SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 359

OSTVARENJE EDF RASPOREĐIVANJA ZA RTLINUX

Hrvoje Knežević

Zagreb, lipanj 2012.

ZAHVALA

Ovom prilikom želim se zahvaliti najviše svojim roditeljima, koji su vlastitim odricanjem omogućili moje školovanje. Želim se zahvaliti i svim kolegama na pruženoj pomoći i podršci tijekom studiranja, kao i profesorima i asistentima fakulteta na uloženom vremenu i prenesenim znanjima.

Posebna zahvala mom mentoru, doc. dr. sc. Leonardu Jelenkoviću na strpljenju, pomoći i vodstvu pri izradi ovog diplomskog rada.

SADRŽAJ

1. Uvod 1

2. Osnovne značajke operacijskog sustava RTLinux 3

2.1. Općenito o sustavu RTLinux 3

2.2. Podsustav za raspoređivanje zadataka 5

2.3. POSIX signali i alarmi u sustavu RTLinux 9

3. Ostvarenje EDF raspoređivanja u RTLinuxu 11

3.1.Princip rada EDF raspoređivača 11

3.2. Ostvarenje raspoređivača u programskom kodu sustava 11

4. Proširenje raspoređivača dodatnim mogućnostima upravljanja zadacima... 15

4.1. Podešavanje ponašanja dretve u slučaju pogreške 16

4.2. Otkrivanje pogrešaka u raspoređivačkom podsustavu 19

5. Demonstracija rada ostvarenih funkcionalnosti 21

5.1. Prikaz rada EDF raspoređivača 21

5.2. Korištenje naprednih mogućnosti upravljanja zadacima 23

5.3. Prikaz rada uz omogućene kontrolne ispise 25

5.4. Završne napomene 26

6. Zaključak 27

Popis korištene literature 28

Popis slika 30

Sažetak 31

Summary 32

Privitak 33

Izvorni kod modula za testiranje raspoređivača 33

1. Uvod

U operacijskim sustavima za rad u stvarnom vremenu uz stroga vremenska ograničenja (engl. *hard real-time systems*) raspoređivač zadataka jedan je od najvažnijih podsustava. U takvim sustavima raspoređivač u ovisnosti o prioritetu zadataka donosi odluke o tome kojem će se zadatku u nekom trenutku dopustiti izvođenje. Drugi vrlo važan čimbenik pri raspoređivanju zadataka je zadovoljavanje vremenskih zahtjeva, odnosno ograničenja vezanih uz trajanje izvođenja zadatka. To znači da zadaci od trenutka pojave nužno moraju završiti izvođenje do nekog zadanog trenutka. Uobičajeno je da se ovo ograničenje definira pomoću **vremena trenutka krajnjeg završetka** (engl. *deadline time*).

Operacijski sustav RTLinux dizajniran je kako bi zadovoljio stroga vremenska ograničenja. Međutim, jedini način raspoređivanja koji se koristi u ovom sustavu je statičko raspoređivanje prema prioritetima, koje prilikom raspoređivanja zadataka uzima u obzir samo prioritet zadatka, a o zadovoljavanju vremenskih zahtjeva potrebno je voditi računa prilikom programiranja aplikacija, dakle briga o vremenskim zahtjevima u potpunosti je zadaća programera. Ovakvo raspoređivanje sprječava mogućnost pojave nepredviđenih situacija u radu sustava i jamči izvođenje prioritetnijih zadataka u trenutku pojave istih. Međutim, u nekim situacijama statički raspoređivač ne zadovoljava zahtjeve aplikacije, te je potrebno koristiti druge kriterije raspoređivanja.

Dinamičko raspoređivanje zadataka ima brojne prednosti, te je naročito korisno ako se u kombinaciji sa zadacima strogih vremenskih ograničenja žele koristiti i neki zadaci kod kojih je dozvoljena određena odgoda izvođenja, te nije potrebna apsolutna vremenska preciznost. Nadalje, ovakvo raspoređivanje omogućava iskorištenje procesorskih resursa do 100%, što je korisno u trenucima kada je apsolutno nužno da se neki zadatak obavi do zadanog roka.

Za ostvarenje dinamičkog raspoređivanja u RTLinuxu odabrana je metoda **raspoređivanja prema trenucima krajnjih završetaka** (engl. *earliest deadline-first, EDF*). Kod ovog načina raspoređivanja, iz skupa zadataka spremnih za izvođenje odabire se zadatak sa najbližim trenutkom krajnjeg završetka (engl. *deadline*). Svaki zadatak određen je pomoću tri parametra: vremena trajanja izvođenja, perioda P i apsolutnog vremena krajnjeg završetka d. Svaki zadatak od trenutka pojave (početka novog perioda rada) mora završiti sa radom do trenutka krajnjeg završetka D relativnog prema trenutku pojave zadatka (engl. *relative deadline*). Dakle, krajnji trenutak do kojeg se zadatak mora

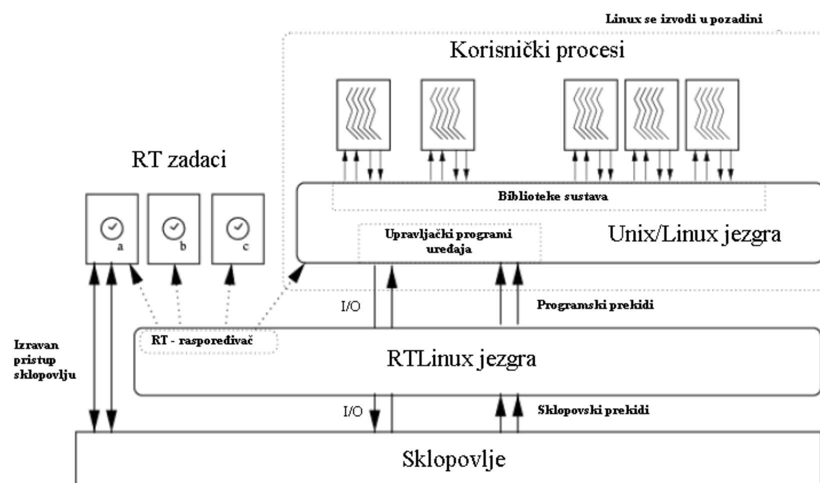
obaviti u k-tom periodu izvođenja je $D = k*P + d$. Ako u nekom trenutku ne postoji zadatak koji čeka na izvođenje, procesor ulazi u stanje mirovanja. EDF raspoređivanje optimalno je za bilo koji skup neovisnih prekidivih (engl. *preemptive*) zadataka.

U nastavku rada opisan je ukratko operacijski sustav RTLinux, analiziran je podsustav za upravljanje raspoređivanjem zadataka, te je osmišljena izmjena postojećeg raspoređivačkog modula kako bi se ostvarilo dinamičko raspoređivanje prema trenucima krajnjih završetaka. Uz samo raspoređivanje zadataka, ostvarene su i neke dodatne mogućnosti naprednog upravljanja zadacima u slučaju prekoračenja vremenskih ograničenja, te se demonstrira funkcionalnost dinamičkog raspoređivanja za vrijeme rada sustava.

2. Osnovne značajke operacijskog sustava RTLinux

2.1. Općenito o sustavu RTLinux

Real-Time Linux (RTLinux) je operacijski sustav za rad u stvarnom vremenu sa strogim vremenskim ograničenjima (engl. *hard real-time operating system*). RTLinux je mala jezgra operacijskog sustava koja koegzistira uz osnovnu jezgru operacijskog sustava Linux. RT jezgra je smještena kao međusloj između osnovne Linux jezgre i sklopovlja računala. Prilikom podizanja sustava RT jezgra pokreće osnovni operacijski sustav. Iz perspektive RT jezgre osnovni sustav je predstavljen kao dretva najnižeg prioriteta, dok osnovna jezgra istovremeno "vidi" sloj RT jezgre kao sklopovlje računala. RT jezgra presreće sve zahtjeve za prekidima od strane sklopovlja, te ih selektivno propušta u osnovnu jezgru sustava. To je ostvareno na način da se svi prekidi vezani uz izvođenje u stvarnom vremenu izvode prvi, sa najvećim prioritetom, te se za njih pokreće odgovarajuća prekidna rutina. Svi ostali prekidi prosljeđuju se osnovnoj jezgri maskirani kao programski prekidi. Na taj način postiže se da se sve zadaće koje to zahtijevaju izvode u stvarnom vremenu. Zadaci koji zahtijevaju rad u stvarnom vremenu imaju izravan pristup sklopovlju i ne koriste straničenje. Aplikacije pisane za osnovni operacijski sustav nije moguće nadograditi kako bi podržavale funkcionalnosti za rad u stvarnom vremenu, već je potrebno u potpunosti redizajnirati svaku aplikaciju kako bi joj se omogućilo korištenje RT funkcionalnosti. Budući da RTLinux podržava rad u strogom stvarnom vremenu, pri izvođenju RT aplikacija sustav garantira vrijeme odziva od 20 mikrosekundi u najgorem slučaju (engl. *worst case latency time*).



Slika 1: Arhitektura operacijskog sustava RTLinux

Slika 1 prikazuje osnovnu arhitekturu operacijskog sustava sa ugrađenim *RT* slojem. Linux jezgra "vidi" RT međusloj kao sklopovlje, te se sva komunikacija između sklopovlja i osnovne jezgre odvija posredstvom RT jezgre. RT jezgra emulira sklopovlje za prekide, te selektivno obrađuje zahtjeve za prekidima od strane sklopovlja: ako zahtjev dolazi od "standardnog" sklopa (koji ne zahtijeva izvođenje u strogom stvarnom vremenu), tada RT sloj prosljeđuje zahtjev za prekidom osnovnoj jezgri u obliku programskog prekida i upravljanje procesima se izvodi iz osnovne jezgre. Ukoliko se pojavi zahtjev za prekidom od strane RT sklopovlja, tada se zaustavljaju svi zadaci osnovne jezgre, upravljanje preuzima RT jezgra i pokreće se prekidna rutina najvećeg prioriteta. Pritom jezgra osnovnog operacijskog sustava ne može izvoditi, odgoditi niti onemogućiti prekide. Drugim riječima, dok se izvodi RT zahtjev, svi drugi zahtjevi se odgađaju i bit će obrađeni kasnije, nakon što se upravljanje ponovno prepusti osnovnoj jezgri operacijskog sustava. O mehanizmima koji ostvaruju ovu funkcionalnost bit će govora kasnije.

