Odlomak iz skripte: Sustavi za rad u stvarnom vremenu
(http://www.zemris.fer.hr/~leonardo/srsv/skripta/SRSV-skripta.pdf)

[...]

4.8. Raspoređivanje dretvi u stvarnim sustavima

[...]

Raspoređivanje značajno ovisi o zahtjevima sustava. Sustavi za rad u stvarnom vremenu samo su jedna skupina sustava u kojoj se koriste računala. U druge skupine možemo ubrojati ugrađene sustave, osobna računala, radne stanice, poslužiteljska računala, prijenosna i mobilna računala (telefoni, tableti, ...). Svojstva zadataka u drugim sustavima nisu jednaka te identični način raspoređivanja dretvi ne mora biti najbolji. Primjerice, multimedijalni program ima svojstva slična zadacima sustava za rad u stvarnom vremenu: svako kašnjenje može izazvati osjetnu degradaciju kvalitete slike ili zvuka. Slično je i s korisničkim sučeljem: znatnija kašnjenja u reakciji na korisničke naredbe korisniku će umanjiti kvalitetu sustava. Ipak, u oba navedena primjera loše raspoređivanje neće izazvati znatne štete (osim smanjenja kvalitete sustava prema ocjeni korisnika). Operacije koje dulje traju, kao što su matematički proračuni, kompresija podataka i prijenos datoteka i ako se još malo odgode neće gotovo ništa umanjiti kvalitetu sustava obzirom da korisnik već očekuje njihovo produljeno trajanje.

Poslove koji se nisu vremenski kritični, koji neće izazvati veće probleme ukoliko zakažu, koji se izvode u običnim sustavima označimo kao *običnim* ili *nekritičnim*. Poslove koji jesu vremenski kritični, čija greška ili kasna reakcija može imati velike posljedice (koji se izvode u SRSV-ima) označimo kao *kritičnima*. Operacijski sustavi najčešće omogućuju različite načine raspoređivanja, jedne za kritične poslove i druge za nekritične. Operacijski sustavi pripremljeni za SRSV-e, osim ostalih potrebnih svojstava imaju i vrlo dobru podršku raspoređivanju kritičnih dretvi, dok operacijski sustavi koji nisu posebice pripremljeni za SRSV-e optimiraju neke druge kriterije i u pogledu raspoređivanja poslova.

Pri ostvarenju sustava, jedan zadatak se preslikava u jednu dretvu te se nastavku koristi termin *dretva* umjesto *zadatka*, odnosno, govori se o *raspoređivanju dretvi*. U nastavku su prikazani uobičajeni načini raspoređivanja dretvi ostvareni u operacijskim sustavima.

4.8.1. POSIX i Linux sučelja

POSIX je zajednički naziv za skupinu IEEE standarda kojima se definira sučelja koja operacijski sustavi trebaju pružati programima, a radi njihove prenosivosti [POSIX]. Prenosivost programa je glavni razlog nastajanja POSIX-a i sličnih standarda. U početku su ciljani operacijski sustavi bile razne inačice UNIX-a, ali se kasnije sučelje počelo širiti tako da obuhvaća i sustave za rad u stvarnom vremenu. Službena oznaka standarda je IEEE 1003, a naziv međunarodnog standarda je ISO/IEC 9945. Standardi su nastali iz projekta koji je počeo oko 1985. Naziv POSIX predložio je Richard Stallman, a naknadno je izvedeno značenje (engl. *backronym*) *portable operating system interface* (prenosivo sučelje operacijskih sustava), pri čemu X predstavlja UNIX.

POSIX standard ostvaruju mnogi operacijski sustavi, pogotovo oni predviđeni za sustave za rad u stvarnom vremenu (najčešće radi omogućavanja prijenosa postojećih programa pripremljenih za druge sustave). U kontekstu SRSV-a POSIX definira nekoliko načina raspoređivanja (klase dretvi):

- `SCHED_FIFO` – prema prioritetu pa po redu prispijeća
- `SCHED_RR` – prema prioritetu pa kružnom podjelom vremena
- `SCHED_SPORADIC` – prema prioritetu, sporadični poslovi
- `SCHED_OTHER` – raspoređivanje nekritičnih dretvi.

Na promjene i proširenje POSIX-a utječu mnogi dionici, posebice oni koji se bave izgradnjom operacijskih sustava. Stoga će u nastavku pored navedenih načina raspoređivanja biti opisani i dodatni načini trenutno ostvareni 'samo' u Linux jezgri. Uz gornje načine, uz izuzetak načina `SCHED_SPORADIC` koji nije ostvaren, u Linuxu su dodatno ostvareni:

- `SCHED_DEADLINE` – raspoređivanje prema krajnjem trenutku dovršetka
- `SCHED_BATCH` – raspoređivanje nekritičnih dugotrajnih dretvi
- `SCHED_IDLE` – raspoređivanje najmanje bitnih dretvi.

U nastavku je najprije prikazano sučelje za postavljanje načina raspoređivanja i dodatnih parametara uz odabrane načine. Obzirom da većina načina koristi prioritete, u sučelja su ugrađene i funkcije koje upravljaju njime.

Isječak kôda 4.1. Postavljanje načina raspoređivanja i ostalih parametara

```
int pthread_setschedparam ( pthread_t thread,
                            int policy,
                            const struct sched_param *param );
```

Isječak kôda 4.2. Postavljanje prioriteta dretvi

```
int pthread_setschedprio ( pthread_t thread,
                           int prio );
```

Ekvivalentne funkcije na razini procesa su `sched_setscheduler` i `sched_setparam` (s istim ili ekvivalentnim parametrima).

Ponekad je jednostavnije prije stvaranja nove dretve definirati njene parametre za raspoređivanje.

