MINISTERUL ÎNVĂȚĂMÂNTULUI
UNIVERSITATEA TEHNICĂ DIN TIMIȘOARA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SOLUȚII PRIVIND MODELAREA, CONDUCEREA ȘI PROGRAMAREA ÎN FABRICAȚIA INTEGRATĂ PRIN CALCULATOR

~TEZĂ DE DOCTORAT~

AUTOR:
Nicolae ROBU

CONDUCĂTOR ȘTIINȚIFIC:
Prof.dr.ing. Nicolae BUDIȘAN

CUPRINS

1
INTRODUCERE

Dezvoltarea industriei după cel de-al doilea război mondial s-a făcut, timp de treizeci de ani, subordonat, dominant, conceptelor economiei de producție [WALD'90]. Motorul acesteia l-a reprezentat obiectivul maximizării profiturilor prin creșterea seriilor de fabricație. Cantitatea de produse promovate pe piață ca rezultat al acestei economii a devenit, la mijlocul deceniului șapte, superioară cererii [OHMA'85]. Situația creată a condus la confruntarea intreprinderilor industriale cu însemnate greutăți. A fost evidențiată marea lor inerție și, în consecință, incapacitatea de a se adapta rapid, în condiții de eficiență, la fluctuațiile de ordin cantitativ ale cererii și la nuanțarea acesteia sub aspect calitativ [ARCH'84].

Analizele efectuate în atarea situație au impus concluzia că întreaga industrie trebuie regândită în jurul postulatului că ***principala problemă este nu a produce, ci a vinde*** [WALD'90]. Iar a vinde, presupune a produce, în fiecare moment, ceea ce se cere, cât se cere, și cum se cere, cu costuri suficient de mici. Este aceasta axioma fundamentală a ceea ce, într-un înțeles actualizat, poartă numele de economie de piață [ARCH'84].

Promovarea economiei de piață moderne a însemnat, în plan tehnic, dar, de altfel, nu numai tehnic, flexibilizarea producției industriale. Locul sistemelor de producție specializate, dezvoltate bazat pe mașini și utilaje, respectiv linii de fabricație, rigide, concepute sub deviza "capacitate de producție, randament energetic și/sau randament substanțial maxime", a fost luat, în multe sectoare, inevitabil, de sistemele de producție suple, bazate pe mașini și utilaje, respectiv pe ansambluri de mașini și utilaje, flexibile, adică apte să treacă rapid și cu cheltuieli mici de la un produs la altul și să asigure profit pentru un interval larg de variație a cererii, sub aspectul cantitativ [MOTT'89].

Flexibilizarea mașinilor și utilajelor, respectiv a ansamblurilor de mașini și utilaje, a fost posibilă, atât din punct de vedere tehnic, cât și din punct de vedere al eficienței economice, abia odată cu avansarea tehnologiei informatice. Aceasta a permis, din anii '80, automatizarea aproape totală a fabricației, într-o concepție nouă, fondată pe ideea integrării, într-o viziune sistemică, a tuturor aspectelor sale intrinseci și a unei însemnate părți a celor conexe [MOTT'89]. Noua concepție are ca element central conceptul, și el nou, de Fabricație Integrată prin Calculator (*CIM: Computer Integrated Manufacturing*). Acest concept subsumează [WALD'90], așa cum se arată în figura 1_1: Fabricația Asistată de Calculator (*CAM: Computer Aided Manufacturing*), Proiectarea Asistată de Calculator (*CAD: Computer Aided Design*), Ingineria Asistată de Calculator (*CAE: Computer Aided Engineering*), și Administrarea Asistată de Calculator (*CAMg: Computer Aided Management*). Liantul acestora în cadrul conceptului *CIM* este, evident, informația.

Importanța deosebită acordată în lume promovării conceptului de fabricație integrată este relevată de numeroasele proiecte lansate în acest sens. Astfel, începând din anul 1984,

Comunitatea Europeană a angajat programele "*ESPRIT I*" și "*ESPRIT II*" (*ESPRIT: European Strategic Program for Research and Developement in Information Technology*), a căror finanțare se ridică la suma de 5 miliarde ECU [WALD'90]. În cadrul acestor programe, s-a evidențiat

