Operaţii fundamentale multitasking

Concurenţa pentru deţinerea resurselor poate genera situaţii conflictuale între taskuri. Pentru evitarea acestora şi pentru ca taskurile să-şi realizeze obiectivele au fost formalizate aşa numitele *operaţii multitasking* între care o importanţă aparte prezintă: *excluderea mutuală, sincronizarea şi comunicarea* [1]

Pentru implementarea acestor operaţii sistemele de operare sau executivele de timp real pun la dispoziţie instrumente cum ar fi: *semafoare, cutii poştale, mesaje de trecere, variabile de tip eveniment, fanioane de excluziune, monitoare, etc.*

1. Resurse şi secţiuni critice

În general termenul de *resursă* desemnează orice element necesar unui task pentru a putea fi executat . Resursele pot fi de două categorii şi anume *materiale* şi *logice.*

În rândul resurselor materiale pot fi considerate:

- procesorul;
- memoria internă;
- memoriile externe;
- dispozitivele de intrare / ieşire;
- ceasul de timp real;
- dispozitivele speciale.

În resursele logice pot fi incluse:

- subrutinele (procedurile);
- masivele de date;
- fişierele;
- variabilele.

O altă clasificare împarte resursele în: *resurse locale* şi *resurse comune.*

- Resursele locale sunt resursele care pot fi *accesate de către un singur task.* Acestea pot fi în primul rând de natură logică (*subrutine, fişiere, variabile*), dar pot fi şi de natură fizică, cum ar fi anumite dispozitive de intrare / ieşire.

[1] În mod obişnuit acestea sunt cunoscute ca *Operaţii Fundamentale Multitasking.*

- Resursele comune sunt acele resurse care pot fi *accesate de mai multe taskuri.* Acestea pot fi atât de natură *logică (subrutine, tampoane de date, variabile, etc.)* cât şi fizică *(procesor, memorie, dispozitive de intrare/ieşire etc.).*

La rândul lor resursele comune pot fi *critice, partajabile, reentrante.*

- O resursă comună asupra căreia la un moment dat se poate desfăşura o singură *sesiune de lucru* respectiv care poate fi accesată la un moment dat de către un singur task se numeşte ***resursă critică.*** În această categorie pot fi incluse *procesorul, locaţiile de memorie, echipamente de intrare, subrutinele care îşi modifică pe parcurs unele date proprii, etc.*
- O resursă comună care admite ca ***n*** *sesiuni de lucru* să fie în derulare asupraei la un moment dat *(respectiv să poată fi accesată de către* ***n*** *taskuri)* se numeşte ***resursă partajabilă*** *cu* ***n*** *puncte de intrare.*
- O resursă comună care admite să fie *oricâte sesiuni de lucru* să fie în derulare la un moment dat se numeşte ***resursă reentrantă***[2]**.**

În continuare vor fi tratate unele aspecte referitoare la utilizarea *neconflictuală* a resurselor critice. Secţiunea dintr-un task în care este accesată o resursă critică se numeşte ***secţiune critică.*** În sistemele multitasking trebuie să existe reglementări pentru accesarea *secţiunilor critice.* În acest sens se impune existenţa unor modalităţi de implementare a *excluderii mutuale.*

Excluderea de către un task aflat într-o secţiune critică referitoare la o resursă a accesului altor taskuri de a accesa propriile *secţiuni critice referitoare la aceiaşi resursă* se constituie în ***excludere mutuală.***

Referitor la implementarea *excluderii mutuale* sunt de menţionat recomandările făcute de prof. Andrew Tanenbaum de la Universitatea Vrijie Amsterdam, Olanda în anul 1987.

Sintetic aceste recomandări au în vedere următoarele aspecte:

1 – orice secţiune critică poate fi executată la un moment dat de către un singur task (aceasta este practic esenţa *excluderii mutuale* care afirmă că un singur task se poate afla la un moment dat în propria secţiune critică referitoare la o resursă);

2 – nu se poate face nici o ipoteză în ceea ce priveşte viteza relativă şi frecvenţa de execuţie a taskurilor (cu alte cuvinte la implementarea *excluderii mutuale* nu pot fi avute în vedere considerente legate de frecvenţa sau viteza de execuţie a taskurilor);

[2] Resursele *reentrante* pot fi considerate resurse *partajabile* cu *n* tinzând către infinit.

3 – orice task are dreptul să acceseze propria *secţiune critică* referitoare ala o anumită resursă după un interval finit de timp;

4 – orice task trebuie să evacueze o *secţiune critică* proprie după un interval finit de timp;

5 – taskurile nu se pot bloca în interiorul propriilor secţiuni critice.

În continuare vor fi prezentate câteva modalităţi de implementare a *excluderii mutuale.*

2. Excluderea mutuală realizată cu semafoare

Un *semafor* reprezintă, conform introdus în anul 1965 de către matematicianul olandez Edsger Wybe Dijkstra, un dublet format dintr-o variabilă de tip întreg *I* şi o coadă de aşteptare *C*, respectiv

$S = (I, C)$.

La iniţializare variabilei I i se atribuie o valoare $I_0 \geq 0$, iar coada C este vidă. Dacă variabila *I* ia numai valorile *0 şi 1* semaforul se numeşte *binar* iar dacă ia şi alte valori întregi, semaforul se numeşte *general.*

Asupra semafoarelor pot fi efectuate două operaţii denumite ***P*** (de la cuvântul olandez ***P****asseren – a trece)* şi ***V*** (de la cuvântul olandez ***V****rigeven – a elibera).* Aceste funcţii sunt indivizibile[3] şi se numesc *primitive.* În ceea ce priveşte *coada de aşteptare* , aceasta poate gestionată conform principiului *FIFO*[4], sau după priorităţi (univoce sau neunuivoce).

Primitiva *P(S)* presupune realizarea următoarelor operaţii (ilustrate în organigrama din figura 1):

1 - *I←I-1;*

2 – dacă $I \geq 0$, funcţia se încheie şi taskul în care aceasta se execută îşi continuă execuţia;

3 – dacă $I < 0$, taskul în care se execută primitiva, iese din rândul taskurilor rulabile, se blochează iar indexul său este înscris în coada de aşteptare *C.* În aceste împrejurări se provoacă un proces de comutare care determină trecerea în rulare (*execuţie fizică*) a altui task.

[3] Indivizibilitatea are în vedere faptul că cele două funcţii nu pot fi întrerupte.
[4] **F**irst **I**nput **F**irst **O**utput

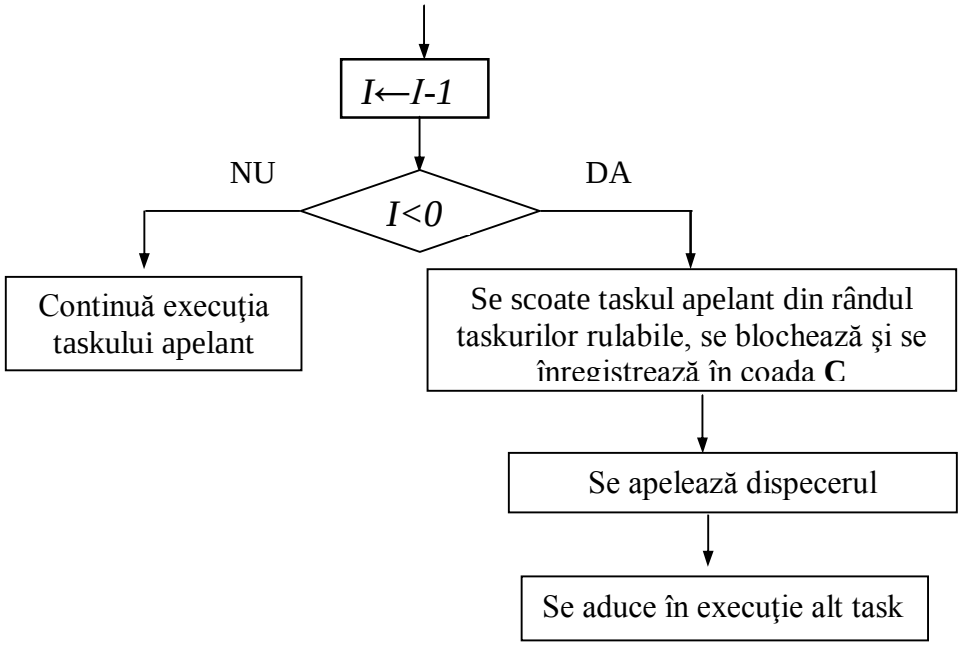

Fig. 1. . Execuţia primitivei *P(S)*.