Svi podsustavi jezgre RTLinuxa ostvareni su kao programski moduli koje je moguće dinamički učitavati u memoriju. Ovakvim pristupom postiže se neovisnost različitih podsustava jezgre, te se optimizira rad sustava i smanjuje zauzeće memorije, budući da se određeni podsustavi mogu učitavati prema potrebi.

RTLinux jezgra može prekidati zadatke usred njihova izvođenja (engl. *preemptive kernel*). RT zadaci koji se izvode u ovakvom sustavu imaju dva važna svojstva: privilegirani su (imaju izravan pristup sklopovlju) i ne koriste straničenje. RT zadaci su, slično kao i podsustavi jezgre, ostvareni kao specijalni moduli koji se mogu dinamički učitavati u memoriju. Oni ne mogu koristiti sistemske pozive osnovnog sustava, izravno pozivati rutine ili pristupati standardnim podatkovnim strukturama. Inicijalizacijski kod RT zadataka pri pokretanju inicijalizira RT strukturu i prosljeđuje RTLinux jezgri informacije o trajanju zadatka, periodu izvođenja zadatka i drugim vremenskim ograničenjima.

Važno je napomenuti kako RTLinux ne radi nikakve promjene na osnovnoj jezgri operacijskog sustava, već pomoću relativno jednostavnih izmjena pretvara osnovni Linux sustav u sustav za rad u stvarnom vremenu sa strogim vremenskim ograničenjima. Zbog jednostavnosti ostvarenja RTLinux ne posjeduje mehanizme za zaštitu memorijskog prostora od nedopuštenog pisanja ili čitanja, kao niti napredne mehanizme sinkronizacije između različitih RT zadataka. Pri dizajniranju aplikacija za rad u stvarnom vremenu programer bi se trebao pobrinuti za ispravnost koda, te predvidjeti i spriječiti moguće situacije pri izvođenju zadatka koje bi mogle dovesti do spomenutih radnji.

2.2. Podsustav za raspoređivanje zadataka

Raspoređivač zadataka u operacijskom sustavu RTLinux ostvaren je u zasebnom programskom modulu, te je na taj način odvojen od ostalih podsustava. Ovakav pristup omogućava jednostavne izmjene postojećeg raspoređivača za potrebe optimizacije ili ostvarenja novih metoda raspoređivanja. Standardni (i jedini) način raspoređivanja u RTLinuxu je statičko raspoređivanje prema prioritetima. Prilikom pokretanja sustava raspoređivački modul RTLinuxa učitava se u jezgru sustava. U tom trenutku u raspoređivaču je prisutna samo jedna dretva, dretva Linux jezgre, koja je u RTLinuxu predstavljena kao "*idle task*", odnosno zadatak za izvođenje "u stanju mirovanja". Ova dretva izvodi se u trenucima kada ne postoji niti jedan RT zadatak koji čeka na izvođenje, te joj je dodijeljen najniži prioritet. Tako se postiže da temeljna dretva jezgre nikad ne može prekinuti RT zadatak za vrijeme izvođenja, odnosno aktivni RT zadatak može prekinuti samo drugi RT zadatak većeg prioriteta.

Važno je u kontekstu operacijskog sustava baziranog na Linux arhitekturi napomenuti da se u ovom tekstu pojmovi *dretva* i *zadatak* ne razlikuju. Naime, rane verzije ovog sustava temeljile su se na procesima (zadacima). Od verzije 2.0 RTLinux je ostvaren kao jednoprocesni sustav u kojem se funkcionalne jedinice ostvaruju u zasebnim dretvama umjesto u zasebnim procesima. Prema tome, u opisu funkcionalnosti raspoređivača oba pojma predstavljaju dretvu kao osnovnu funkcionalnu jedinicu sustava.

Kompletna funkcionalnost statičkog raspoređivača ostvarena je u funkciji `rtl_schedule()`. Raspoređivački sustav raspolaže sa jednim sklopovskim brojačem vremena, `CLOCK_RTL_SCHED`, dok je mjerenje vremena u RT dretvama sustava ostvareno pomoću programskih brojača. Svaka dretva ima na raspolaganju jedan programski brojač koji mjeri trenutak ponovne pojave dretve, periodički ili jednokratno. Istek ovih brojača određuje trenutke u kojima se poziva raspoređivački modul.

Kod raspoređivanja zadataka koriste se tri važne strukture: opisnik dretve, opisnik atributa dretve i raspoređivačka struktura. Opisnik dretve ima sljedeći oblik:

```
struct rtl_thread_struct {
      int *stack; /* hardcoded */
      int fpu_initialized;
      RTL_FPU_CONTEXT fpu_regs;
      int uses_fp;
      int *kmalloc_stack_bottom;
      struct rtl_sched_param sched_param;
      struct rtl_thread_struct *next;
      int cpu;
```

```
        hrtime_t resume_time;
        hrtime_t period;
        hrtime_t timeval;
        struct module *creator;
        void (*abort)(void *);
        void *abortdata;
        int threadflags;
        rtl_sigset_t pending;
        rtl_sigset_t blocked;
        void *user[4];
        int errno_val;
        struct rtl_cleanup_struct *cleanup;
        int magic;
        struct rtl_posix_thread_struct posix_data;
        void *tsd [RTL_PTHREAD_KEYS_MAX];
};
```

Za raspoređivanje zadataka najvažniji parametri u opisniku dretve su struktura `sched_param`, koja sadrži parametre za raspoređivanje (kod standardnog raspoređivača to je samo prioritet dretve), oznaka procesora u kojem se dretva izvodi (što je važno u višeprocesorskim sustavima), apsolutna vrijednost perioda `period`, relativno vrijeme ponovnog javljanja dretve `resume_time`, te skupovi zastavica za opis stanja dretve (`pending`, `blocked`, `thread_flags`). Ovi parametri postavljaju se ili ispituju prilikom raspoređivanja, te određuju hoće li i u kojem trenutku dretva započeti s izvođenjem. Uz ove parametre, opisnik dretve sadrži i neke druge parametre koji se koriste prilikom stvaranja i uklanjanja dretve, ili prilikom korištenja nekih specifičnih resursa sustava, kao što je npr. FPU jedinica.

Druga važna struktura je opisnik atributa dretve. Opisnik atributa koristi se za definiranje parametara rada dretve prilikom inicijalizacije nove dretve, te je definiran na sljedeći način:

```
typedef struct STRUCT_PTHREAD_ATTR {
        size_t stack_size;
        void *stack_addr;
        struct rtl_sched_param sched_param;
        int cpu;
        int use_fp;
        rtl_sigset_t initial_state;
        int detachstate;
} pthread_attr_t;
```

Za raspoređivački podsustav najvažnija je struktura `rtl_sched_cpu_struct`, u kojoj je definirana osnovna struktura raspoređivača. Ova struktura sadrži pokazivače na aktivnu dretvu, dretvu Linux jezgre, red dretvi sustava, vremenski sklop, te zastavice koje se koriste prilikom raspoređivanja zadataka, kao i neke dodatne parametre:

```
struct rtl_sched_cpu_struct {
      struct rtl_thread_struct *rtl_current;
      struct rtl_thread_struct rtl_linux_task;
      struct rtl_thread_struct *rtl_task_fpu_owner;
      struct rtl_thread_struct *rtl_tasks; /* the queue of  RT tasks */
      struct rtl_thread_struct *rtl_new_tasks;
      clockid_t clock;
      spinlock_t rtl_tasks_lock;
      int sched_flags;
      int sched_user[4];
}
```

Sve navedene strukture definirane su u datoteci `rtl_sched.h`, koja sadrži definicije svih važnijih struktura, zastavica, makroa i konstanti koje se koriste u raspoređivačkom modulu sustava.

Važno je napomenuti kako RTLinux nema poseban red pripravnih dretvi, već se sve dretve sustava nalaze u istom redu, a status dretve provjerava se pomoću zastavica u opisniku dretve. To znači da se kod svakog poziva raspoređivača prolazi kroz listu svih dretvi sustava, te se na kraju odabire sljedeća dretva koja će se izvoditi. Ovo može predstavljati problem ako je istovremeno aktivan velik broj dretvi, te može uzrokovati pad performansi sustava.

Raspoređivački podsustav ima tri važne zadaće: dodavanje novih zadataka u red, odabir sljedećeg zadatka koji će se izvoditi, te zamjenu konteksta aktivnog zadatka.

Dodavanje novih zadataka ostvaruje se unutar funkcije `pthread_create()`, kojom se stvara nova RT dretva sustava. Nakon inicijalizacije parametara dretve, pozvat će se funkcija `add_to_task_list()` koja dodaje novu dretvu na kraj reda dretvi sustava.

Odabir sljedećeg zadatka za izvođenje sastoji se od dvije radnje: pronalaženje novog zadatka za izvođenje i pronalaženje zadatka koji može prekinuti aktivni zadatak u izvođenju (engl. *preemptor*). Postupak je sljedeći: prolazi se kroz cijeli red dretvi sustava, te se za svaku dretvu prvo provjerava je li dretva nakon posljednjeg izvođenja označena za ponovno izvođenje u nekom budućem trenutku (zastavica `RTL_THREAD_TIMEARMED`). Zatim se provjerava je li dretvin programski brojač vremena istekao, te se u tom slučaju dretva označava pripravnom za rad. Ukoliko brojač vremena dretve još broji, dretva postaje kandidat za mjesto *preemptora*. U suprotnom, ako je dretva označena kao pripravna i nije blokirana, tada postaje kandidat za novu aktivnu dretvu. Ako prethodno u petlji nije pronađena druga dretva u istom stanju, tada se pokazivač `new_task` postavlja na vrijednost adrese trenutne dretve. Ako je prethodno već pronađena druga dretva koja je spremna na izvršavanje, tada se uspoređuju prioriteti ovih dretvi i `new_task` se postavlja na vrijednost adrese prioritetnije dretve.