Isječak kôda 4.3. Stvaranje nove dretve

```
int pthread_create ( pthread_t *thread,
                     const pthread_attr_t *attr,
                     void *(*start_routine)(void*),
                     void *arg );
```

Drugi parametar funkcije (`attr`) definira atribute za novu dretvu te ga je potrebno postaviti prema željenim svojstvima nove dretve, prije stvaranja nove dretve.

```
int pthread_attr_init ( pthread_attr_t *attr );
int pthread_attr_setinheritsched ( pthread_attr_t *attr,
                                   int inheritsched );
int pthread_attr_setschedpolicy ( pthread_attr_t *attr,
                                  int policy);
int pthread_attr_setschedparam ( pthread_attr_t *attr,
                                 const struct sched_param *param );
```

Preko `pthread_attr_setinheritsched` se definira da li nova dretva nasljeđuje parametre raspoređivanja od dretve koja ju stvara (kada je drugi parametar `PTHREAD_INHERIT_SCHED`) ili ih treba zasebno postaviti (kada je drugi parametar `PTHREAD_EXPLICIT_SCHED`).

Način raspoređivanja postavlja se preko `pthread_attr_setschedpolicy` dok ostali parametri sa `pthread_attr_setschedparam`.

Prioritet dretvi se može promijeniti i pod utjecajem sinkronizacijskih mehanizama. To je objašnjeno uz opis sinkronizacijskih mehanizama u **??**

Ako se u sustavu dretve dinamički stvaraju prema potrebama, onda je potrebno razmatrati

i njihove završetke. Naime, svaka dretva zauzima dio sredstava sustava. Pri završetku dretve sva zauzeta sredstva se ne oslobađaju automatizmom, već neka ostaju zauzeta. Sredstva se u potpunosti oslobađaju kad je status dovršene dretve preuzet od neke druge dretve (npr. pozivom `pthread_join`) ili ako je dretva pri stvaranju označena kao odvojena (engl. *detachable*). Označavanje dretve kao odvojive se postavlja u atributu `attr` pozivom `pthread_attr_setdetachstate` ili to radi sama stvorena dretva pozivom `pthread_detach`.

Stvaranje dretvi koje se raspoređuju po metodama `SCHED_FIFO`, `SCHED_RR`, `SCHED_SPORADIC` i `SCHED_DEADLINE` zahtijeva povlaštenog korisnika (administratora) budući da njegove dretve mogu u potpunosti istisnuti sve druge dretve, pa čak i one operacijskog sustava.

U nastavku su opisani navedeni načini raspoređivanja definirani gornjim sučeljima.

4.8.2. Raspoređivanje dretvi prema prioritetu

Većina operacijskih sustava pripremljena ili prilagođena za sustave za rad u stvarnom vremenu zapravo koristi samo jedan način raspoređivanja – raspoređivanje prema prioritetu. Zato je način dodjele prioriteta dretvama najčešće prema postupku mjere ponavljanja. Odstupanja od tog postupka se koriste kada različite dretve obavljaju poslove različita značaja. Tada arhitekt sustava može nekim dretvama pridijeliti i veći ili manji prioritet od onog koji bi dretva dobila prema postupku mjere ponavljanja.

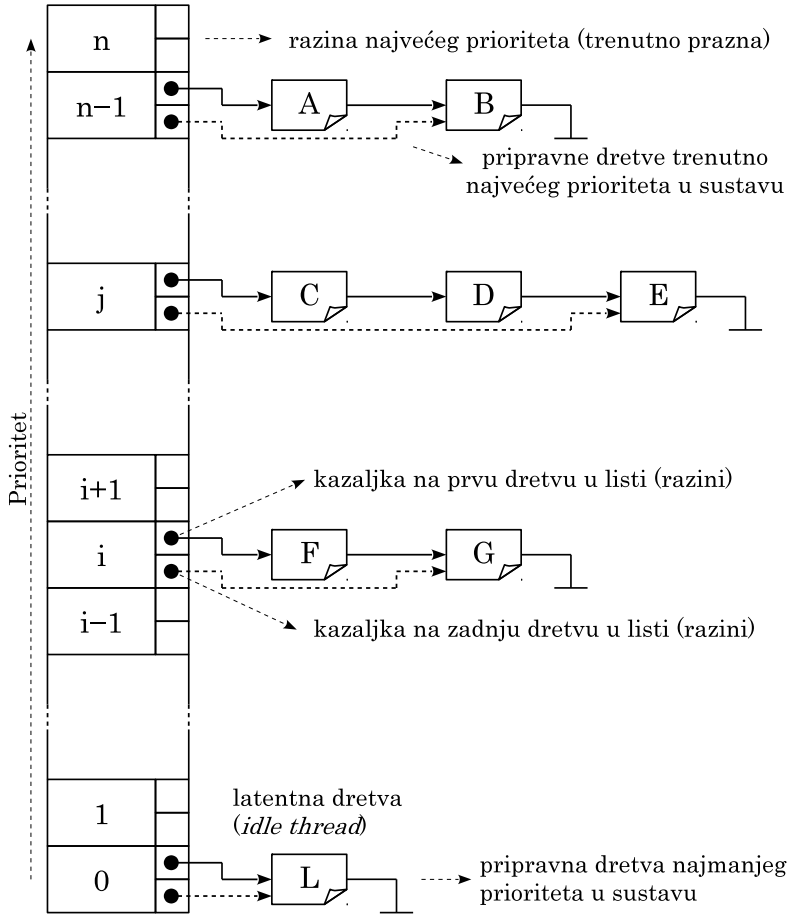
Samo raspoređivanje – određivanje trenutne dretve za izvođenje obavlja se pri radu sustava tako da se u svakom trenutku među dretvama spremnim za izvođenje (*pripravne dretve*) odabire ona najvećeg prioriteta (koja tada postaje *aktivna dretva*). Dretve koje nisu spremne za izvođenje (*blokirane dretve*) se ne razmatraju pri raspoređivanju.

Obzirom da postoji mogućnost da u nekom trenutku najveći prioritet nema samo jedna pripravna dretva već više njih mora se koristiti i dodatni kriterij pomoću kojeg će se odabrati samo jedna dretva. Najčešće korišteni dodatni kriteriji su: prema redu prispijeća (engl. *first in first out – FIFO*) te podjela procesorskog vremena – kružno posluživanje (engl. *round robin – RR*). Obzirom da je prvi kriterij uvijek prioritet, takvi postupci raspoređivanja dobivaju ime prema drugom kriteriju. Tako je i s imenima tih raspoređivača u POSIX-u:

1. `SCHED_FIFO` – raspoređivanje po redu prispijeća
2. `SCHED_RR` – raspoređivanje kružnom podjelom vremena

iako je kod njih primarni kriterij prioritet, a sekundarni (kada primarni daje više od jedne dretve) redoslijed prispijeća za prvi, odnosno, kružna podjela vremena za drugi.