Primitiva *V(S)* presupune realizarea următoarelor operaţii (ilustrate în organigrama din figura 2.):

1 - *I←I+1;*

2 – dacă $I \le 0$, se deblochează taskul aflat pe prima poziţie în coada de aşteptare la semafor şi se înscrie în rândul taskurilor rulabile, după care se face apel la dispecer, după care continuă execuţia taskului apelant ;

3 – dacă *I > 0,* taskul în care se execută primitiva, îşi continuă execuţia.

În timpul execuţiei primitivei *V(S),* dacă în urma incrementării rezultă *I≤0,* după deblocarea taskului aflat pe prima poziţie în coada **C** se dă controlul dispecerului. Acesta decide funcţie de prioritate dacă aduce sau nu în execuţie taskul deblocat[5].

Din definirea primitivelor *P(S)* şi *V(S)* rezultă următoarele precizări referitoare la valorile variabilei *I* aferente semaforului *(S)*:

- dacă *I > 0* , atunci aceasta reprezintă numărul de taskuri care pot executa primitiva *P(S)* fără a se bloca, presupunând că între timp nu se execută nici o primitivă *V(S);*

[5] În figura 2.21 este surprinsă situaţia în care continuă execuţia taskului apelant, deci taskul deblocat este adus în execuţie logică.

- dacă $I \leq 0$, atunci $|I|$ reprezintă numărul de taskuri blocate la semaforul S şi înregistrate în coada **C.**

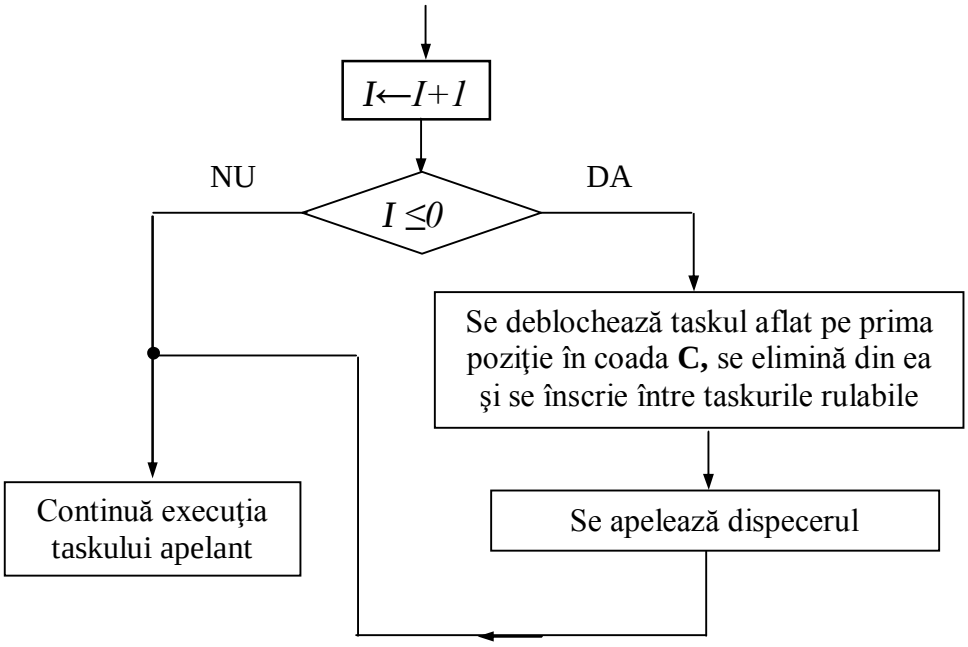

Fig. 3. Execuţia primitivei *V(S)*.

Excluderea mutuală cu semafoare presupune utilizarea unui singur semafor binar, pe care îl vom nota *SEMEX,* care se iniţializează cu valoarea *1*. În fiecare task care trebuie să se excludă mutual, înainte de intrarea în secţiunea critică se execută o directivă *P(SEMEX),* iar după ieşirea din secţiune o directivă *V(SEMEX).*

Este clar că în intervalul de timp în care un task se află în propria secţiune critică referitoare la o anumită resursă *SEMEX=0.* În aceste condiţii orice alt task ce va dori să acceadă în propria secţiune critică referitoare la aceiaşi resursă, va executa directiva *P(SEMEX),* ceea ce va determina blocarea sa întrucât după decrementare *SEMEX =-1*.

La ieşirea din secţiunea critică execuţia directivei *V(SEMEX)* va determina *SEMEX=0* şi conform celor precizate anterior , taskul aflat în prima poziţie în coada **C** aferentă semaforului *S* , va fi deblocat[6] şi va putea pătrunde la rândul său în propria secţiune critică.

[6] Pentru ca soluţia să fie funcţională, semaforul *SEMEX* nu trebuie aservit altor scopuri în afara excluderii mutuale.

Ca exemplu în figura 4 se prezintă schemele logice aferente excluderii mutuale cu semafoare a două taskuri *T1, T2*.. Se observă că cele două taskuri se execută la intervale prestabilite *Δt_ex_T1, Δt_ex_T1*.

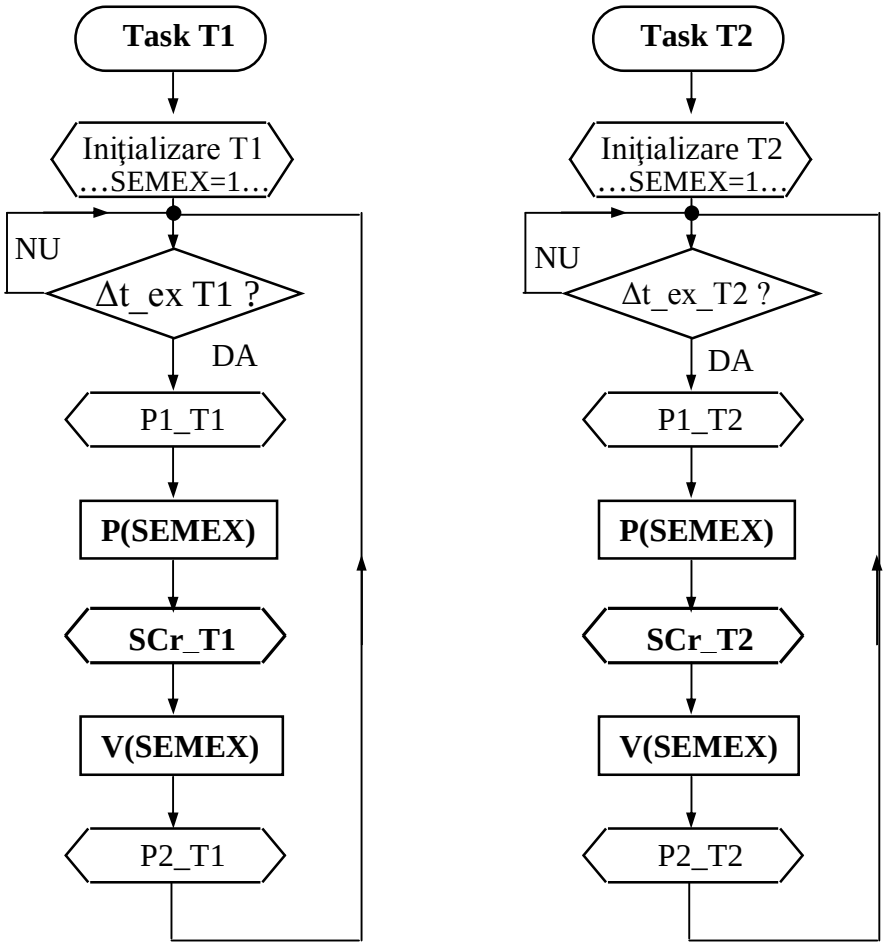

Fig. 4. Excluderea mutuală cu semafoare:
P1_T1, …,P2_T2 proceduri ale taskurilor T1 şi T2.

