MECANISM DE SINCRONIZARE PRIN BLOCURI *RENDEZ-VOUS*

Nicolae ROBU

Universitatea Tehnică Timişoara

Rezumat: lucrarea prezintă soluţiile adoptate în implementarea unui mecanism de sincronizare prin blocuri rendez-vous, 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 rendez-vous* şi de sincronizare de tip *rendez-vous*, iar apoi se arată modul de implementare a blocurilor *rendez-vous* propriu-zise, în executivul **RTC86**.
Se pun în evidenţă funcţiile mecanismului de sincronizare prin blocuri *rendez-vous*, 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 bloc rendez-vous, proces, aşteptare, "time-out".*

1. Introducere

După cum se cunoaşte [3], [4], în anumite aplicaţii, intervin situaţii în care este necesar să se asigure ca un număr de procese să ajungă toate în anumite stadii ale rulării lor, indiferent în ce ordine, şi abia după aceea să poată din nou avansa. Se poate considera că, în aceste situaţii, există un punct în spaţiul multidimensional al acţiunilor respectivelor procese în care ele au stabilit o *întâlnire*, după care, din nou, drumurile lor se despart. Figura 1 oferă o reprezentare a unei asemenea situaţii.

Situaţiile de tipul menţionat ridică o problemă de sincronizare specifică, ce se se rezolvă printr-un mecanism dezvoltat în jurul conceptului de bloc *rendez-vous*.

Din punct de vedere principial, un bloc *rendez-vous* este un ansamblu format dintr-o variabilă întreagă cu rol de contor şi o coadă de aşteptare [1],[2].

Contorul indică iniţial, numărul proceselor care trebuie să se întâlnească, iar curent, numărul proceselor care încă nu au ajuns la întâlnire.

Coada serveşte înregistrării proceselor care au ajuns deja la întâlnire, blocându-se, cu această ocazie, în aşteptarea celorlalte.

De fiecare dată, când un proces ajunge la întâlnire, valoarea contorului se decrementează cu o unitate. Atingerea, prin decrementări succesive, a valorii *zero*, semnifică faptul că întâlnirea a avut loc în formaţie completă şi determină deblocarea tuturor proceselor care au participat la ea. Evident, deblocarea se efectuează pe baza conţinutului cozii.

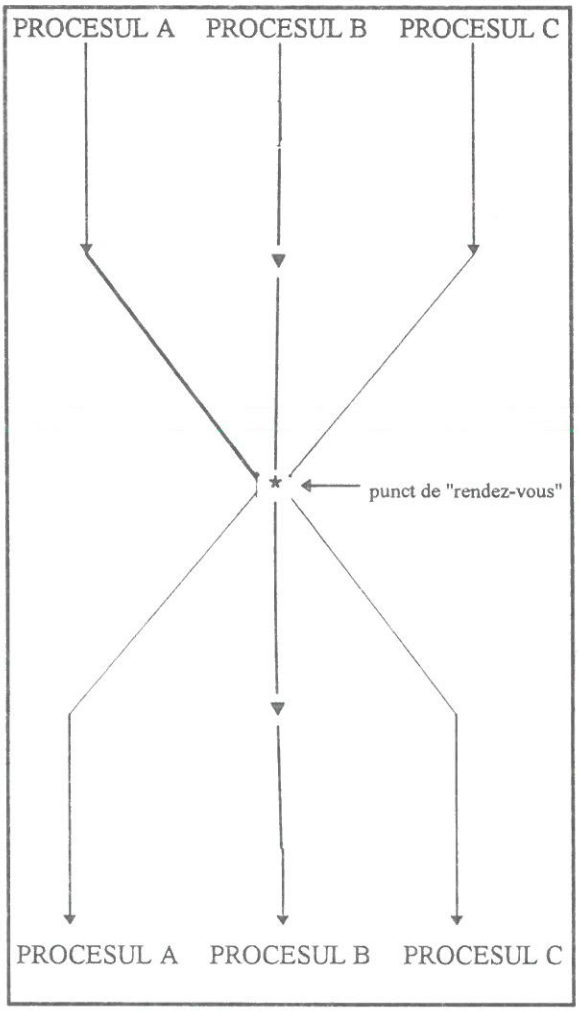

Figura 1. Reprezentarea unui "*rendez-vous*" între trei procese.

Se impune unui bloc *rendez-vous* să asigure şi posibilitatea fixării unui timp limită, în care procesele trebuie să sosească în punctul de întâlnire, pentru a se considera că aceasta a avut loc. Dacă cel puţin un proces nu reuşeşte să se încadreze în timpul fixat, întâlnirea se contramandează şi toate procesele care au ajuns la ea sunt deblocate.

