

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD

**Nadogradnja operacijskog sustava
FreeRTOS za primjenu u kontrolnim
aplikacijama**

Luka JengiĆ

Zagreb, lipanj 2022.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu \izvornik.

SADRŽAJ

1. Uvod	1
2. Operacijski sustavi za rad u stvarnom vremenu	3
2.1. Sustavi za rad u stvarnom vremenu	3
2.2. Periodični zadatci u sustavima za rad u stvarnom vremenu	4
2.3. Algoritmi za raspoređivanje zadataka	6
2.3.1. EDF algoritam	6
2.3.2. RTO algoritam	6
2.3.3. BWP algoritam	7
3. Modifikacija jezgre FreeRTOS-a	9
3.1. Operacijski sustav FreeRTOS	9
3.2. Programska potpora za kontrolu izvršavanja periodičnih zadataka	10
3.2.1. Stvaranje periodičnih zadataka	10
3.2.2. Kontrola izvršenja periodičnih zadataka	11
3.3. Implementacija algoritama za raspoređivanje zadataka	13
4. Implementacija simulatora	14
4.1. Generiranje setova testnih zadataka	14
4.2. Tijek simulacije	15
4.3. Pokretanje simulacije	15
5. Rezultati	17
6. Zaključak	18
6.1. First section	18
6.2. Second section	18
Bibliography	19

1. Uvod

Operacijski sustavi za rad u stvarnom vremenu (eng. Real Time Operating systems) sustavi su u kojima nije bitan samo rezultat operacije, nego je jednako važna i pravovremenost njezina izvršenja. Najvažnija svojstva koja takvi sustavi moraju zadovoljavati su pouzdanost i predvidivost u izvršavanju zadataka. Zbog navedenog svojstva sustavi za rad u stvarnom vremenu (u daljnjem tekstu SRSV) neizostavni su dio svih ugradbenih računalnih sustava u kojima postoje kritični poslovi čije neizvršavanje ili prekasno izvršavanje izaziva katastrofalne posljedice. Primjeri takvih sustava su automobili i ostala prometna sredstva, vojna industrija, multimedijски sustavi, roboti itd.

Ponekad SRSV zbog iznenadne pojave dodatnih zadataka mogu ući u stanje preopterećenja. To je stanje u kojemu procesor ne može izvršiti sve poslove na vrijeme i neke od njih mora preskočiti. Međutim, preskakanje zadataka ne smije biti nasumično, nego mora biti kontrolirano i predvidivo. Za primjer možemo uzeti robota koji ima sustav za izbjegavanje prepreka. Ako takav robot uđe u stanje preopterećenja, prioritet moraju dobiti zadatci koji su vezani uz detekciju prepreke i kontrolu motora kako bi se robot na vrijeme zaustavio ukoliko je to potrebno. Ostale funkcije koje robot obavlja manje su kritične te se mogu propustiti bez štete za sustav.

Potrebno je razmotriti rješenja koja bi osigurala determinističko ponašanje SRSV-a u uvjetima preopterećenja. Što je posao kritičniji, to treba dobiti veći prioritet pri raspoređivanju. Tako se izbjegava šteta koja bi potencijalno nastala propuštanjem takvih zadataka. Komercijalno dostupni SRSV-ovi ne osiguravaju sigurnost sustava kada je on u stanju preopterećenja. Stoga je ideja ovog rada modificirati SRSV otvorenog koda, za što je odabran FreeRTOS, kako bi se poslovi u stanju trajnog preopterećenja mogli kontrolirano propuštati. Modifikacijom jezgre FreeRTOS-a implementiran je strogi sustav za rad u stvarnom vremenu s ublaženim uvjetima (eng. weakly hard). Kod takvog sustava povremeno dopuštamo da se posao ne izvede, ali na način da se unutar određenog broja slijednih poslova samo jedan smije propustiti. Takvim pristupom preskakanja zadataka osigurava se da se niti jedan zadatak trajno ne blokira. Nadalje implementirati će se strategija prekidanja zadataka, prema kojoj se zadatci ne nastavljaju izvršavati nakon što su došli do krajnjeg roka za izvršavanje.

U literaturi se može pronaći mnogo algoritama za raspoređivanje zadataka. Neki od njih

bit će implementirani u raspoređivač zadataka FreeRTOS-a te kasnije uspoređivani nad istim setovima zadataka. Cilj je pronaći algoritam koji će u uvjetima preopterećenja osigurati determinističko ponašanje uz što veću kvalitetu usluge, to jest uz što više izvršenih poslova do njihovog krajnjeg roka završetka

2. Operacijski sustavi za rad u stvarnom vremenu

2.1. Sustavi za rad u stvarnom vremenu

Sustavi za rad u stvarnom vremenu (eng. Real Time System) danas su neizostavan dio mnogih sustava korištenih u svim granama ljudske djelatnosti. Kod njih nam nije bitan samo rezultat izvođenja operacije, nego je jednako važno i vrijeme u kojem se ta operacija izvede. Zbog toga se svakom zadatku pridjeljuje rok do kojeg se mora izvršiti. Izvršenje posla nakon njegovog krajnjeg roka za izvršavanje ekvivalentno je tome da se posao nije niti izvršio.