Având în vedere că *SEMEX* este iniţializat cu valoarea *1,* dacă de exemplu pentru *T1*se impune accesul în secţiunea sa critică, acest lucru va fi posibil întrucât directiva *P(SEMEX)* nu va bloca *Taskul T1*. Dacă în timp ce *T1* se află în secţiunea sa critică *T2* va ajunge la iminenţa intrării în secţiunea sa critică, acest deziderat nu se va putea realiza întrucât directiva *P(SEMEX)* va determina blocarea *Taskului T2* la semaforul *SEMEX*. Deblocarea se va produce când *T1* va termina de executat secţiunea sa critică după care va executa directiva *V(SEMEX).* Ca urmare a acestei directive *SEMEX* va căpăta valoarea *0 (zero)* şi *T2* va deveni rulabil urmând a fi planificat de dispecer pentru execuţia secţiunii sale critice.

Din cele prezentate rezultă că se realizează excluderea mutuală, respectiv un singur task se poate găsi la un moment dat în propria sa secţiune critică.

3. Excluderea mutuală realizată cu mesaje şi cutii poştale

O cutie poştală *(CP)* reprezintă o structură al cărui element central este un tampon circular gestionat conform principiului *FIFO*. Într-o *CP* taskurile pot depune mesaje, accesibile oricărui task. În mod normal o *CP* se specifică prin numărul de poziţii ale tamponului circular şi prin lungimea admisă pentru un mesaj.

Mesajele reprezintă volume de date transferate între taskuri pe parcursul evoluţiei lor. Mesajele pot fi transmise direct între taskuri (caz în care se numesc *mesaje de trecere)* su prin intermediul cutiilor poştale. Din punct de vedere al conţinutului mesajele pot fi cu conţinut *fix* sau *variabil.* Cele cu conţinut fix se numesc *mesaje simbolice* iar celelalte *mesaje informaţionale.* În ceea ce priveşte formatul şi acesta poate fi *fix* sau *variabil.* În mod normal mesajele simbolice sunt mesaje cu format fix.

O funcţie de depunere *(PUT)* a unui mesaj într-o anume *CP* trebuie să conţină un pointer la variabila al cărui conţinut se depune, în timp ce o funcţie de extragere *(GET)* a unui mesaj dintr-o anume *CP* va trebui să conţină un pointer la variabila în care se depune mesajul extras.

Cutiile poştale trebuie astfel gestionate încât să se producă blocarea taskurilor pe *operaţii de depunere* , atunci când *CP* este plină sau pe *operaţii de extragere* atunci când *CP* este vidă.

Excluderea mutuală cu *cutii poştale* presupune utilizarea unei singure *CP* pe care o vom nota *CPEX* în care primul task ce se iniţializează depune un mesaj simbolic, notat *MESEX*. Înainte de intrarea în secţiunea critică fiecare task care trebuie să se excludă mutual va extrage *MESEX* din *CPEX*, iar după ieşirea din secţiunea critică îl va redepune.

Rezultă că în intervalul de timp în care un task se află în secţiunea sa critică referitoare la o anumită resursă cutia poştală *CPEX* este vidă. În aceste condiţii orice alt task ce va dori să acceadă în propria secţiune critică referitoare la aceiaşi resursă, se va bloca la execuţia funcţiei de preluare a mesajului simbolic. Deblocarea[7] se va produce în momentul în care *CPEX* va conţine din nou mesajul simbolic *MESEX*

[7] Utilizarea căsuţei poştală *CPEX* şi a mesajului *MESEX* trebuie să fie numai în excludere mutuală. utilizată numai la implementarea excluderii mutuale.

În figura 5 se prezintă schemele logice aferente excluderii mutuale cu utilizarea cutiilor poştale a două taskuri, *T1*şi *T2*. Ca şi în cazurile precedente cele două taskuri sunt sincronizate cu timpul, intervalele de execuţie fiind *Δt_ex_T1, Δt_ex_T1*.

Având în vedere că la iniţializare cutia poştală *CPEX* va conţine mesajul simbolic *MESEX*, dacă de exemplu pentru *T1*se impune accesul în secţiunea sa critică, acest lucru va fi posibil întrucât prin funcţia de extragere *GET* va putea fi preluat mesajul *MESEX*. Dacă în timp ce *T1* se află în secţiunea sa critică *T2* va ajunge la iminenţa intrării în secţiunea sa critică, acest deziderat nu se va putea realiza întrucât funcţia *GET* va determina *Taskului T2* la cutia poştală *CPEX*. Deblocarea se va produce când *T1* va termina de executat secţiunea sa critică după care va executa funcţia *PUT* de depunere a *MESEX* în *CPEX*.

