MECANISM DE DISPECERIZARE A TASKURILOR ÎNTR-UN EXECUTIV DE TIMP REAL

Nicolae Robu

Universitatea Tehnică din Timişoara

Rezumat: Lucrarea are ca obiect soluţiile adoptate pentru dispecerizarea taskurilor în executivul de timp real **RTC86**, conceput şi realizat de autor, sub sistemul de operare **MS-DOS**, ca extensie a mediului de programare **TURBO C**.
Se prezintă politica de dispecerizare, surprinzînd-o printr-un model. Este abordată implementarea listei de aşteptare la procesor.
Sunt descrise funcţiile mecanismului de dispecerizare. Textele C ale acestora sunt, inclusiv ele, redate.
Cuvinte cheie: politică de dispecerizare, priorităţi neunivoce, listă de aşteptare la procesor, mecanism de dispecerizare.

1. Introducere

Avînd în vedere că, în aplicaţiile de informatică industrială, intervin deopotrivă activităţi de prioritate diferită şi activităţi echiprioritare şi ţinînd seamă de restricţiile temporale severe, pe care aceste aplicaţii le impun procesului de comutare a taskurilor, se apreciază că cea mai raţională politică de dispecerizare pentru executivele de timp real orientate spre această clasă de aplicaţii este cea bazată pe priorităţi neunivoce [4]. Ea presupune că fiecare task are o anumită prioritate, diferită de sau identică cu a altora. În procesul de dispecerizare, primează prioritatea, taskurile de acelaşi prioritate fiind tratate prin rotaţie [2].

În cele ce urmează, se prezintă soluţiile adoptate pentru asigurarea dispecerizării printr-o politică bazată pe priorităţi neunivoce, în cadrul executivului de timp real **RTC86**, conceput şi realizat de autor. Se oferă un model al politicii de dispecerizare, se prezintă implementarea listei de aşteptare la procesor, se descriu funcţiile mecanismului de dispecerizare; de asemenea, sunt redate textele C ale acestora.

Se precizează că **RTC86** este un executiv grefat pe sistemul de operare **MS-DOS**, prezentîndu-se ca o extensie a mediului de programare **TURBO C**. Cu ajutorul acestei extensii, mediului **TURBO C** i se conferă facilităţi de dezvoltare a programelor multitasking în timp real. Prin capabilităţile de care dispune, prin eficienţa de utilizare a procesorului şi timpii de răspuns pe care le asigură la nivelul programelor de aplicaţie, executivul **RTC86** se înscrie în rîndul instrumentelor software destinate domeniului informaticii industriale [3,4].

2. Modelul politicii de dispecerizare

Politica de dispecerizare a executivului de timp real **RTC86** se circumscrie modelului reprezentat în figura 1, model ce corespunde unei situaţii concrete, considerată spre exemplificare.