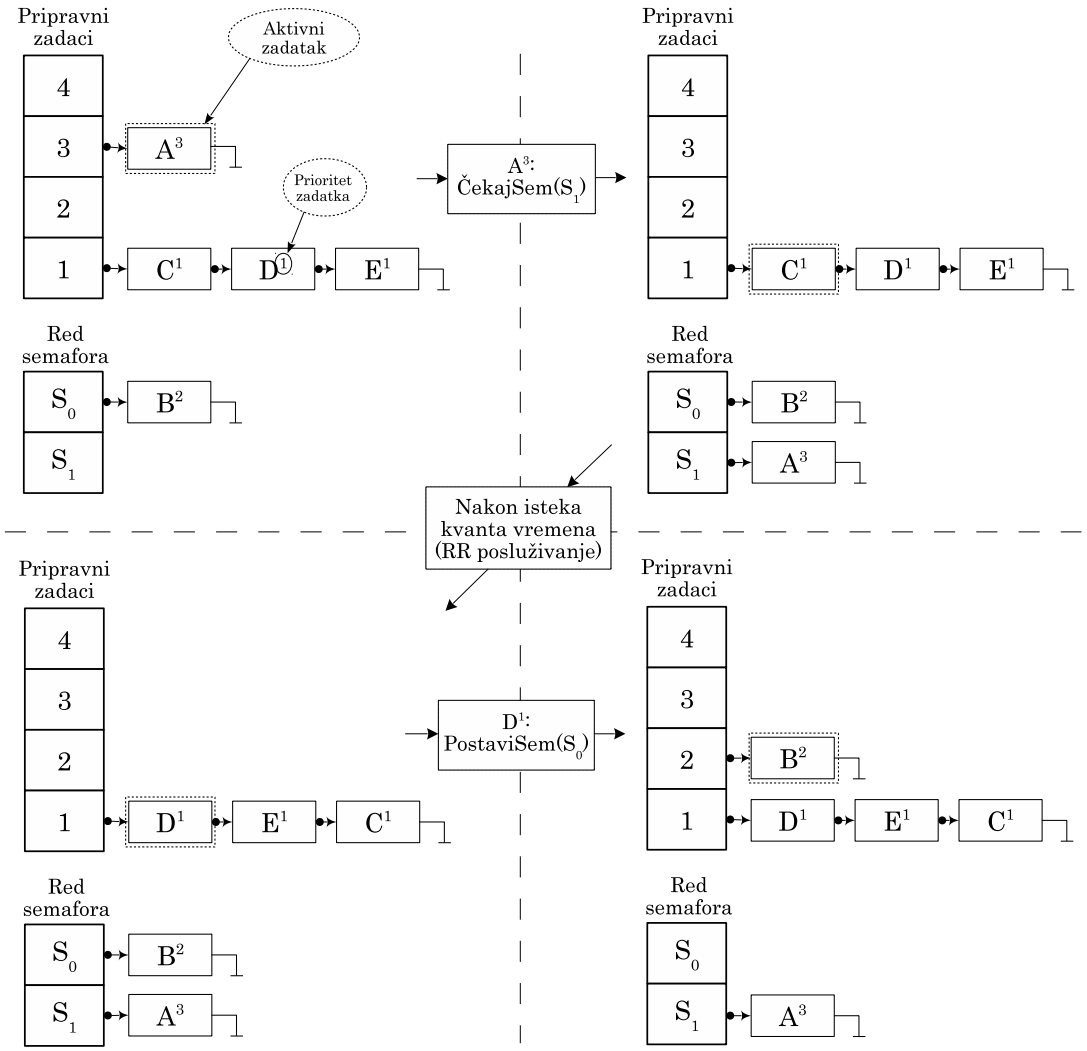
Slika 4.1. prikazuje mogući izgled strukture podataka raspoređivača koji koristi prioritet za raspoređivanje dretvi. Odabir aktivne dretve obavlja se među pripravnim dretvama najvećeg prioriteta, među dretvama A i B. Obzirom da je dretva A prva u listi ona će biti odabrana kao aktivna. Ako je drugi kriterij FIFO dretva A će se izvoditi dok ne završi ili dok se ne blokira. Ako je drugi kriterij RR onda će dretvu A u njenom izvođenju prekinuti raspoređivač nakon što je ona "potrošila" svoj dodijeljeni dio procesorskog vremena (kvant), dretva A bit će stavljena na kraj reda dretvi s istim prioritetom, tj. iza dretva B. Iduća aktivna dretva bit će dretva B.



Slika 4.1. Primjer strukture podataka jezgre za raspoređivanje prema prioritetu

U višeprocesorskim sustavima red pripravnih dretvi (koje se raspoređuju) može biti ostvaren kao na slici 4.1., ali i drukčije. Kod jednog od pristupa se za svaki procesor stvara poseban red pripravnih, prema slici 4.1. Tada postaje potrebno osigurati da se uvijek izvode dretve najveća prioriteta. Ponekad će to zahtijevati "guranje" dretvi u druge redove pripravnih dretvi ili posizanje za dretvama iz redova drugih procesora. Primjerice, kada aktivna dretva A na procesoru I omogući nastavak drugoj dretvi B nastavak svog rada, tada se B može gurnuti procesoru J ako on izvodi dretvu čiji je prioritet manji od dretvi A i B, odnosno i od svih drugih aktivnih dretvi. Slično je i kada neka dretva završi s radom ili se blokira: tada je potrebno odabrati pripravnu dretvu najveća prioriteta među svim pripravnim dretvama – ne samo iz reda procesora koji je sada postao slobodan. U ovom se slučaju radi o "povlačenju" dretve iz reda pripravnih dretvi drugih procesora. Opisani postupci koriste se pri raspoređivanju dretvi u Linux-u.

