MECANISM DE COMUNICARE PRIN CONDUCTĂ

Nicolae ROBU

Universitatea Tehnică din Timişoara

Rezumat: lucrarea prezintă soluţiile adoptate în implementarea unui mecanism de comunicare prin conductă, 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 *conductă*, iar apoi se arată modul de implementare a conductelor propriu-zise, în executivul RTC86.
Se pun în evidenţă funcţiile mecanismului de comunicare prin conductă, 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, conductă, blocare, deblocare, "time-out".

1. Introducere

După cum este cunoscut, o conductă este un tampon circular, gestionat după principiul *FIFO*, asociat unei perechi de procese dintre care unul are exclusiv rolul de consumator. Producătorul depune datele în tampon, octet cu octet, iar consumatorul le extrage, de asemenea, octet cu octet, indiferent ce ce tip sunt ele[4].

Mecanismul de comunicare prin conductă trebuie să asigure blocarea procesului producător, când acesta ajunge în faţa unei operaţii de scriere, iar tamponul este plin, respectiv blocarea pro-cesului consumator,când acesta ajunge în faţa unei operaţii de citire, iar tamponul este vid [1, 2].

Nici un fel de restricţii privind lungimea datelor vehiculate nu intervin la comunicarea prin conductă; datele care nu încap în întregul lor în tampon sunt transmise şi recepţionate fragmentar.

Se prezintă, în continuare, soluţiile adoptate pentru implementarea conductelor 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, prezen-tându-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 posibilităţile 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 *PIPE*

Implementarea, în cadrul unui executiv de timp real, a unui mecanism de comunicare prin conductă, presupune, în primul rând, adoptarea unei accepţiuni concrete asupra noţiunii de conductă. În executivul RTC86, conductele sunt structuri cu numele *PIPE*, ce cuprind:

- -o variabilă de tipul *unsigned short*, denumită *itp*, destinată înregistrării indexului procesului producător;
- -o variabilă de tipul *unsigned short*, denumită *itc*, destinată înregistrării indexului procesului consumator;
- -o variabilă de tipul *unsigned int*, denumită *rs*, cu rolul de a indica, atunci când are o valoare nenulă, pe de o parte, faptul că procesul producător s-a blocat, găsind conducta plină, iar, pe de altă parte, numărul octeţilor ce au mai rămas de scris;
- -o variabilă de tipul *unsigned int*, denumită *rc*, cu rolul de a indica, atunci când are o valoare nenulă, pe de o parte, faptul că procesul consumator s-a blocat, fie pentru că a găsit conducta vidă, fie pentru a-i permite procesului producător să transmită ultimul fragment al unei date, iar, pe de altă parte, numărul octeţilor ce au mai rămas de citit;
- -o variabilă de tipul *unsigned int*, denumită *nocumul*, destinată înregistrării numărului de octeţi cumulaţi la un moment dat în conductă;
- -o variabilă de tipul *pointer* către un caracter fără semn, denumită *ps*, destinată utilizării în procesul de scriere în tampon;
- -o variabilă de tipul *pointer* către un caracter fără semn, denumită *pc*, destinată utilizării în procesul de citire din tampon;
- -un tablou de tipul *unsigned char*, denumit *conducta*, care implementează tamponul circular propriu-zis.

Considerând dimensiunea conductei în octeţi, desemnată prin identificatorul *DIM_COND*, tipul de dată *PIPE* se va declara aşa cum se arată în figura 1.

```
typedef struct {
        unsigned short itp;
        unsigned short itc;
        unsigned int rs;
        unsigned int rc;
        unsigned int nocumul;
        unsigned char *ps;
        unsigned char conducta
        [DIM_COND];
    } PIPE;
```

Figura 1. Tipul de dată *PIPE*.

Executivul RTC86 este înzestrat cu un tablou de structuri de tipul *PIPE*, denumit *_pipe []*, cu dimensiunea *MAX_PIPE*. Din motive lesne de înţeles, tabloul *_pipe []* este de clasă extern:

```
extern PIPE _pipe[MAX_PIPE];
```

Figura 2. Declararea tabloului de structuri conductă *_pipe []*.

Evident, o structură conductă anume, este exploatată cu ajutorul indicelui său în cadrul tabloului *_pipe []*. Pentru sugerarea referirilor, se introduce câte un identificator adecvat, cores-punzător fiecărui indice.