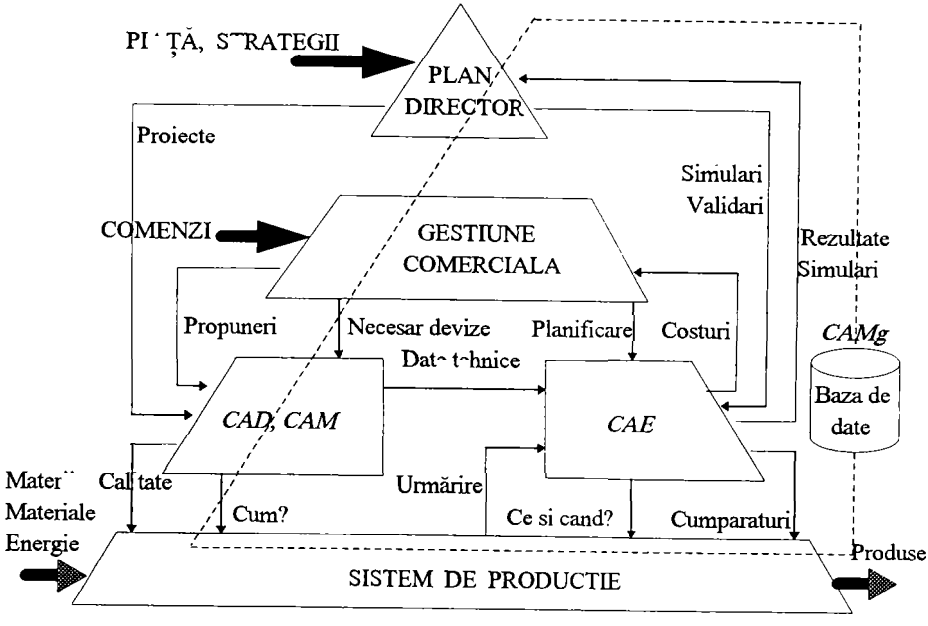


Fig. 1_1. Ilustrarea conceptului *CIM*

proiectul *ECIMA* (*ECIMA: European Computer Integrated Manufacturing Architecture*), care a condus la modelul arhitectural de sistem *CIM* denumit "*CIM-OSA*" (*CIM-OSA: Computer Integrated Manufacturing-Open System Architecture*). În paralel, s-a derulat, începând din anul 1985, proiectul "*CNMA*" (*CNMA: Communications Network for Manufacturing Applications*), iar concernele General Motors și Boeing au scos standardele "*MAP*" (*MAP: Manufacturing Automation Protocol*), respectiv "*TOP*" (*TOP: Technical and Office Protocol*) [CROW'86].

Din vasta problematică a fabricației integrate prin calculator, lucrarea de față și-a propus doar:

- să evidențieze importanța acordată actualmente în lume fabricației integrată prin calculator și, deopotrivă, principiile prin care aceasta este, în prezent, abordată (capitolele 1 și 2);
- să efectueze un excurs asupra sistemelor flexibile -temelie a fabricației integrate prin calculator (capitolul 3);
- să ofere, cu ajutorul unor instrumente moderne și eficace (rețelele Petri de tip condiții-evenimente, rețelele Petri de tip poziții-tranziții, rețelele Petri colorate, rețelele Petri continue) modele ale unor subsisteme și sisteme flexibile de fabricație cu corespondență imediată în realitatea tehnică și, în consecință, cu valoare de întrebuințare (capitolul 4);

- să definească, de pe poziții inginerești, o metodă de rezolvare a problemei ordonanțării în unul dintre cazurile fundamentale de fabricație integrată prin calculator, cu detaliere la nivel de pseudocod a algoritmilor pe care metoda se sprijină (capitolul 5);
- să aducă un set de soluții apte să stea la baza generării unui suport eficient de programare, în regim concurent și în timp real a aplicațiilor din domeniul fabricației integrate prin calculator, dar, inevitabil, nu numai (capitolul 6).

2
VIZIUNI ACTUALE ASUPRA FABRICAȚIEI INTEGRATE PRIN CALCULATOR. CONCEPTUL *CIM-OSA*

Complexitatea problematicii fabricației integrate prin calculator și importanța deosebită pe care acest domeniu o prezintă au determinat marile puteri economice să încerce promovarea unor norme privind abordarea sa [ARSE'84].

Astfel, așa cum, deja, s-a menționat, prin unul dintre programele Comunității Europene, a fost definit modelul arhitectural de sistem de fabricație integrată prin calculator, cu ajutorul conceptului lansat sub sigla "*CIM-OSA*". La baza dezvoltării acestui concept, a stat dorința de a cuprinde, de o manieră sistematică, toate aspectele semnificative, specifice unui sistem de fabricație integrată prin calculator [BARB'92]. A rezultat că trei axe trebuie luate în considerare:

- axa de modelare;
- axa de genericitate;
- axa vederilor.

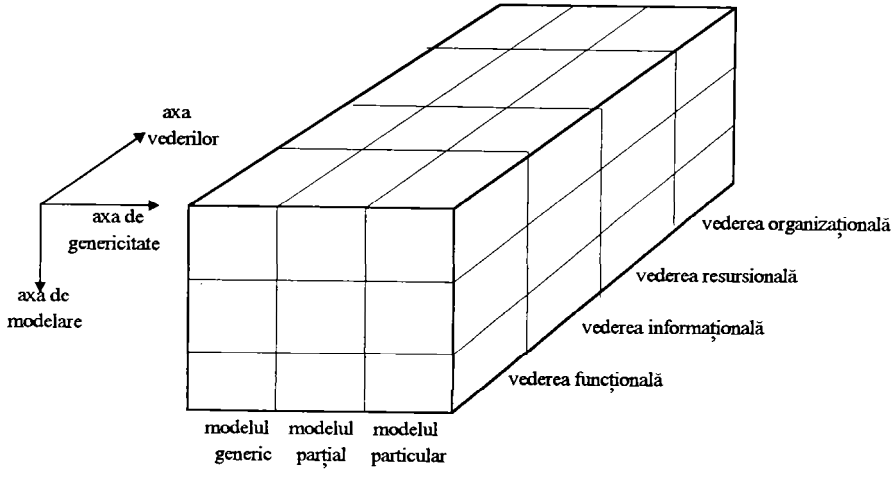


Fig. 2_1. Cubul *CIM-OSA*.

Axa de modelare surprinde aspectele legate de exprimarea cerințelor, specificarea funcțională, și descrierea de realizare.

Axa de genericitate este destinată surprinderii elementelor componente ale intreprinderii, într-o abordare graduală, de la general la particular.

Axa vederilor evidențiază aspectele funcționale, informaționale, resursionale, și organizaționale.

De o manieră sugestivă, modelul *CIM-OSA* se prezintă sub forma unui paralelipiped (care, habitual, este numit "*cubul CIM-OSA*"), format din 36 cuburi, așa cum se arată în figura 2_1.

Cu titlu ilustrativ, în figura 2_2, se prezintă sistemele unei intreprinderi și modul în care ele sunt surprinse de vederile *CIM-OSA* [BARB'92].