Kako bi sustav radio pouzdano mora se osigurati predvidivo raspoređivanje zadataka koje će osigurati da se što više zadataka izvrši na vrijeme. SRSV mora imati implementiranu kontrolu zadataka, kontrolu vremena, raspoređivač zadataka i sustav komunikacije i sinkronizacije među zadatcima.

S obzirom na posljedice koje izaziva propuštanje roka izvršavanja zadatke dijelimo u dvije skupine :

- Strogi zadatci (eng. hard real time tasks)
- Opcionalni zadatci (eng. soft real time tasks)

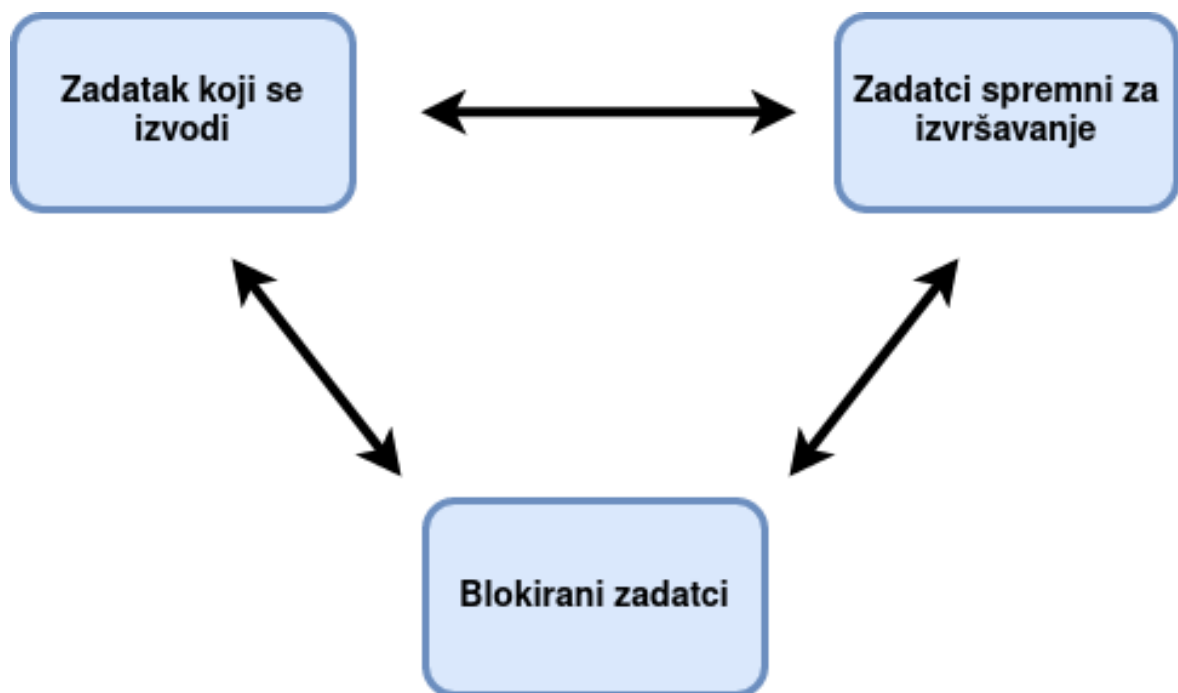
Strogi zadatci niti jednom ne smiju propustiti krajnji rok izvršavanja jer su to zadatci koji obavljaju važan posao. Njihovo propuštanje izaziva katastrofalne posljedice koje mogu biti pogubne za cijeli sustav. Dobar primjer strogog zadatka je detekcija pritiska papučice kočnice na automobilu. Taj zadatak je kritičan i propuštanje njegovog izvršenja za posljedicu može imati ozbiljnu nesreću s ljudskim žrtvama. S druge strane, propuštanje nekih zadataka nije kritično i neće nanijeti štetu sustavu, nego će smanjiti performanse ili mogućnosti sustava. Primjer ovakvog zadatka je paljenje signalne ledice ili prikaz podataka na pokazniku.

Zadatci u operacijskim sustavima za rad u stvarnom vremenu općenito se mogu naći u jednom od 3 stanja, a to su:

- Zadatak koji se trenutno izvodi (eng. running task)
- Zadatci koji su spremi za izvođenje (eng. ready tasks)

- Zadatci koji su blokirani i čekaju određeni događaj (eng. blocked tasks)

Kod sustava s jednom procesorskom jezgrom samo jedan zadatak se u određenom trenutku može izvoditi. Zadatci koji su spremni za izvođenje čekaju oslobodjenje procesora i jedan od njih se odabire za izvršavanje. Dio jezgre SRSV-a koji je zadužen za izbor zadatka koji je spreman za izvođenje i koji će se idući izvršavati naziva se raspoređivač zadataka (eng. Task Scheduler). To je najvažniji dio jezgre SRSV-a jer o raspoređivanju zadataka ovisi kolika će biti kvaliteta usluge i koliko pouzdan će biti cijeli sustav. Neki zadatci mogu biti privremeno ili trajno blokirani i dok su u tom stanju raspoređivač zadataka ih ne uzima u obzir. Zadatci se tijekom rada prebacuju između opisanih stanja, no treba napomenuti da je ovo generalna podjela koja se razlikuje u implementaciji konkretnih SRSV-ova.



