Programe de aplicaţii

MECANISM DE COMUNICARE PRIN CUTIE POŞTALĂ

Nicolae Robu

Universitatea Tehnică din Timişoara

Rezumat: Lucrarea prezintă soluţiile adoptate în implementarea unui mecanism de comunicare prin cutie poştală, 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 *cutie poştală*, iar apoi se arată modul de implementare a cutiilor poştale propriu-zise, în executivul **RTC86**.
Se pun în evidenţă funcţiile mecanismului de comunicare prin cutie poştală, înglobat în executivul **RTC86**, şi principiile care au stat la baza implementării lor. Textul C al acestor funcţii este, şi el, redat.

Cuvinte cheie: comunicare, proces, producător, consumator, cutie poştală, blocare, deblocare, "time-out".

1. Introducere

După cum este cunoscut [1], o cutie poştală reprezintă o structură informaţională al cărei element central este un tampon circular (gestionat după principiul *FIFO*), în care orice proces poate scrie informaţii de un anumit format, invariabil, denumite mesaje, accesibile în citire oricărui alt proces.

Cel mai frecvent, mesajele vehiculate prin cutii poştale cuprind pointerul datei ce se comunică, lungimea datei şi indexul procesului ce o produce.

Mecanismul de comunicare prin cutie poştală trebuie să asigure blocarea proceselor producătoare, cînd acestea ajung în faţa unei operaţii de scriere, iar tamponul este plin, respectiv, blocarea proceselor consumatoare, când acestea ajung în faţa unei operaţii de citire, iar tamponul este vid. Scrierea în şi citirea din tampon reprezintă secţiuni critice.

Este evident, mecanismul de comunicare prin cutie poştală funcţionează după modelul producătorului şi al consumatorului [2]. Operaţiile de sincronizare, care fac obiectul acestui model, sunt intrinseci mecanismului, împreună cu operaţiile de depunere, respectiv de extragere a mesajelor în/din tampon. Aşadar, mecanismul asigură degrevarea utilizatorului de sarcina aplicării explicite a modelului producătorului şi consumatorului.

Se prezintă, în continuare, soluţiile adoptate pentru implementarea cutiilor poştale propriu-zise 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 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 o/îi asigură la nivelul programelor de aplicaţie, executivul **RTC86** se înscrie în rândul instrumentelor software destinate domeniului informaticii industriale [3],[4].

2. Tipul de dată MAILBOX

Pentru implementarea mecanismului de comunicare prin cutie poştală, în primul rând, s-a introdus tipul de dată *MESAJ*, sub forma unei structuri care cuprinde:

- o variabilă denumită *pd*, de tip *pointer generic* către data de comunicat;
- o variabilă denumită *ld*, de tipul *unsigned int*, reprezentând lungimea datei de comunicat;
- o variabilă denumită *ip*, de tipul *unsigned short*, reprezentând indexul procesului producător.

Tipul de dată *MESAJ* se defineşte, deci, aşa cum se arată în figura 1.

```
typedef struct { |
        void *pd
        unsigned int ld;
        unsigned short ip;
        } MESAJ;
```

Figura 1. Tipul de dată *MESAJ*.

În vederea înglobării în mecanismul de comunicare prin cutie poştală a regulilor care definesc modelul producătorului şi al consumatorului [2], s-a introdus, de asemenea,

tipul de dată *MAILBOX*, ca structură ce are în componența sa:

- o variabilă denumită *s1*, de tipul *SEMAPHORE*, destinată realizării excluderii mutuale (contorul semaforului s1 se va inițializa la valoarea 1);
- o variabilă denumită *s2*, de tipul *SEMAPHORE*, destinată controlului operațiilor de citire din tampon (contorul semaforului *s2* se va inițializa la valoarea 0);
- o variabilă denumită *s3*, de tipul *SEMAPHORE*, destinată controlului operațiilor de scriere în tampon (contorul semaforului *s3* se va inițializa la o valoare egală cu dimensiunea tamponului);
- o variabilă *tablou*, denumită *buf* [ ], de tipul *MESAJ*, reprezentând tamponul propriu-zis;
- o variabilă denumită *ps*, de tipul pointer către un *MESAJ*, destinată utilizării în procesul de scriere în tampon;
- o variabilă denumită *pc*, de tipul *pointer* către un *MESAJ*, destinată utilizării în procesul de citire din tampon.