Pentru a se asigura ca procesele vizate pentru o întâlnire să-şi poată continua activitatea în cunoştinţă de cauză, în ceea ce priveşte reuşita sau eşecul respectivei întâlniri, este necesar ca la deblocare să li se furnizeze o informaţie de stare, referitoare la cele două alternative posibile.

Se prezintă, în continuare, soluţiile adoptate pentru implementarea blocurilor *rendez-vous* 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**, prezentându-se

ca o extensie a mediului de programare **TURBO C**. Cu ajutorul acestei extensii, mediului **TURBO C** i se conferă faci-lităţi 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 o/îi asigură la nivelul programelor de aplicaţie, executivul **RTC86** se înscrie în rândul instrumentelor software, destinate dome-niului informaticii industriale [3],[4].

2. Tipul de dată *RENDEZ-VOUS*

În implementarea, în cadrul unui executiv de timp real, a unui mecanism de sincronizare, bazat pe *blocuri rendez-vous*, o primă decizie trebuie luată în ceea ce priveşte statutul pe care îl are blocul *rendez-vous* propriu-zis. În executivul **RTC86**, s-a adoptat ca blocul *rendez-vous* să fie o structură, cu numele *RENDEZ-VOUS*, cuprinzând:

-o variabilă de tipul *unsigned short*, reprezentând contorul blocului *rendez-vous*; s-a ales numele *contor* pentru această variabilă;

- un tablou de tipul *unsigned short*, cu *MAX-TSK* (numărul maxim al proceselor acceptabile în sistem) elemente, dedicat a servi drept coadă a blocului *rendez-vous*; s-a ales numele *coadă* pentru acest tablou;

- o variabilă de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista intrarea în, respectiv ieşirea din coada blocului *rendez-vous*; s-a ales numele*pie* pentru această variabilă;

- o variabilă de tipul *unsigned short*, dedicată memorării timpului limită în care toate procesele vizate trebuie să sosească la întâlnire; s-a ales numele *timeout* pentru această variabilă;

- o variabilă de tipul *unsigned short*, destinată memorării modului în care a decurs întâlnirea; s-a ales numele *status* pentru această variabilă.

Rezultă, deci, că tipul de dată *RENDEZ-VOUS* este introdus aşa cum se arată în figura 2.

```
typedef struct {
        unsigned short contor;
        unsigned short coada[MAX_TSK];
        unsigned short *pie;
        unsigned short timeout;
        unsigned short status;
      } RENDEZ_VOUS;
```

Figura 2. Tipul de dată *RENDEZ_VOUS*.

Este firesc ca toate blocurile *rendez-vous* de care se dispune în sistem să fie grupate sub forma unui tablou. S-a dat numele *_rv* acestui tablou. Din motive lesne de înţeles, el trebuie să fie de clasă *extern*. Dacă admitem că numărul maxim de blocuri *rendez-vous* este *MAX_RV*, atunci tabloul *_rv* se declară astfel:

Evident, un bloc *rendez-vous* concret va fi, în aceste condiţii, exploatat cu ajutorul indicelui său, introducând câte un identificator sugestiv pentru fiecare indice.

```
extern RENDEZ_VOUS _rv[MAX_RV];
```

Figura 3. Declararea tabloului de blocuri *rendez-vous*, *_rv[]*.

3. Funcţiile de operare asupra blocurilor rendez-vous

Funcţiile prin care utilizatorul poate face referiri la un bloc *rendez-vous* sunt:

```
void rv_init (void)
  /* iniţializează tabloul de blocuri */
  /* rendez-vous, prin atribuirea valorii */
  /* zero componentelor contor ale tuturor */
  /*        elementelor sale              */

usshort rv_creat (usshort ntv, usshort timeout)
  /* alocă un element al tabloului _rv [],   */
  /* atribuie componentei contor a elemen- */
  /* tului alocat valoarea specificată prin  */
  /* argumentului ntv (numărul taskurilor */
  /* vizate), reţine valoarea argumentului  */
  /* timeout şi furnizează indicele aferent  */
  /*       respectivului element             */
    void rv_destroy (usshort ind_rv)
  /* dezalocă elementul tabloului _rv [] */
  /* cu indicele ind_rv, repunându-l la */
  /*    dispoziţia funcţiei rv_creat ()     */

    usshort rv_signal (usshort ind_rv)
  /* anunţă sosirea la rendez-vous a pro-  */
  /* cesului curent; dacă nu toate celelalte  */
  /* procese vizate sunt sosite, atunci  */
  /* procesul curent se blochează; altfel,  */
  /* are loc deblocarea tuturor proceselor  */
  /* blocate în aşteptarea întâlnirii; re-  */
  /* turnează o valoare nulă, dacă întâlnirea*/
  /* a reuşit, altfel, una nenulă  */
```

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

Figura 4. Funcţiile de operare asupra blocurilor *rendez-vous*.

