MECANISM DE SINCRONIZARE PRIN BLOCURI EVENIMENT

Nicolae Robu

Universitatea Tehnică din Timişoara

Rezumat: Lucrarea prezintă soluţiile adoptate în implementarea unui mecanism de sincronizare prin blocuri eveniment, în cadrul executivului de timp real **RTC86**, conceput şi realizat de autor, sub sistemul de operare **MS-DOS**, ca extensie a mediului de programare **TURBO C**. Se evocă noţiunea de *bloc eveniment*, iar apoi se arată modul de implementare a blocurilor eveniment propriu-zise, în executivul **RTC86**. Se pun în evidenţă funcţiile mecanismului de sincronizare prin blocuri eveniment, înglobat în executivul **RTC86**, şi principiile care au stat la baza implementării lor. Textul **C** al acestor funcţii este, şi el, redat.

Cuvinte cheie: *sincronizare, eveniment, proces, scheduler, aşteptare, "time-out".*

1. Introducere

După cum se cunoaşte, din punct de vedere conceptual [1,2], un bloc eveniment este un ansamblu format dintr-o variabilă booleană **B** şi o coadă **C**.

Valoarea "**1**" a variabilei semnifică faptul că un eveniment căruia ea i se asociază s-a produs, iar valoarea "**0**" - că evenimentul nu s-a produs. Evident, la crearea blocului eveniment, se asigură pentru variabilă valoarea "**0**", iar coada se videază.

Cînd un proces ajunge în stadiul în care necesită sincronizarea cu un eveniment, se autoblochează şi se înscrie în coada de aşteptare a blocului eveniment asociat evenimentului respectiv. Cînd evenimentul se produce, toate procesele înscrise în coadă de aşteptare la el sunt deblocate deodată. Transpare, din aceste precizări, faptul că mecanismul de sincronizare prin blocuri eveniment **nu asigură memorarea evenimentelor**. De aceea, variabila **B** nici nu trebuie să existe de facto; prezenţa ei în definiţia blocului eveniment este bazată exclusiv pe considerente conceptuale.

Se prezintă, în continuare, soluţiile adoptate pentru implementarea blocurilor eveniment şi a funcţiilor conexe, în executivul de timp real **RTC86**, conceput şi realizat de autor.

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. Tipul de dată *EVENT*

Implementarea unui mecanism de sincronizare prin blocuri eveniment presupune, înainte de toate, adoptarea unei accepţiuni concrete asupra noţiunii de bloc eveniment. S-a adoptat ca blocul eveniment să fie o *structură* formată din:

- un tablou de tipul *unsigned short*, cu *MAX_TSK* (numărul maxim al proceselor acceptabile în sistem) elemente, dedicat a servi drept coadă a blocului eveniment; s-a ales numele *coada* pentru acest tablou;
- o variabilă de tipul *pointer* către un întreg scurt fără semn, avînd rolul de a asista intrarea în şi ieşirea din coada blocului eveniment; s-a ales numele *pie* pentru această variabilă.

Dacă adoptăm numele *EVENT* pentru această structură, atunci ea va putea fi introdusă aşa cum se arată în figura 1.

```
typedef struct {
        usshort coada[MAX_TSK];
        usshort *pie;
      } EVENT;
```

Notă: *usshort* este prescurtare de la *unsigned short*.

Figura 1. Tipul de dată EVENT

Este firesc ca toate blocurile eveniment de care se dispune în sistem să fie grupate sub forma unui tablou. Dăm numele *_evn []* acestui tablou. Dacă admitem că numărul maxim de blocuri eveniment este *MAX_EVN*, atunci tabloul *_evn []*, care trebuie să fie, din motive lesne de înţeles, de clasă *extern*, va fi declarat aşa cum se arată în figura 2.

```
extern EVENT  _evn[MAX_EVN];
```

Figura 2. Declararea tabloului de blocuri eveniment *_evn []*

Un bloc eveniment concret, va fi exploatat cu ajutorul indicelui său în cadrul tabloului de blocuri eveniment, introducînd cîte un identificator sugestiv, pentru fiecare indice.