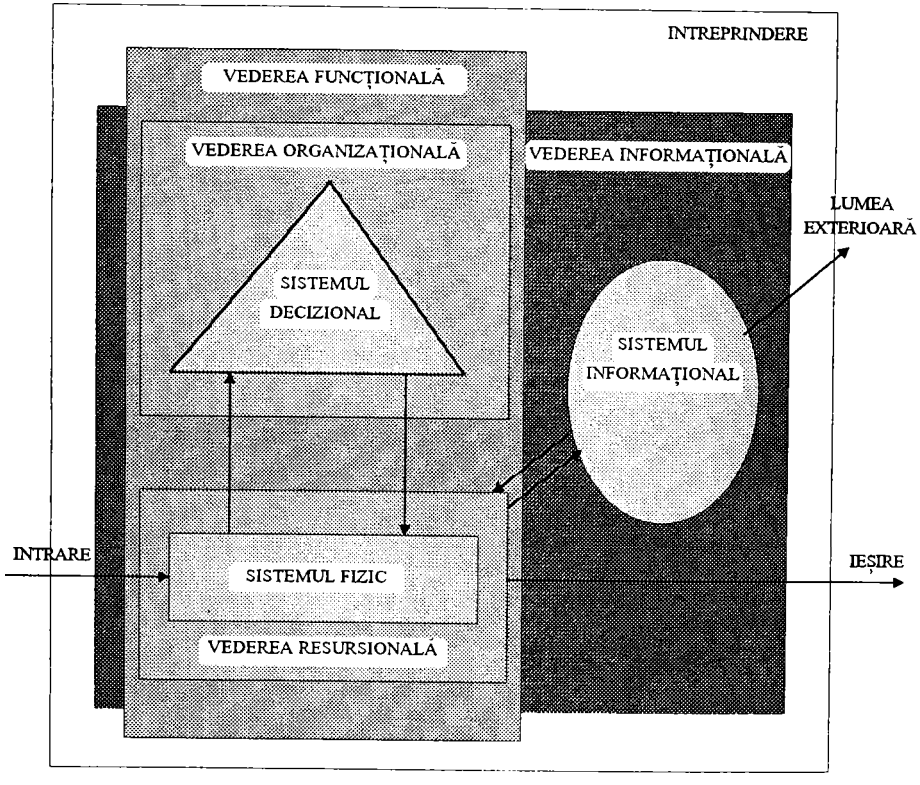


Fig. 2_2. Sistemele unei intreprinderi în accepțiunea *CIM-OSA*.

Sunt de remarcat [BARB'92] următoarele:

a) Sistemul fizic, constituit din ansamblul resurselor fizice, este surprins prin vederea funcțională, vederea resursională, și, în parte, prin vederea informațională.

b) Sistemul decizional, cuprinzând ansamblul funcțiilor de previziune, planificare, control, mentenanță, și administrative, este surprins prin vederea funcțională, vederea organizațională, și, în parte, vederea informațională.
c) Sistemul informațional, constituit din ansamblul datelor și informațiilor specifice, este surprins prin vederea informațională.

Detalii privind conceptul *CIM-OSA* -în special legate de principiile pe care el le conține referitor la tehnologia informațională, principii înglobate în mediul integrat *IDPE* (*IDPE: Integrated Data Processing Environment*) - se găsesc în [ROBU'95a].

3
EXCURS ASUPRA SISTEMELOR FLEXIBILE DE FABRICAȚIE

3.1. Nivelele ierarhice ale structurilor flexibile de fabricație

Nivelul 1

Nivelul 1 cuprinde cele mai mici entități cu funcții autonome în procesul de fabricație. Generic, aceste entități sunt mașini de lucru multifuncționale -cu capabilități de prelucrare sau de asamblare-, roboți de deservire, respectiv dispozitive de transport [KUSI'85d]. Fiecare dintre ele dispune de un echipament de comandă numerică propriu. În ceea ce privește mașinile de lucru multifuncționale, se face mențiunea că, în cele mai multe cazuri, ele se instanțiază prin centre de prelucrare [CĂLI'88], respectiv roboți industriali.

Nivelul 2

Nivelul 2 este definit ca un ansamblu de entități de nivel 1, reunite după criteriul gradului de conjugare a capabilităților lor, în sensul oferirii unor servicii complexe de prelucrare sau de asamblare, respectiv de transport, executabile autonom [KUSI'85d]. În funcție de serviciile pe care le oferă, un asemenea ansamblu este denumit, respectiv, celulă flexibilă de fabricație, celulă flexibilă de asamblare, sau subsistem de transport.

O celulă flexibilă de fabricație cuprinde un număr de mașini de prelucrare multifuncționale (uzual 1...4), unu/două manipulatoare și/sau alte dispozitive de încărcare/descărcare a pieselor și sculelor, respectiv de evacuare a reziduurilor, și, în unele cazuri, depozite tampon de scule, de semifabricate, de piese finite, și de reziduuri [CĂLI'88].

O celulă flexibilă de asamblare cuprinde un număr de roboți industriali -asociați, eventual, cu sisteme senzoriale externe-, și depozite tampon de piese și scule [ARCH'84].

Subsistemele de transport cuprind, de regulă, conveioare și/sau cărucioare. Acestea pot fi ghidate mecanic (prin șine) sau nemecanic (prin procedee optice, inductive, sau prin unde radio) [KUSI'85d] [KUSI'90].

Celulele flexibile, respectiv subsistemele de transport, au, de asemenea, în componența lor, câte un echipament de conducere [HUVE'92].

Nivelul 3

Nivelul 3 cuprinde un număr de celule flexibile de fabricație și/sau de asamblare, și un subsistem de transport, stabilite astfel încât, împreună, să poată achita sarcini de producție complexe, pentru care este justificat un proces de ordonanțare. Desemnăm ansamblul celulelor flexibile prin termenul de subsistem tehnologic [DEWE'88].

De asemenea, nivelul 3 este dotat cu un calculator, cu care se asigură procesul de ordonanțare, precum și comunicarea cu nivelele adiacente [HUVE'92].

Totodată, în cadrul acestui nivel, sunt prevăzute depozite de semifabricate, de scule, de piese finite, și de reziduuri, precum și standuri de calibrare și reparare a sculelor [CĂLI'88].

