DESIGN OF RTOS SYSTEMS

PRIMJER REAL – TIME KERNELA

Karakteristike dizajniranja kernela
Ciljevi su :

- razumjeti osnovne probleme koji se moraju uzeti u obzir kod projektovanja kernela real-time sistema

Objašnjenje je dato korištenjem DICK (Didactic C Kernel)

* pisan je u C jeziku
* upravlja sa periodičnim i aperiodičnim taskovima
* ima eksplicitna vremenska ograničenja
* inter-task komunikacija koja je vremenski prediktabilna

DESIGN OF RTOS SYSTEMS

Struktura real-time kernela

Kernel je :

- unutarnje jezgro svakog operativnog sistema
- direktna povezanost sa hadverom fizičke mašine
- osnovne aktivnosti su:
  * menadjment procesa
  * upravljanje interaptima
  * sinhronizacija procesa

Menadjment procesa:

* je primarni servis koji treba da obezbedi operativni sistem
* uključene su slijedeće funkcije podrške:
  - kreiranje i terminiranje procesa
  - rasporedjivanje poslova
  - dispečing
  - context switching
  - druge vezane aktivnosti

DESIGN OF RTOS SYSTEMS

Upravljanje interaptima

Ciljevi:

Obezbjediti servisiranje interapt zahtjeva koji mogu biti generisani od strane bilo kojeg perifernog uredjaja

Servisi koji se obezbjedjuju:

Izvršenje specifične rutine (tj. drajvera) koji prebacuje podatke izmedju uradjaja i glavne memorije

Razlika u odnosu na normalne OS:

Kod normalnih OS , aplikacioni taskovi se uvjek priemptiraju od strane interapt taskova (tj. drajvera). Kod RTOS (real-time Operating systems), ovo može dovesti do gubljenja deadlajna , zbog toga drajveri se moraju rasporedjivati kao i drugi taskovi.

DESIGN OF RTOS SYSTEMS

Sinhronizacija i komunikacija procesa

Pristup kod klasičnih OS:

semafori, efikasno riješavaju obadva zadatka : i sinhronizaciju i medjusobno isključenje.

Problemi kod RTOS:

Semafori su skloni inverziji prioriteta.

Ovo može prouzrokovati neograničeno blokiranje taskova.

Riješenje

Koristiti specijalne forme " semafora":

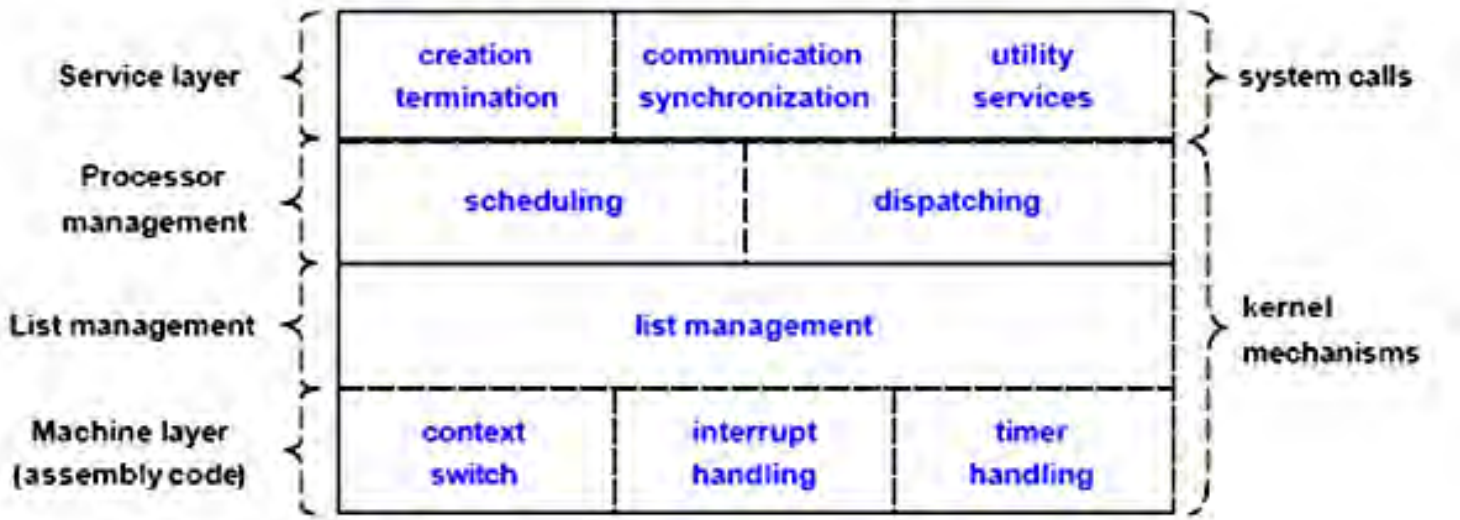
- naslijedjivanje prioriteta
- ograničenje (ceiling) prioriteta
- strategija resursa steka

DESIGN OF RTOS SYSTEMS

Opšta struktura DICK –a

Osnovne predpostavke su :

Svi taskovi su rezidentni u glavnoj memoriji kada prime kontrolu nad procesorom (ovo je uobićajeno riješenje kod svih RTOS):



DESIGN OF RTOS SYSTEMS

Mašinski sloj DICK-a

Mašinski sloj direktno interaktira sa hadverom fizičke mašine

* napisan je u asamblerskom jeziku

Servisi koji su obezbjedjeni:

- context switching
- upravljanje interaptima
- upravljanje tajmerima

* Servisi u mašinskom sloju nisu vidljivi na nivou korisnika

Sloj menadjmenta

* Namjena: da drži evidenciju statusa različitih taskova

DESIGN OF RTOS SYSTEMS

* Servisi : upravlja sa nizom listi. Taskovi sa istim statusom se stavljaju u red u odgovarajuću listu.

* Ovaj sloj obezbjedjuje potrebna umetanja i otklanjanja primitiva

Sloj menadjmenta procesora

* Servisi koje obezbjedjuje:
  - rasporedjivanje
  - dispečing

Servisni sloj

Cilj mu je : da obezbjedi sve servise koji su vidljivi na nivou korisnika kao sistemski pozivi (system call)

DESIGN OF RTOS SYSTEMS

Servisi koje obezbjedjuje:

- kreiranje taskova
- odbacivanje taskova
- suspenzija periodičnih instanci
- aktivacija i suspenzija aperidičnih instanci
- operacije sistemskih upita

Procesna stanja

U svakom kernelu koji podržava konkurentne aktivnosti na jednom procesoru, prisutna su najmanje tri stanja u koja može ući task:

* Running : task ulazi u ovo stanje kada počne da se izvršava na procesoru.
* Ready : task je spreman za izvršenje ali procesor je doznačen drugom tasku. Svi spremni taskovi se stavljaju u red ready .