Considerând dimensiunea tamponului specificată prin identificatorul *DIM_CP*, tipul de dată *MAILBOX* se introduce, deci, așa cum se arată în figura 2.

```
typedef struct {
        SEMAPHORE s1;
        SEMAPHORE s2;
        SEMAPHORE s3;
        MESAJ buf[DIM_CP];
        MESAJ *ps;
        MESAJ *pc;
        } MAILBOX;
```

Figura 2. Tipul de dată *MAILBOX*.

Se precizează că, în executivul **RTC86**, tipul de dată *SEMAPHORE* reprezintă o structură cu următoarele componente:

- un tablou de tipul *unsigned short* (prescurtat: *usshort*), cu *MAX_TSK* elemente, dedicat a servi drept coadă a semaforului; s-a dat numele *coada* acestui tablou;
- o variabilă de tipul *short*, reprezentând contorul semaforului; s-a dat numele *contor* acestei variabile;
- o variabilă de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista intrarea în coada semaforului; s-a dat numele *pi* acestei variabile, ca abreviere de la *pointer de intrare*;
- o variabilă de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista ieșirea din coada semaforului; s-a dat numele *pe* acestei variabile, ca abreviere de la *pointer de ieșire*.

Tipul de dată *SEMAPHORE* este introdus, deci, așa cum se arată în figura 3.

```
typedef struct {
        unsigned short coada[MAX_TSK];
        short contor;
        unsigned short *pi;
        unsigned short *pe;
        } SEMAPHORE;
```

Figura 3. Tipul de dată *SEMAPHORE*.

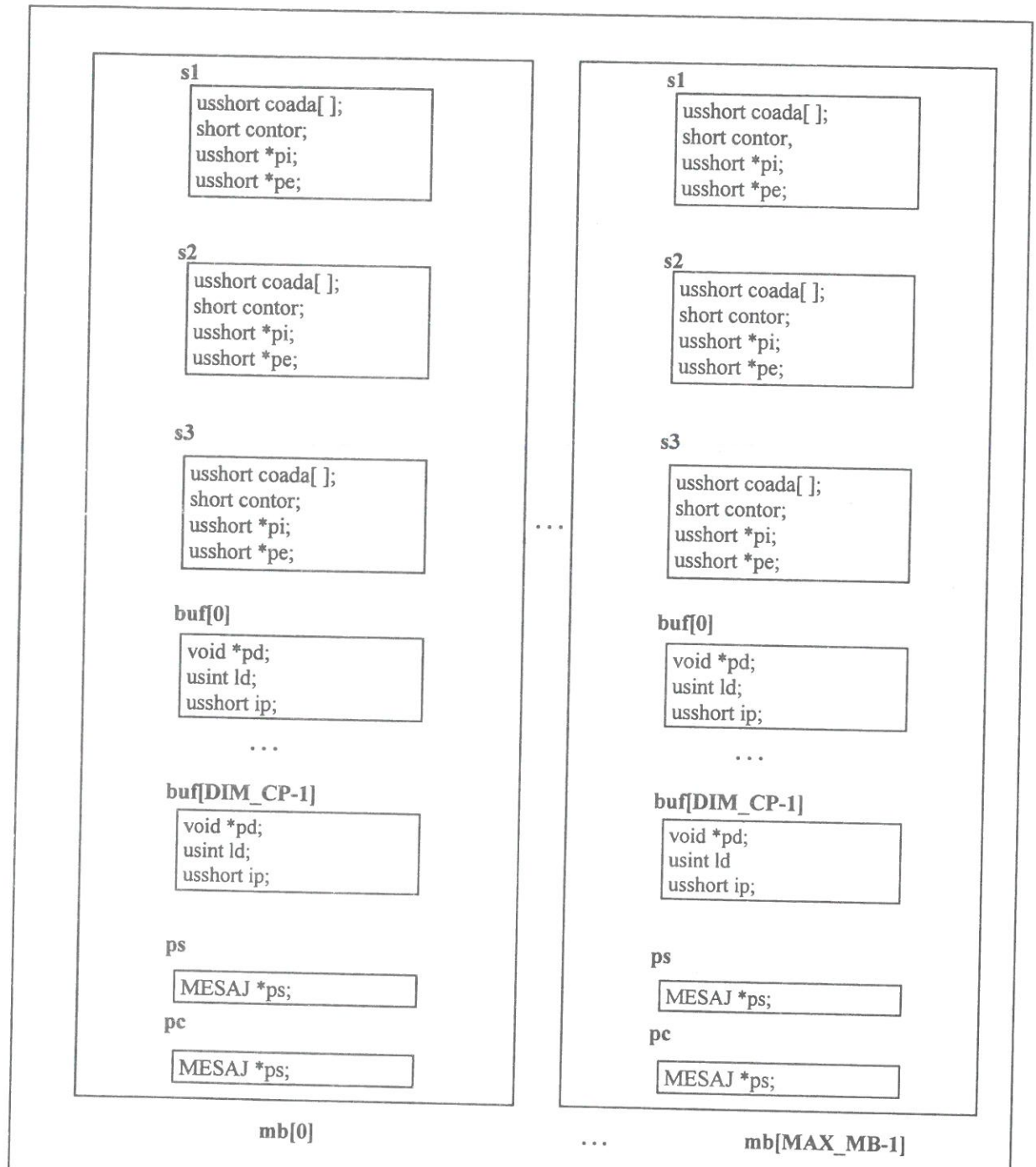
Executivul **RTC86** este înzestrat cu un tablou de structuri de tipul *MAILBOX*, denumit *_mb* []. Considerând dimensiunea acestui tablou specificată prin identificatorul *MAX_MB*, declararea sa se face așa cum se arată în figura 4.

```
extern MAILBOX _mb[MAX_MB]
```