Slika 4.2. prikazuje primjere raspoređivanja korištenjem prioriteta, kružnog posluživanja te događaja povezanih sa sinkronizacijskim funkcijama.



Slika 4.2. Primjeri raspoređivanja prema prioritetu i podjeli vremena

4.8.3. Raspoređivanje prema krajnjim trenucima završetaka

Raspoređivanje prema krajnjim trenucima završetaka je rijetko podržani način raspoređivanja dretvi, čak i među operacijskim sustavima za rad u stvarnom vremenu. U sustavima u kojima jest podržano takvo raspoređivanje, mehanizam njegova korištenja je takav da se posebnim sučeljem dretva označi kao periodička, sa zadanom periodom kojom se dretva budi i pokreće, uzimajući u obzir i druge slične dretve i njihove trenutke krajnjih dovršetaka.

Periodičke dretve se mogu prikazati pseudokodom:

```
periodička_dretva
{
   ponavljaj {
      odradi_periodički_posao;
      odgodi_izvođenje ( ostatak_periode );
   }
}
```

Pseudokod takvih dretvi treba proširiti odgovarajućim pozivima da se željeno ostvari. Uobičajeno sučelje koje nude operacijski sustavi koji podržavaju takvo raspoređivanje prikazano je na primjeru:

```
periodička_dretva
{
   označi_periodičnost ( period );
   ponavljaj {
      čekaj_početak_periode ()
      odradi_periodički_posao;
   }
}
```

Operacijski sustav nudi sučelje koje je u primjeru prikazano funkcijama `označi_periodičnost` i `čekaj_početak_periode`.

Raspoređivanje prema krajnjim trenucima završetka kod Linuxa

Linux jezgra od inačice 3.14 (ožujak 2014.) donosi podršku za ovaj način raspoređivanja pod imenom *Sporadic task model deadline scheduling* te s oznakom `SCHED_DEADLINE`. Parametri svake dretve su:

- C – potrebno vrijeme izvođenja unutar periode (`sched_runtime`)
- d – trenutak krajnjeg završetka u odnosu na početak periode (`sched_deadline`)
- T – perioda ponavljanja (`sched_period`)

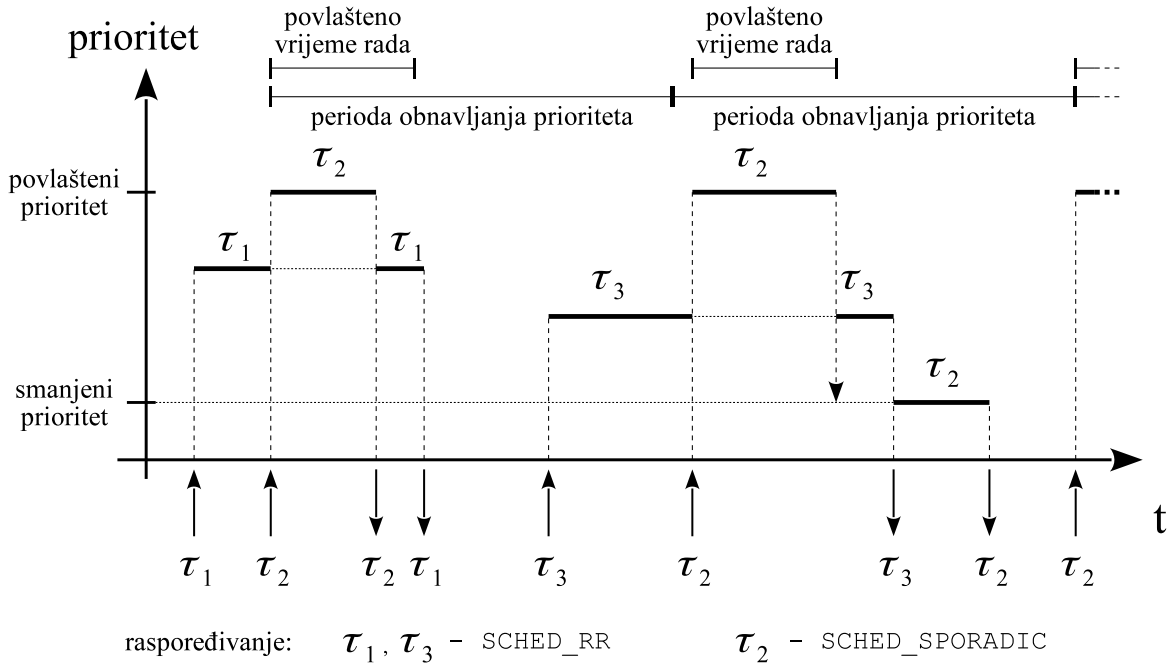
uz ograničenje: $C \leq d \leq T$.

Pri stvaranju nove dretve ovakva načina raspoređivanja provjerava se rasporedivost sustava dretvi koje se raspoređuju prema `SCHED_DEADLINE` uz dodatak opterećenja ove dretve. Ukoliko bi ova dretva narušila rasporedivost, jezgra će odbiti zahtjev za stvaranjem takve dretve ili pretvaranje postojeće u ovakvu. Obzirom da dretve koje se raspoređuju sa `SCHED_DEADLINE` istiskuju sve druge dretve, ovakvim jednostavnim provjerama osigurava se rasporedivost svih dretvi koje se raspoređuju ovim načinom, a koje poštuju nametnuta ograničenja vremena izvođenja unutar periode. Da poneka dretva ne bi ugrozila ostale koje se raspoređuju istim `SCHED_DEADLINE` postupkom, dodatno se koristi postupak rezervacije poslužiteljskog vremena (engl. *Constant Bandwidth Server – CBS*). Naime, ako pojedina dretva od početka periode (njena aktiviranja) 'potroši' zadano vrijeme izvođenja (a prije blokiranja ili poziva `sched_yield()` kojim se označava kraj računanja u periodi) ona je potrošila svoje 'rezervirano vrijeme' u ovoj periodi. Stoga

ona u tom trenutku može biti i istisnuta, ako se za to pojavi potreba od ostalih dretvi u sustavu. Obično se to ostvaruje na način da se njen krajnji trenutak završetka poveća za trajanje periode ('koristi se i njena iduća perioda') te će na red prije njenog nastavka doći ostale dretve s bližim kranjim trenutkom završetka. Pregled ovog i ostalih načina raspoređivanja u Linux jezgri detaljnije je opisan na [Linux scheduling].