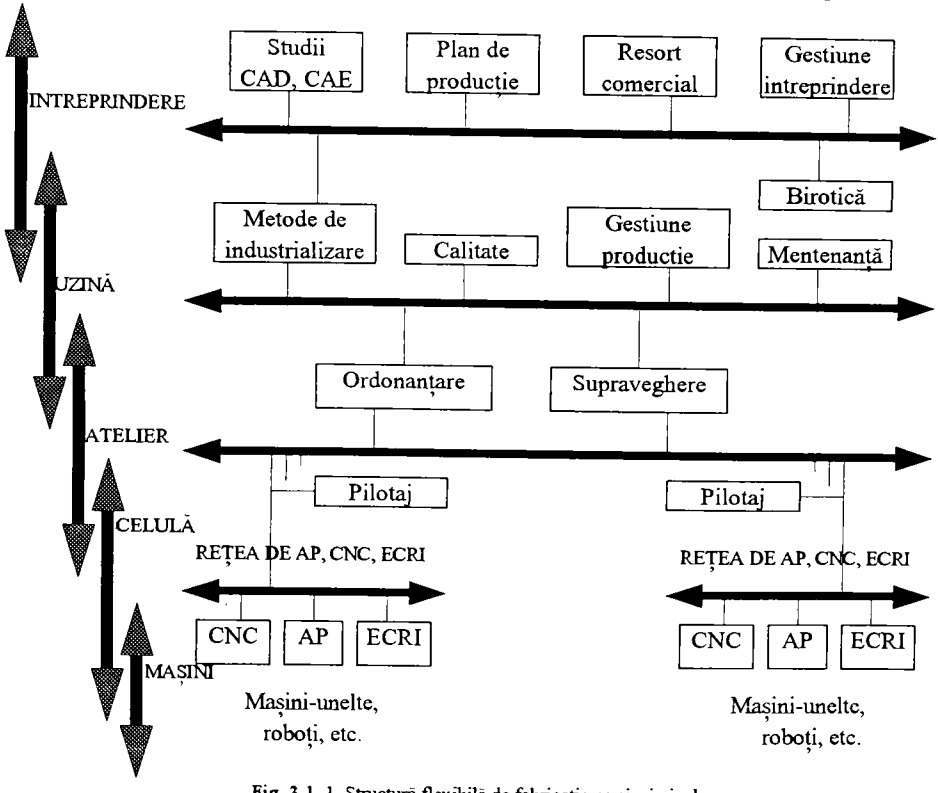


Fig. 3.1_1. Structură flexibilă de fabricație pe cinci nivele .

De remarcat că, dacă nivelele 1 și 2 comportă, în exclusivitate, activități de factură tehnică sau tehnologică -cum sunt: transport scule, semifabricate, piese, și reziduuri; prelucrare semifabricate; măsurare piese; măsurare scule; etc.-, la nivelul 3, intervin activități de un cu totul alt specific: activitățile de ordonanțare [FROM'84].

Se spune că nivelul 3 reprezintă atelierul flexibil [DEWE'88].

Nivelul 4

Nivelul 4 înglobează un ansamblu de ateliere, alături de compartimente dedicate activităților *CAM,* respectiv unor activități de urmărire a calității, de mentenanță, și de management [ARCH'84].

Se spune că nivelul 4 reprezintă uzina flexibilă [WALD'90].

Nivelul 5

Nivelul 5 este nivelul intreprinderii. Aici sunt concentrate compartimentele de marketing, planificare, proiectare (*CAD*), inginerie (*CAE*) [WALD'90].

Figura 3.1_1 ilustrează cele de mai sus, privitoare la structura sistemelor flexibile de fabricație.

Este evident, sistemele flexibile de fabricație au o structură de tip piramidal. La fiecare nivel al structurii, intervin funcții de conducere specifice. Cele mai importante dintre acestea sunt [CĂLI'88]:

1) comanda locală a mașinilor de lucru, a roboților de deservire și a dispozitivelor de transport, precum și a dispozitivelor conexe ale acestora;
2) coordonarea în cadrul celulelor de mașini de lucru și roboți de deservire, respectiv în cadrul subsistemului de transport;
3) ordonanțarea fabricației la nivelul celulelor și subsistemului de transport;
4) ordonanțarea și lansarea fabricației la nivelul atelierului;
5) planificarea fabricației.

Primele trei funcții sunt de tip "*timp real*" și se referă la procesul propriu-zis de fabricație; celelalte țin de procesul de gestiune a fabricației. Se precizează că este manifestă tendința ca și ele să fie exercitate în timp real -desigur, la o altă scară-, ca rezultat al noilor tehnici și tehnologii promovate în domeniu [WIDM'91].

Simpla enumerare a principalelor funcții de conducere intrinseci sistemelor flexibile de fabricație evidențiază postarea într-o anumită relație ierarhică a unora dintre ele în raport cu altele. Ierarhia funcțiilor respectă, în linii mari, ierarhia structurilor, fără a se putea vorbi, însă, de o corespondență bijectivă între ele. Mai mult, abordarea practică a problematicii sistemelor flexibile de fabricație a condus la concluzia că, în privința funcțiilor, este inadecvată considerarea unui model unic de ierarhizare [KUSI'85d] [KUSI'90], exercitarea acestora făcându-se în cadrul unor sisteme distribuite de conducere, concepute în mod specific, de la caz la caz. Întru susținerea acestei concluzii, în subcapitolul următor, se prezintă două topologii tipice de sisteme de conducere.

3.2. Topologii tipice de sisteme de conducere a structurilor flexibile de fabricație

Exemplul 1

Fie sistemul flexibil -un atelier- schematizat în figura 3.2_1 [GURA'86]. Topologia tipică pentru sistemul de conducere aferent unui asemenea sistem flexibil este cea reprezentată în figura 3.2_2 [CĂLI'88].