3. Funcţiile de operare asupra structurilor conductă

În executivul de timp real RTC86, referirile utilizatorului la o structură conductă sunt posibile doar prin intermediul unui set de funcţii dedicate. Ansamblul acestora, prezentat sintetic în figura 3, împreună cu structurile conductă propriu-zise,

```
/* consumator, un element al tabloului */ .
/* _pipe [] şi returnează indicele i      */
    void p_destroy (usshort ind_p)
/* dezalocă elementul tabloului _pipe [] */
/* cu indicele ind_p, repunându-l la     */
/*   dispoziţia funcţiei p_creat ()      */
void p_send (usshort ind_p, void *pdata, usint[2]
ldata)
/* depune în tamponul structurii conductă */
/* cu indicele ind_p cei ldata octeţi ai  */
/*      datei pointate de pdata           */
void p_receive (usshort ind_p, void *pdata, usint
ldata)
/* extrage din tamponul structurii conduct_ */
/* cu indicele ind_p un număr de ldata    */
/* octeţi şi îi atribuie datei pointate   */
/*          de pdata                      */
```

Figura 3. Funcţiile mecanismului de comuni-care prin conductă

Funcţia p_init ()

Această funcţie are rolul de a iniţializa tabloul de structuri conductă, prin stabilirea valorii *NULL* pentru componentele *ps* şi *pc*, respectiv a valorii *0xFF* pentru componentele *itp* şi *itc* ale tuturor elementelor sale.

Textul funcţiei *p_init ()* este redat în figura 4.

```
1  void p_init (void)
2  {
3  register PIPE *p;
4  register usshort j;
5  p=&_pipe[0];
6  for (j=0;j<MAX_PIPE;j++) {
7    p->ps=NULL;
8    p->pc=NULL;
9    p->itp=0xFF;
10   p->itc=0xFF;
11   p++;
12 }
13 }
```

Figura 4. Textul funcţiei *p_init ()*.

```
/* iniţializează tabloul _pipe [], prin */
/* atribuirea valorii NULL componentelor */
/* ps _i pc, respectiv a valorii 0xFF */
/* componentelor itp _i itc ale tuturor */
/*      elementelor sale                */
usshort p_creat (usshort[1] prod, usshort cons)
/* alocă proceselor prod, cu calitatea de */
/* producător, şi cons, cu calitatea de   */
```

constituie mecanismul de comunicare prin conductă.

[1] *usshort* este prescurtare de la *unsigned short*;

[2] *usint* este prescurtare de la *unsigned int*.

Funcţia p_creat ()

Funcţia *p_creat ()* are rolul de a identifica un element liber al tabloului de structuri conductă şi de a-l aloca perechii producător-consumator, specificată prin argumentele sale, returnând indicele respectivului element. De asemenea, funcţiei *p_creat ()* îi revine sarcina de a poziţiona pointerii *ps* şi *pc* ai structurii conductă, alocată pe primul element al tabloului *conducta []* din cadrul acesteia.

Textul funcţiei *p_creat ()* este redat în figura 5.

```
1 usshort p_creat (usshort prod, usshort cons)
2 {
3   register PIPE *p;
4   register usshort n;
5   p=&_pipe[0];
6   n=0;
7   _lock_();
8 while ((p->ps!=NULL) &&(n<MAX_PIPE))
{
9    p++;
10   n++;
11 }
12 if (n==MAX_PIPE) {
13   /* eroare: nici o structură conductă */
14   /*          nu este liberă */
15 }
16 else {
17   p->itp=prod;
18   p->itc=cons;
19   p->rs=0;
20   p->rc=0;
21   p->nocumul=0;
22   p>ps=&(p->conducta[0]);
23   p->pc=&(p->conducta[0]);
24 }
25 _unlock_();
26 return (n);
27 }
```

Figura 5. Textul funcţiei *p_creat ()*.

Funcţia p_destroy ()

Funcţia *p_destroy ()* are rolul de a dezaloca elementul tabloului de structuri conductă *_pipe []*, al cărui indice este precizat prin argumentul său. Dezalocarea constă în atribuirea valorii *NULL* pentru componenta *ps* a respectivului element şi a valoarii *0xFF* pentru componentele *itp* şi *itc*. În acest fel, elementul în cauză este lăsat la dispoziţia funcţiei *p_creat ()*, în vederea unei noi alocări.