DESIGN OF RTOS SYSTEMS

* Waiting : task ulazi u ovo stanje kada izvršava sinhronizacionu primitivu da ćeka na dogadjaj (napr. ćeka na zaključani semafor).
  Task se umeće u red koji je pridružen ovom uslovu. Kada se task ponovo uzme u izvršenje (napr. odključavanjem semafora od strane nekog drugog taska), on se umeće u red spremnih (ready) taskova.

Neaktivno (IDLE) stanje

RT kernelu, koji podržava periodične taskove, potrebno je još jedno dodatno stanje:

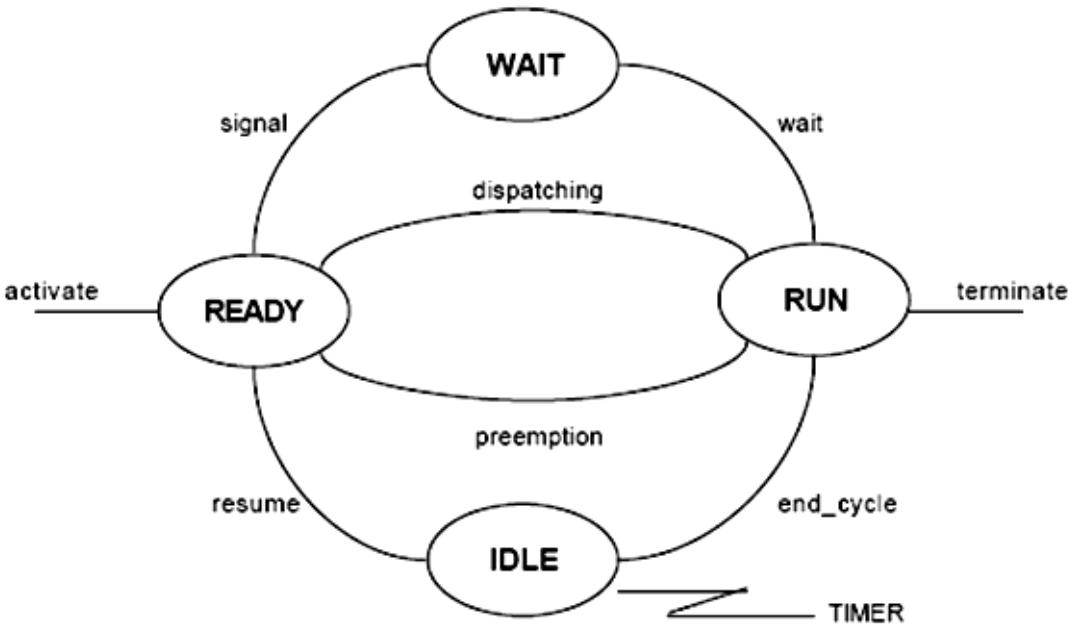
* IDLE : Periodičan posao ulazi u ovo stanje kada kompletira svoje izvršenje i mora da ćeka za početak novog perioda izvršenja, da bude probudjen od strane tajmera.

* Na kraju ciklusa periodični posao obavještava kernel pomoću sistemskog poziva *end cycle*. Ovo stavlja posao u IDLE stanje i doznačuje procesor drugom spremnom poslu.

DESIGN OF RTOS SYSTEMS

* U pravo vrijeme posao u IDLE stanju će biti probudjen od strane kernela i umetnut u ready red.
* Ova operacija se izvršava od strane rutine koja je aktivirana tajmerom. Ona provjerava u svakom otkucaju, da li neki od poslova treba biti probudjen.

Dijagram tranzicije za RT stanja



DESIGN OF RTOS SYSTEMS

Stanja kašnjenja (delay) i prijema (receive)

* **Delay** : U ovo stanje posao ulazi kada on izvrši primitivu *delay* . To će dovesti do suspenzije posla. Zakašnjeli (delayed) poslovi biće probudjeni od strane tajmera nakon nekog specificiranog vremena. (i nakon toga stavljeni u ready red).
* **Receive** : Posao će ući u ovo stanje kada posao izvrši receive operaciju na praznom kanalu. Prouzrokovaće suspenziju posla. Kada send operacija na ovom kanalu je izvršena od strane nekog drugog posla, suspendirani posao napušta **Receive** stanje i postaje spreman (ready).

ZOMBIE STANJE

* Ovo stanje je potrebno za RT sisteme koji podržavaju dinamičku kreaciju i završetak tvrdih (hard) periodičkih taskova.

DESIGN OF RTOS SYSTEMS

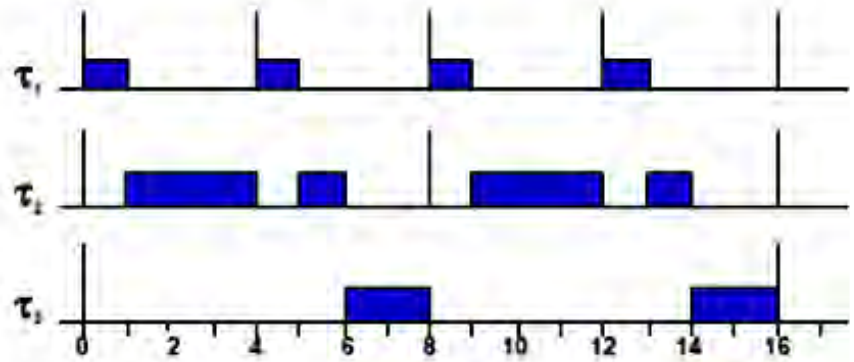
* kada periodički task tk se abortira , njegov faktor Uk iskorištenja ne može se trenutačno oduzeti od ukupnog opterećenja procesora : možda je već zakasnio izvršenje nekog drugog taska.

* Test konzistentne garancije: oduzima Uk samo na kraju tekućeg perioda od tk .
* Da bi se implementirao ovaj koncept : uvesti novo stanje, ZOMBIE. Ako je “ubijen” task će ući u ovo stanje do kraja perioda tekuće instance. Nakon toga on konačno napušta sistem.