SLika 2.1: Stanja zadataka u SRSV-u

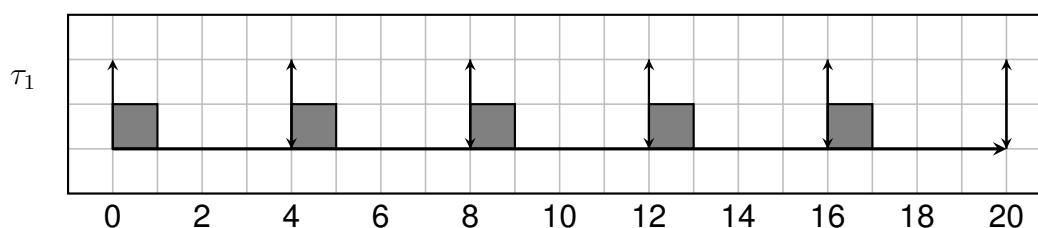
2.2. Periodični zadatci u sustavima za rad u stvarnom vremenu

Periodični zadatci su zadatci koji se iznova ponavljaju u istim vremenskim intervalima T_i . Taj interval naziva se period zadatka. Dio zadatka izvršen u jednom periodu naziva se posao. Svaki zadatak opisuje i vrijeme njegova izvršavanja C_i (eng. Computation time). Period i vrijeme izvršavanja zajedno daju veličinu koju nazivamo faktor opterećenja (eng. Utilization factor) koja daje postotak zauzeća procesora pojedinog zadatka u jednom periodu. Pošto se poslovi ponavljaju jedan za drugim, to je ujedno i veličina koja govori koliko procesorskog

vremena se ukupno troši na taj zadatak.

$$U_i = \frac{C_i}{T_i}$$

Nadalje, bitna veličina koja opisuje svaki zadatak je krajnji rok njegova završetka (eng. Deadline). Krajnji rok izvršavanja je vrijeme do kojeg se posao mora izvršiti kako bi donio korist sustavu. U ovom radu razmatrani su isključivo zadatci čiji krajnji rok završetka je jednak periodu. Kao primjer dan je vremenski dijagram s jednim periodičnim zadatkom. Period zadatka iznosi 4 vremenske jedinice, a vrijeme izvršavanja pojedinog posla 1 vremensku jedinicu. Pomoću ta dva podatka i ranije dane formule proizlazi da je faktor opterećenja zadatka 0.25. Drugim riječima, ovaj zadatak zauzima 25 % ukupnog procesorskog vremena.



SLika 2.2: Primjer periodičnog zadatka

Ukupno opterećenje sustava dobiva se kao zbroj faktora opterećenja svih zadataka. Ako je sustav preopterećen, tj. ukupno opterećenje mu je veće od 1 neki od zadataka se neće izvršiti do svog roka za izvršavanje. U tom slučaju koriste se algoritmi koji će osigurati da se zadatci ne preskaču nasumično, već na kontroliran način kao bi se sustav zaštitio od potencijalnih oštećenja nastalih preskakanjem kritičnih zadataka. Postoje dvije vrste preopterećenja sustava.

- Trajno preopterećenje (eng. Permanent Overload)
- Prolazno preopterećenje (eng. Transient Overload)

Kod prolaznog opterećenja sustava ima ukupno opterećenje manje ili jednako 1, no u nekom trenutku može doći do aktivacije aperiodičnog zadatka koji onda sustav gura u stanje preopterećenja. Nakon što prođe određeno vrijeme i aperiodični zadatak se izvede sustav se vraća u prijašnje stanje i svi zadatci se izvršavaju do svog krajnjeg roka. Drugi slučaj je kada je sustav u stanju trajnog preopterećenja, kada je ukupno opterećenje sustava trajno veće od 1. Tada se svi poslovi neće moći izvršiti i kontinuirano će se pojedini poslovi morati propuštati.

U ovom radu ispitivat će se ponašanje sustava s ublaženo-kritičnim uvjetima (eng. weakly hard constraints). Za razliku od kritičnih zadataka, kod ublaženo-kritičnih zadataka povremeno dopuštamo da se zadatak ne izvede na vrijeme, ali na predvidiv i kontroliran način. Na temelju važnosti pojedinog zadatka za rad cjelokupnog sustava određuje se u koliko slijednih perioda se zadatak može jednom propustiti. Ovaj pristup osigurava znatno bolje ponašanje u uvjetima preopterećenja jer se propuštanje pojedinih poslova odvija deterministički.

2.3. Algoritmi za raspoređivanje zadataka

U ovom radu ispitivat će se različiti algoritmi za raspoređivanje zadataka u sustavu koji se nalazi u stanju trajnog preopterećenja. To je slučaj kada je suma faktora opterećenja svih zadataka veća od 1, i kada se svi zadatci ne mogu izvršiti do svog krajnjeg roka izvršenja. U navedenom slučaju važno je osigurati preskakanje zadataka na predvidiv i za sustav siguran način.

Mjera kojom će se uspoređivati učinkovitost pojedinih algoritama naziva se kvaliteta usluge (eng. Quality of Service). Računa se kao omjer broj zadataka koji su se izvršili do krajnjeg roka završetka i ukupnog broja svih zadataka.

$$QoS = \frac{\text{broj izvršenih zadataka do krajnjeg roka}}{\text{ukupan broj zadataka}}$$

U nastavku su opisani korišteni algoritmi za raspoređivanje zadataka. Uz svaki algoritam prikazan je i raspored izvršavanja seta zadataka danog u tablici. Raspored se prikazuje vremenskim dijagramom u kojem je vidljivo koji zadatak se u danom trenutku izvršava. Za dani set zadataka zadano je da je prvi zadatak kritičan i mora se uvijek izvršiti, drugi zadatak ne utječe na sigurnost sustava, a treći zadatak mora se izvršiti svaki drugi put.

