraće vrijeme rade, a dulje vrijeme spavaju i tako sve dok ispitivanje nije gotovo. Dretve kružnog raspoređivača također su istog prioriteta, ali one samo rade neki dulji vremenski period te nakon toga izlaze iz sustava. Dretve najmanjeg prioriteta, a to su dretve potpuno pravednog raspoređivača, su podijeljene u dvije grupe. Jednu grupu čine dretve istog prioriteta, a drugu grupu dretve s padajućim prioritetom.

Očekivani rezultati za dretve potpuno pravednog raspoređivača su jednaki kao i u prethodnom ispitivanju budući da se vrijeme potpuno pravednog raspoređivača zaustavlja kada se izvršavaju dretve u stvarnom vremenu. Dretve kružnog raspoređivanja trebale bi završiti otprilike u isto vrijeme budući da rade isti posao i ulaze u sustav u isto vrijeme.

Rezultati ispitivanja dobiveni su na temelju sljedećih postavki: kreiraju se dvije FIFO dretve prioriteta 125, četiri dretve kružnog raspoređivanja prioriteta 100, dvije dretve potpuno pravednog raspoređivača prioriteta 32 te tri dretve prioriteta 29, 26 i 23. FIFO dretve prvo rade, a zatim spavaju sve dok ne završi ispitivanje. Trajanje rada je 50 ms, a spavanja 1000 ms. Dretve kružnog raspoređivača rade otprilike 3000 ms i odmah nakon

toga izlaze iz sustava, a dretve potpuno pravednog raspoređivača izvršavaju istu funkciju kao i u prethodnom ispitivanju. Vrijeme ispitivanja postavljeno je na 50 sekundi nakon čega se zaustavljaju sve dretve i ispisuju vrijednosti brojača potpuno pravednih dretvi. Nakon pokretanja nekoliko ispitivanja s navedenim parametrima može se zaključiti da su rezultati prilično konzistentni. Prva dretva kružnog raspoređivanja pojavljuje se u sustavu oko 100-200 ms nakon početka, a sve ostale se pojavljuju 100 ms nakon prethodne jer je vremenski odječak koji se pridjeljuje dretvama kružnog raspoređivača jednak 100 ms. Dakle, u ovom se slučaju sa četiri dretve zadnja kreira do 500 ms nakon pokretanja sustava. Sve dretve završavaju poslije 13. sekunde u razmaku od nekoliko desetaka milisekundi što je i očekivani rezultat. Taj razmak ovisi o FIFO dretvama koje su prekidale dretve kružnog raspoređivača te zbog tog nedeterminizma prekinule normalni tijek izvođenja dretvi kružnog raspoređivača. Nakon završetka dretvi kružnog raspoređivača započinju s radom i potpuno pravedne dretve koje u petlji povećavaju svoje brojače. Vrijednosti brojača blizu su željenog rada sustava. U tablicama 3. i 4. prikazani su dobiveni rezultati za četiri izvršena ispitivanja.

Tablica 3. Počeci i završeci dretvi kružnog raspoređivača

	Ispitivanje 1		**Ispitivanje 2**		**Ispitivanje 3**		**Ispitivanje 4**	
Broj dretve	**Start [ms]**	**Stop [ms]**	**Start [ms]**	**Stop [ms]**	**Start [ms]**	**Stop [ms]**	**Start [ms]**	**Stop [ms]**
1	167	13000	165	12901	166	12503	190	13015
2	267	13202	265	12903	266	12517	290	13318
3	367	13372	365	12796	366	12786	390	13198
4	467	13200	465	12950	466	12779	490	13215

Tablica 4. Vrijednosti brojača potpuno pravednih dretvi na kraju izvođenja

Broj dretve [prioritet]	**Ispitivanje 1** **Vrijednost brojača [milijuni]**	**Ispitivanje 2** **Vrijednost brojača [milijuni]**	**Ispitivanje 3** **Vrijednost brojača [milijuni]**	**Ispitivanje 4** **Vrijednost brojača [milijuni]**
Dretva 1 [32]	5039	5650	5462	5105
Dretva 2 [32]	5024	5608	5458	5074
Dretva 3 [29]	3342	3738	3649	3365
Dretva 4 [26]	2191	2447	2377	2203
Dretva 5 [23]	1441	1603	1586	1449

Valja napomenuti da velike razlike vremenskih vrijednosti između dva ispitivanja proizlaze iz izvođenja u virtualnom okruženju tako da ovi rezultati u apsolutnom smislu ne odgovaraju realnom stanju, ali relativno gledano unutar jednog ispitivanja daju točnu sliku događaja u sustavu. Iz tog bi razloga omjeri brojača u realnom okruženju ostali isti kao i u ovom slučaju, ali bi vremena dobivena u tablici 3 i vrijednosti brojača u tablici 4 između ispitivanja bili konzistentniji.