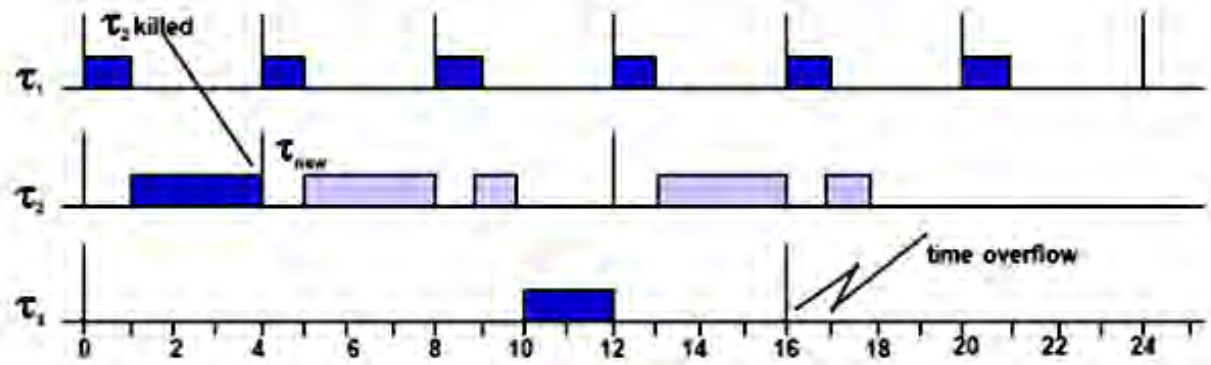
* Za vrijeme ZOMBIE stanja task ne konzumira procesorsko vrijeme nego drži njegov faktor iskorištenja Uk prozvanim.

DESIGN OF RTOS SYSTEMS

Primjer abortiranja



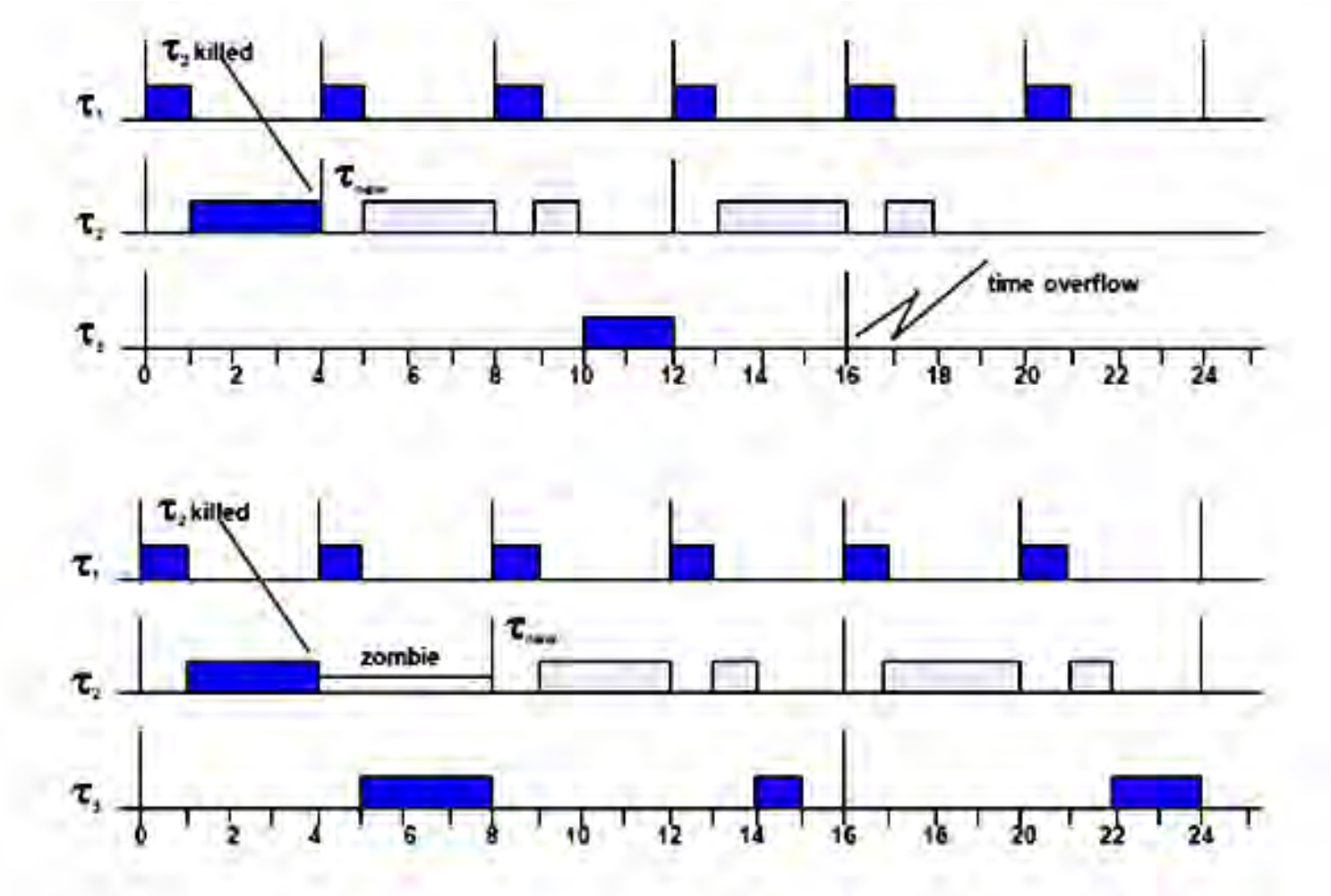
(Periods are harmonic => U=1 is possible!)



predpostavimo da će U2 biti oduzeto u trenutku t=4 , slijedi da će nova instanca od t2 biti prihvaćena. Ali stara je već zakašnjela za t3