Raspoređivanje sporadičnih poslova

Osiguranje unaprijed definiranog procesorskog vremena pojedinom poslu trebalo bi omogućiti i sporadično raspoređivanje. Raspoređivanje `SCHED_SPORADIC` je noviji postupak raspoređivanja (vrlo rijetko još podržan u operacijskim sustavima, primjerice [QNX, 6.3]), pripremljen za periodičke dretve, kod kojih dretva tijekom svake svoje periode treba određeno procesorsko vrijeme. Raspoređivač bi joj to vrijeme trebao dodijeliti uz viši prioritet. Ako u tom vremenu ne obavi svoj periodički posao, prioritet joj se smanjuje tako da suviše ne utječe na ostale dretve sustava. Parametri tog raspoređivanja su: period obnavljanja prioriteta (engl. *replenishment period*), povlašteno vrijeme rada (engl. *initial budget*), povlašteni prioritet (engl. *high priority*) te smanjeni prioritet (engl. *lower priority*). Primjer rada ovog raspoređivača prikazuje slika 4.3.



Slika 4.3. Primjer sporadičnog raspoređivanja

Sporadično raspoređivanje je namijenjeno periodičkim poslovima većeg prioriteta koji uglavnom kratko obave svoje operacije (u svakoj pojavi/periodi). Međutim, ponekada se može dogoditi da posao zahtijeva malo više vremena. Da se u tim slučajevima ne bi narušili ostali poslovi, `SCHED_SPORADIC` će za te dulje obrade smanjiti prioritet dretvi, nakon početnog povlaštenog vremena s većim prioritetom.

Primjer 4.1. Primjer korištenja POSIX sučelja

U ovome primjeru se koriste načini raspoređivanja `SCHED_FIFO` i `SCHED_RR`. Ispis prioriteta dretvi obavlja se unutar kritična odsječka da ne bi došlo do preklapanja u ispisu.

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <pthread.h>

#define THREADS 5
static pthread_mutex_t monitor_handle = PTHREAD_MUTEX_INITIALIZER;

static void print_thread_scheduling_parameters ( long id )
{
    int policy;
    struct sched_param prio;

    pthread_mutex_lock ( &monitor_handle );
    pthread_getschedparam ( pthread_self(), &policy, &prio );
    printf ( "Thread %ld: policy=%d, prio=%d\n", id,
             policy, prio.sched_priority );
    pthread_mutex_unlock ( &monitor_handle );
}

static void *thread_func ( void *param )
{
    print_thread_scheduling_parameters ( (long) param );
    return NULL;
}

int main ()
{
    long min, max, main_prio, i, policy;
    pthread_attr_t attr;
    pthread_t tid[THREADS];
    struct sched_param prio;

    print_thread_scheduling_parameters ( 0 );

    /*! Get priority range for "policy" */
    policy = SCHED_FIFO;
    min = sched_get_priority_min ( policy );
    max = sched_get_priority_max ( policy );

    /* set scheduling policy and priority for main thread */
    main_prio = prio.sched_priority = (min + max) / 2;
    if ( pthread_setschedparam ( pthread_self(), policy, &prio ) ) {
        perror ( "Error: pthread_setschedparam (root permission?)" );
        exit (1);
    }

    print_thread_scheduling_parameters ( 0 );

    /* define scheduling properties for new threads */
    pthread_attr_init ( &attr );
    pthread_attr_setinheritsched ( &attr, PTHREAD_EXPLICIT_SCHED );
    policy = SCHED_RR;
    pthread_attr_setschedpolicy ( &attr, policy );
```

```
    /* create threads */
    for ( i = 0; i < THREADS; i++ ) {
        prio.sched_priority = (min + i) % main_prio;
        pthread_attr_setschedparam ( &attr, &prio );
        if ( pthread_create(&tid[i], &attr, thread_func, (void *)(i+1))
            ) {
            perror ( "Error: pthread_create" );
            exit (1);
        }
    }

    /* wait for threads to finish */
    for ( i = 0; i < THREADS; i++ )
        pthread_join ( tid[i], NULL );