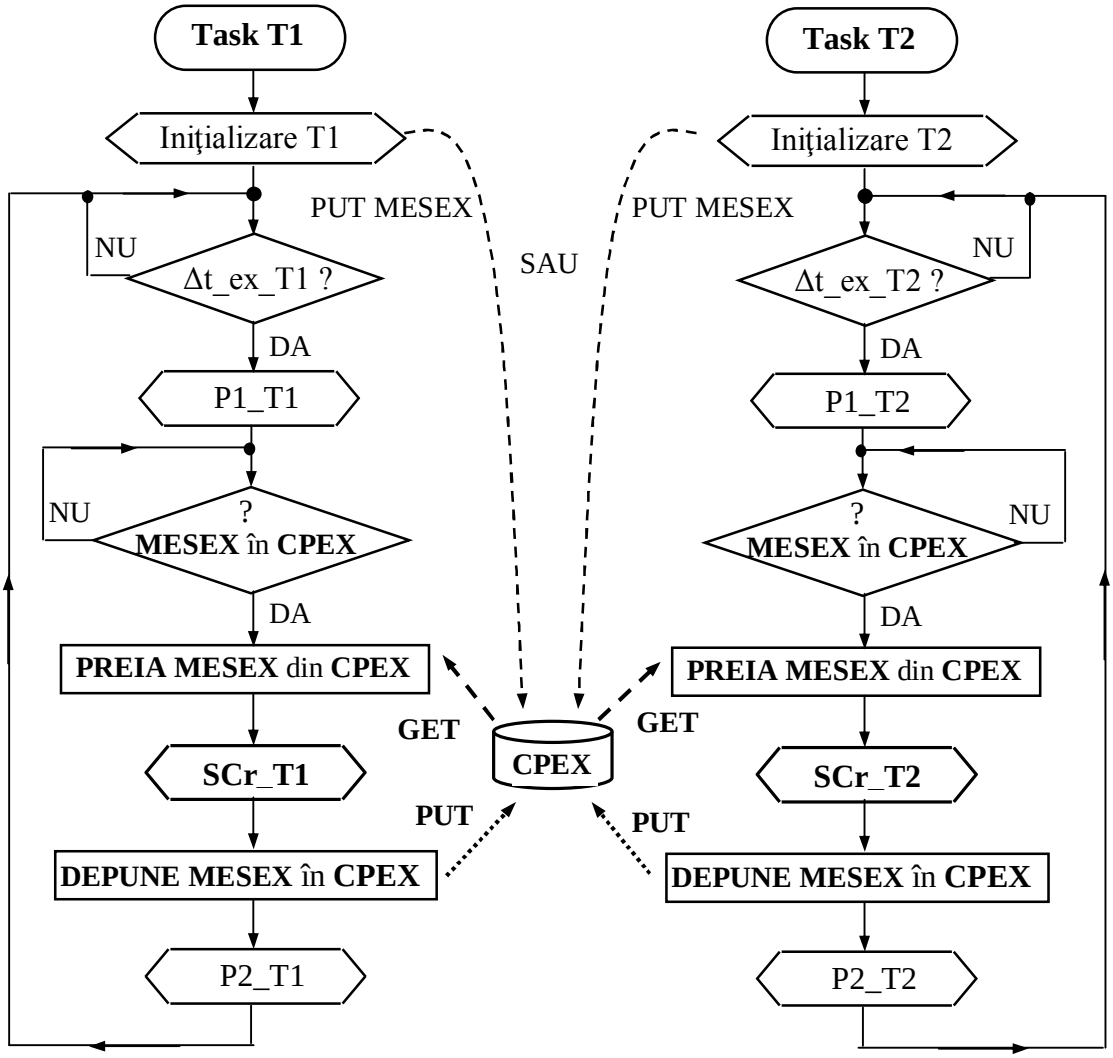

Fig. 5. Excluderea mutuală cu cutii poştale şi mesaje:
P1_T1, …,P2 _T2 proceduri ale taskurilor T1 şi T2.

În urma acestei depuneri *T2* va deveni executabil urmând a fi planificat de dispecer pentru execuţia secţiunii sale critice.

Din cele prezentate rezultă că şi prin utilizarea *cutiilor poştale şi a mesajelor* se poate realiza excluderea mutuală, respectiv se poate asigura prezenţa unui singur task la un moment dat în propria sa secţiune critică.

3. Sincronizarea taskurilor cu utilizarea semafoarelor

De regulă, în evoluţia lor, taskurile unei aplicaţii trebuie să se supună unor relaţii de ordine care să le asigure o anumită *succesiune temporală.*

De exemplu într-o aplicaţie de conducere în timp real un task efectuează achiziţia datelor din proces şi le depune într-un buffer de unde vor fi preluate de taskurile utilizator (reglare, supraveghere, monitorizare etc.). În această situaţie taskurile beneficiare trebuie să aştepte încărcarea buffer-ului, după care în cadrul unei secţiuni critice vor prelua datele.

Sincronizarea taskurilor reprezintă procesul de punerea a unui task în relaţie cu *alt task, cu timpul sau cu un eveniment extern.* Două taskuri se consideră sincronizate dacă se pot stabili relaţii predictibile între anumite momente ale desfăşurării lor. Sensul general al *sincronizării* este acela de coordonare în timp, decorelare.

În mod obişnuit sincronizarea se poate face:

- funcţie de producerea unui eveniment extern taskului;
- funcţie de realizarea unei condiţii de timp.

Sincronizarea cu timpul poate fi tratată ca o sincronizare cu evenimente externe dacă se consideră impulsurile ceasului de timp real ca fiind astfel de evenimente. Pentru realizarea sincronizării sunt necesare funcţii specifice de tratare şi anume:

- *wait* – aşteptare până la producerea evenimentului;
- *signal* – anunţare că evenimentul s-a produs.

Metodele şi mecanismele utilizate pentru realizarea sincronizării se deosebesc sub mai multe aspecte şi anume:

- *natura sincronizării* (cu evenimente externe sau cu timpul);
- *momentul sincronizării (*cu începutul, zona mediană sau sfârşitul unui task):
- *implementarea sincronizării* (prin facilităţi ale limbajelor de programare, ale limbajelor de comandă sau ale monitoarelor).

Din punctul de vedere al sincronizării cu timpul pot exista două situaţii, ilustrate în figura 6, şi anume:

- taskul aşteaptă expirarea unui interval de timp Δt după care se execută (figura 6 a);
- taskul se execută în interiorul cuantei de timp Δt (figura 6 b);.

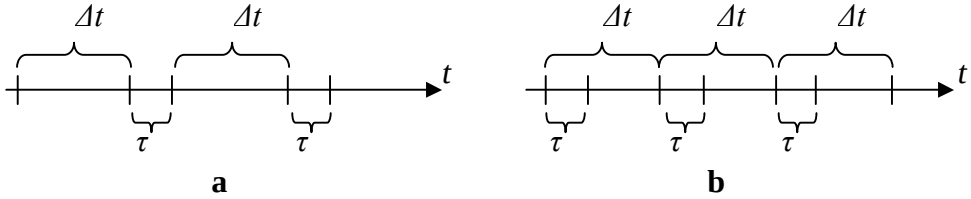

Fig. 6. Sincronizarea cu timpul:
a – intervalul de execuţie τ în afara intervalului de sincronizare Δt;
b – intervalul de execuţie τ în interiorul intervalului de sincronizare Δt.

Se va exemplifica utilizarea semafoarelor pentru un sistem de două taskuri *T1* şi *T2* care trebuie să se sincronizeze reciproc, în fiecare task fiind definit câte un punct de sincronizare *PS1* pentru *T1,* respectiv *PS2* pentru *T2.* Sincronizarea trebuie astfel realizată încât taskul *Ti* să nu poată trece de punctul său de sincronizare *PSi* până când celălalt task *Tj* nu a ajuns în punctul său de sincronizare *PSj* ($i, j \in \{0,1\}$).

Din analiza figurii 7, în care se prezintă sincronizarea cu semafoare, rezultă că punctele de sincronizare se află între procedurile *P1* şi *P2* ale fiecărui task. Cu alte cuvinte un task nu poate executa propria procedură *P2* până când celălalt task nu şi-a executat procedura *P1.*

Sincronizarea se realizează cu ajutorul a doua semafoare binare notate în figura 7 cu *SEMSYNC1* şi *SEMSYNC2* care în taskurile *T1* şi *T2* se iniţializează cu valorile ***1*** respectiv ***0.*** În punctul de sincronizare *PSi* al taskului *Ti* se execută secvenţa *P(SEMSYNCi), V(SEMSYNCj)* cu $i, j \in \{0,1\}$.

Dacă de exemplu *T2* ajunge în *PS2* înainte ca *T1* să îşi execute procedura *P1,* se va bloca întrucât *SEMSYNC2=0.* Taskul se va debloca după ce *T1* depăşeşte *PS1* deoarece va executa directiva *V(SEMSYNC2).*