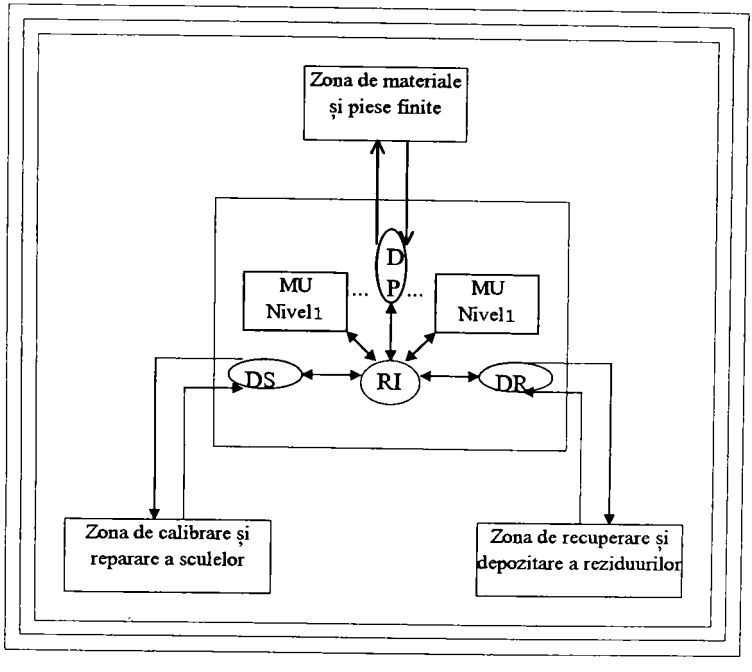


Fig. 3.2_1. Exemplu de atelier flexibil.

Subsistemul tehnologic al sistemului flexibil este constituit din celule care cuprind câte două mașini-unelte și un robot industrial de deservire. Mașinile-unelte, MU, sunt conduse de câte un echipament, ECMU, de tip CNC, iar robotul, RI, -de echipamentul ECRI.

Subsistemul de transport este presupus cu cărucioare filoghidate, CFG. Fiecare cărucior este condus cu câte un automat programabil, AP.

Prin ierarhizare și descentralizare, sarcinile de conducere sunt repartizate la diferite nivele, asigurându-li-se coerența prin legături de coordonare corespunzătoare [DEWE'88].

La nivelul 1, intervin: comanda numerică a mașinilor-unelte (cuprinzând: controlul mișcărilor, controlul operațiilor de prindere/eliberare, etc.), respectiv comanda numerică a cărucioarelor (cuprinzând: controlul mișcărilor, controlul staționărilor, etc.) [BORA'89].

La nivelul 2, se exercită funcțiile de coordonare pentru fiecare celulă flexibilă [HUVE'92], respectiv pentru subsistemul de transport, prin intermediul echipamentelor de coordonare ale celulelor flexibile, ECCF, respectiv prin intermediul echipamentului de coordonare al subsistemului de transport, ECST [HENN'91].

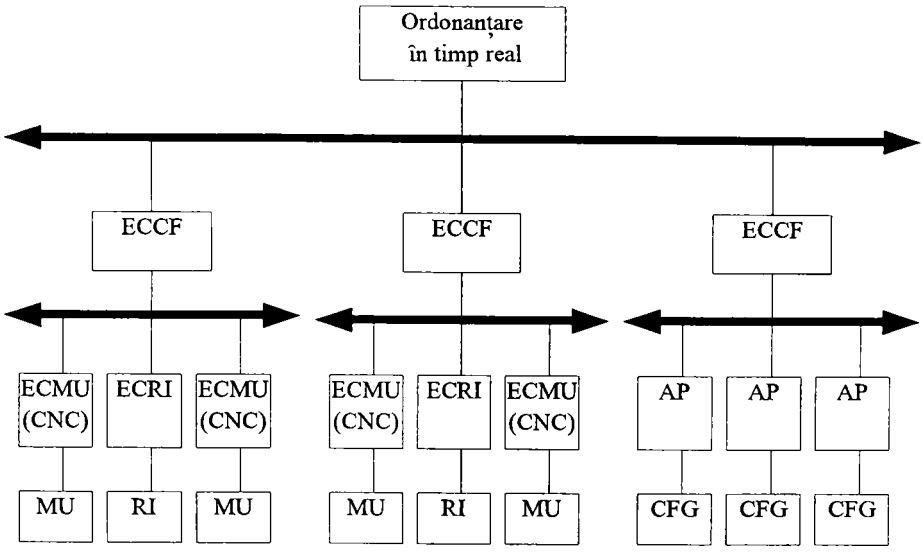


Fig. 3.2_2. Topologie de conducere tipică pentru un atelier flexibil de tipul celui reprezentat în figura 3.2_1.

Echipamentul de coordonare al unei celule dictează echipamentelor de comandă numerică ale mașinilor-unelte programele corespunzătoare prelucrărilor pe care acestea trebuie să le efectueze, stabilește programul pentru echipamentul de comandă al robotului de deservire, și, eventual, pentru automatul programabil al unor dispozitive conexe, și asigură execuția în corelație a tuturor acestor programe.

Echipamentul de coordonare al subsistemului de transport dictează automatelor programabile ale cărucioarelor filoghidate programele de mișcare/staționare, în concordanță cu traseele pe care ele le au de parcurs, și asigură execuția corelată a acestor programe, pentru evitarea coliziunilor.

La nivelul 3, intervin, în principal, funcțiile de ordonanțare, prin care se distribuie sarcinile de prelucrare și de transport către echipamentele nivelului 2 [CARL'88] [WIDM'91].