Broj Zadatka	Period	Vrijeme izvršavanja	Faktor opterećenja
1	6	2	izracunaj
2	8	2	izracunaj
3	4	2	izracunaj

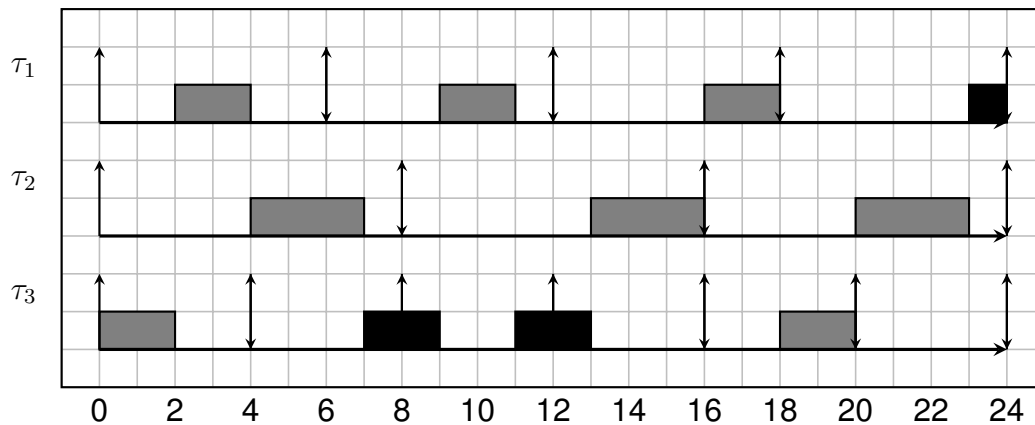
Dani set zadataka nalazi se u preopterećenju, i neki od zadataka neće zadovoljiti svoj krajnji rok završetka. Ukupni faktor opterećenja sustava je 1.08.

2.3.1. EDF algoritam

EDF algoritam (eng. Earliest Deadline First) je algoritam koji pri raspoređivanju zadataka prioritet daje onim poslovima koji imaju raniji rok za završetak. Ukoliko sustav nije preopterećen (ako je ukupni faktor opterećenja manji od 1) ovim algoritmom optimalno će se rasporediti zadatci i svi će se izvršiti. Na vremenskom dijagramu crnom bojom su prikazani poslovi koji su propustili svoj krajnji rok izvršenja.

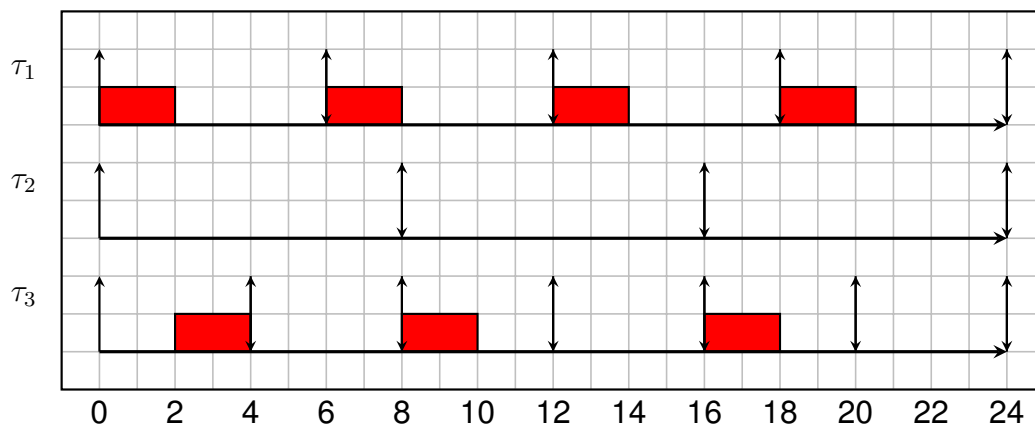
2.3.2. RTO algoritam

RTO algoritam (eng. Red Tasks Only) je algoritam prema kojemu se samo izvršavaju strogi zadatci, dok se oni koji se mogu preskočiti uvijek preskaču. Prema ovom algoritmu osigurano je poštivanje zadanih uvjeta, no pri manjim faktorima opterećenja ovaj algoritam nije optimalan.



