SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 000

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

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 sretstva,vojna industrija, multimedijski sustavi, roboti itd. Ponekad SRSV zbog iznenadne pojave dodatnih zadataka mogu ući u stanje preopterećenja. To je stanje u kojemu procesor ne može izršiti 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. (OVO DODAJ !!!!)

Potrebno je razmotriti rješenja koja će osigurati da najkritičniji poslovi dobiju veći prioritet za izvršavanje. Na taj način se izbjegava šteta koja bi potencijalno nastala propuštanjem takvih zadataka.

Ideja u ovom radu je implementirati strogi sustav za rad u stvarnom vremenu sa 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 se osigurava da nikad ne dođe do blokade pojedinog zadatka te da se njegovo izvršavanje uvijek preskače.

U literaturi se može pronaći mnogo algoritama za raspoređivanje zadataka. Neki od njih bit će implementirani i kasnije uspoređivani na istim setovima zadataka. Cilj je pronaći algoritam koji će optimalno rasporediti zadatke u uvjetima preopterećenja, to jest kod kojeg će biti zadovoljeni svi uvjeti strogog SRVS-a s ublaženim uvjetima uz što manje propuštenih poslova.

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. Kako bi sustav radio pouzdano mora se osigurati predvidivo raspoređivanje zadataka koje će osigurati da se što više zadataka izvrši na vrijeme.

S obzirom na posljedice koje izaziva propuštanje roka izvršavanjazadatke djelimo u dvije skupine :

- Kritični zadatci (eng. hard real time tasks)
- Zadatci koji se mogu preskočiti (eng. soft real time tasks)

Kritični 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 kritičnog zadatka je detekcija pritiska papučice kočnice na automobilu. S druge strane, propuštanje nekih zadataka nam nije kritično i neće nanjeti štetu sustavu, nego će smanjiti performanse ili mogućnosti sustava. Primjer ovakvog zadatka je paljenje signalne ledice.

U ovom radu ispitivati će se različiti algoritmi za raspoređivanje zadataka u sustavu koji se nalazi u stanju 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 stopa uspješnosti izvršenja zadataka (eng. Quality of Service). Računa se kao omjer broj zadataka koji su se izvršili do krajnjeg roka završetka i ukupnopg broja svih zadataka.

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

Zadatci u operacijskim sustavima za rad u stvarnom vremenu općenito se mogu podjeliti u 3 grupe :

- 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)

(OPISI OVO !!)

2.2. Opis periodičnih zadataka u sustavima za rad u stvarnom vremenu

Periodični zadatci su zadatci koji se iznova ponavljaju u istim vremenskim intervalima Ti. Taj interval naziva se period zadatka. Svaki zadatak opisuje i vrijeme njegova izvršavanja Ci (eng. Computation time). Period i vrijeme izvršavanja zajedno daju veličinu koju nazivamo faktor opterećenja (eng. Utilization) koja opisuje postotak zauzeća procesora pojedinog zadatka u jednom periodu.

$$U_{\mathrm{i}} = \frac{C_{\mathrm{i}}}{D_{\mathrm{i}}}$$

Nadalje, bitna veličina koja opisuje svaki zadatak je krajnji rok njegova završetka (eng. deadline). U ovom radu razmatrani su isključivo zadatci čiji krajnji rok završetka je jednak periodu.

Ukupno opterećenje sustava dobiva se kao zbroj faktora opterećenja svih zadataka. Ukoliko 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 moraju se koristiti 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.

U ovom radu ispitivat će se ponašanje sustava sa 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 na siguran način preskaćemo izvođenje pojedinih zadataka.

2.3. Algoritmi za raspoređivanje zadataka

2.3.1. EDF algoritam

EDF algoritam (eng. Earliest Deadline First) je algoritam koji pri raspoređivanju zadataka prioritet daje onim zadatcima koji imaju raniji rok za završetak. Ukoliko sustav nije preopterećen (ukoliko je ukupni faktor opterećenja manji od 1) ovim algoritmom optimalno će se rasporediti zadatci i svi će se izvršiti.

2.3.2. RTO algoritam

RTO algoritam (eng. Red Tasks Only) je algoritam prema kojemu se samo izvršavaju kritični 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. 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.