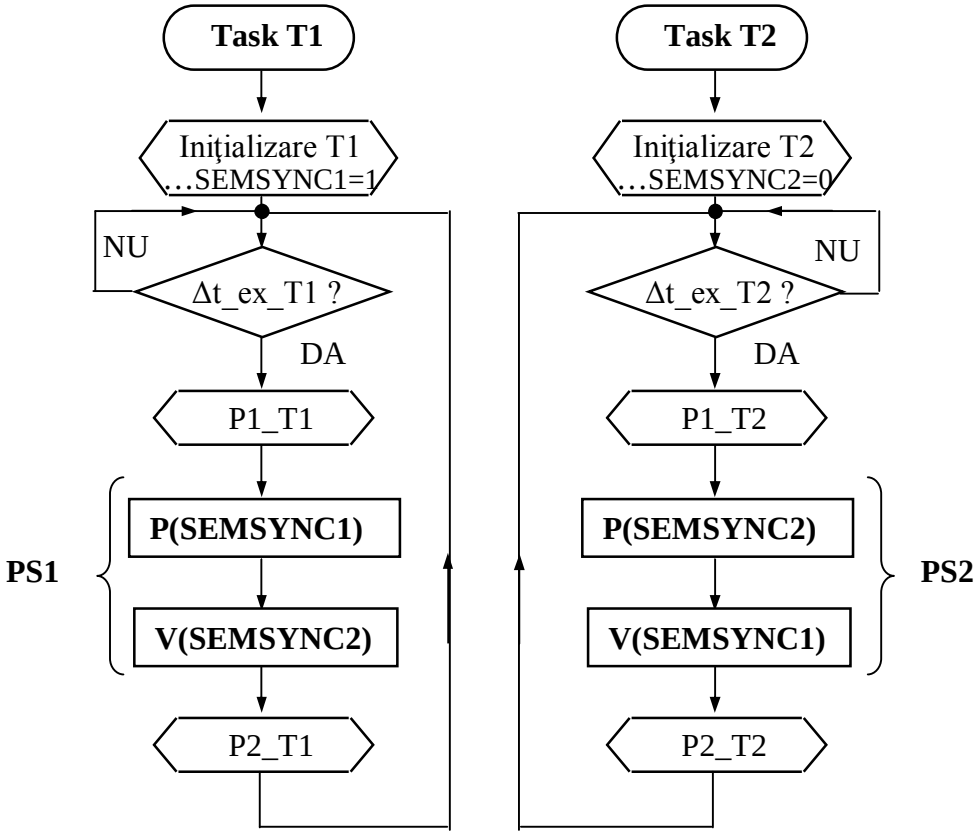

Fig. 7. Utilizarea semafoarelor pentru sincronizarea a două taskuri: P1_T1, …,P2_T2 proceduri ale taskurilor T1 şi T2.

În figura 8 este ilustrată utilizarea sincronizării cu un eveniment extern. Taskul *T1* trebuie să îşi execute procedura *P2* numai după producerea evenimentului extern *EVEXT*. În acest scop se utilizează semaforul de sincronizare *SYMSYNC* care se iniţializează cu valoarea ***0.*** Asupra acestui semafor se execută o directivă de tip *V* în taskul *T0* care supraveghează producerea evenimentului.

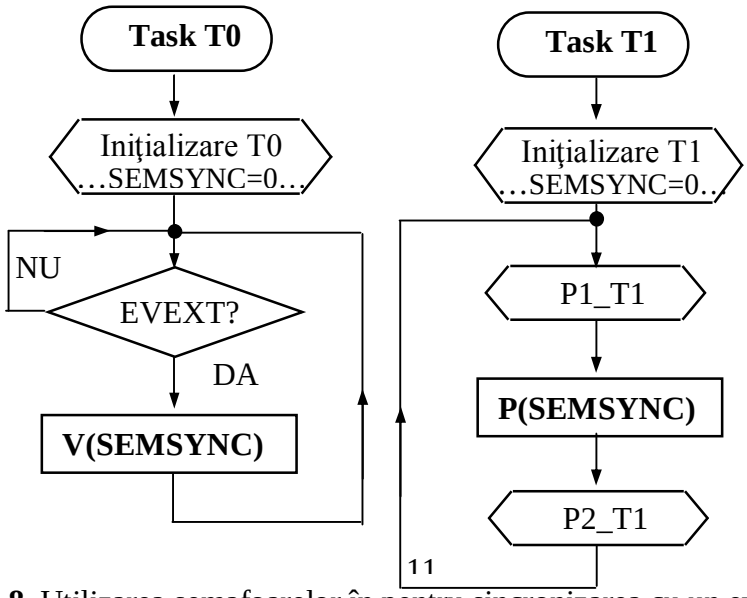

Fig. 8. Utilizarea semafoarelor în pentru sincronizarea cu un eveniment extern: P1_T1, …,P2_T1 proceduri ale taskului T1.

Semafoarele pot fi utilizate şi în sincronizarea cu timpul, în figura 9 fiind prezentat un exemplu în acest sens. Taskul *T1* trebuie să se execute la intervale de timp *Δt_ex_T1* . În acest scop se construieşte taskul *T0* cu rol de planificator. Acesta execută o aşteptare temporizată la semaforul de sincronizare *SYMSYNC* iniţializat cu valoarea ***0.*** Aşteptarea temporizată presupune execuţia directivei *V(SEMSYNC)* la expirarea intervalului *Δt_ex_T1*. Această directivă va debloca taskul *T1,* care între momentele de execuţie aşteaptă pe o funcţie *P* la acelaşi semafor, rezultatul fiind acela că taskul *T1* se va executa cu periodicitatea *Δt_ex_T1*[8] .

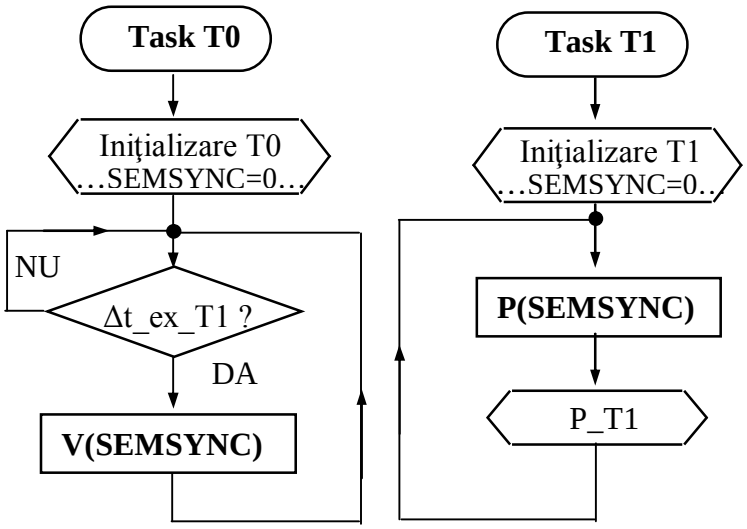

Fig. 9. Utilizarea semafoarelor în pentru sincronizarea cu timpul:
P_T1 procedură a taskului T1.

4. Sincronizarea realizată cu mesaje şi cutii poştale

Cutiile poştale şi mesajele pot fi utilizate atât pentru implementarea sincronizării cu timpul, cât şi cu evenimente externe.

În cazul sincronizării cu timpul se utilizează de regulă *aşteptarea temporizată* la o cutie poştală vidă. Mesajele implicate sunt de regulă *mesaje simbolice,* respectiv mesaje cu format şi conţinut fix. În continuare vor fi prezentate două soluţii de sincronizare bazate pe cutii poştale şi mesaje.

O primă soluţie, ilustrată în figura 10 presupune existenţa a două taskuri *T1- cu rol de planificator* şi *T2 – care trebuie să se execute la intervale Δt.* Soluţia implică prezenţa a trei cutii poştale *C0, C1, C2* cu următoarele funcţii:

[8] Timpul de execuţie al taskului *T1* este inclus în acest interval.