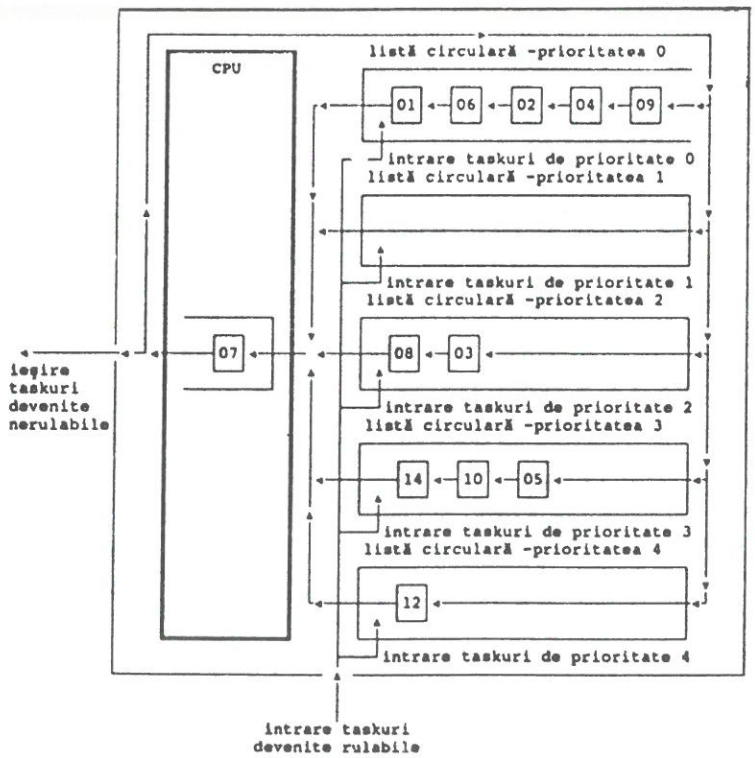

Figura 1. Modelul politicii de dispecerizare a executivului de timp real RTC 86

În situația concretă surprinsă în model, se remarcă următoarele:

- lista de așteptare la procesor cuprinde cinci liste circulare, corespunzătoare celor cinci niveluri de prioritate, aflate la dispoziția taskurilor rulabile; lista de prioritate 1 este vidă;
- taskul 07, aparținînd listei circulare a nivelului de prioritate 0, este în rulare, ocupînd, în consecință, ultima poziție a acestei liste;
- în ipoteza conservării conținutului listei de așteptare la procesor, prin procese de comutare succesive, vor intra în rulare taskurile 01, 06, 02, 04, 09, apoi din nou 07, 01 ș.a.m.d.;
- taskurile de pe nivelul de prioritate 2 pot intra în rulare imediat după vidarea listei de prioritate 0, întrucît lista de prioritate 1 este vidă; dacă vidarea listei de prioritate 0 se produce, va intra în rulare mai întîi taskul 08, trecînd, cu această ocazie, pe ultima poziție a listei ce-l conține; îi va urma taskul 03 etc.
- în general, taskurile de pe un nivel de prioritate pot intra în rulare doar dacă toate listele corespunzătoare nivelurilor de prioritate mai puternice sunt vide.

3. Lista de așteptare la procesor

Lista de așteptare la procesor, în care se înscriu toate taskurile rulabile, reprezintă o înlănțuire de liste circulare, fiecare dintre acestea corespunzînd taskurilor de o anumită prioritate; ordinea de succedare a listelor circulare este cea de slăbire a priorităților taskurilor înscrise în ele.

O asemenea înlănțuire de liste, ca și listele însele, pot fi implementate cu ajutorul a două tablouri corelate: un tablou al taskurilor și un tablou al priorităților [4].

Tabloul taskurilor implementează listele circulare. Un element al tabloului, împreună cu indicele său, reprezintă un nod al unei liste. Indicele are semnificația de index al taskului aferent nodului, iar elementul -semnificația de index al succesorului acestui task. Considerîndu-se că în sistem pot fi maximum *MAX_TSK* taskuri, tabloul taskurilor va avea *MAX_TSK* elemente, cu indicii *0...MAX_TSK-1*. Valoarea elementelor corespunzătoare taskurilor neprinse în nici o listă va fi *MAX_TSK*.

Tabloul priorităților are *MAX_TSK+1* elemente, cu indicii *0...MAX_TSK*. Indicii *0...MAX_TSK-1* se pun în corespondență biunivocă cu prioritățile pe care le pot avea taskurile. Elementele de tablou cu acești indici primesc ca valori indexurile taskurilor aflate în coada listelor corespunzătoare priorităților respective. La fiecare schimbare a cozii unei liste, valoarea elementului asociat ei este actualizată. Dacă o listă de o anumită prioritate este vidă, valoarea elementului cu indicele egal cu respectiva prioritate va fi *MAX_TSK*.

Elementul de indice *MAX_TSK* are un rol aparte. Valoarea sa indică cel mai puternic nivel de prioritate a cărui listă circulară este nevidă, respectiv nivelul de prioritate al taskului aflat în rulare. Cînd toate listele circulare sunt vide, elementul de indice *MAX_TSK* are valoarea *MAX_TSK*.

Notînd cu *_lap_t* tabloul taskurilor și cu *_lap_p* tabloul priorităților, rezultă, evident, că valoarea expresiei:

$$\_lap\_p\ [\_lap\_p\ [MAX\_TSK]]$$

reprezintă indexul taskului aflat în rulare, iar valoarea expresiei:

$$\_lap\_t\ [\_lap\_p\ [\_lap\_p\ [MAX\_TSK]]]$$

reprezintă indexul taskului din capul listei căreia îi aparține taskul aflat în rulare.