    return 0;
}

/* Example run: (on single processor system !!!)
$ gcc scheduling.c -pthread -Wall
$ sudo ./a.out
Thread 0: policy=0, prio=0
Thread 0: policy=1, prio=50
Thread 5: policy=2, prio=5
Thread 4: policy=2, prio=4
Thread 3: policy=2, prio=3
Thread 2: policy=2, prio=2
Thread 1: policy=2, prio=1
*/
```

Iz ispisa prikazanome uz kod vidi se da se stvorene dretve izvode prema prioritetu: naprije ona prirriteta 5, a najzadnja ona prioriteta 1, iako je redoslijed njihova pokretanja bio obrnut. Međutim, na višeprocesorskim sustavima to ne mora biti tako. Naime, u prikazanome primjeru je posao svake dretve samo ispis parametara raspoređivanja što je vrlo kratko, kraće od stvaranja nove dretve. Stoga ona dretva manjeg prioriteta stigne obaviti svoje prije pojave dretve većeg prioriteta koja bi ju inače istisnula.

4.8.4. Raspoređivanje nekritičnih dretvi

Pri raspoređivanju nekritičnih dretvi mogu se koristiti razna načela:

- pravednost – pravedno raspodijeliti procesorsko vrijeme svim dretvama u sustavu (u skladu njihovim prioritetima)
- veća procesorska iskoristivost – uz višu iskoristivost više će se posla obaviti
- veći broj završenih dretvi – dovršiti što više dretvi – favorizirati kratke dretve
- minimizacija čekanja u redovima – smanjuje se prosječno vrijeme koje dretve provedu u redu prije nego li su poslužene (npr. smanjiti kvant vremena)
- kraće vrijeme odziva interaktivnih dretvi – one rijetko koriste procesor, pa im zato odmah posvetiti pažnju čim budu pripravne (obično upravljaju s korisničkim sučeljem ili UI jedinicama)
- optimizacija korištenja priručnih spremnika – u sustavima s više procesorskih jedinki zadržavati dretvu na istom procesoru jer u sukcesivnim izvođenjima (nakon zamjene s drugim dretvama) te dretve mogu pronaći svoje podatke još uvijek u priručnom spremniku i time smanjiti vrijeme njihovog dohvata pri radu dretve, i općenito gledajući povećati učinkovitost sustava.

U operacijskim sustavima opće namjene dretve koji nisu vremenski kritične uglavnom se raspoređuju principom pravedne podjele procesorskog vremena. Dretve i tu imaju atribut prioriteta, ali se prioritet koristi za određivanje koliko će procesorskog vremena te dretve dobiti (izračun se obavlja uzimajući sve pripravne dretve i njihove prioritete).

Uobičajeni princip raspoređivanja koji se koristi za takve dretve naziva se višerazinsko raspoređivanje s povratnom vezom (engl. *multilevel feedback queue – MFQ*). Ciljevi koje to raspoređivanje nastoji ostvariti su:

- dati prednost dretvama s kratkim poslovima
- dati prednost dretvama koje koriste ulazno-izlazne naprave
- na osnovi rada dretve ustanoviti u koju skupinu dretva pripada te ju prema tome dalje raspoređivati.

Duge dretve, tj. dretve koje su procesorski intenzivne, koje trebaju puno procesorskog vremena (koje bi cijelo vrijeme koristile procesor) se žele potisnuti. Naime, takve dretve ionako duže traju, te se očekuje da će kasnije biti gotove te će njihova kratka odgoda zbog izvođenja drugih dretvi manje utjecati na sustav nego odgoda kratkih dretvi, koje, primjerice upravljaju korisničkim sučeljem i čija reakcija mora biti promptna (inače se kod korisnika stvara dojam sporosti sustava).

Sam postupak raspoređivanja koji nastoji poštovati navedena načela može se opisati skupom pravila ponašanja raspoređivača nad skupom dretvi koje su složene prema trenutnim prioritetima u redove prema redu prispijeća (slično slici 4.1.):

- Uvijek se raspoređuje prva dretva iz reda najvećeg prioriteta (najviši neprazni red).
- Procesor se dretvi dodjeljuje za određeni interval vremena (kvant vremena).
- Ako dretva završi u tom intervalu, ona napušta sustav (ne razmatra se više u postupku raspoređivanja).
- Ako se dretva pri svom izvođenju u tom njoj dodijeljenom intervalu blokira, miče se iz reda pripravnih, ali se kasnije, pri odblokiranju stavlja u isti red pripravnih dretvi iz kojeg i otišla (‘zadržava prioritet’), ili ovisno o vremenu provedenom u blokiranom stanju, čak se postavlja i u više redove (podiže joj se prioritet).
- Ako se dretva izvodi cijeli interval i nije ni gotova niti se blokirala za vrijeme tog izvođenja, onda ju raspoređivač prekida i miče u red niže (smanjuje joj prioritet za jedan).
- Postupak se ponavlja dok god ima pripravnih dretvi i dok one ne dođu do najnižeg reda (reda najmanjeg prioriteta). U tom redu se dretve poslužuju podjelom vremena (ne idu u red niže na kraju intervala, već samo na kraj istog reda).

- Kada u sustav dođe nova dretva, ona se stavlja u najviši red (najprioritetniji red), na kraj tog reda.

Opisani postupak vrlo brzo procjenjuje dretvu: je li ona spada u kategoriju kratkih ili dugih. Ako je kratka, ostaje joj prioritet, a ako je duga prioritet joj pada tijekom izvođenja.

Višerazinsko raspoređivanje s povratnom vezom se ne ostvaruje u operacijskim sustavima upravo prema prikazanim pravilima, ali se sličnim pristupima ostvaruju navedena načela.

Razmotrimo raspoređivanje nekritičnih dretvi u operacijskim sustavima zasnovanim na Linux jezgri te sustavima zasnovanim na porodici operacijskih sustava Microsoft Windows.

Primjer 4.2. Raspoređivači ugrađeni u Linux jezgru

Operacijski sustavi zasnovani na Linux jezgri podržavaju načine raspoređivanja `SCHED_FIFO`, `SCHED_RR` i `SCHED_DEADLINE` (tek od 3.14) za kritične dretve te `SCHED_OTHER`, `SCHED_BATCH` (od 2.6.16) i `SCHED_IDLE` (od 2.6.23) za obične, nekritične dretve. Prioriteti kritičnih dretvi kreću se od 1 do 99 (veći broj označava veći prioritet). Za raspoređivanje nekritičnih dretvi koristi se prioritet 0.