- *C0* – CP destinată aşteptării temporizate;
- *C1* – CP destinată transferului mesajului de sincronizare *MES_SYNC;*
- *C2* – CP destinată transferului mesajului de confirmare *MES_CONF.*

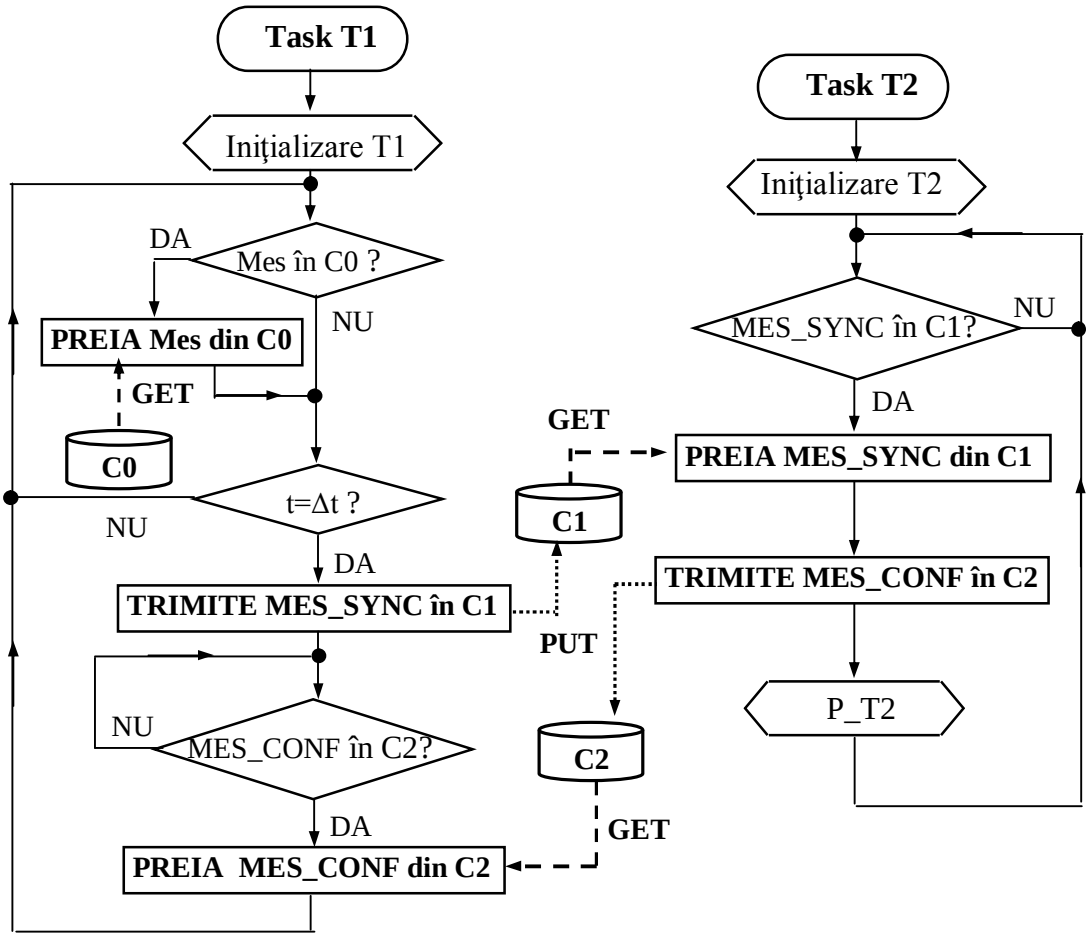

Fig. 10. Utilizarea cutiilor poştale pentru sincronizarea cu timpul:
C0, C1, C2 – cutii poştale; P T2 procedură ale taskului T2.

Taskul *T1* realizează următoarea secvenţă de operaţii:

1.1 – aşteaptă un interval de timp *Δt (intervalul de sincronizare)* la cutia poştală vidă *C0;*

1.2 – la expirarea acestui intervalul transmite în cutia poştală *C1* mesajul de sincronizare *MES_SYNC;*

1.3 – aşteaptă şi preia din cutia poştală *C2* mesajul de confirmare *MES_CONF;*

1.4 – se revine la pasul 1.1 , intrându-se din nou în starea de aşteptare temporizată.

Mesajul de confirmare *MES_CONF* este un mesaj cu formă fixă care informează taskul planificator *T1* în legătură cu funcţionalitatea taskului *T2*, care trebuie să se execute temporizat.

În ceea ce priveşte taskul *T2* , acesta execută următoarea secvenţă:

2.1 – aşteaptă şi preia din cutia poştală *C1* mesajul de sincronizare *MES_SYNC;*

2.2 - transmite în cutia poştală *C2* mesajul de confirmare *MES_CONF;*

2.3 – execută procedura *P-T2.*

Din secvenţa de mai sus rezultă faptul că execuţia taskului *T2* se produce în interiorul cuantei de timp *Δt* , aspect evidenţiat în figura 11.

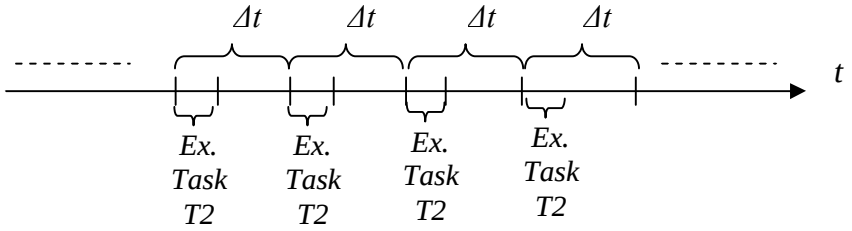

Fig. 11. Sincronizarea cu timpul: execuţia taskului *T2* se face în interiorul cuantei de timp *Δt.*

Este posibilă execuţia taskului *T2* în afara cuantei *Δt,* conform reprezentării din figura 12.

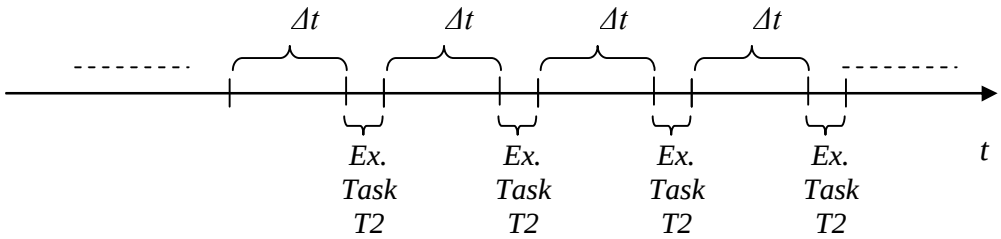

Fig. 12. Sincronizarea cu timpul: execuţia taskului *T2* se face în exteriorul cuantei de timp *Δt.*

În această situaţie nu se mai impune existenţa taskului planificator *T1*. Taskul *T2* aşteaptă la cutia poştală vidă *C0* un interval de timp *Δt,* deblocarea producându-se la expirarea acestei cuante de timp. În figura 13 se prezintă structura adaptată a taskului *T2* pentru acest tip de sincronizare.

Se observă că nu mai sunt necesare mesaje de sincronizare şi de confirmare. Dacă se doreşte monitorizarea funcţionalităţii taskului *T2,* atunci se menţine taskul *T1* care va primeşte la fiecare execuţie a lui *T2* câte un mesaj de confirmare.

Procedeul sincronizării cu timpul, în condiţiile existenţei unui task planificator poate extins şi la implementarea sincronizării cu un eveniment extern, corespunzător reprezentării din figura 14.

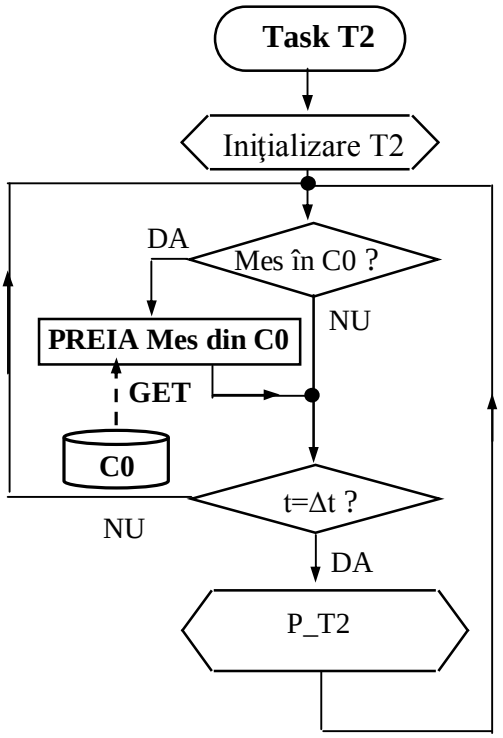

Fig. 2.33. Utilizarea unei cutii poştale vide pentru sincronizarea cu timpul: C0 – cutie poştală; P_T2 procedură ale taskului T2.