3. Funcţiile de operare asupra blocurilor eveniment

În executivul de timp real **RTC86**, referirile utilizatorului la blocurile eveniment sunt posibile doar prin intermediul unui set de funcţii dedicate. Ansamblul acestora, prezentate, sintetic în figura 3, împreună cu blocurile eveniment propriu-zise, constituie mecanismul de sincronizare.

```
                void e_init(void)
/* iniţializează tabloul de blocuri                    */
/* eveniment, prin atribuirea valorii                  */
/* NULL componentelor pie ale tuturor                  */
/*          elementelor sale                           */
```

```
        usshort e_creat (void)
/* alocă un element al tabloului _evn []      */
/* și furnizează indicele respectivului       */
/*            element                         */

        void e_destroy (usshort ind_evn)
/* dezalocă elementul tabloului _evn []       */
/* cu indicele ind_evn, repunîndu-l la        */
/*    dispoziția funcției e_creat ()          */

usshort e_wait (usshort ind_evn, usshort timeout)  */
/* blochează procesul curent și-l înscrie     */
/* în coada blocului eveniment cu indi-       */
/* cele ind_evn; reține valoarea argu-        */
/* mentului timeout ca timp limită cît        */
/* procesul curent poate rămîne blocat        */
/* la blocul eveniment; returnează o          */
/* valoare nulă, dacă deblocarea se           */
/* produce înaintea expirării timpului,       */
/*          altfel - una nenulă;              */

        void e_signal (usshort ind_evn)
/* deblochează toate procesele aflate în      */
/* coada blocului eveniment cu indicele       */
/*            ind_evn                         */
```

Figura 3. Funcțiile de operare asupra blocurilor eveniment

Funcția *e_init ()*

Această funcție are rolul de a inițializa tabloul de blocuri eveniment, prin stabilirea valorii *NULL* pentru componentele *pie* ale tuturor elementelor sale.
Textul funcției *e_init ()* este redat în figura 4.

```
1  void e_init (void)
2  {
3   register EVENT *e;
4   register unsigned j;
5   e=&_evn[0];
6   for (j=0;j<MAX_EVN;j++) {
7    e→pie=NULL;
8    e++;
9   }
10 }
```

Figura 4. Textul funcției *e_init ()*

Funcția *e_creat ()*

Funcția *e_creat ()* are rolul de a identifica un element liber al tabloului de blocuri eveniment *_evn* [] și de a-l aloca, furnizînd indicele - i. De asemenea, funcției *e_creat ()* îi revine sarcina de a poziționa pointerul *pie* al blocului eveniment pe care îl alocă, pe primul element al cozii.
Textul funcției *e_creat ()* este redat în figura 5.

```
1  usshort e_creat (void)
2  {
3   register EVENT *e;
4   register unsigned n;
5   e=&_evn[0];
6   n=0;
7   _lock_();
8   while ((e→pie)&&(n<MAX_EVN)) {
9    e++;
10   n++;
11  }
12  /* în ciclul while, s-a considerat că un bloc      */
13  /* eveniment încă nealocat se distinge prin        */
14  /* valoarea NULL a componentei sale pie            */
15  /*  (a se vedea e_init () și e_destroy ())         */
16  if (n<MAX_EVN) {
17   e→pie=&(e→coadă[0]);
18   /* se poziționează pie pe                         */
19   /* primul element al cozii                        */
20  }
21  else {
22   /* eșec: nici un bloc eveniment nu este liber     */
23  }
24  _unlock_();
25  return (n);
26  /* se furnizează indicele                         */
27  /* blocului eveniment alocat                      */
28 }
```

Figura 5. Textul funcției *e_creat ()*.

Funcția *e_destroy ()*

Funcția *e_destroy ()* are rolul de a dezaloca elementul tabloului de blocuri eveniment *_evn* [] cu indicele specificat prin valoarea argumentului ei, atribuind valoarea *NULL* componentei *pie* a acestui element. În acest fel, elementul în cauză este lăsat la dispoziția funcției *e_creat ()*, în vederea unei noi alocări.
Pentru a se exclude posibilitatea apariției unor situații critice ca urmare a unor erori de programare, se interzice distrugerea unui bloc eveniment a cărui coadă este nevidă. Funcția *e_destroy ()* va efectua o verificare a cozii blocului eveniment vizat a fi distrus, procedînd la distrugere doar dacă aceast coadă este vidă. Altfel, acțiunea funcției se va rezuma la generarea unui mesaj de eroare.
Textul funcției *e_destroy ()* face obiectul figurii 6.

```
1  void e_destroy (usshort ind_evn)
2  {
3   register EVENT *e;
4   e=&_evn[ind_evn];
5   _lock_();
6   if (e→pie!=&(e→coada[0])) {
     /* eroare: coadă este nevidă                    */
7   }
8   else {
9    e→pie=NULL;
10  }
11  _unlock_();
12 }
```

Figura 6. Textul funcției *e_destroy ()*.