Textul funcţiei *p_destroy ()* este redat în fig. 6.

```
1  void p_destroy (usshort ind_p)
2  {
3  register PIPE *p;
4  p=&_pipe[ind_p];
5  p->ps=NULL;
6  p->itp=0xFF;
7  p->itc=0xFF;
8  }
```

Figura 6. Textul funcţiei *p_destroy ()*.

Funcţia p_send ()

Funcţia *p_send ( )* asigură depunerea în tamponul structurii conductă cu indicele precizat prin argumentul *ind_p* a unui număr de *ldata* octeţi ai datei pointate de argumentul *pdata*.

Pentru a fi posibilă comunicarea datelor de orice tip de la un proces la altul, argumentul *pdata* este un *pointer generic*; în cadrul funcţiei, el va fi convertit în *pointer de caracter fără semn* şi utilizat ca atare.

Cînd funcţia *p_send ()* constată că tamponul este plin, ea procedează astfel:

- -înregistrează în componenta *rs* a stucturii conductă numărul de octeţi pe care îi mai are de transmis, pentru a-l face cunoscut funcţiei *p_receive ()*, prevăzută în procesul consumator;
- -verifică dacă procesul consumator este blocat în interiorul funcţiei *p_receive ()*, ca urmare a faptului că a găsit tamponul vid, şi, în caz afirmativ, îl anunţă că situaţia s-a schimbat în această privinţă şi îl deblochează;
- -blochează procesul în care ea se execută (procesul producător), lăsîndu-l în aşteptare pasivă a deblocării de către procesul consumator.Procesul consumator asigură deblocarea producătorului, dacă îl găseşte blocat

la tampon plin, cu ocazia execuţiei funcţiei *p_receive ()* prevăzută în cadrul lui, atunci când în tampon este creat suficient spaţiu liber pentru depunerea tuturor octeţilor de transmis rămaşi restanţi sau, oricum, atunci când această funcţie găseşte tamponul vid sau termină de preluat o dată.

```
1   p_send (usshort ind_p, void *pdata, usint
ldata)
2  {
3   register PIPE *p;
4   register uschar *q;
5   usint nos;
6   p=&_pipe[ind_p];
7   if (p->itp!=_task_crt) {
8    /* eroare: procesul curent nu este */
9    /*      asociat la conductă          */
10  }
11  else {
12   _lock_();
13   q=(uschar *)pdata;
14   nos=ldata;
15   while (nos) {
16    if (p->nocumul<DIM_COND){
17          *p->ps=*q++;
18          if (p->ps==&(p->conducta [DIM
_COND-1])) {
19          p->ps=&(p->conducta[0]);
20          }
21          else {
22          p->ps++;
23          }
24          nos--;
25          p->nocumul++;
26    }
27    else {
28          p->rs=nos;
29          if (p->rc) {
30           p->rc=0;
31           wakeup (p->itc);
32          }
33          while (p->rs){
34           sleep (0);
35          }
36    }
37   }
38   if (p->rc) {
39    p->rc=0;
40    wakeup (p->itc);
41   }
42   _unlock_();
43  }
44 }
```

Notă: *_lock_()* şi *_unlock_()* sunt macro-instrucţii ale executivului, destinate definirii de secţiuni neîntreruptibile.

Figura 7. Textul funcţiei *p_send ()*.

Când termină de pus o dată în tampon, funcţia *p_send ()* verifică dacă procesul consumator este blocat din cauză că a întâlnit tamponul vid şi, în caz afirmativ, îl anunţă că tamponul nu mai este vid şi îl deblochează.

Se precizează că funcţia *p_send ()* realizează blocarea, respectiv deblocarea proceselor, făcând apel la două funcţii dedicate, şi ele parte a executivului RTC86, funcţii denumite *sleep ()*, respectiv *wakeup ()*. Funcţia *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 - situaţie de "time-out"-, se produce o deblocare automată.

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

Funcţia p_receive ()

Funcţia *p_receive ()* asigură extragerea din tamponul structurii conductă al cărui indice este precizat prin argumentul *ind_p* a unui număr de *ldata* octeţi şi atribuirea lor variabilei pointate de argumentul *pdata*.