2.3.3. BWP algoritam

BWP algoritam (eng. Blue When Possible) je poboljšanje ranije opisanog RTO algoritma. Kod BWP algoritma prioritet imaju kritični 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.

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 implmenetacija i izvedba raspoređivanja zadataka. Kod FreeRTOS-a organiziran je u svega nekoliko datoteka i zauzima svega nekoliko kilobajta. Datoteka u kojoj je implementirano upravljanje zadatcima i njihovim raspoređivanjem je tasks.c.

Kod FreeRTOS-a zadatci su raspoređeni u liste za čekanje na način da postoji posebna lista za sve moguće prioritete zadataka. Stoga se pri raspoređivanju zadataka naprije pronađe lista najvišeg prioriteta koja nije prazna. Zadatci 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

Zadatci u FreeRTOS-u su upravljani kroz strukturu za upravljanje zadatcima (eng. TCB - Task Control Block). bla bla opisi jos ovo...

Konfiguriranje FreeRTOS-a vrši se putem FreeRTOSconfig.h datoteke. U toj datoteci nalaze se konstante preko kojih se uključuju pojedine funkcionalnosti ili postavljaju vrijednosti bitne za rad sustva.

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 omogučeno uključenje programske potpore za periodičke zadatke na jednostavan i brz način.

```
#if ( configUSE_PERIODIC_TASK == 1 )

#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 poglavljima.

```
#if ( configUSE_PERIODIC_TASK == 1 )
    uint8_t xTaskId;
    TickType_t xTaskPeriod;
    TickType_t start_time;
    TickType_t xTaskDuration;
    TickType_t xDeadline;
    TickType_t xRemainingTicks;

    // variables used for weakly hard conditions control
    uint8_t weakly_hard_constraint;
    uint8_t previous_deadline_met;
#endif
```

Isječak koda 3.2: Varijable dodane u strukturu za kontrolu zadataka

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

```
BaseType_t xTaskCreatePeriodic( TaskFunction_t pxTaskCode,
                        uint8_t id,
                        const char * const pcName,
                        const configSTACK_DEPTH_TYPE usStackDepth,
                        void * const pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * const pxCreatedTask,
                        TickType_t period,
                        TickType_t duration,
                        int weakly_hard_constraint);
```

Isječak koda 3.3: Prototip funkcije xTaskCreatePeriodic

3.2.2. Kontrola izvršenja priodičnih 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 implementrana 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 spremani 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 pxItemToAdd);
2 void addTaskToWaitList(TCB_t * const pxItemToRemove);
```

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 zadatcima. Funkcionalnost prekidanja poslova implementirana je u funkciji killTasks. U njoj prolazimo po svim zadatcima 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

DODAJ GRAF FINO DA SE VIDI !!!!

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

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.

```
struct periodic_task{
    TaskHandle_t handler;
    char * name;
    double u;
    TickType_t period;
    TickType_t duration;
    int weakly_hard_constraint;
    int numOfPeriods;
    bool report[MAX_PERIOD_CNT];
    int missed_deadlines;
    int times_killed;
} Task_Set[MAX_TASK_CNT];
```

Isječak koda 4.1: Struktura periodic_task

Za svaki zadatak definirana je jedna inačica ove strukture. Infomacije 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.

```
void startTaskSetGenerator(double utilization,int n, char * report_file){
    TASK_CNT = n;
    total_utilization = utilization;
    file_path = report_file;
    calculateUtilization(utilization);
    readTaskPeriods();
    readWeaklyHardConstraint();
    calculateTaskDuration();
    calculateHiperperiod();
    calculateNumOfPeriods();
    generateTaskNames();
    resetTimesKilled();
    resetReports();
}
```

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 skripa kojom se najprije stvara csv datoteka u kojoj će biti pohranjen izvještaj sa podatcima o svim simulacijama pokrenutim za određeni set testnih zadataka. 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 prosljeđuju 3 parametra, broj zadataka, faktor opterećenja te datoteka na čiji će se kraj upisivati izvještaj nakon završene simulacije. Faktor opterećenja se radi jednostavnosti implementacije prosljeđuje pomožen sa 100, jer bash ne podržava rad sa decimalnim brojevima.

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.