De asemenea, la nivelul 3, sunt exercitate funcții de mentenanță, având menirea de a superviza funcționarea sistemului, și, în caz de defecțiune, de a emite diagnostice și de a asigura funcționarea degradată, cu limitarea, într-o cât mai mare măsură, a consecințelor [DEWE'88].

Se punctează importanța de prim rang ce revine ordonanțării în cadrul unui sistem de fabricație, ea punându-și amprenta în mod substanțial asupra gradului de utilizare a capacităților de care el dispune, și, în definitiv, asupra eficienței economice.

Exemplul 2

Fie sistemul flexibil -ca și în exemplul 1: un atelier- schematizat în figura 3.2_3 [BORA'89].

Topologia tipică a sistemului de conducere aferent unui asemenea sistem flexibil este cea reprezentată în figura 3.2_4.

Subsistemul tehnologic al sistemului flexibil este constituit, în principal, din patru mașini-unelte, MU, care, împreună cu dispozitivele lor auxiliare, joacă rolul de celule flexibile, fiind, în consecință, deopotrivă, structuri de nivelul 1 și structuri de nivelul 2 [KUSI'85]. Fiecare mașină-unealtă este condusă de câte un echipament CNC.

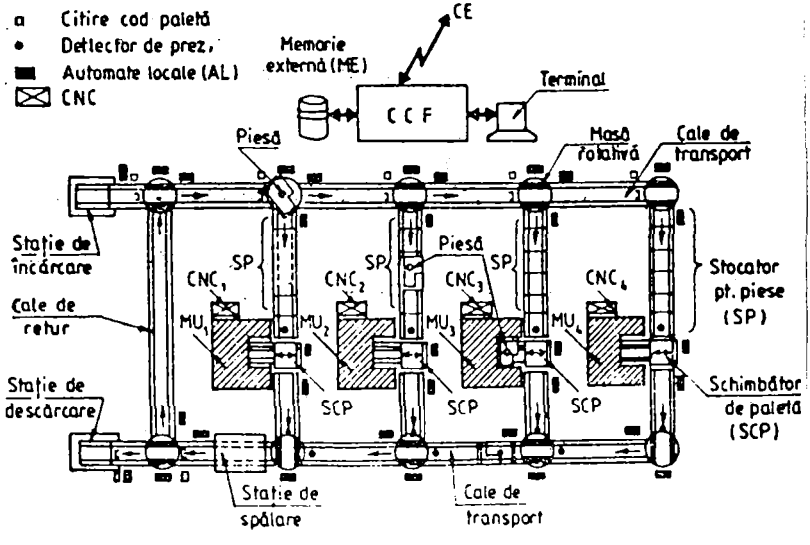


Fig. 3.2_3. Exemplu de atelier flexibil.

Subsistemul de transport este de tip conveior, cu căi delimitate, prevăzut cu mese rotative pentru orientarea paletelor în punctele de ramificație. El este condus cu automate programabile, AP, cu sarcini locale, la nivel de tronson.

În componența sistemului, sunt cuprinse, de asemenea, schimbătoare de palete, SCP, depozit tampon pentru piese, SP, o stație de încărcare a pieselor, SI, o stație de descărcare a pieselor, SD și, în amonte de aceasta, o stație de spălare, toate conduse cu automate programabile.

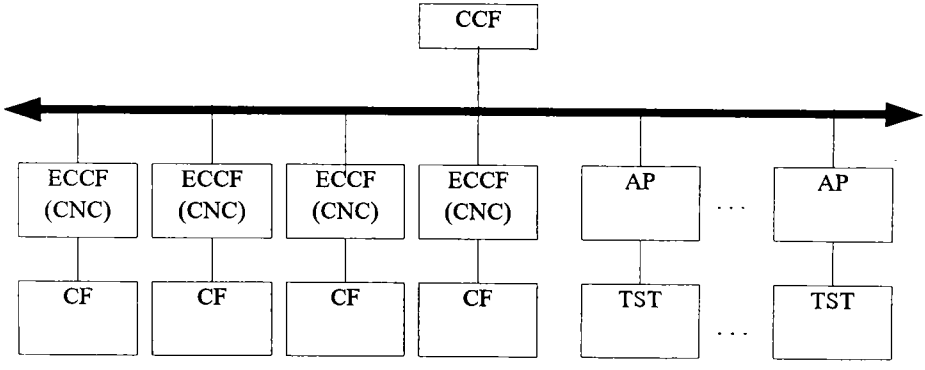


Fig. 3.2_4. Topologie de conducere tipică pentru un atelier flexibil de tipul celui reprezentat în figura 3.2_3.

Figura 3.2_4 evidențiază ierarhizarea pe două nivele a sistemului de conducere.

Nivelul 1 cuprinde echipamentele CNC ale celor patru mașini-unelte, automatele programabile, AP, de comandă a tronsoanelor subsistemului de transport, și cititoare de coduri de palete, dispuse în punctele de ramificație.

Nivelul 2 cuprinde un așa numit "*calculator de conducere a fabricației*" [BORA'89], CCF, care dispune de o memorie externă și de un terminal. El realizează ordonanțarea, distribuind programe adecvate către echipamentele CNC și către automatele programabile, și, prin coordonare, asigură execuția în corelație a respectivelor programe, astfel încât operațiile de prelucrare și fluxul de materiale să se deruleze în mod corespunzător.

Exemplul 3

Fie sistemul flexibil schematizat în figura 3.2_5 [CĂLI'88]. Acest sistem nu diferă, principial, de cel din exemplul 2, dar este mai complex.

Subsistemul tehnologic al sistemului flexibil este constituit, în esența lui, din opt mașini-unelte, MU, care, împreună cu dispozitivele lor auxiliare, joacă rolul de celule flexibile, fiind, în consecință, deopotrivă, structuri de nivelul 1 și structuri de nivelul 2 -ca și în exemplul precedent-, și o stație de verificare a pieselor, SV. Fiecare mașină-unealtă este condusă de câte un echipament CNC, iar stația de verificare, de asemeni; echipamentul CNC corespunzător acesteia este, însă, unul special.