Figura 4. Declararea tabloului de cutii poștale, *_mb []*.

Așa stând lucrurile, referirile la o cutie poștală se fac prin intermediul indicelui său în cadrul tabloului *_mb[]*, folosind câte un identificator sugestiv pentru fiecare indice.

Pentru crearea unei imagini de ansamblu asupra structurii de date a mecanismului care face obiectul lucrării, în figura 5, se oferă o reprezentare sinoptică a sa.



Note:
1)*usshort* este prescurtare de la *unsigned short*;
2)*usint* este prescurtare de la *unsigned int*.

Figura 5. Structura de date a mecanismului de comunicare prin cutie poştală reprezentare sinoptică.

3. Funcţiile de operare asupra cutiilor poştale

Comunicarea prin cutie poştală este instrumentată cu ajutorul setului de funcţii prezentate, grupat, în figura 6.

```
        void mb_init (void)
/*iniţializează tabloul _mb [] prin*/|
/* atribuirea valorii NULL componen- */
/* telor ps ale elementelor sale */

        usshort mb_creat (void)
/* aloc_ un element al tabloului */
/* _mb [ ] şi returnează indicele-i */

        void mb_destroy (usshort ind_mb)
/* dezalocă elementul tabloului */
/* _mb [ ] cu indicele ind_mb, repu- */
/* nându-l la dispoziţia funcţiei */
/* mb_creat ( ) */

        void mb_send (usshort ind_mb, void
            *pdata, usint ldata)
/* depune în cutia poştală cu indi- */
/* cele ind_mb mesajul corespunzător */
/* datei pointate de argumentul */
/* pdata, având lungimea, în octeţi, */
/* specificată de argumentul ldata */

        usshort mb_receive (usshort ind_mb,
            void *pdata)
/* extrage din cutia poştală cu in- */
/* dicele ind_mb mesajul urm_tor, */
/* atribuind valoarea variabilei pe */
/* care acesta o reprezintă variabi- */
/* lei pointate de argumentul pdata */
/* şi returnează indexul procesului */
/*producător */
```

Figura 6. Funcţiile mecanismului de comunicare prin cutie poştală.

Funcţia mb_init ( )

Această funcţie are rolul de a iniţializa tabloul de structuri cutie poştală, prin stabilirea valorii *NULL* pentru componentele *ps* ale tuturor elementelor sale.

Textul funcţiei *mb_init* () este redat în figura 7.

Funcţia mb_creat ( )

Funcţia *mb_creat ()* are rolul de a identifica un element liber al tabloului de structuri cutie poştală şi de a-l aloca, returnând indicele său.

```
1   void mb_init (void)
2   {
3   register MAILBOX *p;
4   register unsigned j;
5   p=&_mb[0];
6   for (j=0;j<MAX_MB;j++)  {
7   p->ps=NULL;
8   p++;
9   }
10  }
```

Figura 7. Textul funcţiei *mb_init ( )*.