SLika 2.3: Raspored zadataka EDF algoritmom

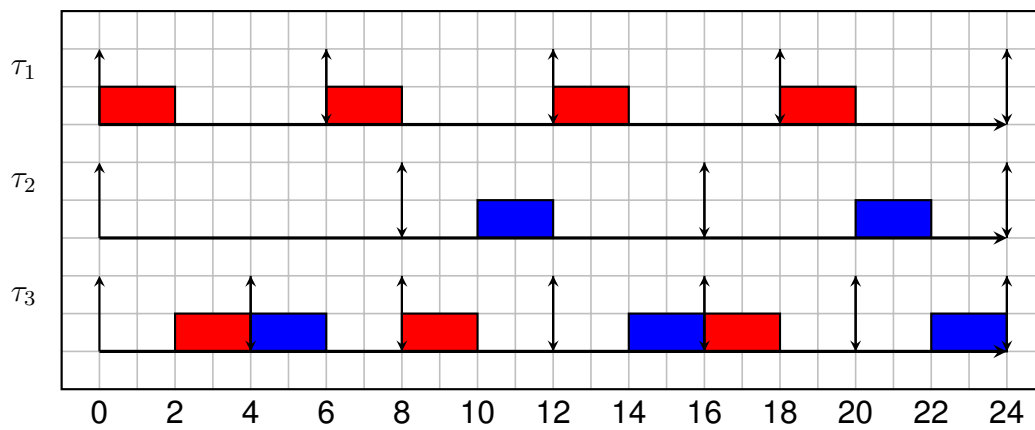
Razlog tomu je što postoji slobodno procesorsko vrijeme u kojem bi se mogli izvršiti zadatci koje nije nužno izvršiti, no oni se automatski izbacuju iz rasporeda. Kritični zadatci raspoređuju se prema ranije opisanom EDF algoritmu.



SLika 2.4: Raspored zadataka RTO algoritmom

2.3.3. BWP algoritam

BWP algoritam (eng. Blue When Possible) je poboljšanje ranije opisanog RTO algoritma. Kod BWP algoritma prioritet imaju strogi zadatci, no i zadatci koji se ne moraju nužno izvršiti su u listi čekanja. Na taj način, ukoliko se svi kritični zadatci izvrše, na red će doći i opcionalni zadatci. Ovom modifikacijom znatno se poboljša stopa uspješnosti izvršenja zadataka (eng. Quality of Service), pogotovo pri manjim faktorima opterećenja. Zadatci su u dijagramu obojeni prema nazivu algoritma, crvena boja za stroge zadatke, a plava za zadatke čije izvršavanje je opcionalno.



SLika 2.5: Raspored zadatka BWP algoritmom

3. Modifikacija jezgre FreeRTOS-a

3.1. Operacijski sustav FreeRTOS

FreeRTOS je operacijski sustav za rad u stvarnom vremenu otvorenog koda (eng. open source). U ovom odjeljku bit će objašnjena njegova implementacija i izvedba raspoređivanja zadataka. Kod FreeRTOS-a organiziran je u nekoliko datoteka i zauzima svega nekoliko kilobajta. Datoteka u kojoj je implementirano upravljanje zadatcima i njihovim raspoređivanjem je `tasks.c`.

Zadaci u FreeRTOS-u su upravljani kroz strukturu za upravljanje zadatcima (eng. TCB - Task Control Block). U toj strukturi nalaze se sve informacije koje opisuju pojedini zadatak. Pri stvaranju novog zadatka u memoriji se alocira prostor za novu inačicu ove strukture u koju se potom upisuju parametri novokreiranog zadatka. Zadaci su nadalje smješteni u različite liste, ovisno o tome u kojem stanju se zadatak nalazi. Postoje liste za zadatke koji su spremni za izvršavanje, za zadatke koji su blokirani te za zadatke koji čekaju određeno vrijeme kako bi se ponovo mogli izvršavati. U TCB-u svakog zadatka nalazi se informacija kojoj listi zadatak u danom trenutku pripada. Upravljanje i rad listi ostvaren je strukturama `ListItem_t` i `List_t`. `ListItem_t` je struktura koja sadrži podatke o pojedinom članu liste (pokazivači na prethodni i slijedeći `ListItem_t` te pokazivač na TCB zadatka na kojeg se odnosi). U `List_t` spremljeni su podatci o listi (broj elemenata, pokazivač na trenutni element te pokazivač na kraj liste).

Informacija o tome koji zadatak se trenutno izvršava pohranjena je u varijabli `pxCurrentTCB`, koja je tipa pokazivač na TCB strukturu. Zadaci koji su spremni za izvršavanje podjeljeni su u različite liste ovisno o tome kolikog su prioriteta (postoji posebna lista za sve moguće prioritete zadataka). Stoga se pri raspoređivanju zadataka naprije pronađe lista najvišeg prioriteta koja nije prazna. Zadaci iz te liste dijele procesorsko vrijeme na način da se svakom zadatku iz liste dodjeljuje jedan vremenski odsječak procesorskog vremena (eng. tick). Nakon što se zadatak izvede u jednom odsječku, vraća se na kraj liste čekanja. Ovaj način raspoređivanja zadataka naziva se dijeljenje procesorskog vremena među zadatcima (eng. Time Slicing).

FreeRTOS se konfigurira putem `FreeRTOSConfig.h` datoteke i korisnik treba mijenjati

samo tu datoteku. U toj datoteci nalaze se sve konstante preko kojih se uključuju pojedine funkcionalnosti ili postavljaju konstante bitne za rad sustva.

Podrška za periodične zadatke nije implementirana u FreeRTOS-u te ju je potrebno dodati. Također navedni pristup ne rješava problem preopterećenja sustava te je potrebno modificirati jezgru FreeRTOS-a kako bi se omogućilo predvidivo ponašanje ukoliko sustav uđe u trajno preopterećenje. Potrebne modifikacije detaljno su opisane u narednim podglavljljima.