Subsistemul de transport este de tip conveior, având mai multe tronsoane, TST, și dispunând de toate facilitățile tipice de paletizare/depaletizare, ramificare/deramificare, etc. Fluxul de scule este separat de fluxul de piese, fiind asigurat de o zonă dedicată a subsistemului de transport, STS. Conducerea subsistemului de transport este realizată cu automate programabile, AP.

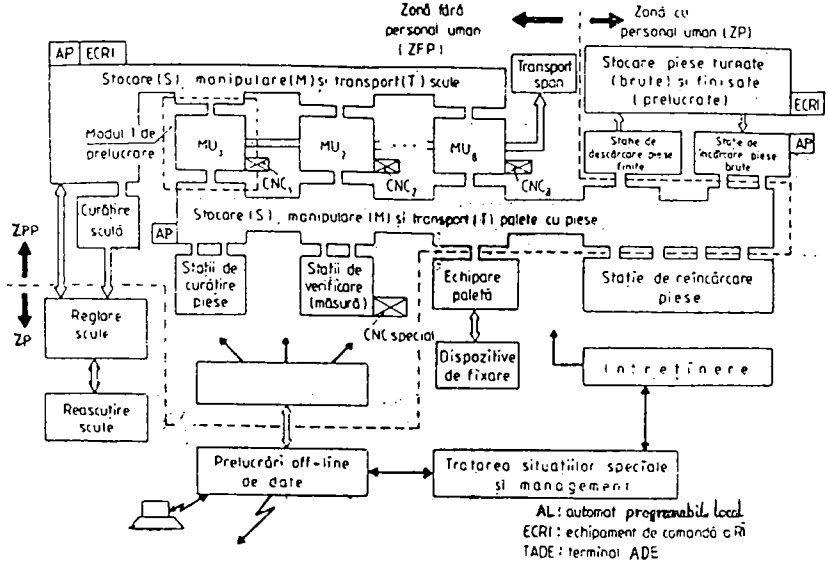


Fig. 3.2_5. Exemplu de atelier flexibil.

În componența sistemului sunt cuprinși, de asemenea, un robot industrial, RI, pentru deservirea zonei de depozitare a pieselor brute și finite, și un robot industrial, RI, pentru deservirea proceselor de stocare și transport al sculelor. Roboții dispun de propriile lor echipamente de conducere, ECRI.

Topologia sistemului de conducere aferent unui asemenea sistem flexibil este reprezentată în figura 3.2_6.

Se remarcă organizarea pe trei nivele a acestui sistem de conducere.

Nivelul 1 cuprinde echipamentele CNC ale celor opt mașini-unelte și al stației de verificare a pieselor, respectiv echipamentele ECRI de comandă a celor doi roboți industriali, și automatele programabile, AP, de comandă a tronsoanelor subsistemului de transport al pieselor, respectiv a zonei subsistemului de transport dedicată fluxului de scule.

Nivelul 2 cuprinde un calculator pentru conducerea în timp real a fabricației, CCTRF, având rolul de a distribui programe către echipamentele CNC, ECRI, și AP, de la nivelul 1, și de a coordona aceste echipamente, astfel încât ansamblul acțiunilor pe care ele le asigură să aibă coerența cerută de procesul de fabricație. De asemenea, calculatorul nivelului 2 realizează supravegherea întregului proces.

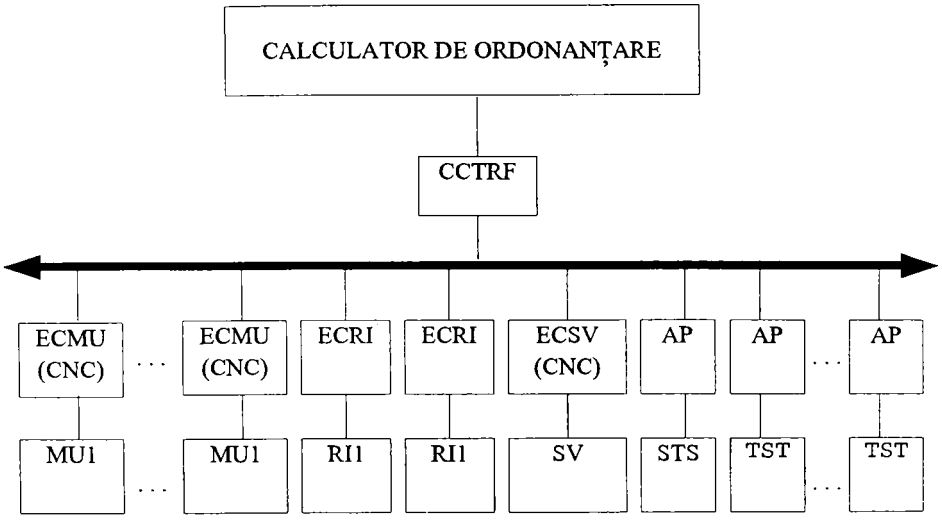


Fig. 3.2_6. Topologie de conducere tipică pentru un atelier flexibil de tipul celui reprezentat în figura 3.2_5.

Nivelul 3 cuprinde un calculator a cărei principală funcție este de a realiza ordonanțarea, comunicând calculatorului nivelului 2 rezultatele acesteia, pentru ca, pe baza lor, el să selecteze și să distribuie către nivelul 1 programele specifice procesului de fabricație în cauză. Evident, s-a presupus că biblioteca de programe este implementată la nivelul 2. Este posibil, însă, ca, în parte sau în totalitate, ea să fie situată la nivelul 3, ceea ce ar presupune ca, în urma ordonanțării, să aibă loc o comunicare de programe de la acest nivel către nivelul 2. Dincolo de cele menționate, calculatorul nivelului 3 efectuează prelucrări ale unor date despre proces, primite de la nivelul 2, și calcule "*off-line*", cu precădere de gestiune a fabricației [DEWE'88].