Presupunînd *MAX_TSK = 15*, pentru situația din figura 1, tablourile *_lap_t* și *_lap_p* vor arăta astfel:

MAX_TSK−1 ↓

_lap_t

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
15	06	04	08	09	14	02	01	03	07	05	15	12	15	10

MAX_TSK ↓

_lap_p

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
07	15	03	05	12	15	15	15	15	15	15	15	15	15	15	00

Figura 2. Tablourile *_lap_t* și *_lap_p* corespunzătoare situației din figura 1.

4. Funcțiile mecanismului de dispecerizare

Mecanismul de dispecerizare a taskurilor din executivul de timp real RTC86 cuprinde patru funcții [1], cu ajutorul cărora se operează asupra listei de așteptare la procesor. Aceste funcții sunt prezentate sintetic în figura 3.

```
void ins_lap (usshort ind_tsk, usshort prior)
/*          inserează în lap                          */
/* taskul cu indexul ind_tsk și prioritatea prior     */
  void elim_lap (usshort ind_tsk, usshort prior)
/* elimină din lap taskul cu indexul ind_tsk          */
          usshort urm_lap (void)
/* determină indexul taskului aflat în capul lap      */
void init_lap     (void)
/* inițializează lista de așteptare la procesor       */
```

Notă: *usshort* reprezintă prescurtare pentru *unsigned short*.

Figura 3. Funcțiile de gestionare a listei de așteptare la procesor.

Funcția *ins_lap ()*

Această funcție are, în primul rînd, sarcina să verifice dacă lista circulară asupra căreia trebuie să opereze este vidă sau nu.

În cazul în care această listă este vidă, inserarea în

ea a taskului cu indexul precizat prin argumentul *ind_tsk* se face prin înscrierea în elementul de indice *ind_tsk* din tabloul taskurilor a valorii *ind_tsk*, întrucît taskul în cauză se va succede în respectiva listă pe el însuși, reprezentînd, deopotrivă, capul și coada ei. Totodată, valoarea argumentului *ind_tsk* se înscrie, de astă dată cu semnificația de coadă de listă, în elementul din tabloul priorităților al cărui indice este egal cu valoarea argumentului *prior*.

În cazul în care lista nu este vidă, inserarea în ea a taskului cu indexul precizat prin argumentul *ind_tsk* presupune instalarea acestui task în capul listei corespunzătoare priorității indicată de argumentul *prior*. Acest lucru se asigură prin copierea în elementul din tabloul taskurilor cu indicele *ind_tsk* a indexului taskului surprins în postura de cap de listă, index aflat în elementul din același tablou corespunzător cozii listei în cauză, urmat de înscrierea valorii *ind_tsk* în acest element corespunzător cozii. Se reamintește că indicele elementului din tabloul taskurilor asociat cozii unei liste de o anumită prioritate se găsește în tabloul priorităților, în elementul aferent priorității respective.

Revenim la cazul în care lista asupra căreia trebuie să opereze funcția *ins_lap ()* este găsită vidă, pentru a adăuga că, în acest caz, se impune, suplimentar, compararea priorității taskului nou inserat cu valoarea înscrisă în elementul de indice *MAX_TSK* din tabloul priorităților (valoare care reprezintă cea mai puternică prioritate căreia îi corespunde o listă circulară nevidă) și, dacă este nevoie, înlocuirea acestei valori cu cea a argumentului *prior*.

Rezultă, așadar, pentru funcția *ins_lap ()*, textul din figura 4.

```
1   void ins_lap (usshort ind_tsk, usshort prior
2  {
3   if (_lap_p [prior]==MAX_TSK) {
4     _lap_t [ind_tsk]=ind_tsk;
5     _lap_p [prior]=ind_tsk;
6     if (prior<_lap_p [MAX_TSK]) {
7       _lap_p [MAX_TSK]=prior;
8     }
9   }
10  else {
11    _lap_t [ind_tsk]=_lap_t [_lap_p [prior]];
12    _lap_t [_lap_p [prior]]=ind_tsk;
13  }
14 }
```

Figura 4. Textul funcției *ins_lap ()*.