De asemenea, funcţiei *mb_creat ()* îi revine sarcina de a iniţializa semafoarele din componenţa structurii cutie poştală alocată, în conformitate cu rolul lor în cadrul mecanismului de comunicare şi anume:

- semaforul *s1* cu componenta *contor* la valoarea **1** şi componentele *pi* şi *pe* la valoarea de pointare a elementului **0** al componentei *coada []*;
- semaforul *s2* cu componenta *contor* la valoarea **0** şi componentele *pi* şi *pe* la valoarea de pointare a elementului **0** al componentei *coada []*;
- semaforul *s3* cu componenta *contor* la valoarea *DIM_CP* şi componentele *pi* şi *pe* la valoarea de pointare a elementului **0** al componentei *coada []*.

În plus, funcţia *mb_creat ( )* asigură iniţializarea componentelor *ps* şi *pc* ale cutiei poştale alocată la valoarea de pointare a elementului **0** al componentei *buf []* a acesteia.

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

```
1   usshort mb_creat (void)
2   {
3   register MAILBOX *p;
4   register unsigned n;
5   p=&_mb[0];
6   n=0;
7   _lock_();
8   while ((p->ps!=NULL)&&(n<MAX_MB)) {
9       p++;
10      n++;
11  }
12  if (n==MAX_MB) {
13      /* eroare: nici o cutie poştală */
14      /*      nu este liberă      */
15  }
16  else {
17      _mb[n].s1.contor=1;
18      _mb[n].s1.pi=&(_mb[n].s1.coada[0]);
19      _mb[n].s1.pe=&(_mb[n].s1.coada[0]);
20      _mb[n].s2.contor=0;
21      _mb[n].s2.pi=&(_mb[n].s2.coada[0]);
22      _mb[n].s2.pe=&(_mb[n].s2.coada[0]);
23      _mb[n].s3.contor=DIM_CP;
24      _mb[n].s3.pi=&(_mb[n].s3.coada[0]);
25      _mb[n].s3.pe=&(_mb[n].s3.coada[0]);
26      p->ps=&(p->buf[0]);
27      p->pc=&(p->buf[0]);
28  }
29  _unlock_( );
30  return (n);
31  }
```

Figura 8. Textul funcției *mb_creat ( )*.

Funcția mb_destroy ( )

Funcția *mb_destroy* ( ) are rolul de a dezaloca elementul tabloului de structuri cutie poştală al cărui indice este precizat prin argumentul său, atribuind valoarea *NULL* componentei sale *ps*. În acest fel, elementul în cauză este lăsat la dispoziția funcției *mb_creat* ( ), în vederea unei noi alocări.

Textul funcției *mb_destroy* ( ) este următorul:

```
1   void mb_destroy (usshort ind_mb)
2   {
3   register MAILBOX *p;
4   p=&_mb[ind_mb];
5   p->ps=NULL;
6   }
```

Figura 9. Textul funcției *mb_destroy ( )*.

Funcția mb_send ()

Funcția *mb_send* () are rolul de a depune în cutia poştală cu indicele *ind_mb* mesajul corespunzător datei pointate de argumentul *pdata*, având lungimea, în octeți, precizată de argumentul *ldata*.

În conformitate cu modelul producătorului şi al consumatorului, depunerea va fi precedată de o secvență de instrucții indivizibilă, care execută operațiile ce definesc primitiva **P**, asupra semafoarelor *s3* şi *s1*, asigurându-se, astfel, evitarea depunerilor în tamponul plin, respectiv excluderea mutuală a secțiunilor critice de operare asupra tamponului.

După depunerea în tampon a pointerului către data în cauză, reprezentat de argumentul *pdata*, a lungimii datei, reprezentată de argumentul *ldata* şi a indexului procesului producător (care, evident este tocmai procesul în care funcția se execută), se actualizează pointerul de scriere, ținând seamă de gestiunea circulară a tamponului.

În final, o nouă secvență de instrucții indivizibilă execută operațiile ce definesc primitiva **V**, asupra semafoarelor *s2* şi *s1*; se semnalează, astfel, faptul că un nou mesaj a fost depus în cutia poştală, respectiv că secțiunea critică s-a încheiat.

Se precizează că, funcția *mb_send* () realizează blocarea, respectiv deblocarea proceselor, făcând apel la două funcții dedicate, şi ele parte a executivului **RTC86**, funcții denumite *sleep* ( ), respectiv *wakeup* ( ). Funcția *sleep* ( ) necesită la apel un argument. Dacă acest argument este nul, atunci blocarea se face pe timp nelimitat. Dacă argumentul este nenul, atunci el reprezintă timpul limită după care, dacă procesul va fi încă găsit

blocat - situație de "time-out"-, se produce o deblocare automată.