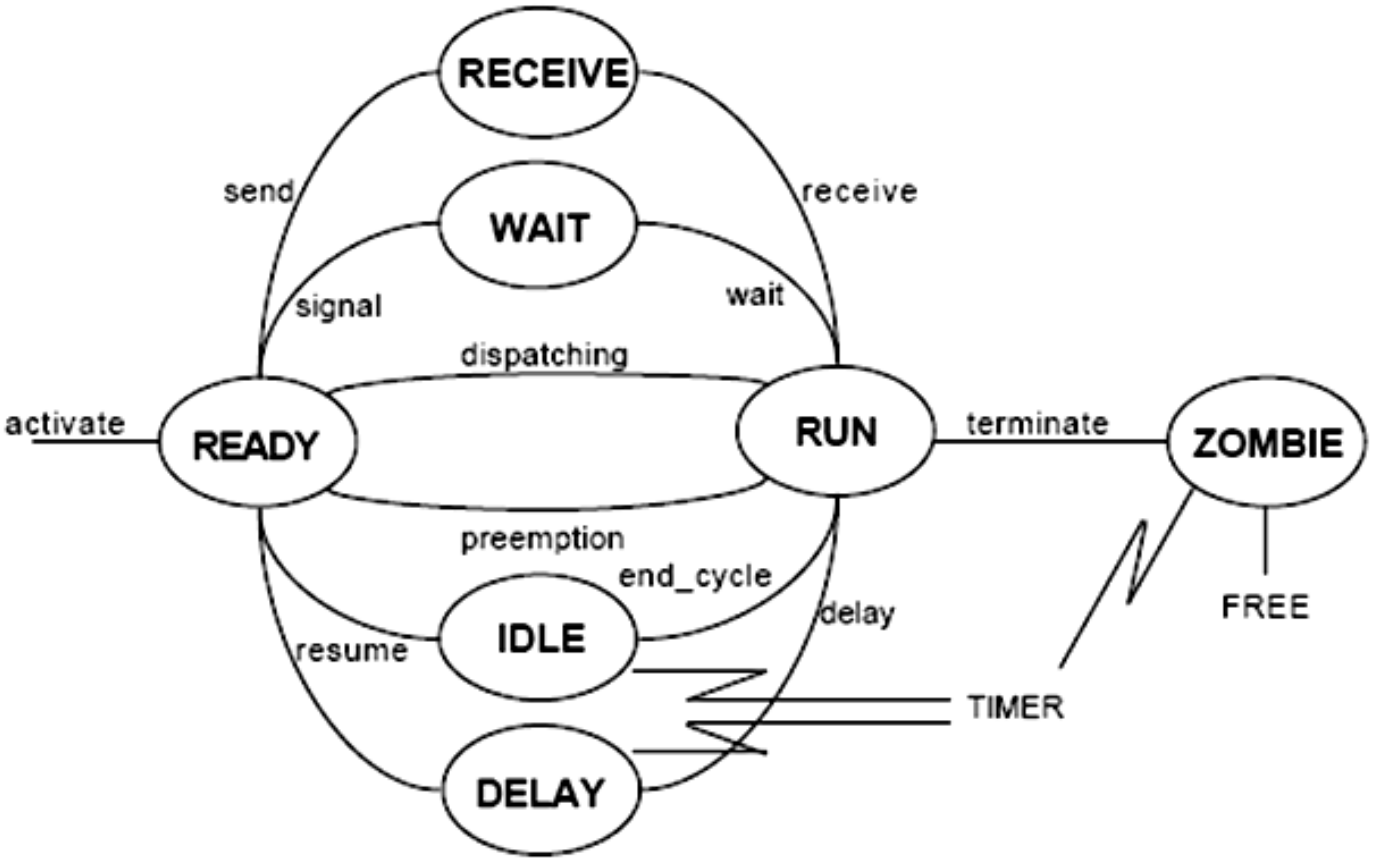
DESIGN OF RTOS SYSTEMS

Riješenje problema abortiranja korištenjem ZOMBIE stanja



DESIGN OF RTOS SYSTEMS

Kompletan dijagram RT tranzicije stanja



DESIGN OF RTOS SYSTEMS

DICK dijagram tranzicije stanja

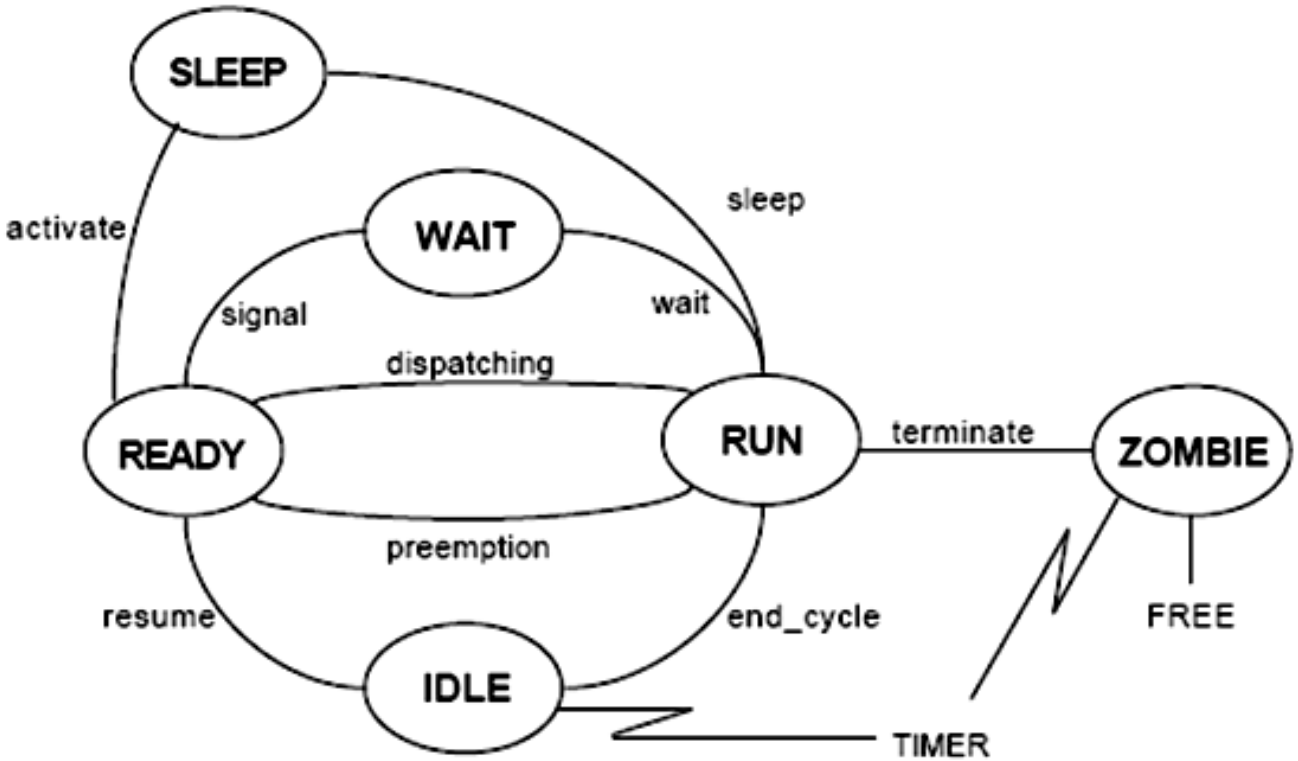
* U DICK-u stanja (i sa njima vezani servisi) RECEIVE i DELAY nisu prisutni (ali se mogu lako dodati).