3.2. Programska potpora za kontrolu izvršavanja periodičnih zadataka

Uključenje funkcionalnosti za podršku periodičnim zadatcima ostvareno je putem konstante unutar FreeRTOSconfig.h datoteke. Svi djelovi programskog koda zaduženi za periodične zadatke pisani su u odsječcima koji se kompajliraju samo ako je korištenje periodičnih zadataka uključeno. Navedeno je ostvareno pomoću pretprocesorske naredbe #if. Time je korisniku omogućeno uključenje programske potpore za periodičke zadatke na jednostavan i brz način.

```
1 #if ( configUSE_PERIODIC_TASK == 1 )
2
3 #endif
```

Isječak koda 3.1: Pretprocesorska naredba za uključenje periodičnih zadataka

3.2.1. Stvaranje periodičnih zadataka

Prvi korak pri implementaciji programske podrške za izvršavanje periodičnih zadataka je proširenje TCB-a veličinama koje opisuju periodičan zadatak. Ovdje su definirane sve veličine bitne za kontrolu periodičkih zadataka koje su opisane u ranijim poglavljljima.

```
1 #if ( configUSE_PERIODIC_TASK == 1 )
2     uint8_t xTaskId;
3     TickType_t xTaskPeriod;
4     TickType_t start_time;
5     TickType_t xTaskDuration;
6     TickType_t xDeadline;
7     TickType_t xRemainingTicks;
8
9     // variables used for weakly hard conditions control
10    uint8_t weakly_hard_constraint;
11    uint8_t previous_deadline_met;
12 #endif
```

Isječak koda 3.2: Varijable dodane u strukturu za kontrolu zadataka

Nadalje napisana je funkcija `xTaskCreatePeriodic` koja se koristiti za stvaranje periodičnih zadataka. Navedena funkcija je proširenje funkcije `xTaskCreate` s novim varijablama potrebnim za opis periodičnih zadataka.

```
1 BaseType_t xTaskCreatePeriodic( TaskFunction_t pxTaskCode ,
2                               uint8_t id ,
3                               const char * const pcName ,
4                               const configSTACK_DEPTH_TYPE usStackDepth ,
5                               void * const pvParameters ,
6                               UBaseType_t uxPriority ,
7                               TaskHandle_t * const pxCreatedTask ,
8                               TickType_t period ,
9                               TickType_t duration ,
10                              int weakly_hard_constraint );
```

Isječak koda 3.3: Prototip funkcije `xTaskCreatePeriodic`

3.2.2. Kontrola izvršenja periodičnih zadataka

Izvršavanje zadataka u SRSV-ovima podjeljeno je u vremenske odsječke. U određenim vremenskim intervalima prekida se izvođenje poslova i određuje se koji posao se treba dalje izvršavati. U konkretnom slučaju simulacije na posix sustavu taj interval iznosi 1 milisekundu. U FreeRTOS-u navedena funkcionalnost implementirana je u `xTaskIncrementTick` funkciji. Prekidni sustav periodički poziva navedenu funkciju i u njoj je potrebno dodati funkcionalnost kontrole periodičnih zadataka.

Za potrebe ovog projekta zadatci su spremni u dvije liste. Jedna lista u kojoj se čuvaju zadatci spremni za izvršavanje, ranije implementirana u FreeRTOS-u i novododana lista nazvana `xWaitTaskList` u kojoj su zadatci koji su na čekanju. Za dodavanje zadataka u liste napisane su dvije funkcije, za jedna po svakoj listi.

```
1 void addTaskToReadyList(TCB_t * const pItemToAdd);
2 void addTaskToWaitList(TCB_t * const pItemToRemove);
```

Isječak koda 3.4: Definicije funkcija za dodavanje zadataka u opisane liste

Kontrola izvršavanja u ovom projektu realizirana je apsolutno na višekratnike perioda zadataka od početka simulacije.

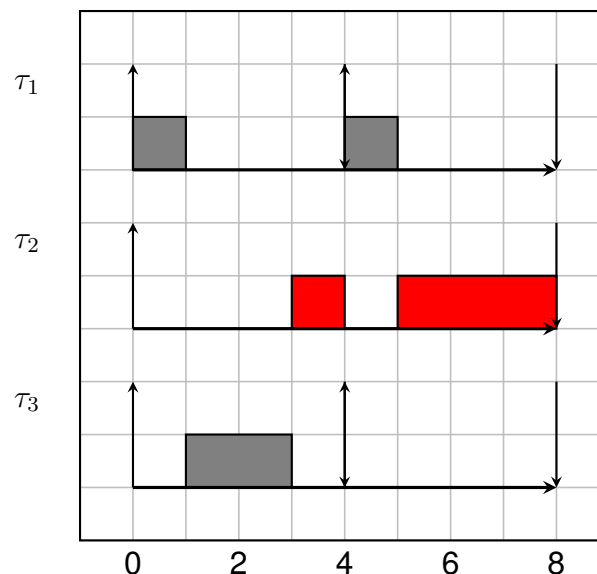
U svakom pozivu funkcije `xTaskIncrementTick` potrebno je proći po svim zadatcima u obje liste. Zadatci koji su spremni za izvršavanje možda se nebi uspjeli izvršiti do roka izvršenja čak kad bi dobili svo procesorsko vrijeme. Pošto nema koristi od posla koji se djelomično izvršio takve zadatke potrebno je odmah smjestiti u listu za čekanje i detektirati njihovo propuštanje roka izvršenja. Ova strategija zove se prekidanje poslova koji se ne mogu izvršiti do svog krajnjeg roka izvršenja(eng. job killing). Time se podiže kvaliteta