7. Zaključak

Raspoređivač zadataka bitan je dio svakog višezadaćnog sustava što znači da je dio većine današnjih operacijskih sustava. Njegov je cilj omogućiti kako korisnicima tako i jezgri operacijskog sustava kreiranje dretvi te brinuti o njihovom raspoređivanju na slobodne procesore. Sučelje raspoređivača zadataka mora biti neovisno o samom sklopovlju (npr. broj procesora) te raspoređivač mora sam brinuti o broju dretvi u sustavu i broju raspoloživih procesora. Različiti operacijski sustavi imaju različite primjene iz čega proizlazi da su njihovi zahtjevi za raspoređivanjem dretvi bitno različiti. Tako primjerice sustavi za rad u stvarnom vremenu najčešće koriste prioritet kao jedini kriterij raspoređivanja. Dakle, u bilo kojem trenutku na takvom se sustavu mora izvoditi jedna od dretvi najvišeg prioriteta. S druge strane, prioritet dretve u potpuno pravednom raspoređivaču označava koliko neka dretva mora dobiti vremena u odnosu na ostale i ne postoji zahtjev da se u bilo kojem trenutku mora izvoditi dretva najvišeg prioriteta. Iz navedenih je tvrdnji očito da kriteriji koje moraju zadovoljiti raspoređivači ovise o primjeni za koju su namijenjeni.

Pri izgradnji podsustava raspoređivanja u ovom jednostavnom operacijskom sustavu pažnja je bila usmjerena na modularno ostvarenje raspoređivanja koje omogućuje ugrađivanje različitih kombinacija raspoređivača u sustav na temelju parametara zadanih prije izgradnje slike sustava. Na taj je način moguće implementirati više algoritama raspoređivanja koji se koriste za različite potrebe i na temelju stvarnih potreba izgraditi sliku sustava sa željenim raspoređivačem. Time je omogućeno da ovaj operacijski sustav može zadovoljavati mnoge primjene čak i istovremeno jer je moguće učitati proizvoljan broj raspoređivača. Na korisniku je da odluči koje raspoređivače u sustavu želi te izgradi sustav sa željenim parametrima, a to su tipovi raspoređivača u sustavu i rasponi njihovih prioriteta. Kako je osnovna svrha ovog sustava rad u ugrađenim okruženjima, ne postoje strogi zahtjevi za brz odziv kao što je to u interaktivnim sustavima. Iz tog razloga nije implementirano nikakvo dinamičko podizanje prioriteta za nove zadatke u sustavu, niti se koriste heuristike koje procjenjuju stupanj interaktivnosti zadatka. Naglasak u ostvarenju ovog raspoređivača bio je na zadovoljavanju kriterija pravednosti i iskoristivosti procesora.

Potpuno pravedan raspoređivač prvi je od raspoređivača u većim operacijskim sustavima koji nije koristio liste za svoj rad već crveno-crna stabla što mu je povećalo efikasnost te smanjilo troškove operacijskog sustava. Ostvarenje pravednosti preko

logičkih vremena svake dretve omogućilo je bolje raspoređivanje u slučajevima sa mnogo dretvi u sustavu kada je većina ostalih raspoređivača nepravedna.

Ispitivanja potpuno pravednog raspoređivača pokazala su da implementacija približno odgovara idealnom slučaju. Za manji broj dretvi vrijeme koje dretve dobivaju je vrlo blizu idealnom, dok su za veći broj dretvi vremenski odsječci koji se pridjeljuju dretvama maleni pa dolazi do izražaja nepreciznost sata te su prisutna veća odstupanja od idealnom slučaja. Kao moguće rješenje predložen je drugačiji način izračuna logičkog vremena koji povećava troškove operacijskog sustava pa stoga nije prikladan. Druga je mogućnost implementacija sučelja za komunikaciju s preciznijim satnim mehanizmom koji se nalazi u većini modernih sustava. No, iako odstupanja postoje, ona su konzistentna kroz dulji vremenski period te su pokazatelj ispravnog rada raspoređivača.

Potpuno pravedan raspoređivač u ovakvoj osnovnoj implementaciji pokazao se prikladan za jednostavan operacijski sustav. Resursi koje raspoređivač koristi nisu preveliki i vremenski troškovi koje unosi u raspoređivanje su prihvatljivi. Da bi se troškovi što više smanjili ne daje se nikakva prednost novim dretvama u sustavu niti je implementiran bilo kakav mehanizam za povećanje odziva nekih dretvi. Jedini kompromis koji je napravljen za bolji odziv dretvi u sustavu je izračun vremenskog odsječka na temelju ukupne težine svih dretvi u sustavu i težine dretve koja se treba izvoditi.