Nakon prolaska kroz cijeli red dretvi, odabrana je sljedeća dretva za izvođenje i *preemptor*. Ako je pronađen *preemptor*, vrijeme prekida za ponovni poziv raspoređivača postavlja se na vrijeme pojave *preemptora*. U suprotnom se raspoređivač poziva periodički frekvencijom HZ/2. Ovaj trenutak postavlja se u jedinom brojaču vremena kojime raspolaže raspoređivački sustav, `sched->clock`, a koji je definiran u strukturi `rtl_sched_cpu struct`.

Ukoliko prolaskom kroz red svih dretvi sustava nije pronađen novi zadatak za izvođenje, sustav prelazi u stanje mirovanja, tj. prepušta procesorsko vrijeme dretvi Linux jezgre. Ako je ipak pronađen novi zadatak, raspoređivač prelazi u posljednju fazu, **zamjenu konteksta** aktivnog zadatka, što se izvodi pozivom funkcije `rtl_switch_to()`. Ova funkcija će obaviti zamjenu sadržaja registara sustava i pohraniti kontekst prethodne dretve na programski stog. Konačno, nakon zamjene konteksta aktivnoj dretvi se prosljeđuju signali sustava. Naime, u RTLinuxu signali se ne dostavljaju dretvi u trenutku pojave, već se svi signali zadržavaju do poziva raspoređivača i tek se onda obrađuju. U trenutku pojave određenog signala za vrijeme rada dretve postavlja se vrijednost bitova maske `pending` smještene u kontekstu dretve. Kada dretva ponovno postaje aktivna poziva se funkcija `do_signal()`, koja provjerava stanje bitova maske te izvodi radnje vezane uz određeni signal. Takvim pristupom onemogućuju se prekidi aktivne dretve uslijed izvođenja, te se prekid aktivne dretve može dogoditi samo prilikom isteka raspoređivačkog brojača vremena `sched->clock`.

Opisani redoslijed radnji prilikom poziva raspoređivača može se sažeto prikazati sljedećim pseudokodom:

```
Prođi kroz listu svih dretvi sustava{
      Ako (timer(dretva)->postavljen == 1){
            Ako (vrijeme_sada > timer(dretva)->vrijeme){
                  dretva->flags(pripravna) = 1;
            }
            Inače{
                  preemptor= Provjeri_i_postavi_preemptor(dretva);
            }
      }
      Ako (dretva->flags(pripravna) && !dretva->flags(blokirana) &&
          (!nova_dretva || (dretva->prioritet > nova_dretva->prioritet)){
            nova_dretva = dretva;
      }
}
```

```
Ako(pronađen preemptor)
      vrijeme_prekida = preemptor->vrijeme;
Ako(pronađena nova_dretva){
      Ako (nova_dretva != trenutna_dretva){
            Zamjena_konteksta(trenutna_dretva, nova_dretva);
            Dostavi_signale(trenutna_dretva);
      }
}
```

2.3. POSIX signali i alarmi u sustavu RTLinux

Podrška za alarme i signale prema POSIX standardu uključena je u RTLinux od verzije 3.2 *pre*-2. Ranije je spomenuto kako se svi signali u RTLinuxu dostavljaju dretvama u trenutku kada se dretvi dodijeli procesorsko vrijeme u raspoređivaču, odnosno u trenutku kada dretva postane aktivna. Alarmi predstavljaju mehanizam kojim je moguće poslati signal dretvi u zadanom vremenskom trenutku ili u trenutku isteka zadanog vremenskog intervala. Ova podrška omogućuje korištenje dodatnih programskih alarma u sustavu, te proširuje mogućnosti upravljanja vremenom i radom sustava.

Alarme je moguće podesiti prema dva osnovna kriterija: načinu rada i vremenskim odnosima. Prema načinu rada alarm je moguće definirati kao **periodički** i **jednokratni** (engl. *one-shot*). Periodički način rada omogućuje generiranje signala prilikom svakog isteka vremenskog perioda definiranog u strukturi alarma, dok se u slučaju jednokratnog alarma signal generira samo jednom, te se zatim alarm trajno deaktivira. Prema definiciji vremenskih odnosa alarm može biti **apsolutni**, odnosno odbrojavati vrijeme u apsolutnom vremenskom intervalu koji mu je zadan, ili **relativni**, gdje alarm odbrojava vrijeme u odnosu na neki drugi vremenski sklop, npr. vrijeme do pojave određene vrijednosti sklopovskog brojača vremena.

Osnovno sučelje prema programskim alarmima predstavljaju sljedeće funkcije:

```
int timer_create(clockid_t clockid, struct sigevent *restrict evp,
                 timer_t *restrict timer_id);

int timer_delete(timer_t *timer_id);

int timer_settime(timer_t timer_id, int flags,
                  const struct itimerspec *new_setting,
                  struct itimerspec *old_setting);

int timer_gettime(timer_t timer_id, struct itimerspec *expires);

int timer_getoverrun(timer_t timer_id);
```

Ove funkcije koriste se za stvaranje, postavljanje i brisanje alarma. Funkcije za stvaranje i brisanje alarma dozvoljeno je pozivati isključivo u *init* i *cleanup* funkcijama modula, odnosno prilikom pokretanja i uklanjanja modula. Drugim riječima, nije dozvoljeno dinamičko stvaranje i brisanje alarma za vrijeme rada programa.

Definicije parametara alarma prosljeđuju se pomoću posebnih struktura. Prilikom stvaranja novog alarma definiraju se vrijednosti signala koji će alarm generirati, te se te vrijednosti prosljeđuju pomoću strukture `sigevent`:

```
struct sigevent {
      int sigev_notify;
      int sigev_signo;
      union sigval sigev_value;
}
```

Ova struktura sadrži tri vrijednosti, zastavicu `sigev_notify` kojom se definira vrsta obavijesti u trenutku isteka alarma, oznaku signala koji se generira, te jedinstvenu oznaku alarma za koji se definira događaj.

Druga važna struktura je `itimerspec`, koja se koristi za definiranje vrijednosti vremena alarma:

```
struct itimerspec{
      struct timespec it_interval;
      struct timespec it_value;
}
```

U ovu strukturu pohranjuje se vrijednost intervala `it_interval` (ako se alarm koristi u periodičkom načinu rada), odnosno vrijednost `it_value` (ako se alarm koristi u jednokratnom načinu rada). Svaka od ovih vrijednosti ima sljedeći oblik:

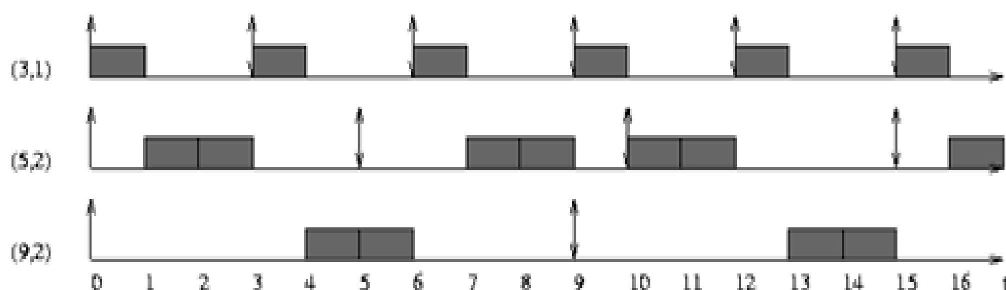
```
struct timespec {
      time_t tv_sec;
      long tv_nsec;
}
```

Budući da je potonja struktura naslijeđena izravno iz Linuxa, može uzrokovati probleme u slučaju da se dogodi preljev vrijednosti `tv_nsec` kod postavljanja vrijednosti alarma. To se može dogoditi ako se u alarm pokuša izravno zapisati vrijednost vremena sklopovskog brojača, koji poprima 64-bitne vrijednosti (*`hrtime_t`*). U RTLinuxu postoji i struktura `rtl_itimerspec` koja sadrži `hrtime_t` vrijednosti za zapis vremena, ali funkcije za upravljanje alarmima nisu prilagođene za rad sa navedenom strukturom.

3. Ostvarenje EDF raspoređivanja u RTLinuxu

3.1.Princip rada EDF raspoređivača

Dinamičko raspoređivanje prema trenucima krajnjih završetaka (EDF) određuje prioritete zadataka dinamički, prema trenutku krajnjeg završetka izvođenja zadatka (u nastavku TKZ). U trenutku pojave nekog događaja vezanog uz raspoređivanje (zadatak završio sa radom, dodavanje novog zadatka u red čekanja,...), pretražuje se red pripravnih zadataka i pronalazi se zadatak koji ima najbliži trenutak krajnjeg završetka. Tom zadatku se dodjeljuje najveći prioritet i on će biti izveden sljedeći.



Slika 2: Raspoređivanje zadataka prema trenucima krajnjih završetaka

RTLinux u svojoj osnovnoj inačici koristi samo statičko raspoređivanje prema prioritetima, pa je za ostvarenje dinamičkog raspoređivača potrebno izmjeniti postojeće strukture i funkcije raspoređivača, te dodati neke nove.