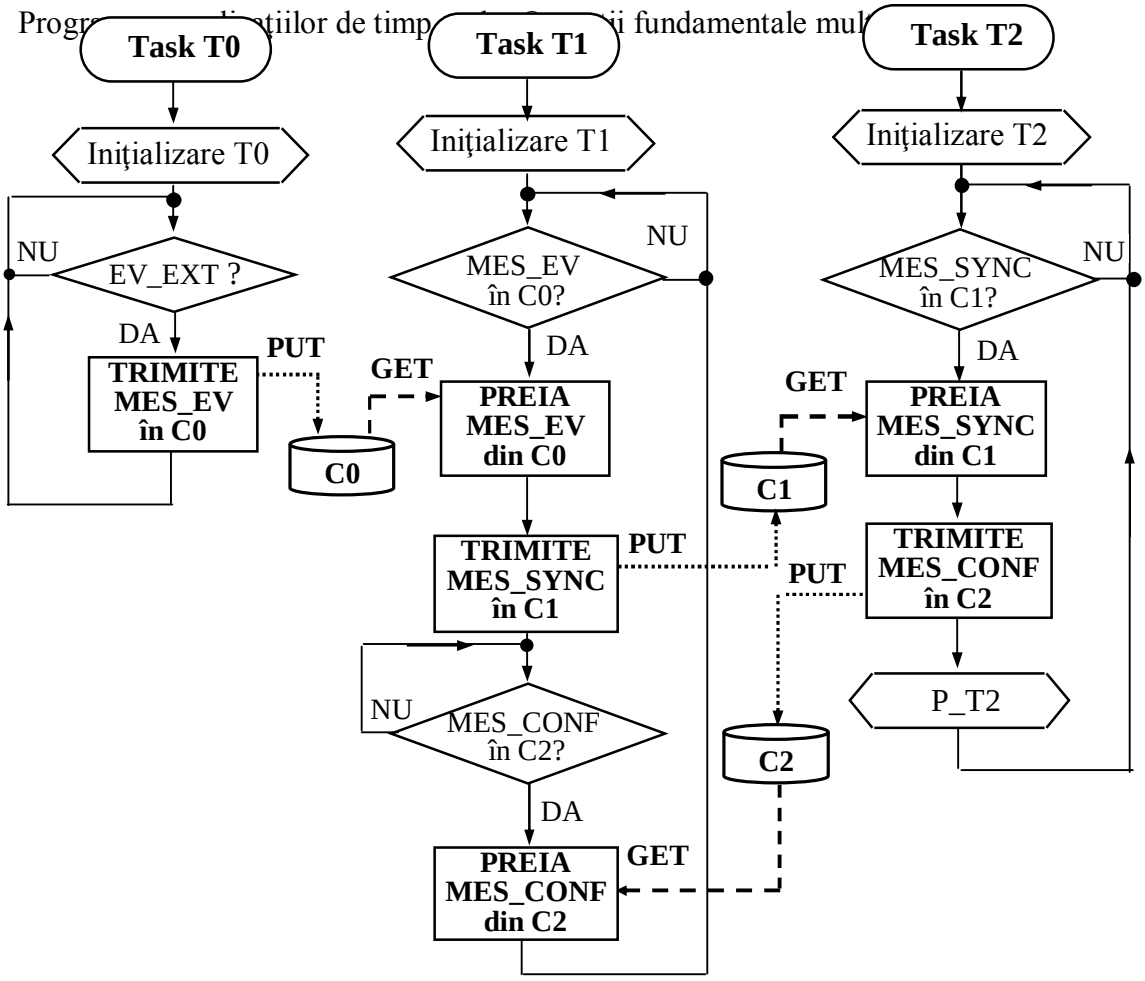

Fig. 14. Utilizarea cutiilor poştale pentru sincronizarea cu evenimente externe: C0, C1, C2 – cutii poştale; P_T2 procedură ale taskului T2.

După cum se observă, înainte de producerea evenimentului *EV_EXT* taskurile *T0, T1, T2* sunt blocate după cum urmează:

- *T0* – în aşteptarea producerii evenimentului *EV_EXT;*
- *T1* – la cutia poştală *C0* în aşteptarea mesajului *MES_EV* care confirmă producerea evenimentului;
- *T2* – la cutia poştală *C1* în aşteptarea mesajului de sincronizare *MES_SYNC.*

Este important de subliniat faptul că după primirea mesajului de sincronizare, *T2* trebuie să depună în cutia poştală *C2* mesajul de confirmare *MES_CONF.*

Ca şi în cazul sincronizării cu timpul, taskul *T1* poate fi eliminat în situaţia în care nu se doreşte monitorizarea funcţionalităţii taskului *T2*, soluţia principială fiind prezentată în figura 15.

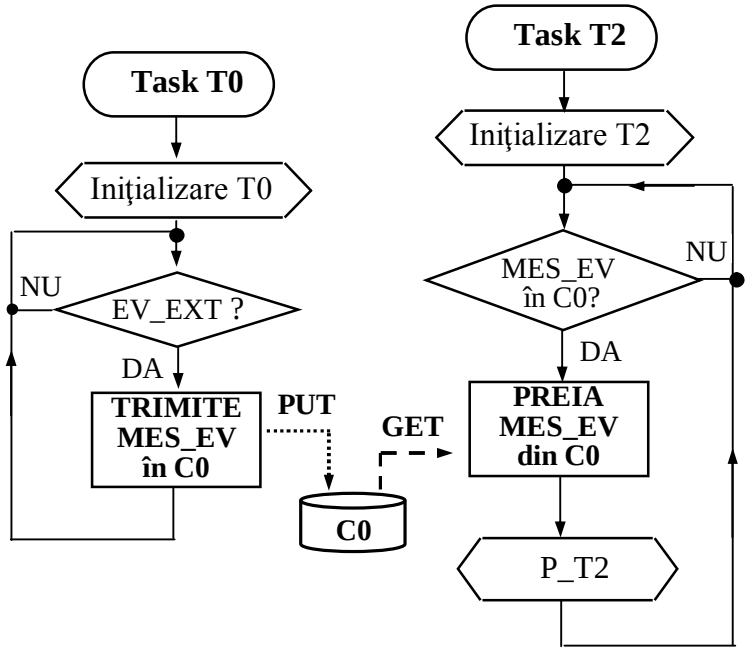

Fig. 15. Utilizarea unei cutiilor poştale pentru sincronizarea cu evenimente externe: C0, – cutie poştale; P_T2 procedură ale taskului T2.

După cum se observă între momentele producerii evenimentului extern *EV_EXT* taskurile *T0* şi *T2* sunt blocate după cum urmează:

- *T0* – în aşteptarea producerii evenimentului *EV_EXT;*
- *T2* – la cutia poştală *C0* în aşteptarea mesajului *MES_EV* referitor la producerea evenimentului extern.

În aceste condiţii taskul *T2* este deblocat în momentul în care în cutia poştală *C0* este depus *MES_EV* asociat producerii evenimentului extern.