usluge, jer je osigurano da poslovi koji se nikako neće izvršiti na vrijeme ne zauzimaju procesorsko vrijeme ostalim zadacima. Funkcionalnost prekidanja poslova implementirana je u funkciji `killTasks`. U njoj prolazimo po svim zadacima koji se nalaze u listi zadataka spremnih za izvršavanje i na temelju podataka o trenutnom stanju posla odlučujemo treba li ga prekinuti ili ne. U nastavku je priložen navedeni uvjet i graf koji opisuje navedenu situaciju prekidanja zadataka.

```
1 if ((xTaskGetTickCount() - Tcb->start_time + Tcb->xRemainingTicks) >
2 Tcb->xTaskPeriod)
```

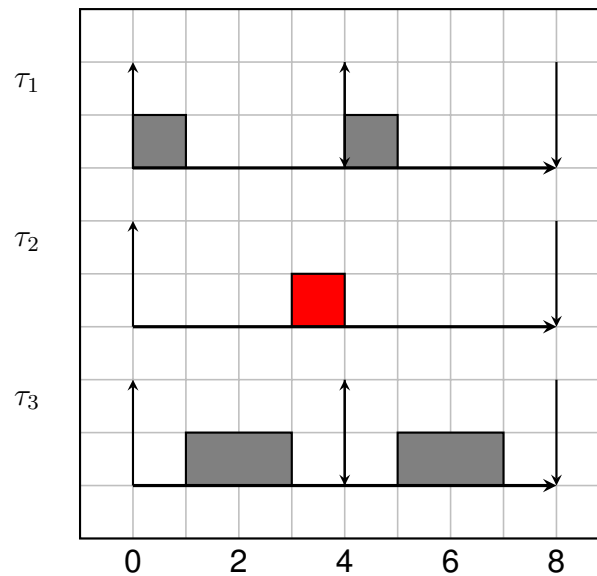
Isječak koda 3.5: Uvjet za prekidanje izvođenja zadatka

Opisana situacija prikazana je na vremenskom dijagramu 3.1. U primjeru imamo 3 zadatka, od kojih je zadatak broj 1 većeg prioriteta dok su preostala dva zadatka manjeg prioriteta. Periodi zadataka su slijedno 4, 8, i 4, dok su vremena izvršavanja 1, 5, i 2. (dodaj ovo ispod slike u figure radije, bit će bolje i preglednije) Vidimo da je zadatak broj 2 potrošio 4 vremenska odjeka procesorskog vremena, a na kraju se nije uspio izvršiti na vrijeme. Strategija prekidanja poslova uvodi provjeru svaki vremenski odjevak može li se zadatak izvesti do roka. U navedenom primjeru u trenutku $t = 5$ zadatak bi bio prekinut. Tada bi raspored poslova izgledao kao na dijagramu 3.2. Treba primjetiti da se uz prekidanje poslova povećala kvaliteta usluge jer se je zadatak broj 3 sad uspio izvršiti u oba perioda. Važna napomena uz navedeni primjer je da služi samo za objašnjenje strategije prekidanja poslova te je stoga pretpostavljeno da su svi zadatci jednako kritični.



SLika 3.1: Primjer rasporeda bez prekidanja zadataka

Nadalje, u funkciji `wakeTasks` provjerava se trebaju li se zadatci u listi za čekanju prebaciti u listu zadataka spremnih za izvršavanje. Ukoliko je zadatak u listi za čekanju i ukoliko se program nalazi na višekratniku njegova perioda, zadatak se dodaju u listu zadataka spremnih



SLika 3.2: Primjer rasporeda s prekidanjem zadataka

za izvršavanje. Pri tome je potrebno u varijablu `start_time` upisati trenutno vrijeme (trenutak u kojem se zadatak krenio izvršavati). Vrijeme početka izvršavanja je važno za provjeru treba li zadatak prekinuti i je li se izvršio do krajnjeg roka završetka.

Varijabla `xRemainingTicks` u svakom trenutku pamti koliko je vremenskih odsječaka ostalo poslu da se do kraja izvrši. Svaki put kada se posao izvršava ta varijabla se umanjuje za 1. Ukoliko se vrijednost `xRemainingTicks` smanji na 0, posao je gotov i zabilježava se njegovo pravovremeno izvršavanje.

3.3. Implementacija algoritama za raspoređivanje zadataka

Algoritmi za raspoređivanje zadataka podrazumijevaju logiku kojom se odabire koji zadatak će se idući poslati na izvršavanje. Liste implementirane u FreeRTOS-u imaju mogućnost sortiranja po vrijednosti varijable `xItemValue` koju sadrži svaki član liste (u `ListItem_t` strukturi). To svojstvo je iskorišteno kako bi zadatke koji su spremni za izvršavanje poredali po željenom redosljedu.