Pentru a fi posibilă comunicarea datelor de orice tip de la un proces la altul, ca şi în cazul funcţiei *p_send ()*, argumentul *pdata* este un *pointer generic*; în cadrul funcţiei, el va fi convertit în *pointer de caracter fără semn* şi utilizat ca atare.

Pe parcursul extragerii octeţilor din tampon, funcţia *p_receive ()* verifică, utilizând compo-nenta *rs* a structurii conductă, dacă procesul producător este blocat în interiorul funcţiei *p_send ()*, ca urmare a faptului că a găsit tamponul plin. În caz afirmativ, când se ajunge ca în tampon să fie suficient spaţiu liber pentru depunerea tuturor octeţilor rămaşi restanţi, funcţia *p_receive ()* anunţă procesul producător asupra acestui lucru şi îl deblochează, înregistrând în componenta *rc* a structurii conductă numărul octeţilor neextraşi ai datei curente şi blocând procesul în care ea se execută (procesul consumator).

Cînd funcţia *p_receive ()* constată că tamponul este vid, procedează astfel:

-înregistrează în componenta *rc* a structurii conductă numărul de octeţi pe care îi mai are de recepţionat;

-verifică dacă procesul producător este blocat în interiorul funcției *p_send ()* ca urmare a faptului că a găsit tamponul plin și, în caz afirmativ, îl anunță că situația s-a schimbat în această privință și îl deblochează;

-blochează procesul în care ea se execută (procesul consumator), lăsându-l în așteptare pasivă a deblocării de către procesul producător. Procesul producător asigură deblocarea consumatorului, dacă îl găsește blocat la tampon vid, cu ocazia execuției funcției *p_send ()* prevăzută în cadrul lui, atunci când această funcție găsește tamponul plin sau termină de depus o dată.

Când termină de extras o dată din tampon, funcția *p_receive ()* verifică dacă procesul producător este blocat din cauză că a întâlnit tamponul plin și, în caz afirmativ, îl anunță că tamponul nu mai este plin și îl deblochează.

Ca și funcția *p_send ()*, funcția *p_receive ()* realizează blocarea și deblocarea proceselor cu ajutorul funcțiilor *sleep ()*, respectiv *wakeup ()*.

Textul funcției *p_receive ()* este redat în fig. 8.

4. Concluzii

Soluțiile prezentate sunt înglobate în exe-cutivul de timp real RTC86. Ele conferă acestuia capacitatea de a realiza comunicarea inter-procese sigur, versatil și eficient. Timpul procesor consumat de mecanismul de comunicare dezvoltat pe baza lor este relativ scăzut.

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.**: Systemes Temps Réel -Méthodes de Specification et de Concep-tion, Dunod, Paris, 1990.
4. **TSCHIRHART, D.**: Commande en Temps Réel, Dunod, Paris, 1990.


```
1  p_receive (usshort ind_p, void *pdata, usit ldata)
2  {
3  register PIPE *p;
4  register uschar *q;
5  usint noc;
6  p=&_pipe[ind_p];
7  if (p->itc!=_task_crt) {
8   /* eroare: procesul curent nu este */
9   /*     asociat la conductă      */
10 }
11 else {
12  _lock_();
13  q=(uschar *)pdata;
14  noc=ldata;
15  while (noc) {
16   if (p->nocumul) {
17    *q++=*p->pc;
18              if (p->pc==&(p->conducta[DIM_COND-1])) {
19     p->pc=&(p->conducta[0]);
20    }
21    else {
22     p->pc++;
23    }
24    noc--;
25    p->nocumul--;
26    if (p->rs&&
27              (p->rs<=(DIM_COND-(p->nocumul)))) {
28     p->rc=noc;
29     p->rs=0;
30     wakeup (p->itp);
31     while (p->rc)
32      sleep (0);
33     }
34    }
35   }
36   else {
37    p->rc=noc;
38    if (p->rs) {
39     p->rs=0;
40     wakeup (p->itp);
41    }
42    while (p->rc)
43     sleep (0);
44    }
45   }
46  }
47  if (p->rs) {
48   p->rs=0;
49   wakeup (p->itp);
50  }
51  _unlock_();
52 }
53 }
```

Figura 8. Textul funcției *p_receive ()*.