5. Utilizarea cutiilor poştale şi mesajelor în comunicare

După cum s-a arătat procesarea datelor în regim concurent multitasking implică pe lângă utilizarea de resurse în comun şi schimbul de informaţie între taskuri concretizat în operaţia de *comunicare*. Uzual taskurile transferă informaţia sub formă de mesaje care, după cum s-a văzut la prezentarea excluderii mutuale pot fi cu conţinut *variabil (mesaje informaţionale)* sau *fix (mesaje simbolice).*

În mod obişnuit comunicarea se implementează prin mecanismul *producător-consumator,* taskurile putând fi din acest punct de vedere

- taskuri de tip *producător;*
- taskuri de tip *consumator.*

Pentru facilitarea comunicării sistemele de operare trebuie să pună la dispoziţie instrumente specifice, unul dintre acestea fiind conducta *(pipe).* O conductă reprezintă un tampon unidirecţional gestionat conform principiului *FIFO,* în care un task producător depune mesaje pe care le preia un task consumator.

Mecanismul de comunicare prin conductă trebuie să asigure blocarea taskului *producător* care ajunge în faţa unei operaţii de scriere şi conducta este plină. Acelaşi mecanism va trebui să blocheze un task de tip consumator aflat în situaţia de preluare a unui mesaj (citire) dintr-o conductă goală.

În situaţia în care două taskuri au atât rol de producător, cât şi de consumator vor trebui utilizate două conducte, situaţie evidenţiată în figura 16.

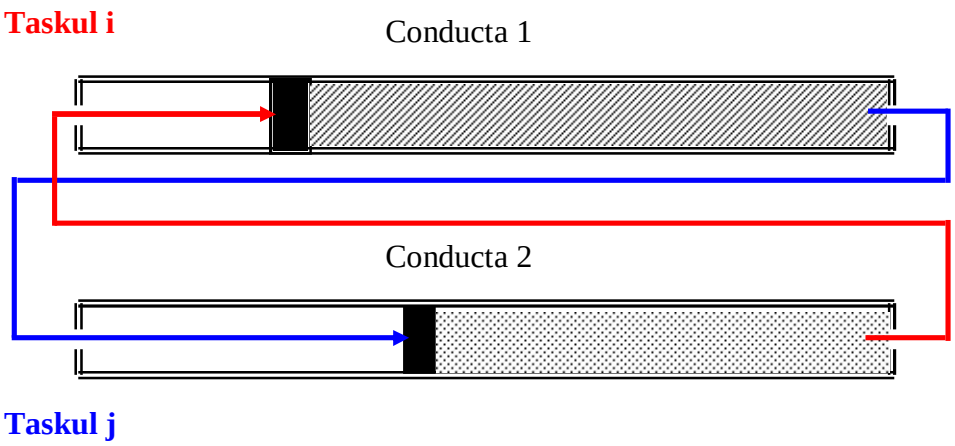

Fig. 16. Ilustrarea comunicării prin conducte între două taskuri.

După cum se observă *Conducta 1* este deschisă la scriere pentru *taskul i* şi la citire pentru *taskul j,* cu alte cuvinte cele două taskuri îndeplinesc rolurile de *producător* respectiv *consumator.* În ceea ce priveşte *Conducta 2,* aceasta este deschisă la scriere pentru *taskul j* şi la citire pentru *taskul i,* rolurile celor două taskuri fiind inversate respectiv *taskul i – consumator, taskul j – producător.*

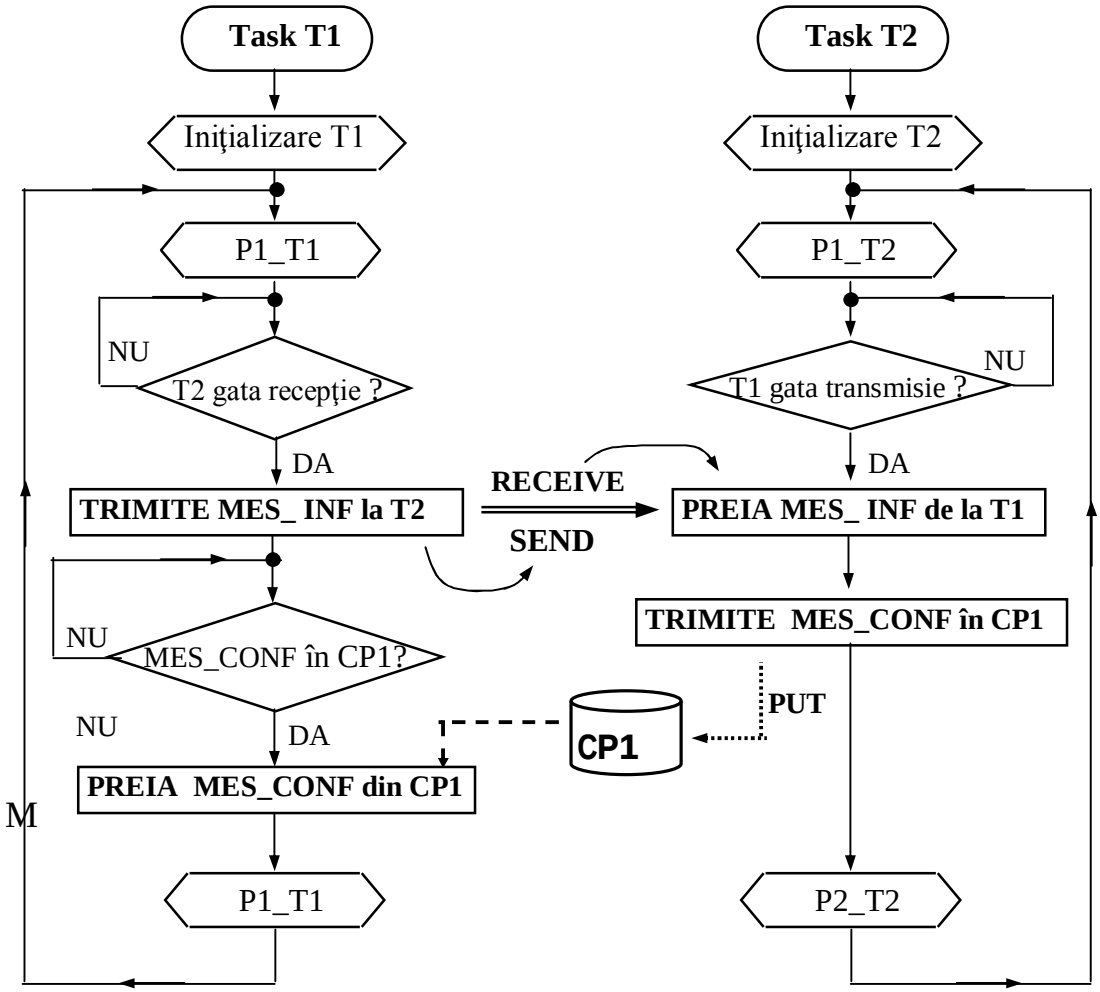

Fig. 17. Utilizarea mesajelor de trecere şi a cutiilor poştale în comunicare.

Mesajele de trecere şi cutiile poştale reprezintă alte două mijloace destinate asigurării comunicării dintre taskuri. Mesajele se trimit direct de la un taskul emiţător către cel receptor, în timp ce cutiile poştale reprezintă facilităţi de comunicare care pot fi utilizate de către toate taskurile aferente aplicaţiei de timp real.

În figura 17 se prezintă un exemplu de comunicare între două taskuri în care se utilizează ambele modalităţi. Mesajul informaţional *MES_INF* este transmis de *Taskul T1* sub forma unui mesaj de trecere în timp ce *Taskul T2* va trimite un mesajul simbolic de confirmare *MES_CONF* prin intermediul cutiei poştale *CP1*.

Este important de subliniat faptul că transmiterea unui mesaj de trecere este posibilă dacă taskul receptor este gata să îl primească. În ceea ce priveşte receptorul acesta se va bloca în aşteptarea mesajului dacă taskul emiţător nu este gata să îl transmită.

În ceea ce priveşte mesajul de confirmare, *Taskul T1* se blochează până la depunerea acestuia în cutia poştală *CP1*.