* Novo stanje SLEEP se uvodi za aperiodične taskove. Aperiodični task ulazi u SLEEP izvršavanjem sleep primitive. Task napušta ovo stanje (prebacivanjem u READY stanje) , samo kada eksplicitna aktivacija je izvršena nad drugim taskom.

*Kreiranje taska stavlja ovaj task u SLEEP stanje. Ovo ima samo tehničke rezone da bi se smanjio runtime overhead.

DESIGN OF RTOS SYSTEMS

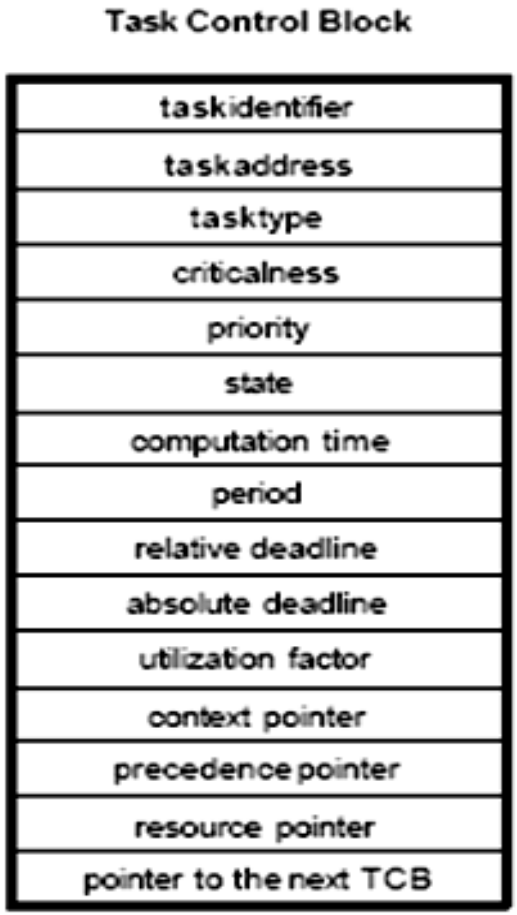
DICK dijagram tranzicije stanja



DESIGN OF RTOS SYSTEMS

Strukture podataka

U svakom OS informacija o tasku se pohranjuje u strukturu podataka koja se zove : Task control block (TCB) , tj. kontrolni blok taska.



DESIGN OF RTOS SYSTEMS

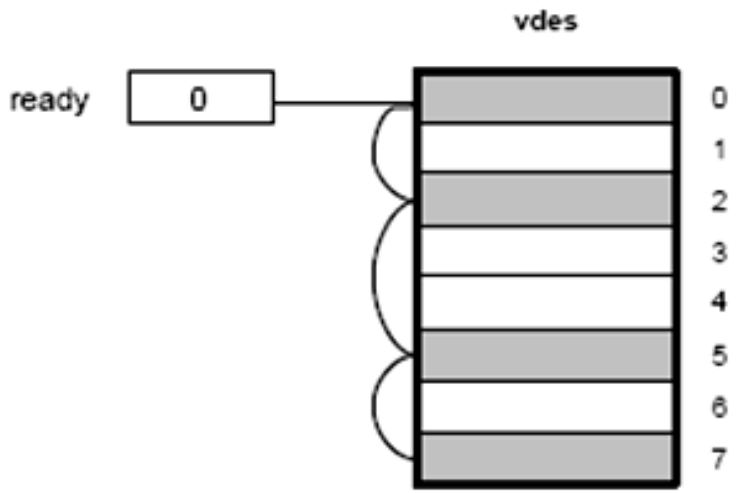
Pohranjivanje kontrolnih blokova taska

U okviru DICK-a :

TCB je element polja (array) vdes [MAXPROC] , velićine jednake maksimalnom broju taskova kojima upravlja kernel.

Svaki TCB se identificira sa jedinstvenim indeksom jednakim pozicijom u vdes. Red taskova se može identificirati pomoću indeksa vrha reda.

Primjer na narednoj slici pokazuje READY red unutar vdes:

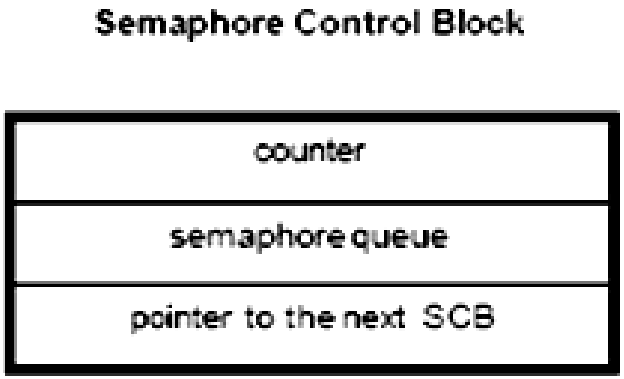


DESIGN OF RTOS SYSTEMS

Kontrolni blok semafora

Informacija o semaforu se pohranjuje u strukturu koja se naziva :

Semaphore control block (SCB)



Svaki SCB je element polja vsem [MAXSEM] .

Organizacija ove strukture je slićna onoj za vdes.

DESIGN OF RTOS SYSTEMS

Prikaz implementacije DICK struktura: vdes, scb

```
struct  scb   {

     int            count ;                 /*  semaphore count          */
     queue          qsem ;                  /*  semaphore queue          */
     sem            next ;                  /*  pointer to the next      */
};
```

```
struct  tcb         vdes[MAXPROC] ;         /* tcb array                 */
struct  scb         vsem[MAXSEM] ;          /* scb array                 */
```

DESIGN OF RTOS SYSTEMS

Menadjment vremena

Cilj: Generisanje vremenske reference

Pristup : - Tajmersko kolo se programira da interaptira proces u fiksnim intervalima vremena.