Funcția *elim_lap ()*

Această funcție trebuie, în primul rînd, să efectueze identificarea în tabloul taskurilor a elementului care conține indexul taskului de eliminat, index precizat prin argumentul *ind_tsk*.

Procesul de căutare prin care se realizează această identificare se poate efectua analizînd succesiv conținuturile elementelor tabloului respectiv, de exemplu, de la primul către ultimul.

O soluție mai eficientă, în ceea ce privește timpul necesar găsirii elementului căutat, se poate obține prin limitarea căutării doar la nivelul subsetului de elemente ale tabloului taskurilor, implicate în implementarea listei circulare în care trebuie să figureze taskul de eliminat. Dispunîndu-se, la intrarea în funcție, de prioritatea taskului, ca valoare a argumentului *prior*, prin citirea valorii înscrise în elementul din tabloul priorităților cu indicele egal cu această prioritate, se obține indexul taskului aflat în coada listei în cauză. Cu elementul din tabloul taskurilor avînd indicele egal cu acest index, se poate începe procesul de căutare, ținînd seamă că, valoarea înscrisă în el, ca de altfel și în celelalte cu care el este înlănțuit, reprezintă fie indexul taskului de eliminat, fie indicele elementului următor în care acest index s-ar putea găsi.

Odată identificat elementul care conține indexul taskului de eliminat, se impune tratarea diferită a cazului în care acest task este coadă de listă, de cazul în care el nu este așa ceva. O comparare a valorii argumentului *ind_tsk* cu valoarea înscrisă în elementul din tabloul priorităților desemnat de argumentul *prior* pune în evidență cazul în speță: dacă cele două valori sunt identice, taskul de eliminat este coadă de listă, altfel, nu.

În primul caz, dacă indicele elementului din tabloul taskurilor în care a fost găsit indexul taskului de eliminat este egal cu acest index, înseamnă că acest task era unicul din listă și, drept urmare, prin eliminarea lui, lista rămîne vidă. Eliminarea se efectuează, în această situație, prin înscrierea valorii *MAX_TSK*, atît în elementul din tabloul taskurilor care a conținut indexul taskului de eliminat, cît și în elementul din tabloul priorităților, corespunzător priorității acestui task.

Tot în acest prim caz, dacă indicele elementului din tabloul taskurilor în care a fost găsit indexul taskului de eliminat nu este egal cu acest index, înseamnă că acest task nu este unicul în listă. Eliminarea presupune, în această situație, două acțiuni. Prima consistă în instalarea în coada listei a predecesorului taskului de eliminat, prin înscrierea indexului său, egal cu indicele elementului din tabloul taskurilor în care a fost găsit indexul taskului de eliminat, în elementul din tabloul priorităților cu indicele precizat prin valoarea argumentului *prior*. A doua acțiune constă în copierea în elementul din tabloul taskurilor în care a fost găsit indexul taskului de eliminat a indexului succesorului acestui task, aflat în acest tablou în elementul de indice *ind_tsk*.

În al doilea caz, caracterizat prin faptul că taskul de eliminat nu este coadă de listă, eliminarea se efectuează prin simpla copiere în elementul din tabloul taskurilor în care a fost găsit indexul taskului de eliminat a indexului succesorului acestui task, aflat, cum s-a precizat mai sus, în elementul acestui tablou cu indicele *ind_tsk*.

Revenim la cazul în care taskul de eliminat este

unicul din lista circulară corespunzătoare priorității sale, pentru a adăuga că, în acest caz, se impune, suplimentar, să se verifice dacă nu cumva această prioritate era cea mai puternică dintre cele care aveau asociate liste circulare nevide.

Verificarea se efectuează prin compararea valorii argumentului *prior* cu valoarea înscrisă în elementul de indice *MAX_TSK* din tabloul priorităților. Egalitatea semnifică faptul că prioritatea taskului eliminat era cea mai puternică și conduce la necesitatea actualizării valorii elementului de indice *MAX_TSK*. Actualizarea se efectuează prin înscrierea în elementul în discuție a valorii celui mai mic indice din tabloul priorităților, care are asociat un element ce conține o valoare diferită de *MAX_TSK*.