Textul funcției *mb_send* () este redat în figura 10.

Funcția mb_receive ( )

Funcția *mb_receive* ( ) are rolul de a extrage din cutia poștală cu indicele *ind_mb* mesajul următor, atribuind valoarea variabilei pe care el o reprezintă variabilei pointate de argumentul *pdata*. Funcția returnează indexul procesului producător.

```
/*****************************************************************************/
/*                  funcția mb_send ( ) mizează pe declarațiile:             */
/*                                                                           */
/*                  typedef struct                                           */
/*                          usshort coada[MAX_TSK];                          */
/*                          short contor,                                    */
/*                          usshort *pi;                                     */
/*                          usshort *pe;                                     */
/*                          } SEMAPHORE                                      */
/*                                                                           */
/*                  typedef struct {                                         */
/*                          void *pd;                                        */
/*                          usint ld;                                        */
/*                          usshort ie;                                      */
/*                          } MESAJ;                                         */
/*                                                                           */
/*                  typedef struct {                                         */
/*                          SEMAPHORE s1;                                    */
/*                          SEMAPHORE s2;                                    */
/*                          SEMAPHORE s3;                                    */
/*                          MESAJ buf[DIM_CP];                               */
/*                          MESAJ *ps;                                       */
/*                          MESAJ *pc;                                       */
/*}                         MAILBOX;                                         */
/*                                                                           */
/*                  extern MAILBOX _mb[MAX_MB];                              */
/*****************************************************************************/

1       void mb_send (usshort ind_mb,
2                                   void *pdata, usint ldata)
3       {
4       register usshort tsk;
5       SEMAPHORE *p1, *p2, *p3;
6       p1=&_mb[ind_mb].s1;
7       p2=&_mb[ind_mb].s2;
8       p3=&_mb[ind_mb].s3;
9       _lock_( );
10      if (--p3->contor<0) {
11      *p3->pi=_task_crt;
```

Figura 10. Textul funcției *mb_send ( )*.(partea I)

```
12              if (p3->pi= =&(p3->coada[MAX_TSK-1])) {
13                      p3->pi=&(p3->coada[0]);
14              }
15              else {
16                      p3->pi++;
17              }
18              sleep (0);
19          }
20          if (--p1->contor<0) {
21              *p1->pi=_task_crt;
22              if (p1->pi==&(p1->coada[MAX_TSK-1])) {
23                      p1->pi=&(p1->coada[0]);
24              }
25              else {
26                      p1->pi++;
27              }
28              sleep (0);
29          }
30          _mb[ind_mb].ps->pd=pdata;
31          _mb[ind_mb].ps->ld=ldata;
32          _mb[ind_mb].ps->ip=_task_crt;
33          if (_mb[ind_mb].ps==
34                      &(_mb[ind_mb].buf[DIM_CP-1])) {
35          _mb[ind_mb].ps=&(_mb[ind_mb].buf[0]);
36          }
37          else {
38          _mb[ind_mb].ps++;
39          }
40          if (++p2->contor<=0) {
41              tsk=*p2->pe;
42              if (p2->pe==&(p2->coada[MAX_TSK-1])) {
43                  p2->pe=&(p2->coada[0]);
44          }
45          else {
46                  p2->pe++;
47          }
48          wakeup (tsk);
49      }
50      if (++p1->contor<=0) {
51              tsk=*p1->pe;
52              if (p1->pe==&(p1->coada[MAX_TSK-1])) {
53                      p1->pe=&(p1->coada[0]);
54              }
55              else {
56                      p1->pe++;
57              }
58              wakeup (tsk);
59      }
60  _unlock_();
61  }
```

Figura10. Textul funcției *mb_send ( )*. (partea a II-a)

În conformitate cu modelul producătorului și al consumatorului, depunerea este precedată de o secvență de instrucții indivizibilă, care execută operațiile ce definesc primitiva **P**, asupra semafoarelor *s1* și *s2*, asigurându-se, astfel, evitarea extragerilor din tamponul vid, respectiv excluderea mutuală a secțiunilor critice de operare asupra tamponului.