- interno sistemsko vrijeme je predstavljeno kao integer varijabla.

Interval vremena programiran u tajmeru, definira jedinicu vremena u sistemu, koja se zove sistemski tick.

Kod DICK kernela: sistemsko vrijeme je predstavljeno sa long integer : sys_clock, i sadrži vrijednost tickova kao jedinica vremena, koje su pohranjene kao float varijabla .

```
Unsigned  long    sys_clock ;    /* system time          */
float             time_unit ;    /* unit of time (ms)    */
```

DESIGN OF RTOS SYSTEMS

Rezolucija nasprema životnom opsegu (life time)

U svakom trenutku vremena, sys_clock sadrži broj interapta koje je generisao tajmer , od trenutka inicijalizacije sistema.
Stvarno vrijeme koje je prošlo je:

$$t = n * time\_unit$$

Maksimalno vrijeme koje se može predstaviti od strane kernela naziva se životnim opsegom (lifetime). Za fiksna predstavljanja n (napr. 32 bitni integer), zavisi od time_unit

tick	lifetime
1 ms	50 days
5 ms	8 months
10 ms	16 months
50 ms	7 years

DESIGN OF RTOS SYSTEMS

Upravljanje sa tajmerskim interaptom

Kod RTOS sistema upravljanje sa tajmerom za interapte je krucijalno.

Njegova uloga je da:

- ažurira vrijednost internog vremena
- provjeri za potencijalne propuste za deadline, zbog nekorektne WCET (worst case execution time) procjene
- monitoring životnog vijeka
- aktiviranje periodičnih taskova u IDLE stanju
- budjenje taskova u DELAY stanju
- provjera za uslove deadlocka
- završavanje taskova u ZOMBIE stanju

DESIGN OF RTOS SYSTEMS

DICK rutina za upravljanje tajmerom za interapte

* Pohranjuje kontekst taskova u izvršenju
* Inkrementira sistemsko vrijeme
* Ako je trenutno vrijeme veće nego sistemski životni opseg, generisaće grešku tajminga
* ako je tekuće vrijeme veće nego neki tvrdi deadline, generisaće grešku : time-overflow

* budi IDLE taskove, ako ih ima, koji treba da započnu novi period
* ako je barem jedan task bio probudjen , poziva rasporedjivač (scheduler)
* otklanja sve ZOMBIE taskove za koje je njihov deadline prošao

DESIGN OF RTOS SYSTEMS

* puni kontekst tekućeg taska
*vraća se iz interapta

Klase taskova i algoritam rasporedjivanja

U DICK-u postoje samo dvije klase taskova:

* HARD taskovi, koji imaju kritični deadline
* non-real time (NRT) taskovi, koji imaju fiksan prioritet

Aktivacija HARD taskova zavisi od toga kako je instanca okončana:

* pomoću end_cycle : task se stavlja u IDLE stanje i automatski aktivira pomoću tajmera na početku svog narednog perioda.
* pomoću end_aperiodic : task se stavlja u SLEEP stanje , odakle može biti ponovno aktiviran (resumed) , samo pomoću eksplicitne aktivacije.

DESIGN OF RTOS SYSTEMS

Rasporedjivanje u DICK-u

* Tvrdi (hard) taskovi se rasporedjuju koristeći EDF
* NRT taskovi se izvršavaju u pozadini na bazi njihovih prioriteta

*Da bi se integrisala rasporedjivanja ovih klasa taskova (u samo jednom redu rasporedjivanja) , potrebno je da se prioriteti NRT taskova transformišu u deadline koji su uvjek veći od HARD deadlajna.

INICIJALIZACIJA

RT okruženje podržavano od strane DICK, starta izvršavanjem ini_system primitive. Nakon što je ova funkcija izvršena, tada glavni program postaje NRT task u kojem novi konkurentni taskovi se mogu kreirati.

DESIGN OF RTOS SYSTEMS

Aktivnosti koje izvršava ini_system.

* inicijalizacija svih redova u kernelu
* postavljanje svih interapt vektora
* pripremanje TCB pridruženo sa glavnim procesom
* postavljanje perioda tajmera za sistemski tick

DESIGN OF RTOS SYSTEMS

Primitive kernela

Nisko nivovske primitive: save context

```
/*------------------------------------------------------------------------------------------------------------ */
/*    save_context--of the task in execution                                                                   */
/*------------------------------------------------------------------------------------------------------------*/
void        save_context(void)
{
int         *pc;                        /*pointer to the context of pexe*/
      <disable interrupts>
      pc = vdes[pexe].context;
      pc[0] = <register_0>              /*save register 0*/
      pc[1] = <register_1>              /*save register 1*/
      pc[2] = <register_2>              /*save register 2*/
            .........

      pc[n] = <register_n>              /*save register n*/



}
```

DESIGN OF RTOS SYSTEMS

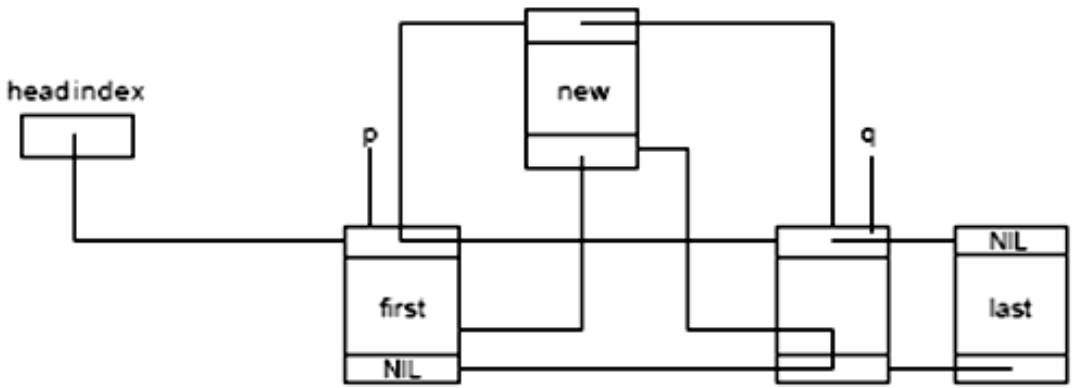
Nisko nivovske primitive: Load context

```
/*------------------------------------------------------------------------------------------------------------------------ */
/*    load_context -- of the task to be executed                                                                          */
/*------------------------------------------------------------------------------------------------------------------------*/
void        load_context(void)
{
int         *pc;                            /*pointer to the context of pexe*/

      pc = vdes[pexe].context;
       <register_0> = pc[0]                 /*load register 0*/
       <register_1> = pc[1]                 /*load egister 1*/
            .........

        <register_n> = pc[n]                /*load register n*/

      <return from interrupts>

}
```