Funcţia rv_init ()

Această funcţie are rolul de a iniţializa tabloul de blocuri *rendez_vous*, prin atribuirea valorii *zero* componentelor *contor* ale tuturor elementelor sale.

Textul funcţiei *rv_init ()* este următorul:

```
1  void rv_init ()
2  {
3  register RENDEZ_VOUS *rv;
4  register usshort j;
5  rv=&_rv[0];
6  for (j=0;j<MAX_RV;j++) {
7     rv->contor=0;
8  }
9  }
```

Figura 5. Textul funcţiei *rv_init ()*.

Funcţia rv_creat ()

Funcţia *rv_creat ()* are rolul de a identifica un element liber al tabloului de blocuri *rendez-vous*, *_rv []*, şi de a-l aloca, furnizând indicele-i. De asemenea, funcţiei *rv_creat ()* îi revin sarcinile de a poziţiona pointerul *pie* al blocului pe care îl alocă pe primul element al cozii, de a înregistra argumentele *ntv* şi *timeout* în componentele *contor*, respectiv *timeout* ale acestui bloc, şi de a forţa la *zero* componenta *status*.

Textul funcţiei *rv_creat ()* este redat în figura 6

```
1     unsigned rv_creat (usshort ntv, usshort
timeout)
2  {
3   register RENDEZ_VOUS *rv;
4   register usshort j;
5   rv=&_rv[0];
6   j=0;
7   _lock_();
8   while ((rv->contor)&&(j<MAX_RV)) {
9    rv++;
10   j++;
11  }
12 /* în ciclul while, s-a considerat ca semn */
13   /* distinctiv pentru un bloc rendez_vous
încă */
14   /* nealocat, valoarea zero a componentei
sale */
15  /* contor, stabilită prin funcţiile rv_init (), */
16  /* rv_destroy () şi, de asemenea, rv_signal ()
*/
17  if (j<MAX_RV) {
18   rv->contor=ntv;
19   rv->pie=&(rv->coada[0]);
20   /* se poziţionează pointerul pie     */
21   /* pe primul element al cozii        */
22   rv->timeout=timeout;
23   rv->status=0;
24  }
25  else {
26   /* eroare: nici un bloc rendez-vous */
27   /*          nu este liber           */
28  }
29  _unlock_();
30  return (j);
31 }
```

Figura 6. Textul funcţiei *rv_creat ()*.

Funcţia rv_destroy ()

Funcţia *rv_destroy ()* are rolul de a dezaloca elementul tabloului de blocuri *rendez-vous* cu indicele specificat prin valoarea argumentului ei, atribuind valoarea *zero* componentei *contor* a acestui element. În acest fel, elementul în cauză este lăsat la dispoziţia funcţiei *rv_creat ()*, în vederea unei noi alocări.

Pentru evitarea unor anomalii, funcţia *rv_destroy ()* efectuează o verificare a cozii blocului *rendez_vous* vizat a fi distrus, procedând la distrugere doar în situaţia în care coada este vidă. Altfel, acţiunea funcţiei se rezumă doar la generarea unui mesaj de eroare.

Textul funcţiei *rv_destroy ()* este redat în figura 7

```
1  void rv_destroy (ind_rv)
2  {
3  register RENDEZ_VOUS *rv;
4  rv=&_rv[ind_rv];
5  _lock_();
6  if (rv->pie!=&(rv->coada[0]) {
7   /* eroare: se încearcă distrugerea unui bloc
*/
8     /* rendez_vous a cărui coadă nu este
vidă */
9  }
10  else {
11   rv->contor=0;
12  }
13  _unlock_();
14 }
```

Figura 7. Textul funcţiei *rv_destroy ()*.

Funcţia rv_signal ()

Funcţia *rv_signal ()* are rolul de a anunţa sosirea la *rendez-vous* a procesului în care ea se execută.

În primul rând, funcţia verifică, uzând de componenta *contor* a blocului *rendez-vous*, dacă

nu cumva toate procesele vizate pentru *rendez-vous* şi-au anunţat deja sosirea.

În caz afirmativ, se poziţionează la valoarea *zero* componenta *status* a blocului *rendez-vous*, iar apoi are loc deblocarea de către procesul sosit ultimul a tuturor celorlalte procese implicate în întâlnire, pe baza conţinutului cozii. Deblocarea se efectuează prin apelul unei funcţii numite *wakeup ()*, repetat de câte ori este necesar.

În caz negativ, taskul în care se execută funcţia *rv_signal ()* este blocat şi înscris în coada blocului *rendez-vous* cu indicele *ind_rv*. Operaţia de blocare se efectuează prin apelul unei funcţii numită *sleep ()*, căreia i se transmite ca argument valoarea pe care o are componenta *timeout* a blocului *rendez-vous* în cauză.