3.2. Ostvarenje raspoređivača u programskom kodu sustava

EDF raspoređivanje u RTLinuxu ostvareno je izmjenom programskog koda operacijskog sustava, odnosno raspoređivačkog modula sustava. Osnovni koncept ostvarenja bio je da se novi način raspoređivanja može opcionalno uključiti prilikom podešavanja RTLinux jezgre, te da standardni, statički raspoređivač prema prioritetima, bude u potpunosti neovisan o izmjenama unutar jezgre. Opcija za uključivanje EDF raspoređivača ostvarena je izmjenom datoteke `/scripts/config.in`, pomoću koje je dodana nova opcija u izbornik za podešavanje sustava. Odabirom spomenute opcije generirat će se zastavica `CONFIG_EDF` u datoteci `/include/rtl_conf.h`, čija će se vrijednost provjeravati u pretprocesorskim naredbama u kodu. Sve izmjene koda potrebne za ostvarenje EDF algoritma raspoređivanja smještene su u izvornom kodu unutar `#ifdef` blokova, te se prevode i izvode u slučaju kada je postavljena zastavica `CONFIG_RTL_EDF`.

Za ostvarenje EDF raspoređivanja potrebno je napraviti izmjene u raspoređivačkom modulu sustava, preciznije u samo dvije datoteke, `/schedulers/rtl_sched.c` i `include/rtl_sched.h`. Ostvarene promjene ne utječu niti na jedan drugi dio jezgre osim raspoređivačkog modula, te je na taj način očuvan integritet i neovisnost ostatka operacijskog sustava.

U datoteci `rtl_sched.h` definirana je nova raspoređivačka politika, `SCHED_EDF`, te je dodana nova struktura, `rtl_sched_param`, koja uz statički zadan prioritet dretve sadrži i novi parametar raspoređivanja, `hrtime_t sched_deadline`. Struktura `sched_param` u standardnom raspoređivaču sadrži samo prioritet dretve, ali nije izravno mjenjana zbog sukladnosti sa statičkim raspoređivačem.

U opisnik dretve dodani su novi parametri, `relative_deadline` i `policy`. Također, i u opisnik atributa dretve dodan je novi parametar *`policy`*, budući da je sada moguće koristiti više od jednog načina raspoređivanja. Ovaj parametar trenutno nema bitnu funkciju, ali u daljem razvoju sustava može se koristiti za definiranje politike raspoređivanja, te tako zamijeniti pretprocesorsku zastavicu u slučaju da dinamički raspoređivač bude potpuno integriran u operacijski sustav.

U funkciji `thread_attr_init()` potrebno je inicijalizirati nove atribute dretve. Inicijalna vrijednost atributa `attr->sched_param.sched_deadline` postavlja se na `HRTIME_INFINITY`, što je maksimalna vrijednost koju može poprimiti 64-bitni tip podatka `hrtime_t`, kako bi svaka nova dretva prilikom stvaranja imala najniži prioritet.

Za ostvarenje funkcionalnosti EDF raspoređivača izmjenjene su funkcije raspoređivača za postavljanje i dohvat parametara raspoređivanja u opisniku dretve i opisniku atributa:

```
extern inline int pthread_setschedparam(pthread_t thread, int policy,
const struct rtl_sched_param *param)

extern inline int pthread_getschedparam(pthread_t thread, int *policy,
struct rtl_sched_param *param)

extern inline int pthread_attr_setschedparam(pthread_attr_t *attr, const
struct rtl_sched_param *param)

extern inline int pthread_attr_getschedparam(const pthread_attr_t *attr,
struct rtl_sched_param *param)
```

Dodane su nove funkcije za odabir i postavljanje raspoređivačke politike, kao i funkcije za dohvat i postavljanje vrijednosti TKZ iz opisnika dretve i opisnika atributa:

```
extern inline int pthread_attr_setschedpolicy(pthread_attr_t *attr, int
policy)

extern inline int pthread_attr_getschedpolicy(pthread_attr_t *attr, int
*policy)

extern inline int pthread_setdeadline(pthread_t thread, hrtime_t
deadline)

extern inline int pthread_getdeadline(pthread_t thread, hrtime_t
*deadline)

extern inline int pthread_attr_setdeadline(pthread_attr_t *attr, hrtime_t
deadline)

extern inline int pthread_attr_getdeadline(pthread_attr_t *attr, hrtime_t
*deadline)
```

Nakon što su definirane i pripremljene sve potrebne strukture i sučelja u raspoređivaču, moguće je ostvariti funkcionalnost dinamičkog raspoređivača u sustavu. Ovo se postiže izmjenama u datoteci `rtl_sched.c`.

Najvažnija promjena funkcionalnosti raspoređivača sadržana je u tome što je dodana nova funkcija `cmp_prio()`, koja se koristi za određivanje sljedeće dretve koja će se izvoditi. Naime, kod statičkog raspoređivača ovo je ostvareno izravnom usporedbom prioriteta dodijeljenih dretvama unutar funkcije `rtl_schedule()`. Međutim, u novom ostvarenju prvo se uspoređuju relativne vrijednosti krajnjih završetaka dretvi, te se tako odabire ona sa najbližim TKZ. Ova funkcija sukladna je sa statičkim raspoređivačem, jer se prvo uspoređuju statički prioriteti dretvi, a budući da se oni ne postavljaju kod EDF raspoređivanja, ovo ne utječe na odabir dretve kod dinamičkog raspoređivanja. Ukoliko bi neka dretva ipak imala zadan statički prioritet, tada bi ona uvijek imala prednost izvođenja te bi u tom slučaju raspoređivanje zadataka bilo ostvareno na mješoviti način. Ova funkcija poziva se iz najvažnije funkcije raspoređivača. Prva takva funkcija je `rtl_schedule()`, koja se izvodi kod svakog događaja u sustavu (pojava nove dretve, završetak rada aktivne dretve...), te implementira funkcionalnost raspoređivanja zadataka.

U funkciji `pthread_make_periodic_np()` dodan je odsječak koda za izračun i dodjelu vrijednosti relativnog TKZ, koja se računa prema sljedećoj formuli:

```
p->relative_deadline = start_time + (p)->sched_param.sched_deadline
```

U funkciji jezgre `__pthread_create()` dodano je postavljanje inicijalnih vrijednosti atributa `task->policy` i relativnog i apsolutnog TKZ kod stvaranja nove

dretve. Za potrebe dinamičkog raspoređivača izmjenjena je i funkcija pthread_wait_np(). U osnovnoj izvedbi ova funkcija privremeno zaustavlja izvođenje dretve, postavlja novu vrijednost trenutka ponovnog pojavljivanja, te poziva funkciju rtl_schedule() kako bi se procesorsko vrijeme dodijelilo drugoj dretvi na čekanju. U izmjenjenoj izvedbi ovoj funkciji dodan je odsječak koda za računanje vrijednosti relativnog TKZ dretve. Naime, kada dretva ode na čekanje (zadatak je obavljen do kraja), postavlja se nova vrijednost relativnog TKZ u idućem periodu rada, što se postiže uvećanjem relativnog TKZ dretve za vrijednost trajanja perioda izvođenja.

Konačna promjena, kojom se ostvaruje željena funkcionalnost, napravljena je u funkciji init_module(). U ovoj funkciji dodjeljuje se memorija glavnoj dretvi osnovne Linux jezgre, te se inicijalne vrijednosti TKZ za tu dretvu postavljaju na HRTIME_INFINITY. Tako se postiže da dretva operacijskog sustava uvijek ima najniži prioritet izvođenja, te tako može uvijek biti prekinuta od strane bilo kojeg RTLinux procesa.

Na ovaj način ostvaren je potpuno funkcionalan dinamički raspoređivač zadataka, uz minimalne promjene u operacijskom sustavu, i uz zadržavanje funkcionalnosti statičkog raspoređivača, budući da je kod stvaranja nove dretve jedino potrebno preko parametra odabrati način raspoređivanja.

4. Proširenje raspoređivača dodatnim mogućnostima upravljanja zadacima

Prilikom ostvarivanja opisane funkcionalnosti raspoređivanja zadataka pojavljivali su se određeni problemi i poteškoće vezane uz sam operacijski sustav. U poglavlju 2.1. spomenuto je da RTLinux ne posjeduje nikakve zaštitne mehanizme, te da je zadaća programera da se pobrine za ispravnost koda i da predvidi sve moguće probleme koji bi mogli dovesti do neispravnosti rada sustava. Također, postupak otklanjanja pogrešaka prilikom implementacije i testiranja novog raspoređivača dugotrajan je i težak, budući da RTLinux ne posjeduje mehanizam otkrivanja funkcionalnih pogrešaka ili obavještavanja o istima, što može uzrokovati nepredviđeno ponašanje i rezultirati potpunim zastojem sustava.

Nadalje, prilikom programiranja neke greške je vrlo teško pronaći, jer se može dogoditi da sustav neko vrijeme radi ispravno, te se potom dogodi potpuni zastoj. Iscrpno testiranje pokazalo je da se takvo ponašanje sustava očituje kada je vrijeme rada dretve blisko ili veće od perioda dretve. Naime, ako dretva ima vrijeme izvođenja duže od perioda, njoj se dodjeljuje 100% procesorskog vremena, te se ta dretva ne može prekinuti u radu. Nakon završetka rada, periodička dretva postavlja novo vrijeme ponovnog javljanja (parametar `resume time`) tako što prethodnu vrijednost uveća za vrijednost atributa `period`. Ako je taj trenutak već prošao (dretva je prekoračila vrijeme perioda), prilikom poziva raspoređivača procesorsko vrijeme se ponovno dodjeljuje istoj dretvi. Ovakvo ponašanje rezultira beskonačnim ponavljanjem iste dretve, što se očituje prividnim potpunim zastojem sustava, budući da će izvođenje dretve operacijskog sustava biti potpuno blokirano.

Kako bi se navedeni problemi mogli uočiti i ispraviti, ostvareni su dodatni mehanizmi upravljanja radom dretvi i pronalaženja pogrešaka u radu. Kao i kod novog raspoređivačkog mehanizma, i ovdje su korištene pretprocesorske naredbe za provjeru zastavica postavljenih prilikom podešavanja RTLinux jezgre, kako bi se nove funkcionalnosti odvojile od ostatka sustava i po potrebi mogle uključivati i isključivati.