Nekritične dretve će dobiti procesorsko vrijeme tek kada nema niti jedne kritične dretve u redu pripravnih. Iznimno, od Linux jezgre 2.6.25 moguće je rezervirati dio procesorskog vremena i za nekritične dretve. Uobičajeno je to postavljeno na 5 % procesorskog vremena. Navedena rezervacija omogućava zaustavljanje dretve koja se raspoređuje prema `SCHED_FIFO`, `SCHED_RR` ili `SCHED_DEADLINE` i koja ima grešku (npr. beskonačnu petlju i koristila bi svo procesorsko vrijeme), korištenjem naredbi u ljuski koje koriste obično raspoređivanje (`SCHED_OTHER`).

Dretve koje spadaju u klase `SCHED_OTHER`, `SCHED_BATCH` i `SCHED_IDLE` imaju osnovni prioritet postavljen na nulu (manji od najmanjeg za kritične dretve), ali za međusobnu usporedbu (raspoređivanje) koriste drugu vrijednost, takozvanu *razinu dobrote* (engl. *nice level*) koja se kreće od -20 (najveća) do $+19$ (najmanja). Dobrote ispod nule može postavljati samo administrator (korisnik *root*). Dobrota dretve utječe na to koliko će procesorskog vremena dretva dobiti u odnosu na ostale dretve različite dobrote. Primjerice, dretva s razinom dobrote q trebala bi dobiti 10 do 15 % više procesorskog vremena od dretve razine $q + 1$.

Načini raspoređivnja `SCHED_OTHER` i `SCHED_BATCH` su slični, jedino što će raspoređivač uvijek pretpostaviti da je dretva u klasi `SCHED_BATCH` duga dretva i zbog toga biti neznatno u lošijem položaju od ostalih dretvi (u klasi `SCHED_OTHER`).

Dretve u klasi `SCHED_IDLE` imaju najmanju dobrotu i neće se izvoditi dokle god ima drugih dretvi koje nisu u istoj klasi.

Od inačice Linux jezgre 2.6.23 za raspoređivanje nekritičnih dretvi (`SCHED_OTHER`, `SCHED_BATCH` i `SCHED_IDLE`) koristi se novi raspoređivač naziva *potpuno pravedan raspoređivač* (engl. *completely fair scheduler – CFS*). Korištenjem izračunatog virtualnog vremena koje pripada pojedinoj dretvi i stvarno dodijeljenog vremena, tj. razlike tih vremena izgrađuju se crveno-crna stabla s pripravnim dretvama te se za aktivnu odabire ona dretva kojoj sustav najviše duguje (s najvećom razlikom).

Prikažimo rad potpuno pravednog raspoređivača na pojednostavljenom primjeru.

Neka se u sustavu u početnom trenutku nalazi pet dretvi iste razine dobrote $\{D_1, \ldots, D_5\}$. Raspoređivač će odabrati jednu, recimo D_1, i njoj dati kvant vremena T_q. Nakon što taj kvant istekne, raspoređivač će ažurirati virtualna vremena koja pripadaju pojedinim dretvama: u tom intervalu svaka od dretvi dobiva jednaki dio virtualnog vremena, tj. $T_q/5$. Obzirom da se samo D_1 izvodila, jedino će se njoj povećati dodijeljeno vrijeme za T_q što

će uzrokovati pomak te dretvu u stablu – ona više neće biti zajedno s ostalima već zadnja u ovom trenutku. Zato će raspoređivač u idućem trenutku odabrati neku drugu dretvu $\{D_2, \ldots, D_5\}$ kao aktivnu.

Primjer 4.3. Microsoft Windows operacijski sustavi

Raspoređivanje u porodicama operacijskih sustava Microsoft Windows (NT, 2000, XP, 2003, Vista, 7, 8) obavlja se korištenjem prioriteta dretvi. Prioritet dretve računa se na osnovu prioritetne klase procesa i prioritetne razine dretvi, prema tablicama 4.1. i 4.2.

Tablica 4.1. Prioritetne klase procesa

oznaka klase	skraćeno
IDLE_PRIORITY_CLASS	ID
BELOW_NORMAL_PRIORITY_CLASS	BN
NORMAL_PRIORITY_CLASS	N
ABOVE_NORMAL_PRIORITY_CLASS	AN
HIGH_PRIORITY_CLASS	H
REALTIME_PRIORITY_CLASS	RT

Tablica 4.2. Prioritetne razine dretvi

oznaka razine	skraćeno
THREAD_PRIORITY_IDLE	ID
THREAD_PRIORITY_LOWEST	L
THREAD_PRIORITY_BELOW_NORMAL	BN
THREAD_PRIORITY_NORMAL	N
THREAD_PRIORITY_ABOVE_NORMAL	AN
THREAD_PRIORITY_HIGHEST	H
THREAD_PRIORITY_TIME_CRITICAL	TC

Tablice 4.3. i 4.4. prikazuju dodjeljivanje prioriteta na osnovu prioritetnih klasa i razina.

Tablica 4.3. Izračun prioriteta za normalne dretve (ID, BN, N, AN, H)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ID	ID	L	BN	N	AN	H									TC
BN	ID			L	BN	N	AN	H							TC
bN	ID				L	BN	N	AN	H						TC
fN	ID						L	BN	N	AN	H				TC
AN	ID							L	BN	N	AN	H			TC
H	ID										L	BN	N	AN	H,TC

Prioritet 0 (najniži prioritet) rezerviran je za posebne dretve operacijskog sustava (koje brišu oslobođene stranice u postupku upravljanja spremnikom straničenjem).

Tablica 4.4. Izračun prioriteta za kritične dretve (RT)

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ID	−7	−6	−5	−4	−3	L	BN	N	AN	H	3	4	5	6	TC

Raspoređivanje kritičnih dretvi (klasa REALTIME_PRIORITY_CLASS) se ponešto razlikuje od ostalih klasa, odnosno, raspoređivanje je identično prethodno opisanom načinu SCHED_RR, uz raspon prioriteta od 16 do 31.

Raspoređivanje nekritičnih dretvi obavlja se korištenjem njihovih prioriteta – dretve najvećeg prioriteta se odabiru za izvođenje, slično SCHED_RR, ali uz iznimke:

- Prve iznimke su dretve procesa u klasi NORMAL_PRIORITY_CLASS (N) koji je trenutno u fokusu korisnika (engl. *foreground process*) koji dobiva malo povećanje prioriteta (prefiks f u tablici 4.3.) u odnosu na dretve koje nisu u fokusu (prefiks b).
- Druge iznimke su pripravne dretve koje dugo nisu dobile ništa procesorskog vremena (izgladnjele dretve). Njima raspoređivač povremeno dodijeljuje dio procesorskog vremena da izbjegne njihovo potpuno izgladnjivanje i također da ublaži problem inverzije prioriteta. Više o problemu inverzije prioriteta u poglavlju **??**

4.9. Upravljanje poslovima u uređajima napajanim baterijama