DESIGN OF RTOS SYSTEMS

Menadjment lista

* DICK : Taskovi se rasporedjuju prema EDF algoritmu: slijedi da svi redovi u kernelu su uredjeni sa opadajućim deadlajnima, (vrh reda = task sa najranijim deadlajnom).
* Sve liste su implementirane koristeći biderkcionalne pokazivače (pointere) (next, prev).
* Funkcija Insert se poziva sa dva parametra: indeks taska koji se treba unjeti i pokazivač na red.
* Koristi dva pomoćna pokazivača : p i q.



DESIGN OF RTOS SYSTEMS

Primitiva rasporedjivanja verificira da li task koji se izvršava je onaj koji ima najraniji deadline.

```
*-------------------------------------------------------------------------------------------------*/
/*    schedule ---- selects the task with the earliest deadline                                    */
/*-------------------------------------------------------------------------------------------------*/
void        schedule( void)
{
      if (firstdline(ready) < vdes[pexe].dline)  {
            vdes[pexe].state =READY;
            insert (pexe, &ready);
            dispatch( );
      }
}
```

DESIGN OF RTOS SYSTEMS

Mehanizam rasporedjivanja : dispečing

Primitiva dispatch jednostavno doznačuje procesor na prvi task koji je spreman:

```
*-------------------------------------------------------------------------------------------------*/
/*    dispatch --- assigns the cpu to the first ready task                                         */
/*-------------------------------------------------------------------------------------------------*/
void                    dispatch( void)
{
        pexe = getfirst(&ready);
        vdes[pexe].state = RUN;
}
```

DESIGN OF RTOS SYSTEMS

Rutina za upravljanje tajmerom interapta

Rutina wake_up radi slijedeće aktivnosti:

*** inkrementira varijablu sys_clock**
*** provjerava za životni vijek sistema**

*** provjerava za moguće propuste u deadlajnima**

*** otklanja one taskove u ZOMBIE stanju čiji deadlajni su istekli**
*** ponovo preuzima (resume) one periodične taskove u IDLE stanju na početku njihovog slijedećeg perioda**
Ako najmanje jedan task je bio ponovno aktiviran , tada rasporedjivač se poziva i desiće se priempcija.

DESIGN OF RTOS SYSTEMS

Rutina DICK-a za upravljanje tajmerom interapta

```
*-------------------------------------------------------------------------------------------------*/
/*   wake_up --- timer interrupt handling routine                                                  */
/*-------------------------------------------------------------------------------------------------*/
void                wake_up( void)
{
    proc            p;
    int             count = 0;
        save_context ( );
        sys_clock++;
        if (sys_clock >= LIFETIME) abort (TIME_EXPIRED);
        if (vdes[pexe].type == HARD)
                if(sys_clock  > vdes[pexe].dline)
                        abort (TIME_OVERFLOW);
        while ( !empty(zombie) && (firstdline(zombie) <= sys_clock))
                {
                        p = getfirst (&zombie);
                        util_fact = util_fact - vdes[p].util;
                        vdes[p].state = FREE;
                        insert (p, &freetcb);
                }
}
```

DESIGN OF RTOS SYSTEMS

Rutina DICK-a za upravljanje tajmerom interapta - nastavak

```
While ( !empty(idle) && (firstdline(idle) <= sys_clock))
    {
        p = getfirst(&idle);
        vdes[p].dline += (long)vdes[p].period;
        vdes[p].state = READY;
        insert (p, &ready);
        count ++;
    }
    if (count > 0) schedule ();
    load_context ( );
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : Create

```
        vdes[p].name = name ;
        vdes[p].addr = addr ;
        vdes[p].type = type;
        vdes[p].state = SLEEP;
        vdes[p].period = (int)(period / time_unit);
        vdes[p].wcet = (int)(wcet / time_unit); ;
        vdes[p].util= wcet / period ;
        vdes[p].prt = (int)period ;
        vdes[p].dline = MAX_LONG + (long) (period - PRT_LEV);
        if(vdes[p]. type = HARD)
              if( !guarantee(p)) return (NO_GUARANTEE);
        <initialize proces stack>
        <enable cpu interrupts>
        return(p);
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : Create - nastavak

```
*------------------------------------------------------------------------------------------------------------*/
/*    create --- creates a task and puts it in sleep state                                                  */
/*------------------------------------------------------------------------------------------------------------*/
proc                create(
                    char      name[MAXLEN +1],      /* task name              */
                    proc      (*addr)(),            /* task address           */
                    int       type,                 /* type (HARD, NRT)       */
                    float     period,               /* period or priority     */
                    float     wcet )                /*execution time          */

{
proc          p;
          <disable cpu interrupts>
          p =  getfirst( &freetcb);
          if (p == NIL) abort(NO_TCB);
```

DESIGN OF RTOS SYSTEMS

Menadjment task : garancija izvodivosti

```
*-------------------------------------------------------------------------------------------------*/
/*    guarantee -- guarantees the feasibility of a hard task                                       */
/*-------------------------------------------------------------------------------------------------*/
int          guarantee( proc  p)
{
     util_fact = util_fact + vdes[p].util;
     if (util_fact > 1.0)  {
          util_fact = util_fact - vdes[p].util;
          return(FALSE);
     }
     else return (TRUE);
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : Activate

Ovaj sistemski poziv izvršava tranziciju : SLEEP –READY

```
/*-------------------------------------------------------------------------------------------------*/
/*    activate -- inserts a task in the ready queue                                                */
/*-------------------------------------------------------------------------------------------------*/
int          activate( proc  p)
{
     save_context ( );
     if (vdes[p].type == HARD)
          vdes[p].dline = sys_clock + (long) vdes[p].period ;
     vdes[p].state = READY ;
     insert (p, &ready);
     schedule ();
     load_context () ;
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : Sleep

Sistemski poziv sleep izvršava tranziciju RUN → SLEEP :
Primjetimo da ova primitiva djeluje na pozivajući task:

```
*-------------------------------------------------------------------------------------------------*/
/*    sleep -- suspends itself in a sleep state                                                    */
/*-------------------------------------------------------------------------------------------------*/
void      sleep( void)
{
     save_context ( );
     vdes[p].state = SLEEP ;
     dispatch ();
     load_context () ;
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : End_Cycle

Ovaj task realizuje završavanje periodične instance. Kernel mora biti informisan o vremenu u kojem tajmer mora ponovno da aktivira posao.