După extragerea din tampon a pointerului către data recepționată, a lungimii respectivei date și a indexului procesului care a produs-o, are loc copierea datei, octet cu octet, în variabila pointată de argumentul *pdata*, iar apoi, actualizarea pointerului de citire, ținând seamă de gestiunea circulară a tamponului.

```
/*************************************************************************/
/*          funcţia mb_receive ( ) mizează pe declaraţiile:              */
/*                  typedef struct {                                     */
/*                          usshort coada[MAX_TSK];                      */
/*                          short contor;                                */
/*                          usshort *pi;                                 */
/*                          usshort *pe;                                 */
/*                          } SEMAPHORE;                                 */
/*                  typedef struct {                                     */
/*                          void *pd;                                    */
/*                          usint ld;                                    */
/*                          usshort ie;                                  */
/*                          } MESAJ;                                     */
/*                  typedef struct {                                     */
/*                          SEMAPHORE s1;                                */
/*                          SEMAPHORE s2;                                */
/*                          SEMAPHORE s3;                                */
/*                          MESAJ buf[DIM_CP];                           */
/*                          MESAJ *ps;                                   */
/*                          MESAJ *pc;                                   */
/*                          } MAILBOX;                                   */
/*                                                                       */
/*                  extern MAILBOX _mb[MAX_MB];                          */
/*************************************************************************/
1   usshort mb_receive (usshort ind_mb, void *pdata)
2   {
3   uschar *p,*q;
4   register usint ldata;
5   register usshort tsk;
6   usshort indprd;
7   SEMAPHORE *p1,*p2,*p3;
8   p1=&_mb[ind_mb].s1;
9   p2=&_mb[ind_mb].s2;
10  p3=&_mb[ind_mb].s3;
11  _lock_( );
12  if(--p2->contor<0) {
13          *p2->pi=_task_crt;
14          if (p2->pi==&(p2->coada[MAX_TSK-1])) {
15                  p2->pi=&(p2->coada[0]);
16          }
17          else {
18                  p2->pi++;
19          }
20          sleep (0);
21  }
```

Figura 11. Textul funcţiei *mb_receive ( )*. (partea I)

În final, o nouă secvenţă de instrucţii indivizibilă execută operaţiile ce definesc primitiva **V**, asupra semafoarelor *s3* şi *s1*; se semnalează, astfel, faptul că un nou mesaj a fost extras din cutia poştală, respectiv că secţiunea critică s-a încheiat.

Ca şi funcţia *mb_send* ( ), funcţia *mb_receive* () realizează blocarea şi deblocarea proceselor cu ajutorul funcţiilor *sleep* ( ), respectiv *wakeup* ( ).

Textul funcţiei *mb_receive* ( ) este redat în figura 11.

4. Concluzii

Soluţiile prezentate sunt înglobate în executivul de timp real **RTC86**. Ele conferă acestuia capabilitatea de a realiza comunicarea interprocese sigur, versatil şi eficient. Timpul procesor consumat de mecanismul de comunicare dezvoltat pe baza lor este relativ scăzut.

```
22      if (--p1->contor<0) {
23              *p1->pi=_task_crt;
24              if (p1->pi==&(p1->coada[MAX_TSK-1])) {
25                      p1->pi=&(p1->coada[0]);
26              }
27              else {
28                      p1->pi++;
29              }
30              sleep (0);
31      }
32      p=(uschar *)pdata;
33      q=(uschar *)(_mb[ind_mb].pc->pd);
34      ldata=_mb[ind_mb].pc->ld;
35      while (ldata--) {
36              *p++=*q++;
37      }
38      indprd=_mb[ind_mb].pc->ip;
39      if (_mb[ind_mb].pc= =
40                      &(_mb[ind_mb].buf[DIM_CP-1])) {
41              _mb[ind_mb].pc=&(_mb[ind_mb].buf[0];
42      }
43      else {
44      _mb[ind_mb].pc++;
45      }
46      if (++p3->contor<=0) {
47              tsk=*p3->pe;
48              if (p3->pe==&(p3->coada[MAX_TSK-1])) {
49                      p3->pe=&(p3->coada[0]);
50              }
51              else {
52                      p3->pe++;
53              }
54              wakeup (tsk);
55      }
56      if (++p1->contor<=0) {
57              tsk=*p1->pe;
58              if (p1->pe==&(p1->coada[MAX_TSK-1])) {
59                      p1->pe=&(p1->coada[0]);
60              }
61              else {
62                      p1->pe++;
63              }
64              wakeup (tsk);
65      }
66      _unlock_( );
67      return (indprd);
68      }
```

Notă: *uschar* este prescurtare de la *unsigned char*.

Figura 11. Textul funcției *mb_receive ( )*. (partea a II-a)

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.