4.1. Podešavanje ponašanja dretve u slučaju pogreške

Za ostvarenje naprednih mogućnosti upravljanja ponašanjem dretve u slučaju pogreške korištena je podrška za alarme i signale prema POSIX standardu. Osnovno sučelje za upravljanje alarmima opisano je u poglavlju 2.3.

Podešavanje ponašanja dretve u slučaju prekoračenja vremena krajnjeg završetka zamišljena je tako da se željena akcija odabire prilikom stvaranja periodičke dretve. U tu svrhu dodana je funkcija `make_periodic()`, koja za razliku od standardne funkcije `pthread_make_periodic_np()` prima dodatni parametar, `int alarm_action`. Funkcija postavlja istoimeni parametar definiran u opisniku dretve, te potom poziva funkciju `pthread_make_periodic_np()`. Vrijednost zadanog parametra određuje akciju koja se izvodi u slučaju prekoračenja vremena krajnjeg završetka aktivne dretve, te može poprimiti sljedeće vrijednosti definirane u datoteci `schedulers/rtl_sched.h`:

```
#define ALARM_NOACTION 0
#define ALARM_NOTIFICATION  1
#define ALARM_STOP_THREAD 2
#define ALARM_TRY_UNTIL_PERIOD 3
#define ALARM_STOP_SYSTEM 4
#define ALARM_RUN_UNTIL_PERIOD 5
```

Prilikom poziva funkcije `make_periodic()` moguće je zadati prvih pet definiranih vrijednosti zastavice `alarm_action`, dok se vrijednost `ALARM_RUN_UNTIL_PERIOD` postavlja unutar raspoređivača i ima posebnu namjenu. Ponašanje dretve u ovisnosti o vrijednostima ove zastavice opisana je u nastavku rada.

U opisnik dretve dodan je pokazivač na strukturu alarma `struct time_t *alarm_timer` . Ova struktura stvara se za svaku dretvu prilikom stvaranja dretve u funkciji `pthread_create()`, te se koristi za dojavljivanje o prekoračenju vremena krajnjeg završetka. Uz zastavicu `alarm_action` ovo je jedina izmjena u opisniku dretve potrebna za ostvarenje ove funkcionalnosti. Prilikom uklanjanja dretve iz sustava, u funkciji `pthread_delete_np()` prvo se zaustavlja pripadajući alarm za dretvu, te se potom oslobađa iz memorije.

Postavljanje i deaktivacija alarma izvodi se pomoću funkcije alarm_settime(). Ova funkcija je zapravo izmjena funkcije timer_settime(), potpuno jednake funkcionalnosti, osim što za ulazne parametre koristi pokazivače na strukturu rtl_itimerspec, koja sadrži 64-bitne hrtime_t vrijednosti. Standardne funkcije rtl_gettime() i rtl_settime() primaju kao ulazni parametar pokazivač na strukturu itimerspec, koja sadrži vrijednosti vremena i intervala tipa long i time_t. Budući da se u alarme izravno prosljeđuju hrtime_t vrijednosti iz opisnika dretve, ovakva izvedba omogućava izravan upis vremenskih vrijednosti bez potrebom za pretvorbom vremena iz jednog u drugi format, te ovakav pristup optimizira i ubrzava proces postavljanja i zaustavljanja alarma. Također postoji dokumentirani problem vezan uz funkciju rtl_settime() u slučaju preljeva kod uvećanja parametra it_value na vrijednost veću od dvije sekunde, što može dovesti do nepredviđenog ponašanja sustava. Korištenjem hrtime_t tipa podataka za zapis vremena otklanja se ovaj problem. Sukladno tome, dodana je i funkcija alarm_gettime() koja se koristi za dohvat trenutne vrijednosti alarma. Kao i funkcija za postavljanje alarma, i ova funkcija je dobivena izmjenama standardne funkcije timer_gettime() kako bi podržavala rad sa hrtime_t vremenskim strukturama.

Prilikom svake promjene aktivne dretve, u glavnoj funkciji raspoređivača rtl_schedule() postavlja se alarm sa zadanom vrijednošću TKZ. Dakle, kada neka druga dretva dobije procesorsko vrijeme uslijed nekog događaja raspoređivača (završetak rada trenutne dretve, pojava nove dretve u redu pripravnih), tada se vrijednost njezinog alarma postavlja na vrijeme krajnjeg završetka te se u tom trenutku aktivira njezin alarm. Za signaliziranje isteka vremena koristi se korisnički prekidni signal SIGALRM. Prekidna akcija na pojavu signala uvijek se definira za tekuću dretvu prije postavljanja alarma.

Ako dretva završi sa radom prije isteka vremena krajnjeg završetka, poziva se funkcija pthread_wait_np() te se deaktivira alarm za aktivnu dretvu. Deaktivacija se vrši jednostavnim postavljanjem vrijednosti alarma na nulu. U ovoj funkciji odvijaju se i dodatne radnje za određene postavke ponašanja dretve, koje će biti podrobnije opisane u nastavku poglavlja.

U slučaju da se dretva ne izvede do trenutka isteka vremena krajnjeg završetka, dolazi do pojave signala SIGALRM te se izvođenje dretve prekida i poziva se funkcija alarm_handler(). Ova funkcija određuje ponašanje dretve u ovisnosti o vrijednosti

zastavice alarm_action u opisniku dretve. Mogući su sljedeći obrasci ponašanja za zadane vrijednosti zastavice:

1)ALARM_NOACTION – dretva nastavlja s radom. Ova vrijednost zastavice simulira standardno ponašanje dretve i nije sigurna za korištenje, jer će u slučaju prekoračenja perioda doći do prividnog potpunog zastoja sustava, tj. aktivna dretva nastavit će se izvoditi u beskonačnoj petlji. Ova zastavica postoji isključivo kako bi se minimalizirala potreba za promjenama u programima kada se želi ispitati ponašanje u standardnim uvjetima rada sustava. Za postupak ispitivanja funkcionalnih pogrešaka u sustavu ne preporučuje se korištenje ovog načina rada.

2)ALARM_NOTIFICATION – u trenutku prekoračenja vremena krajnjeg završetka ispisat će se obavijest o grešci te će aktivna dretva nastaviti sa izvođenjem. Kada dretva konačno završi sa izvođenjem, u funkciji pthread_wait_np() ispitat će se koliko perioda rada je dretva potrošila na izvođenje, te će se u naredbenom prozoru ispisati obavijest. Vrijednost trenutka ponovnog javljanja dretva postavlja se na sljedeća vrijednost perioda u budućnosti, te se sukladno tome određuje i novo vrijeme krajnjeg završetka dretve. Ovaj način rada iznimno je koristan u slučaju da su vrijeme krajnjeg završetka i vrijeme perioda vrlo blizu, te se otklanja mogućnost potpunog zastoja u slučaju da dretva prekorači vrijeme perioda.

3)ALARM_STOP_THREAD – u slučaju prekoračenja vremena krajnjeg završetka dretva se zaustavlja te se u potpunosti uklanja iz sustava pozivom funkcije pthread_exit(). Nakon uklanjanja dretve iz sustava poziva se raspoređivač te se procesorsko vrijeme dodjeljuje sljedećoj dretvi najvećeg prioriteta. Iako ovakvim obrascem ponašanja sustav nastavlja s radom, potrebno je voditi računa o tome je li aktivna dretva u trenutku prekida pristupala zajedničkim resursima, budući da dretva prilikom uklanjanja iz sustava ne oslobađa resurse. Zbog toga je pri korištenju ovog obrasca ponašanja važno voditi računa i o utjecaju na druge zadatke sustava, te o obrascu ponašanja za te zadatke.

4)ALARM_TRY_UNTIL_PERIOD – u trenutku prekoračenja vremena krajnjeg završetka dretvi se postavlja vrijednost zastavice alarm_action na vrijednost ALARM_RUN_UNTIL_PERIOD, alarm se postavlja na trenutak isteka perioda, te dretva nastavlja s radom. Time se trenutak krajnjeg završetka dretve privremeno postavlja na vrijeme isteka perioda. Ako se dretva ne uspije završiti do isteka perioda, zaustavlja se i uklanja iz sustava, te se ispisuje obavijest o ovom događaju. Ovaj obrazac ponašanja zahtijeva promjene i u drugim dijelovima raspoređivačkog modula, ponajprije u funkciji

cmp_prio(), gdje se provjerava vrijednost zastavice alarm_action za svaku od dretvi čija se vremena krajnjeg završetka uspoređuju, te se sukladno tome odabire dretva koja će se sljedeća izvoditi. Također, u funkciji rtl_schedule(), prilikom zamjene konteksta za dretvom kojoj je zastavica postavljena na ovu vrijednost, postavlja se vrijednost alarma na vrijeme isteka perioda umjesto na vrijeme krajnjeg završetka. Ako dretva ipak uspije završiti s radom prije isteka perioda, u funkciji pthread_wait_np() vrijednost zastavice ponovno se postavlja na vrijednost ALARM_TRY_UNTIL_PERIOD, te se na ekran ispisuje obavijest. Ovaj obrazac ponašanja koristan je kada se ispituje rad dretve u situacijama gdje zbog nekog mogućeg zastoja u radu postoji mogućnost da izvođenje dretve traje nešto duže od vremena krajnjeg završetka, ali ta odgoda nije vremenski kritična u odnosu na neki drugi zadatak.