Textul funcției *elim_lap ()*, scris în conformitate cu cele de mai sus, face obiectul figurii 5.

```
1  void elim_lap (usshort ind_tsk, usshort prior)
2  {
3   register usshort j;
4   j=_lap_p [prior];
5   if (j!=MAX_TSK) {
6    while((_lap_t[j]!=ind_tsk)
7                &&(_lap_t [j]!=_lap_p [prior])) {
8     j=_lap_t [j];
9    }
10   if (_lap_t [j]==ind_tsk) {
11    if (ind_tsk==_lap_p [prior]) {
12     if (j==ind_tsk) {
13      _lap_t [ind_tsk]=MAX_TSK;
14      _lap_p [prior]=MAX_TSK;
15      if (_lap [MAX_TSK]==prior) {
16       j=prior;
17       do {
18        j=j+1;
19       } while (_lap_p [j]==MAX_TSK);
20       _lap_p [MAX_TSK]=j;
21      }
22     }
23     else {
24      _lap_p [prior]=j;
25      _lap_t [j]=_lap_t [ind_tsk];
26      _lap_t [ind_tsk]=MAX_TSK;
27     }
28    }
29    else {
30     _lap_t [j]=_lap_t [ind_tsk];
31     _lap_t [ind_tsk]=MAX_TSK;
32    }
33   }
34  }
35 }
```

Figura 5 Textul funcției *elim_lap ()*.

Funcția *urm_lap ()*

Această funcție are sarcina de a determina și indica succesorul taskului aflat în rulare. Acest succesor este taskul aflat curent în capul listei circulare cu cea mai puternică prioritate. Indexul său se află înscris în elementul din tabloul taskurilor cu indicele dat de valoarea expresiei:

_lap_p [_lap_p [MAX_TSK]]

Totodată, funcția *urm_lap ()* asigură instalarea în coada listei circulare în cauză a taskului desemnat ca "următorul", înscriindu-i indexul în elementul din tabloul priorităților cu indicele *MAX_TSK*. Această operație este însoțită, implicit, de avansarea în cadrul listei cu o poziție și a celorlalte taskuri pe care ea le include.

Textul funcției *urm_lap ()* este redat în figura 6.

```
1  usshort urm_lap (void)
2  {
3   lap_p [_lap_p [MAX_TSK]]=
4            _lap_t [_lap_p [_lap_p [MAX_TSK]]];
5   return (_lap_p [_lap_p [MAX_TSK]]);
6  }
```

Figura 6. Textul funcției *urm_lap ()*.

Funcția *init_lap ()*

Această funcție are sarcina de a asigura inițializarea listei de așteptare la procesor, prin înscrierea valorii *MAX_TSK* în toate elementele tablourilor *_lap_t* [], respectiv *_lap_p* [].

Textul funcției *init_lap ()* este prezentat în figura 7.

```
1  void init_lap (void)
2  {
3   register usshort i;
4   for (i=0;i<MAX_TSK;i++) {
5    _lap_t [i]=MAX_TSK;
6    _lap_p [i]=MAX_TSK;
7   }
8   _lap_p [MAX_TSK]=MAX_TSK;
9  }
```

Figura 7. Textul funcției *init_lap ()*.

5. Concluzii

Soluțiile prezentate au dat satisfacție deplină în cadrul executivului de timp real **RTC86**. Simplitatea lor a asigurat eficiență procesului de dispecerizare. Este de menționat că timpul în care executivul **RTC86** asigură comutarea taskurilor este de circa 60 μs, pe un procesor **INTEL 80386**, la 33 MHz. Se amintește că timpii de comutare pretinși de domeniul informaticii industriale sunt inferiori pragului de 100 μs [4].

Bibliografie:

1. **ALLWORTH, S.T.**: Introduction to Real-Time Software Design, Macmillan Press Ltd., London, 1981.
2. **BALTAC V., ș.a.**: Sisteme interactive și limbaje conversaționale. Utilizare și proiectare, Editura Tehnică, București, 1984.
3. **PEREZ, J.P.**: Systèmes Temps Réel -Méthodes de Spécification et de Conception, Dunod, Paris, 1990.
4. **TSCHIRHART, D.**: Commande en Temps Réel, Dunod, Paris, 1990.