Mnogi su računalni sustavi (računala) napajana iz baterija, bez priključka na izvor stalne energije (na električnu mrežu). U takvim je sustavima osobito bitno osigurati željeni period samostalnog rada. Dok se za prijenosna računala, tablete i pametne telefone to vrijeme mjeri satima, za neke druge to mogu biti dani, tjedni, mjeseci pa i godine! Radi omogućavanja navedena načina rada treba prilagoditi i sklopovlje i programe koji će biti i manje zahtjevni i koji će znati iskoristiti posebnosti sklopovlja.

Veliki potrošač energije jest procesor jer najčešće on može raditi na najvećoj frekvenciji. Stoga se on pokušava izvesti u posebnoj tehnologiji manje potrošnje, s pojednostavljenim operacijama u odnosu prema procesoru namjenjenome stolnim računalima. Mogućnosti upravljanja takvim procesorom radi uštede energije uglavnom spadaju u slijedeće kategorije:

- prilagodljiva frekvencija rada – manja kada je računalo napajano baterijom ili nema potrebe za većom procesorskom snagom
- zaustavljanje procesora posebnim instrukcijama kada nema nikakva posla (npr. pri izvođenju latentne/idle dretve)
- zaustavljanje nepotrebnih jezgri u višejezgrenome procesoru.

Za ostale komponente računa mogu vrijediti ista načela kao i za procesor: izvedba u tehnologiji manje potrošnje, mogućnost zaustavanja rada naprave te rada u načinu sa smanjenom potrošnjom.

Operacijski sustav treba poznavati mogućnosti naprava i koristiti one načine uštede energije koji naprave pružaju. Ovdje spadaju i isključivanje zaslona, postavljanje u stanje niske potrošnje ili čak i gašenje računala nakon nekog intervala nekorištenja.

Razni tipovi računala se koriste na razne načine.

Prijenosno računalo je zapravo zamjena za stolno te nasljeđuje njegova svojstva i načine korištenja. To zapravo znači da korisnik pokreće neke programe koje operacijski sustav raspoređuje prema utvrđenim kriterijima, primjerice, prema prioritetu.

Pametni telefon i tablet računalo se naizgled koriste na isti način. Međutim, oni su u začetku zamišljeni za jednog korisnika i jedan program s kojim korisnik u jednom trenutku komunicira. Stoga je i sama unutarnja arhitektura tih sustava ponešto drukčija. Program koji komunicira s korisnikom je u tom trenutku najvažniji i njemu treba dati sve potrebna sredstva sustava, dok ostale treba zaustaviti. Obzirom na taj očekivani način rada, programi pisani za takve sustave imaju dio koda koji se izvodi dok je program aktivan (komunicira s korisnikom), dio koda koji se izvodi kada program prestaje biti aktivan, dio koda koji se izvodi kada program ponovno postaje aktivan i slično. Ukoliko program treba izvoditi neke periodičke radnje i dok nije aktivan onda se on mora koristiti posebnim sučeljem operacijskog sustava za izvođenje tih periodičkih aktivnosti. Primjerice, slušanje glazbe zahtijeva periodički dohvat dijela skladbe, obradu tog dijela te prijenos prema zvučnom podsustavu računala koji će ga dalje proslijediti u pravom obliku do zvučnika.

Ostala ugrađena računala manje procesne snage prilagođavaju se svojoj funkciji. Najčešće to nisu računala za opću uporabu i nadogradnju s programima različite namjene, već služe samo za jednu ili do nekoliko namjena. U ovu klasu računala možemo ubrojiti i računala koja se sada nazivaju 'Internet stvari' (engl. *Internet of Things – IoT*), a za koje se predviđa da će u bliskoj budućnosti brojčano znatno nadmašiti sva ostala računala zajedno. Radi osiguravanja dugog rada takvih sustava potrebno je izdvojiti one komponente koje potencijalno troše najviše energije i prilagoditi njihov rad da se potrošnja smanji. S današnje perspektive to su dijelovi računala predviđeni za ostvarenje komunikacije za koje je za sada potrebna najveća energija ukoliko bi željeli stalnu povezanost tih računala s ostalima ('na Internet'). Već i sada postoji nekoliko protokola koji omogućuju komunikaciju koja nije toliko zahtjevna kao dosadašnje tehnologije. Ipak, još uvijek se radi na osmišljavanju još boljih, tj. manje zahtjevnih protokola i tehnologija, koje bi omogućile još dulji rad takvih ugrađenih računala i sa samo povremenom povezanošću s ostalim računalima, npr. čak i ne izravno na mrežnu infrastrukturu već do nje preko ostalih sličnih računala, npr. bežične mreže osjetila (engl. *wireless sensor network – WSN*). Primjerice, ugrađeno računalo koje bi se moglo ugraditi u čovjeka (npr. ispod kože), a koje bi mjerilo temperaturu, krvni tlak, udio neke materije u krvi, pratilo rad srca i slično, trebalo bi biti što manje, u mogućnosti raditi što duže (mjereno u godinama!). S druge strane takvo bi računalo trebalo svoja mjerenja u nekim intervalima pokušati proslijediti svoja očitanja prema drugom računalu koje bi pohranilo ta očitanja te možda napravila i neke složenije analize tih podataka. Osnovnu analizu bi možda moglo napraviti i ugrađeno računalo te poslati poruke upozorenja ukoliko su očitani podaci problematični.

Ukoliko se projektira ugrađeno računalo napajano baterijom, pored uobičajenih problema ostvarenja logičke i vremenske ispravnosti treba voditi računa i o samostalnom radu računala u predviđenom periodu njegova rada te koristiti i postupke kojim će se smanjiti potrošnja energije, a da bi se željena samostalnost mogla ostvariti. Odabir redoslijeda izvođenja raznih poslova/operacija će stoga vjerojatno biti različit od do sada opisivanih postupaka koji nisu uzimali u obzir potrošnju energije.

Literatura

[POSIX] *POSIX sučelja, 7. izdanje*, 2008, http://pubs.opengroup.org/onlinepubs/9699919799.

[Linux scheduling] *Linux Programmer's Manual: sched – overview of scheduling APIs*, http://man7.org/linux/man-pages/man7/sched.7.html.

[QNX, 6.3] *The QNX Neutrino Microkernel – Scheduling*, 2013., http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/sys_arch/kernel.html#SCHEDULING