5)ALARM_STOP_SYSTEM – sve RT dretve se zaustavljaju u slučaju zastoja trenutne dretve. Nakon ispisa obavijesti poziva se funkcija stop_all_rt_tasks(), koja će prvo zaustaviti izvođenje svih dretvi koje čekaju u redu pripravnih na procesoru koji je zauzet aktivnom dretvom, a zatim će njihovo izvođenje biti trajno zaustavljeno. Budući da nije moguće sve dretve ukloniti iz sustava u kontekstu trenutne dretve (jer se uklanjanjem bilo koje dretve poziva raspoređivač i procesor se dodjeljuje nekoj drugoj pripravnoj dretvi), svim dretvama osim aktivne se trajno zaustavlja izvođenje te one postaju tzv. "*zombie*" dretve, odnosno dretve koje se ne izvode, ali i dalje su prisutne u sustavu. Nakon toga se dretva koja je izazvala zastoj potpuno uklanja iz sustava. Važno je napomenuti kako je ova funkcionalnost ostvarena i testirana za jednoprocesorske sustave, dok je za višeprocesorske sustave potrebno u budućnosti dodatno izmjeniti funkciju stop_all_rt_tasks(). Opisani obrazac ponašanja primjenjiv je kada se želi točno odrediti dretva koja izaziva probleme u sustavu, ili se žele zaštititi vitalni resursi sustava u slučaju kritične pogreške.

Ostvarenjem opisanih funkcionalnosti višestruko je olakšan razvoj aplikacija za RTLinux, jer se omogućava preciznije određivanje pogrešaka u radu modula, te se povećava ukupna pouzdanost sustava u radu. Izvorni kod ostvarenih funkcionalnosti nalazi se na mediju priloženom uz rad.

4.2. Otkrivanje pogrešaka u raspoređivačkom podsustavu

Prilikom ostvarivanja dinamičkog raspoređivača i naprednih mehanizama upravljanja dretvama, često su se pojavljivale funkcionalne pogreške kojima je teško

odrediti uzrok ili točno mjesto pojave greške. Opisani problemi naročito su bili izraženi prilikom testiranja mehanizama upravljanja radom dretvi, budući da kašnjenje drevi i potencijalno prekoračenje perioda, odnosno vremena krajnjeg završetka utječe na čitav niz drugih događaja. RTLinux ne posjeduje mehanizme za otkrivanje funkcionalnih pogrešaka, kao ni mogućnost ispisa obavijesti o procesima sustava.

Prilikom razvoja novih funkcionalnosti u programski kod dodane su brojne provjere uvjeta, povratnih vrijednosti pojedinih funkcija i događaja u raspoređivačkom sustavu. Za svaku provjeru dodan je i ispis obavijesti o uspjehu, odnosno neuspjehu izvođenja određene radnje. Nakon uspješnog ostvarenja sustava ove provjere i ispisi zadržani su u programskom kodu, kako bi se mogle koristiti za detaljno testiranje novih funkcionalnosti, ali i upotrijebiti u svrhe daljeg razvoja sustava. Kao i kod prethodnih izmjena, i ovdje su korištene pretprocesorske naredbe za opcionalno uključivanje ove mogućnosti prilikom podešavanja sustava.

Kontrolni ispisi koriste se prilikom stvaranja nove dretve ili brisanja dretve, gdje se provjerava uspješnost stvaranja, odnosno brisanja strukture alarma. Također se provjerava uspješnost postavljanja alarma u funkcijama `pthread_wait_np()` i `rtl_schedule()`, kako bi se provjerila ispravnost aktiviranja alarma za pojedine dretve. U funkciji `rtl_schedule()` dodani su kontrolni ispisi prilikom zamjene konteksta aktivne dretve u procesoru, kako bi se omogućilo praćenje procesa rada raspoređivača. U kombinaciji sa opcijom ispisa poruka uslijed pogreške u radu dretve, ovo pruža dobar uvid u stanje sustava i rad raspoređivača. Tako je moguće pratiti što se događa u slučaju zamjene konteksta dretve, u kojim situacijama se aktivira Linux dretva, što se događa u raspoređivaču prilikom uklanjanja dretve uslijed pogreške itd. Time je ostvaren bolji uvid u funkcije raspoređivača, te je omogućen lakši razvoj novih poboljšanja u budućnosti.

Važno je napomenuti da je za korištenje opisanih dodatnih mogućnosti upravljanja prilikom podešavanja jezgre potrebno omogućiti podršku za POSIX signale i alarme, budući da nove mogućnosti ovise o navedenoj podršci.

5. Demonstracija rada ostvarenih funkcionalnosti

U svrhu demonstracije rada ostvaren je modul za testiranje, čiji se izvorni kod nalazi u privitku rada. Pomoću navedenog modula moguće je prikazati sve ostvarene funkcionalnosti, što se postiže jednostavnim promjenama parametara dretvi u izvornom kodu modula.

Ostvareni EDF raspoređivač testiran je pomoću modula koji periodički pokreće četiri dretve istog statičkog prioriteta, ali sa različitim periodima pojavljivanja, trenucima krajnjih završetaka i vremenima trajanja izvođenja. Nakon što je uspješno testirana osnovna funkcionalnost raspoređivača (raspoređivanje, *preempting*), program je proširen dodavanjem dijeljenih resursa u vidu mehanizama međusobnog isključivanja (*mutex*). Po dvije dretve dijele zajedničke resurse, te svaka dretva obavlja 60% svog posla u zaštićenom prostoru. RTLinux za upravljanje dijeljenim resursima koristi protokol stropnog prioriteta (Priority Ceiling Protocol , PCP), te se ovo rješenje pokazalo učinkovitim za obavljanje danog zadatka. Korištenje naprednih mogućnosti upravljanja zadacima za pojedine dretve moguće je koristiti promjenom vrijednosti zastavice `alarm_action` za svaku dretvu, koja se postavlja pozivom funkcije `make_periodic()`. Za svaku dretvu moguće je u za to predviđenim strukturama postaviti proizvoljne vrijednosti trajanja perioda, trenutka krajnjeg završetka, radnog opterećenja i načina rada.

5.1. Prikaz rada EDF raspoređivača

Osnovni rad raspoređivača prikazan je pomoću modula za testiranje uz sljedeće parametre:

```
int compute_time[4] =  {1500000, 3000000,  4500000, 6000000};
hrtime_t deadline[4] = {70000000,30000000, 100000000, 300000000};
hrtime_t period[4] =   {90000000,150000000,170000000, 610000000};
int flag[4] = {0,0,0,0};
```

Izvođenje modula za testiranje prikazano je u vremenskom rasponu od približno 500 ms. Pritom su prikazane sve važne funkcionalnosti raspoređivača, raspoređivanje prema vremenima krajnjih završetaka, kao i prekidanje trenutne dretve usred pojave druge dretve sa bližim vremenom krajnjeg završetka:

```
(...)

Start thread 0xDF1AC200(tid = 2), time = 520796, deadline = 620680
Thread 0xDF1AC200(tid = 2): LOCK Mutex 0
Thread 0xDF1AC200(tid = 2): UNLOCK Mutex 0
End thread 0xDF1AC200(tid = 2), time = 526237
Start thread 0xDF1AC800(tid = 0), time = 570232, deadline = 639526
Start thread 0xDF1AC600(tid = 1), time = 570634, deadline = 600381
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 573787
Thread 0xDF1AC800(tid = 0): LOCK Mutex 0
Thread 0xDF1AC800(tid = 0): UNLOCK Mutex 0
End thread 0xDF1AC800(tid = 0), time = 576084
Start thread 0xDF1AC800(tid = 0), time = 659655, deadline = 729526
Thread 0xDF1AC800(tid = 0): LOCK Mutex 0
Thread 0xDF1AC800(tid = 0): UNLOCK Mutex 0
End thread 0xDF1AC800(tid = 0), time = 661371
Start thread 0xDF1AC200(tid = 2), time = 690803, deadline = 790680
Thread 0xDF1AC200(tid = 2): LOCK Mutex 0
Thread 0xDF1AC200(tid = 2): UNLOCK Mutex 0
End thread 0xDF1AC200(tid = 2), time = 696221
Start thread 0xDF1AC600(tid = 1), time = 720502, deadline = 750381
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 723943
Start thread 0xDF1AC800(tid = 0), time = 749658, deadline = 819526
Thread 0xDF1AC800(tid = 0): LOCK Mutex 0
Thread 0xDF1AC800(tid = 0): UNLOCK Mutex 0
End thread 0xDF1AC800(tid = 0), time = 751438
Start thread 0xDF1AC800(tid = 0), time = 839650, deadline = 909526
Thread 0xDF1AC800(tid = 0): LOCK Mutex 0
Thread 0xDF1AC800(tid = 0): UNLOCK Mutex 0
End thread 0xDF1AC800(tid = 0), time = 840805
Start thread 0xDF1AC200(tid = 2), time = 860803, deadline = 960680
Thread 0xDF1AC200(tid = 2): LOCK Mutex 0
Thread 0xDF1AC200(tid = 2): UNLOCK Mutex 0
End thread 0xDF1AC200(tid = 2), time = 865717
Start thread 0xDF1AC600(tid = 1), time = 870702, deadline = 900381
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 874228
Start thread 0xDF1ACA00(tid = 3), time = 890981, deadline = 1190856
Thread 0xDF1ACA00(tid = 3): LOCK Mutex 1
Thread 0xDF1ACA00(tid = 3): UNLOCK Mutex 1
End thread 0xDF1ACA00(tid = 3), time = 897940
Start thread 0xDF1AC800(tid = 0), time = 929648, deadline = 999526
Thread 0xDF1AC800(tid = 0): LOCK Mutex 0
Thread 0xDF1AC800(tid = 0): UNLOCK Mutex 0
End thread 0xDF1AC800(tid = 0), time = 931022
Start thread 0xDF1AC800(tid = 0), time = 1019937, deadline = 1089526
Thread 0xDF1AC800(tid = 0): LOCK Mutex 0
Start thread 0xDF1AC600(tid = 1), time = 1020517, deadline = 1050381
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 1025136
Thread 0xDF1AC800(tid = 0): UNLOCK Mutex 0
End thread 0xDF1AC800(tid = 0), time = 1034891

(...)
```