Literatura

1. Process, 20.3.2011., *Process (computing)*, http://en.wikipedia.org/wiki/Process_(computing), 15.4.2011.
2. Threads, 14.4.2011., *Threads (computer science)*, http://en.wikipedia.org/wiki/Thread_(computer_science), 15.4.2011.
3. Context Switching, 6.9.2010., *Context Switch*, http://wiki.osdev.org/Context_Switching, 15.4.2011.
4. Computer multitasking, 18.4.2011., *Computer multitasking*, http://wiki.osdev.org/Context_Switching, 22.4.2011.
5. Real-time computing, 30.5.2011., *Real-time computing*, http://en.wikipedia.org/wiki/Real-time_computing, 4.6.2011.
6. Scheduling, 20.5.2011. *Scheduling (computing)*, http://en.wikipedia.org/wiki/Scheduling_(computing), 25.5.2011.
7. Franco Callari, Types of scheduling, *Types of scheduling,* http://www.cim.mcgill.ca/~franco/OpSys-304-427/lecture-notes/node37.html, 11.5.2011.
8. Jennifer Aber Black, Process Scheduling, *Process Scheduling*, http://homepages.uel.ac.uk/u0214248/jenny2.htm, 25.5.2011.
9. Cem Özdoğan, Scheduling Criteria, 14.2.2011., *Scheduling Criteria*, http://siber.cankaya.edu.tr/OperatingSystems/ceng328/node120.html, 23.5.2011.
10. Shortest Job Next, 8.5.2011., *Shortest Job Next*, http://en.wikipedia.org/wiki/Shortest_job_next, 13.5.2011.
11. Yair Amir, Operating Systems, *Process Control and Scheduling*, http://www.cs.jhu.edu/~yairamir/cs418/os2/sld028.htm, 30.4.2011.
12. Multilevel feedback queue, *Multilevel feedback queue*, 9.9.2010., http://en.wikipedia.org/wiki/Multilevel_feedback_queue, 7.5.2011.
13. Josh Aas, Linux CPU Scheduler, 17.2.2005., *Understanding the Linux 2.6.8.1 CPU Scheduler*, http://joshaas.net/linux/linux_cpu_scheduler.pdf, 2.5.2011.
14. Linux: The Completely Fair Scheduler, 18.4.2007., *Linux: The Completely Fair Scheduler*, http://kerneltrap.org/node/8059, 12.4.2011.
15. Chandandeep Singh Pabla, Completely Fair Scheduler, 1.8.2009., *Completely Fair Scheduler*, http://www.linuxjournal.com/article/10267, 15.5.2011.

16. Avinesh Kumar, Multiprocessing with the Completely Fair Scheduler, 8.1.2008., *Multiprocessing with the Completely Fair Scheduler*, http://www.ibm.com/developerworks/linux/library/l-cfs/index.html, 15.5.2011.
17. M. Tim Jones, Inside the Linux 2.6 Completely Fair Scheduler, 15.12.2009., *Inside the Linux 2.6 Completely Fair Scheduler,* http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/, 15.5.2011.
18. Max Hailperin, Changes in the Linux Scheduler as of 2.6.23, *Changes in the Linux Scheduler as of 2.6.23*, https://gustavus.edu/+max/os-book/updates/CFS.html, 27.4.2011.

Sažetak

Naslov rada: Ostvarenje raspoređivanja dretvi

U ovom je radu opisan način ostvarenja raspoređivanja u jednostavnom operacijskom sustavu. U teoretskom dijelu opisani su često korišteni algoritmi raspoređivanja. U praktičnom je dijelu prikazan način na koji je ostvareno raspoređivanje u sustavu. Nakon toga su objašnjeni konkretni raspoređivači zadataka koji su ostvareni u tom podsustavu, a to su raspoređivanje po redu prispijeća, kružno raspoređivanje i potpuno pravedan raspoređivač. Naglasak je stavljen na potpuno pravedan raspoređivač pa je on podrobnije opisan u teoretskom i praktičnom dijelu. Provedena su i ispitivanja kojima se pokazalo da se ostvarenje potpuno pravednog raspoređivača poklapa s očekivanim ponašanjem uz mala odstupanja.

Ključne riječi: raspoređivač zadataka, proces, dretva, promjena konteksta, vremenski odsječak, potpuno pravedan raspoređivač, raspoređivanje po redu prispijeća, kružno raspoređivanje,

Summary

Title: Thread Scheduling Implementation

In this thesis the thread scheduling implementation in a simple operating system is described. The theoretical part describes frequently used scheduling algorithms while the experimental part presents the scheduling subsystem implementation in an operating system. Furthermore, implementation of all three schedulers in system is explained – First In First Out Scheduling, Round Robin Scheduling and Completely Fair Scheduler. This thesis is focused on Completely Fair Scheduler, therefore it is explained in full detail both in theoretical and experimental part. The tests were carried out which showed that Completely Fair Scheduler implementation satisfies expected behaviour with a small variation.

Key words: task scheduler, process, thread, context switch, time slice, Completely Fair Scheduler, FIFO scheduling, Round Robin Scheduling