Deblocarea unui proces înscris în coada unui bloc *rendez-vous* se poate produce în trei situaţii, ce pot fi deosebite prin valorile pe care le au componentele *contor* şi *status* ale blocului, imediat după deblocare.

În prima dintre ele, caracterizată prin valoarea nulă, atât a contorului, cât şi a componentei *status*, deblocarea este rezultatul ajungerii în timp util la întâlnire a ultimului proces aşteptat. În această situaţie, oaricare dintre procesele implicate în *rendez-vous* nu vor mai avea altceva de făcut, decât să-şi revalideze întreruperile şi să părăsească funcţia *rv_signal ()*, cu returnarea valorii componentei *status* a blocului *rendez-vous* la care au fost în aşteptare.

În a doua situaţie, caracterizată prin valoarea nenulă a contorului, deblocarea este rezultatul expirării timpului afectat pentru realizarea întîlnirii înainte ca toate procesele să ajungă în punctul prevăzut. În această situaţie, unul dintre procesele care au fost blocate la blocul *rendez-vous* -firesc, primul- va fi reintrodus în rulare prin deblocare automată. Acest proces va poziţiona, mai întâi, la o valoare *nenulă* componenta *status* a blocului *rendez-vous* în cauză, va aduce componenta *contor* la *zero*, iar apoi va debloca, pe baza cozii, toate celelalte procese blocate în aşteptarea întâlnirii. Deblocarea se efectuează prin apelul funcţiei *wakeup ()*, repetat de câte ori este necesar. Componenta *contor* a blocului este adusă la *zero* de către procesul deblocat în mod automat, pentru a asigura proceselor pe care el le deblochează posibilitatea de a se autolocaliza în situaţia a treia.

A treia situaţie, caracterizată prin valoare nulă a componentei *contor* şi valoare *nenulă* a componentei *status*, este proprie oricăror procese deblocate de către partenerul lor refăcut rulabil în mod automat, în cazul eşuării întâlnirii prin expirarea timpului afectat pentru realizarea ei. Acestor procese nu le-a mai rămas altceva de făcut decât să-şi revalideze întreruperile şi să părăsească funcţia *rv_signal ()*, bineînţeles, cu returnarea valorii componentei *status* a blocului *rendez-vous* la care au fost în aşteptare.

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

```
1  unsigned rv_signal (usshort ind_rv)
2  {
3   register RENDEZ_VOUS *rv;
4   register usshort tsk;
5   rv=&_rv[ind_rv];
6   _lock_();
7   rv->contor--;
8   if (!(rv->contor)) {
9    rv->status=0;
10   while (rv-pie!=&(rv->coada[0])) {
11    rv->pie--;
12    tsk=*rv->pie;
13    wakeup (tsk);
14    /* sunt deblocate toate procesele găsite */
15    /* în coada blocului rendez-vous: cazul*/
16  /* întâlnirii reuşite, subcazul procesului */
17  /*          sosit ultimul                  */
18   }
19  }
20  else {
21   *rv->pie=_task_crt;
22   rv->pie++;
23   sleep (rv->timeout);
24   /* procesul curent se blochează */
25   /* în aşteptarea întâlnirii */
26   if (rv->contor) {
27    rv->status=1;
28    rv->contor=0;
29    while (rv->pie!=&(rv->coada[0])) {
30     rv->pie--;
31     tsk=*rv->pie;
32     wakeup (tsk);
33 /* sunt deblocate toate procesele găsite */
34 /* în coada blocului rendez-vous: cazul */
35 /* întâlnirii eşuate,subcazul procesului */
36 /*          deblocat primul                */
37    }
38   }
39   }
40  _unlock_();
41  return (rv->status);
42/* în afară de procesul sosit ultimul-caz*/
43/* întâlnirii reuşite-respectiv de procesul */
44 /* deblocat primul, în mod automat, din motiv
*/
45 /* de "time-out" -cazul întâlnirii eşuate-, */
46 /* toate celelalte vor parcurge doar liniile: */
47 /*          1...8, 20...26 _i 40...41          */
48 }
```

Figura 8. Textul funcţiei *rv_signal ()*.

4. Concluzii

Soluţiile prezentate sunt înglobate în executivul de timp real **RTC86**. Ele conferă acestui executiv capabilitatea de a realiza elegant sincronizarea interprocese -evident, în situaţiile în care se impune ca aceasta să fie de tipul *rendez-vous* - în condiţiile unui consum relativ 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 Réel - Méthodes de Spécification et de Conception, Dunod, Paris, 1990.
4. **TSCHIRHART, D.**: Commande en Temps Réel, Dunod, Paris, 1990.