5.2. Korištenje naprednih mogućnosti upravljanja zadacima

Za demonstraciju naprednih mogućnosti upravljanja odabrani su sljedeći parametri:

```
int compute_time[4] =  {15000000, 30000000, 45000000,  60000000};
hrtime_t deadline[4] = {70000000, 90000000, 100000000, 100000000};
hrtime_t period[4] =   {90000000, 150000000,170000000, 310000000};
int flag[4] = {1,3,1,2};
```

Ovaj primjer demonstrira nekoliko različitih mogućnosti upravljanja. Dretve 0 i 2 podešene su tako da u slučaju pogreške ispisuju obavijest, dretva 1 u slučaju prekoračenja TKZ ispisuje obavijest i pokušava završiti sa radom do kraja perioda, dok se dretva 3 trenutno zaustavlja u slučaju prekoračenja. Svim dretvama vrijeme izvođenja je povećano deset puta kako bi se stvorili uvjeti za pojavu pogrešaka. Sljedeći odsječak ispisa programa prikazuje rad raspoređivača u navedenim uvjetima:

```
(...)

Start thread 0xDF1ACA00(tid = 0), time = 1231732, deadline = 1301478
Thread 0xDF1ACA00(tid = 0): LOCK Mutex 0
Thread 0xDF1ACA00(tid = 0): UNLOCK Mutex 0
End thread 0xDF1ACA00(tid = 0), time = 1250120
Start thread 0xDF1AC200(tid = 1), time = 1263118, deadline = 1352125
Thread 0xDF1AC200(tid = 1): LOCK Mutex 1
Thread 0xDF1AC200(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC200(tid = 1), time = 1302671
Start thread 0xDF1AC400(tid = 3), time = 1303625, deadline = 1372703
Thread 0xDF1AC400(tid = 3): LOCK Mutex 1
Thread 0xDF1AC400(tid = 3): UNLOCK Mutex 1

Thread 0xDF1AC400 deadline overrun! Ending thread...

Start thread 0xDF1ACA00(tid = 0), time = 1373852, deadline = 1391478
Thread 0xDF1ACA00(tid = 0): LOCK Mutex 0
Thread 0xDF1ACA00(tid = 0): UNLOCK Mutex 0

NOTIFICATION: Thread 0xDF1ACA00 deadline overrun!

End thread 0xDF1ACA00(tid = 0), time = 1392168


NOTIFICATION: Thread 0xDF1ACA00 missed deadline, but finished before
period overrun!
Start thread 0xDF1AC600(tid = 2), time = 1392775, deadline = 1402495

NOTIFICATION: Thread 0xDF1AC600 deadline overrun!

Thread 0xDF1AC600(tid = 2): LOCK Mutex 0

(...)
```

Potrebno je obratiti pozornost na to da zaustavljanje neke dretve uslijed pogreške ne oslobađa dijeljene resurse koje je dretva zauzela prije pojave pogreške, tako da oni ostaju nedostupni drugim dretvama koje ih koriste. Zbog toga je potrebno pažljivo odabrati način ponašanja za dretve koje koriste zajedničke resurse. Primjer lošeg odabira koji može dovesti do nestabilnog rada sustava ostvaren je uz sljedeće parametre:

```
int compute_time[4] =  {150000000, 30000000, 45000000,  60000000};
hrtime_t deadline[4] = {70000000, 90000000, 100000000, 300000000};
hrtime_t period[4] =   {90000000, 150000000,170000000, 610000000};
int flag[4] = {3,4,1,2};
```

Cilj ovog primjera je simulirati zastoj dretve 0 u kritičnom odsječku. Dretva 2 koristi isti *mutex*, ali je podešena tako da samo ispiše obavijest o pogrešci, te će zauvijek ostati u stanju čekanja na zajednički resurs:

```
(...)

Start thread 0xDF1AC200(tid = 0), time = 3090400, deadline = 3160272
Thread 0xDF1AC200(tid = 0): LOCK Mutex 0

Thread 0xDF1AC200 deadline overrun!
Thread 0xDF1AC200 will try to finish task until the end of period...


Thread 0xDF1AC200 period overrun! Ending thread...

Start thread 0xDF1AC600(tid = 1), time = 3181575, deadline = 3241047
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 3217857
Start thread 0xDF1AC400(tid = 2), time = 3218912, deadline = 3271447

NOTIFICATION: Thread 0xDF1AC400 deadline overrun!

Start thread 0xDF1AC600(tid = 1), time = 3301172, deadline = 3391047
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 3337910
Start thread 0xDF1AC600(tid = 1), time = 3451171, deadline = 3541047
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 3489430
Start thread 0xDF1AC600(tid = 1), time = 3601172, deadline = 3691047
Thread 0xDF1AC600(tid = 1): LOCK Mutex 1
Thread 0xDF1AC600(tid = 1): UNLOCK Mutex 1
End thread 0xDF1AC600(tid = 1), time = 3634502
Start thread 0xDF1ACA00(tid = 3), time = 3636504, deadline = 3911674
Thread 0xDF1ACA00(tid = 3): LOCK Mutex 1

(...)
```

5.3. Prikaz rada uz omogućene kontrolne ispise

Ako se prilikom podešavanja jezgre omoguće kontrolni ispisi, prilikom izvođenjanja modula za testiranje prikazivat će se dodatne informacije o procesima koji se odvijaju u raspoređivaču. Demonstracija ove funkcionalnosti provedena je sa istim parametrima kao u prethodnom primjeru, te prikazuje i informacije prilikom učitavanja i uklanjanja modula za testiranje iz jezgre:

```
<< SUCCESS >> __pthread_create(): created alarm timer for thread
0xDF124800
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124800
Switched to Linux thread...
<< SUCCESS >> __pthread_create(): created alarm timer for thread
0xDF124400
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124400
Switched to Linux thread...
<< SUCCESS >> __pthread_create(): created alarm timer for thread
0xDF124600
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124600
Switched to Linux thread...
<< SUCCESS >> __pthread_create(): created alarm timer for thread
0xDF124200
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124200
Switched to Linux thread...
Switched to RT thread 0xDF124800
Start thread 0xDF124800(tid = 0), time = 2696435, deadline = 2766293
Thread 0xDF124800(tid = 0): LOCK Mutex 0

Thread 0xDF124800 deadline overrun!
Thread 0xDF124800 will try to finish task until the end of period...


Thread 0xDF124800 period overrun! Ending thread...

Switched to RT thread 0xDF124400
Start thread 0xDF124400(tid = 1), time = 2787484, deadline = 2847029
Thread 0xDF124400(tid = 1): LOCK Mutex 1
Thread 0xDF124400(tid = 1): UNLOCK Mutex 1
End thread 0xDF124400(tid = 1), time = 2824327
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124400
Switched to RT thread 0xDF124600
Start thread 0xDF124600(tid = 2), time = 2824976, deadline = 2877494
Switched to Linux thread...
Switched to RT thread 0xDF124600

NOTIFICATION: Thread 0xDF124600 deadline overrun!

Switched to Linux thread...
Switched to RT thread 0xDF124400
Start thread 0xDF124400(tid = 1), time = 2907161, deadline = 2997029
Thread 0xDF124400(tid = 1): LOCK Mutex 1
Thread 0xDF124400(tid = 1): UNLOCK Mutex 1
End thread 0xDF124400(tid = 1), time = 2945965
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124400
Switched to Linux thread...
Switched to RT thread 0xDF124400
Start thread 0xDF124400(tid = 1), time = 3057166, deadline = 3147029
```

```
Thread 0xDF124400(tid = 1): LOCK Mutex 1
Thread 0xDF124400(tid = 1): UNLOCK Mutex 1
End thread 0xDF124400(tid = 1), time = 3094181
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124400
Switched to Linux thread...
Switched to RT thread 0xDF124400
Start thread 0xDF124400(tid = 1), time = 3207158, deadline = 3297029
Thread 0xDF124400(tid = 1): LOCK Mutex 1
Thread 0xDF124400(tid = 1): UNLOCK Mutex 1
End thread 0xDF124400(tid = 1), time = 3240319
<< SUCCESS >> wait_np() : alarm timer disarmed for thread 0xDF124400
Switched to RT thread 0xDF124200
Start thread 0xDF124200(tid = 3), time = 3242429, deadline = 3517756
Thread 0xDF124200(tid = 3): LOCK Mutex 1
Thread 0xDF124200(tid = 3): UNLOCK Mutex 1
Switched to Linux thread...
<< SUCCESS >> pthread_delete_np(): timer disarmed before deleting thread
0xDF124800
<< SUCCESS >> pthread_delete_np(): timer_deleted in thread 0xDF124800
Switched to RT thread 0xDF124800
RT thread 0xDF124800 ending...

Switched to Linux thread...
<< SUCCESS >> pthread_delete_np(): timer disarmed before deleting thread
0xDF124400
<< SUCCESS >> pthread_delete_np(): timer_deleted in thread 0xDF124400
Switched to RT thread 0xDF124400
RT thread 0xDF124400 ending...

Switched to Linux thread...
<< SUCCESS >> pthread_delete_np(): timer disarmed before deleting thread
0xDF124600
<< SUCCESS >> pthread_delete_np(): timer_deleted in thread 0xDF124600
Switched to RT thread 0xDF124600
RT thread 0xDF124600 ending...

Switched to Linux thread...
<< SUCCESS >> pthread_delete_np(): timer disarmed before deleting thread
0xDF124200
<< SUCCESS >> pthread_delete_np(): timer_deleted in thread 0xDF124200
Switched to RT thread 0xDF124200
RT thread 0xDF124200 ending...

Switched to Linux thread...
```

5.4. Završne napomene

Opisane funkcionalnosti ostvarene su i testirane u operacijskom sustavu Red Hat Linux 8.0 (Psyche). Za razvoj raspoređivača korištena je Linux jezgra verzije 2.4.18 i RTLinux jezgra verzije 3.2 pre-2. Zbog opsega i veličine programskog ostvarenja, izvorni kodovi nisu navedeni u privitku rada, već se nalaze na mediju priloženom uz rad.