Funcția *e_wait ()*

Funcția *e_wait ()* are rolul de a bloca procesul care execută și de a-l înscrie în coada blocului eveniment cu indicele specificat prin argumentul *ind_evn*. Operația de

locare a procesului în cauză se asigură prin apelul unei ıncţii denumită *sleep ()*. Această funcţie realizează ıclusiv limitarea timpului cît un proces poate rămîne locat. Cel de-al doilea argument al funcţiei *e_wait ()* este ransmis de către ea funcţiei *sleep ()*. Neproducerea venimentului în intervalul stabilit prin acest argument va onduce la deblocarea automată a taskului în cauză. O semenea deblocare reprezintă o anomalie care trebuie ratată în mod corespunzător de către utilizator. În sprijinul icestuia, funcţia *e_wait ()* returnează o valoare *unsigned hort* nenulă, în cazul în care anomalia s-a petrecut, şi nulă, ltfel. Această valoare se generează, primar, la nivelul *cheduler*-ului, fiind, apoi, preluată de către funcţia *sleep)* şi returnată funcţiei *e_wait ()*.

n conformitate cu cele de mai sus, funcţia *e_wait ()* a fost mplementată prin textul din figura 7.

```
1  usshort e_wait (usshort ind_evn, usshort timeout)
2  {
3   register EVENT *e;
4   register usshort aux;
5   e=&_evn[ind_evn];
6   _lock_():
7   *e→pie=_task_crt;
8   e→pie++;
9   aux=sleep (timeout);
10  _unlock_():
11  return (aux);
12 }
```

Figura 7. Textul funcţiei *e_wait ()*.

Funcţia *e_signal ()*

Funcţia *e_signal ()* are rolul de a debloca toate procesele ıflate în coada blocului eveniment cu indicele specificat rin argumentul său *ind_evn*. Deblocarea proceselor se sigură prin apelul unei funcţii cu numele *wakeup ()*. Dacă ıici un proces nu aşteaptă evenimentul în cauză, funcţia *_signal ()* nu are nici un efect.

Textul funcţiei *e_signal ()* este redat în figura 8.

```
1  void e_signal (usshort ind_evn)
2  {
3   register EVENT *e;
4   register usshort tsk;
5   e=&_evn[ind_evn];
6   _lock_();
7   while (e→pie!=&(e→coada[0])) {
8    e→pie--;
9    tsk=*e→pie;
10   wakeup (tsk);
11  }
12  _unlock_();
13 }
```

Figura 8. Textul funcţiei *e_signal ()*

Se poate înţelege, din cele de mai sus, că procesele care utilizează pentru sincronizare mecanismul blocurilor eveniment trebuie să execute, cînd ajung în punctul în care se impune sincronizarea, funcţia *e_wait ()*, cu referire la blocul eveniment asociat evenimentului în cauză. Semnalarea evenimentului se va asigura prin funcţia *e_signal ()* cu referire la acelaşi bloc eveniment. Funcţia *e_signal ()* poate fi prevăzută fie într-un alt proces, fie într-o rutină de tratare de întrerupere, după cum evenimentul este abstract, respectiv concret. Figura 9 ilustrează cele de mai sus, pentru cazul sincronizării a două procese, **A** şi **B**, cu un eveniment **e**, de tip concret.

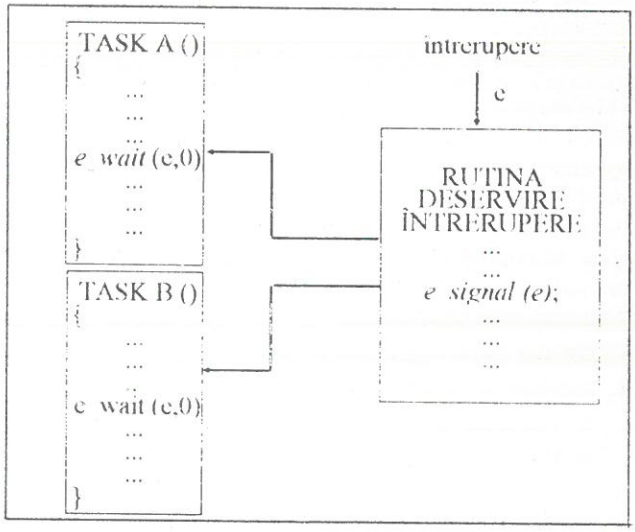

Figura 9. Sincronizarea proceselor A şi B cu evenimentul legat de întreruperea e.

4. Concluzii

Soluţiile prezentate sînt înglobate în executivul de timp real **RTC86**. Ele conferă acestui executiv capabilitatea de a realiza elegant sincronizarea interprocese, respectiv sincronizarea proceselor cu diverse evenimente concrete, în condiţiile unui consum redus de timp procesor.

Bibliografie

1. **ALLWORTH, S.T.**: Introduction to Real-Time Software Design, Macmillan Press Ltd., London, 1981.
2. **BALTAC V., ş.a.**: Sisteme interactive şi limbaje conversaţionale. Utilizare şi proiectare. Editura Tehnică, Bucureşti, 1984.
3. **PEREZ, J.P.**: Systemes Temps Reel -Methodes de Specification et de Conception, Dunod, Paris, 1990.
4. **TSCHIRHART, D.**: Commande en Temps Reel, Dunod, Paris, 1990.