```
/*-------------------------------------------------------------------------------------------------*/
/*     end_cycle -- inserts a task in the idle queue                                               */
/*-------------------------------------------------------------------------------------------------*/
void end_cycle( void)
{
long          dl;
              save_context ( );
              dl = vdes[pexe].dline;
              if (sys_clock < dl)    {
                            vdes[pexe].state= IDLE ;
                            insert (pexe , &idle);
              }
              else {
                            dl = dl + (long)vdes[pexe].period;
                            vdes[pexe].dline= dl;
                            vdes[pexe].state = READY;
                            insert (pexe , &ready);
              }
              dispatch () ;
              load_context () ;
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : Primjer tipičnog taska

Tipični periodični task upravljan od strane DICK-a izgleda kao slijedeći kod:

```
proc     cycle()
{
     while (TRUE)  {
          <periodic code>
          end_cycle () ;
     }
}
```

DESIGN OF RTOS SYSTEMS

Menadjment task : End Process

Sistemski poziv end_process završava task koji se izvršava (tj. pozivajući) task. Ako je to HARD , onda če on ići u ZOMBIE stanje.

```
/*-------------------------------------------------------------------------------------------------------------- */
/*    end_process -- terminates the running task                                                                 */
/*-------------------------------------------------------------------------------------------------------------- */
void      end_process( void)
{
     <disable cpu interrupts>
     if (vdes[pexe].type == HARD)
          insert (pexe, &zombie);
     else {
          vdes[pexe].state = FREE;
          insert (pexe , &freetcb);
     }
     dispatch ();
     load_context () ;
}
```

DESIGN OF RTOS SYSTEMS

Task menadjment : Kill

Sistemski poziv kill završava proces indiciran sa parametrom. Ako je tvrd (HARD) ići će u ZOMBIE stanje.

```
/*------------------------------------------------------------------------------------------------------------ */
/*    kill ---terminates a task                                                                                */
/*------------------------------------------------------------------------------------------------------------ */
void        kill( proc p)
{
     <disable cpu interrupts>
     if (pexe == p) {
          end_process();
          return;
     }
      if(vdes[p].state == READY)  extract (p,&ready);
      if(vdes[p].state == IDLE)  extract (p,&idle);
      if(vdes[p].type== HARD)
          insert (p, &zombie);
     else {
            vdes[p].state = FREE ;
            insert (p, &freetcb);
     }
     <enable cpu interrupts>
 }
```

DESIGN OF RTOS SYSTEMS

SEMAFORI

DICK upravlja sa sinhronizacijom i medjusobnim isključenjem putem semafora.

Postoje četiri operacije koje se izvršavaju sa onemogućenim interaptima.

* Newsem doznačuje novi SCB i inicijalizira brojač za vrijednost koja je prošla.
* Delsem dealocira SCB, unoseći ga u red slobodnih semafora.
* Wait ćeka na dogadjaj udružen sa semaforom. Ako je brojač semafora ›0 , on se dekrementira i task se nastavlja. Ako je ≤ 0, task je blokiran i umeće se u red za semafor.

* Signal omogućava tasku da signalizira dogadjaj koji je udružen sa semaforom. Ako nema blokiranih taskova na ovom semaforu brojač se povećava. Ako ima blokiranih taskova, onaj sa najranijim deadlajnom se vadi.

DESIGN OF RTOS SYSTEMS

Semafori: Newsem

```
/*------------------------------------------------------------------------------------------------------ */
/*    newsem -- allocates and initializes a semaphore                                                    */
/*------------------------------------------------------------------------------------------------------ */
sem         newsem( int n)
{
sem         s;
      <disable cpu interrupts>
      s = freesem ;                              /*first free semaphore index*/
      if (s == NIL) abort (NO_SEM);
      freesem = vsem[s].next ;                   /*update the free  sem list  */
       vsem[s].count = 0;
      vsem[s].qsem = NIL;                        /*initilize counter          */
      <enable cpu interrupts>
      return(s);
}
```

DESIGN OF RTOS SYSTEMS

Semafori : Delsem

```
/*------------------------------------------------------------------------------------------------------ */
/*    delsem -- deallocates   a semaphore                                                                 */
/*------------------------------------------------------------------------------------------------------ */
void        delsem( sem s)
{
      <disable cpu interrupts>
      vsem[s].next = freesem;                    /*inserts  s at the head       */
      freesem = s;                               /*of the freesem list          */
      < enable cpu interrupts>
}
```

DESIGN OF RTOS SYSTEMS

Semafori : Wait

```
/*------------------------------------------------------------------------------------------------------ */
/*    wait -- waits for an event                                                                        */
/*------------------------------------------------------------------------------------------------------ */
void        wait ( sem s)
{
     <disable cpu interrupts>
     if (vsem[s].count > 0 ) vsem[s].count -- ;
     else {
          save_context ();
          vdes[pexe].state = WAIT;
          insert(pexe , &vsem[s].qsem);
          dispatch();
          load_context();
     }
     <enable cpu interrupts>
}
```

DESIGN OF RTOS SYSTEMS

Semafori : Signal

```
/*------------------------------------------------------------------------------------------------------------ */
/*    signal -- signals anevent                                                                                 */
/*------------------------------------------------------------------------------ ---------------------------- */
void        signal ( sem s)
{
proc      p;
     <disable cpu interrupts>
     if ( !empty(vsem[s].qsem ))     {
          p = getfirst (&vsem[s].qsem);
          vdes[p].state = READY;
          insert(p , &ready);
           save_context ();
          schedule ( );
          load_context();
     }
     else              vsem[s].count ++ ;
     <enable cpu interrupts>
}
```