6. Zaključak

Modularni pristup izgradnji jezgre operacijskog sustava RTLinux omogućava implementaciju novih funkcionalnosti na relativno brz i jednostavan način. EDF raspoređivač ostvaren je izmjenama samo dvije datoteke, bez potrebe za bilo kakvim promjenama u ostalim segmentima sustava. Ostvareni raspoređivač proširuje mogućnosti korištenja RTLinuxa, te je u skladu sa osnovnom idejom RTLinuxa – jednostavnost izvedbe doprinosi strogoj vremenskoj ispravnosti sustava. Detaljna ispitivanja novog raspoređivača pokazala su da EDF raspoređivač zadovoljava osnovne kriterije sustava. Vremena obrade raspoređivanja i odziva sustava nisu se bitno povećala dodavanjem nove raspoređivačke politike u sustav.

S druge strane, proširenja mogućnosti ponašanja dretve u slučaju pogreške mogu, za pojmove sustava sa strogim vremenskim ograničenjima, stvoriti značajno kašnjenje sustava, međutim to isključivo ovisi o specifikacijama RT sustava i zadataka koji se izvode u njemu. Osnovna namjena ovih proširenja je traženje i otklanjanje pogrešaka tijekom procesa oblikovanja RT sustava, te su se ove mogućnosti pokazale iznimno korisnima i bitno olakšavaju razvoj novih aplikacija. Unatoč tome, pri razvoju RT aplikacija nije dobro u potpunosti se osloniti na mehanizme pronalaženja pogrešaka, te je uvijek dobro i potrebno razmišljati unaprijed i pokušati predvidjeti potencijalne probleme prije nego što se uopće i dogode. Ovakav stav uvelike doprinosi pouzdanosti RT sustava, te dugoročno štedi vrijeme i novac.

Izvedba opisanog raspoređivača otvara nove mogućnosti daljeg razvoja i poboljšanja, kako raspoređivačkog sustava, tako i RTLinuxa u cjelini.

Potpis:

____

Popis korištene literature

[1] Yodaiken, V. *The RTLinux Manifesto*, znanstveni članak, Department of Computer Science, New Mexico Institute of Technology, 1997.

[2] Yodaiken, V. *An Introduction to RTLinux*, 19.11.1999. , preuzeto sa http://www.linuxdevices.com/articles/AT3694406595.html, 15.11.2011.

[3] Barabanov, M. *A Linux-Based Real-Time Operating system,* magistarski rad, New Mexico Institute of Mining and Technology, 1997.

[4] Hofrat, D.H. *Introducing RTLinux/GPL*, 2002.

[5] Elkayam, M. *Wind River's Real-Time Solutions*, Wind River, 2007.

[6] Sojka, M. *Case Studies for RT Linux*, 2. izdanje, Ocera, 2005.

[7] *Getting Started with RTLinux*, tehnički priručnik, FSM Labs Inc. , 2001.

[8] Yodaiken, V.: *The RT-Linux approach to hard real-time*, znanstveni članak, Department of Computer Science, New Mexico Institute of Technology, 1997.

[9] Kaski, S. *RealTime Linux: seminar on RTLinux*, 14.2.2001.

[10] Divarkan, D. *RTLinux HOWTO: RTLinux Installation and writing realtime programs in Linux*, 29.8.2002., preuzeto sa http://tldp.org/HOWTO/RTLinux-HOWTO.html, 17.11.2011.

[11] Balbastre, P, Ripoll, I: *Integrated Dynamic Priority Scheduler for RTLinux*, 1997.

[12] Sousa, P.B, Ferreira, L.L. *Implementing a new real-time scheduling policy for Linux,* Polytechnic Institute of Porto, 2010.

[13] Ming-Tsu Y. *A multi-level scheduler in Real-Time Linux*, doktorski rad, Department of Electrical Engineering, NCKU Taiwan, 2009.

[14] Hanssen, F.T.Y. *Scheduling and resource allocation with Real-Time Linux*, diplomski rad, preuzeto sa http://www.croky.net/publications/PDFs/hanssen-1998-01.pdf, Department of Computer Science, University of Twente, 30.10.1998.

[15] Vidal, J., Gonzálvez, F., Ripoll, I. *POSIX TIMERS implementation in RTLinux*, tehnička dokumentacija, 2002.

[16] Vidal, J., Gonzálvez, F., Ripoll, I. *POSIX Signals implementation in RTLinux*, tehnička dokumentacija, 2002.

[17] web stranice tvrtke FSMLabs, http://www.fsmlabs.com, 3.1.2012.

[18] web stranice RTLinuxa, http://www.rtlinuxfree.com/, 14.11.2011.

Popis slika

Slika 1: Arhitektura operacijskog sustava RTLinux.. 3
Slika 2: Raspoređivanje zadataka prema trenucima krajnjih završetaka............................. 11

Sažetak

Naslov: Ostvarenje EDF raspoređivanja za RTLinux

Sažetak: U ovom radu predstavljen je operacijski sustav RTLinux i njegovi mehanizmi raspoređivanja zadataka, te je dodatno ostvaren dinamički EDF raspoređivač. Jednostavnim izmjenama izvornog koda postojećeg statičkog raspoređivača prema prioritetima postignuta je potpuna funkcionalnost dinamičkog raspoređivanja. Pri ostvarenju dinamičkog raspoređivanja raspoređivački sustav proširen je tako da se omogući dinamičko raspoređivanje, ali je pritom zadržana i funkcionalnost statičkog raspoređivača. Uz novi način raspoređivanja ostvarene su i napredne mogućnosti upravljanja radom dretvi u slučaju pogreške, te nove mogućnosti pronalaženja i ispravljanja pogrešaka u radu sustava. Na taj način povećana je pouzdanost sustava te olakšan razvoj aplikacija za RTLinux.

Ključne riječi: RTLinux, operacijski sustavi, raspoređivanje zadataka, raspoređivanje prema trenucima krajnjih završetaka

Summary

Title: Implementation of the EDF scheduling for RTLinux

Summary: This paper presents the RTLinux operating system and its task scheduling mechanisms, and additionally describes new implementation of the Earliest-deadline first (EDF) scheduler. Dynamic scheduling functionality is thoroughly achieved through simple modifications of the existing static priority scheduler. Scheduling system has been extended in order to provide dynamic scheduling, but at the same time keeping the existing static scheduler functionality intact. In addition to the new scheduling policy, the advanced thread management options are implemented, together with new run-time error detection and correction mechanisms. These additional features increase the overall system reliability and allow easier development of RTLinux applications.

Keywords: RTLinux, operating systems, task scheduling, earliest-deadline scheduling

Privitak

Izvorni kod modula za testiranje raspoređivača

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>

MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("Hrvoje Knezevic");
MODULE_DESCRIPTION("Test module for EDF scheduler with configurable alarm
timers on multiple threads");

pthread_t thread[4];
pthread_mutex_t mutex[2];
struct sched_param data[4];

int x;
hrtime_t now;
int compute_time[4] =  {15000000, 30000000, 45000000,  60000000};
hrtime_t deadline[4] = {70000000, 90000000, 100000000, 300000000};
hrtime_t period[4] =   {90000000, 150000000,170000000, 610000000};
int flag[4] = {1,1,1,1};

void * start_routine(void *arg)
{
      int id = (int) arg;
      int tid = (int)pthread_self();
      int mut;
      int comp = compute_time[id];
      int work = 1, x = 0;
      mut = id % 2;
      hrtime_t start_time = gethrtime() + 100000000;

      make_periodic(pthread_self(), start_time , period[id], flag[id]);

      while (1)
      {
            pthread_wait_np ();

            rtl_printf("Start thread 0x%X(tid = %d), time = %d,
                     deadline = %d\n", tid, id,
                     (unsigned)(gethrtime())/1000,
                     (unsigned)(pthread_self()->relative_deadline)/1000);

            for(x=0;x<comp/5;x++)
            {
                  work+=work;
            }

            pthread_mutex_lock(&mutex[mut]);
            rtl_printf ("Thread 0x%X(tid = %d): LOCK Mutex %d\n",
                        tid, id, mut );

            for(x=0;x<3*comp/5;x++)
            {
                  work+=work;
```

```
			}

			pthread_mutex_unlock(&mutex[mut]);
			rtl_printf ("Thread 0x%X(tid = %d): UNLOCK Mutex %d\n",
					tid, id, mut );
			for(x=0;x<comp/5;x++)
			{
				work+=work;
			}

			rtl_printf("End thread 0x%X(tid = %d), time = %d\n",
					tid, id, (unsigned)(gethrtime())/1000);
		}
		return 0;
}

int init_module(void)
{
		pthread_attr_t attr;
		pthread_mutexattr_t mutexattr;

		for (x=0;x<4;x++)
		{
			//data[x].sched_priority = 0 ;
			data[x].sched_deadline = deadline[x];
		}

		pthread_mutexattr_init(&mutexattr);
		pthread_mutexattr_setprotocol(&mutexattr, PTHREAD_PRIO_PROTECT);

		for (x=0;x<2;x++)
		{
			pthread_mutex_init(&mutex[x], &mutexattr);
		}

		for (x=0;x<4;x++)
		{
			pthread_attr_init(&attr);
			pthread_attr_setschedparam(&attr, &data[x]);
			pthread_attr_setdeadline(&attr, deadline[x]);
			pthread_attr_setschedpolicy(&attr, SCHED_EDF);
			pthread_create (&(thread[x]), &attr,
					start_routine, (void *)x);
			pthread_attr_destroy(&attr);
		}
		return 0;
}

void cleanup_module(void)
{
		for (x=0;x<4;x++)
		{
			pthread_delete_np (thread[x]);
		}

		for (x=0;x<2;x++)
		{
			pthread_mutex_destroy(&mutex[x]);
		}
}
```