4. Implementacija simulatora

4.1. Generiranje setova testnih zadataka

Generiranje seta zadataka implementirano je u taskSetGenerator.c datoteci. U strukturi periodic_task sadržane su sve informacije potrebne za opis pojedinog zadatka, kontrolu njegovog izvođenja te prikupljanje podataka o poštivanju rokova izvršavanja tijekom simulacije.

```
1 struct periodic_task {
2     TaskHandle_t handler;
3     char * name;
4     double u;
5     TickType_t period;
6     TickType_t duration;
7     int weakly_hard_constraint;
8     int numOfPeriods;
9     bool report[MAX_PERIOD_CNT];
10    int missed_deadlines;
11    int times_killed;
12 } Task_Set[MAX_TASK_CNT];
```

Isječak koda 4.1: Struktura periodic_task

Za svaki zadatak definirana je jedna inačica ove strukture. Informacije o periodima i weakly hard ograničenjima zadataka zadane su u tekstuanim datotekama i ranije generirane (OPISI TO KAD NAPRAVIS). Ti podatci se procitaju iz datoteka i upišu u odgovarajuće varijable strukture pojedinog zadatka. Za generiranje faktora opterećenja za set testnih zadataka korišten je UUniFast algoritam. Algoritam prima ukupan broj zadataka i sumu faktora opterećenja svih zadataka te uniformno raspoređuje faktore opterećenja. Vremenska složenost algoritma je $O(n)$. Vrijeme izvršavanja pojedinog zadatka dobiveno je kao umnožak perioda i faktora opterećenja. Imena zadataka generiraju se u obliku Task_xx, gdje xx predstavlja id pojedinog zadatka, počevši od 1 do broja zadataka.

Prije početka svake simulacije poziva se funkcija startTaskSetGenerator koja poziva potrebne funkcije kako bi se izvršili svi ranije navedeni koraci. Navedena funkcija kao argumente prima ukupan broj zadataka koje je potrebno generirati, zadani faktor opterećenja i putanju do datoteke u koju će se pohraniti rezultati simulacije.

```

1 void startTaskSetGenerator(double utilization ,int n, char * report_file){
2     TASK_CNT = n;
3     total_utilization = utilization;
4     file_path = report_file;
5     calculateUtilization(utilization);
6     readTaskPeriods();
7     readWeaklyHardConstraint();
8     calculateTaskDuration();
9     calculateHiperperiod();
10    calculateNumOfPeriods();
11    generateTaskNames();
12    resetTimesKilled();
13    resetReports();
14 }

```

Isječak koda 4.2: Funckija startTaskSetGenerator

4.2. Tijek simulacije

Tijekom izvođenja simulacije za svaki zadatak se pamti je li izvršen do roka završetka u svakom periodu. Na taj način se dobije informacija o izvođenju svakog zadatka u svakom vremenskom intervalu perioda. To je implementirano pomoću bool niza čije su vrijednosti na početku simulacije postavljene na 0 i nakon što se zadatak uspješno izvede prije roka završetka ta vrijednost postavi se u 1. Simulacija se prekida nakon hiperperioda (najmanjeg zajedničkog višekratnika perioda svih zadataka). Nakon završetka simulacije, na temelju ranije navedenih podataka, za svaki zadatak se zbraja broj propuštenih rokova završetaka. te se u csv (eng. comma-separated values) datoteku izvještaja dodaje novi redak koji opisuje trenutno provedenu simulaciju. Podatci koje je bitno upisati u izvještaj su ukupni broj zadataka koji se nisu izvršili do roka završetka, kao i broj propuštenih rokova završetka svakog pojedinog zadatka. Također je bitno znati jesu li bili zadovoljeni svi weakly hard uvjeti postavljeni nad simulacijom te kolika je stopa uspješnosti izvršenja zadataka (eng. Quality of Service).

4.3. Pokretanje simulacije

Radi usporedbe različitih algoritama nad generiranim setom zadataka potrebno je pokrenuti program uz različite faktore opterećenja. Za to je implementirana bash skripta kojom se najprije stvara csv datoteka u kojoj će biti pohranjen izvještaj sa podacima o svim simulacijama pokrenutim za određeni set testnih zadataka. Za stvaranje datoteke izvještaja napisan je poseban c program koji osim što stvori datoteku zapiše u nju prvi red sa nazivima pojedinih

stupaca. Skripta nadalje pokreće simulaciju od faktora opterećenja 0.9 do 1.5, uz korake 0.01 te se nakon svakog pokretanja simulatora u datoteku izvještaja upisuje jedna linija koja sadrži ranije opisane podatke. Pri pokretanju simulacije programu se preko argumenata prosleđuju 3 parametra, broj zadataka, faktor opterećenja te datoteka na čiji će se kraj upisivati izvještaj nakon završene simulacije. Faktor opterećenja se radi jednostavnosti implementacije prosleđuje pomožen sa 100, jer bash ne podržava rad sa decimalnim brojevima. U nastavku je priložena slika datoteke izvještaja.

DODAJ SLIKU IZVJEŠTAJA !!!

OPISI KAKO SE IZ IZVJEŠTAJA DOBIJU GRAFOVI!!!

5. Rezultati

tu ce biti rezultati

6. Zaključak

tu ce nam biti zakljucak, kad ga naravno napisem

6.1. First section

ovo je prvo odlomak

6.2. Second section

ovo je drugi. laku noc !

BIBLIOGRAPHY

Nadogradnja operacijskog sustava FreeRTOS za primjenu u kontrolnim aplikacijama

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.