3.3. Soluții industriale de realizare a rețelelor informaționale din conducerea sistemelor flexibile de fabricație

3.3.1. Soluția *BRING*

Această soluție a fost elaborată de firma ECS. Elementul de economicitate specific unei rețele *BRING* (*BRING: Bus RING*) consistă în utilizarea unui singur cablu coaxial, cu impedanță joasă, capabil să permită viteze de transmisie de 1 MBit/s, cu o codificare bifazică. Protocolul utilizat este *HDLC*, recunoscut ca eficient și pentru pachete de date de mică dimensiune [BORA'89].

Pe magistrala *BRING* pot fi conectate atât module active, cât și module pasive. Dacă în rețea este conectat un singur modul activ, atunci comunicația este de tip "*polling*". Dacă în rețea sunt conectate mai multe module active, atunci comunicația este de tip "*token passing*". Distanța maximă între două module este de 300 m, dar lungimea inelului nu este, practic, limitată.

Sistemul de operare care gestionează transmisia este numit "*BRINGOS*" (*BRINGOS: BRING Operating System*). El poate suporta până la 256 module de diverse tipuri.

Structura în inel impune ca un modul să primească și să regenereze orice pachet de date, indiferent dacă acesta îi este destinat sau nu.

După ce un pachet este recepționat de către modulul căruia îi era destinat, acesta efectuează retransmiterea lui, adăugându-i un cod de control, cu rolul de a certifica recepția corectă. În eventualitatea unei erori de transmisie, sistemul *BRINGOS* încearcă să reia legătura. După un număr limitat de eșecuri, modulul defect este scos din funcțiune, prin scurtcircuitarea liniei.

Există cinci tipuri de module care pot intra în componența unei rețele *BRING*:

- tipul procesor de date (P);
- tipul transmițător/receptor (TR);
- tipul procesor de ax (PA);
- tipul consolă (CS);
- tipul "miscellanea" (M).

Modulele P sunt active. Ele cumulează "inteligența" necesară gestiunii întregului sistem, și programele pentru executarea funcțiilor tipice de automat programabil sau de echipament CNC.

Modulele TR sunt pasive. Ele asigură schimbul de date logice și analogice.

Modulele SA sunt de tip pasiv. Ele sunt destinate realizării controlului axelor de mașini-unelte sau de roboți industriali, fiind capabile să recepționeze informații de la traductoarele de poziție și să prescrie referințe pentru regulatoare externe de viteză.

Modulele CS permit interacțiunea cu operatorul, fiind realizate în diferite moduri, de regulă, clasice [BORA'89].

Modulele M asigură interfața dintre sistemul *BRING* și unitățile externe.

În concluzie, *BRING* poate fi privit ca un sistem informațional distribuit, dedicat rezolvării comunicației în mediul industrial, mai ales între echipamente de tipurile automat programabil și CNC.

3.3.2. Soluția *MODIAC*

O rețea locală din aceeași zonă cu *BRING*, dar mai evoluată decât aceasta, este denumită *MODIAC* [BORA'89]. Modernitatea sa constă în compatibilitatea parțială a comunicației cu nivelele *ISO-OSI* [ROBU'95a].

În esență, sistemul *MODIAC* permite conectarea la o magistrală de mare viteză a unor noduri de tip monoprocesor sau multiprocesor, fiecare nod având obiective specifice (achiziție de date analogice și/sau numerice, control, etc.). Nodurile multiprocesor au o arhitectură bazată pe o magistrală paralelă, denumită "*MODOSK*", dispunând de un sistem de operare rezident, cu facilități pentru multitasking [BORA'89]. În figura 3.3.2_1, este prezentată schematic corespondența între comunicația *MODIAC* și modelul *ISO-OSI*.

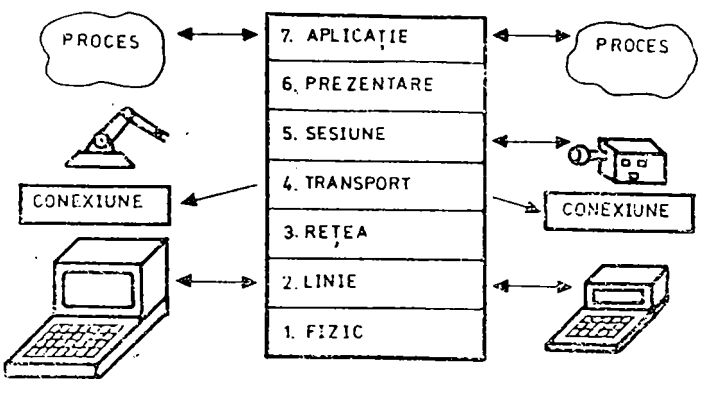


Fig. 3.3.2_1. Corespondența între comunicația *MODIAC* și modelul *ISO-OSI*.

Până în prezent, nu sunt prevăzute servicii pentru nivelele REȚEA, respectiv PREZENTARE. Serviciile REȚEA nu sunt necesare, deoarece topologia rețelei *MODIAC* este de tip "magistrală comună" [BORA'89], iar lipsa serviciilor PREZENTARE se explică prin utilizarea nodurilor omogene.

Comunicarea la primele două nivele poate fi făcută, ca și la *BRING*, în două variante: "*token passing*", respectiv "*CSMA*" (*CSMA: Carrier Sense Multiple Acces*). Nivelul TRANSPORT asigură fragmentarea mesajelor care trebuie transmise -unități de date de serviciu transport, TSDU-, în unități de date de transport protocol, TPDU, care pot fi gestionate direct la nivelul LEGĂTURĂ. În esență, la nivelul TRANSPORT, rețeaua MODIAC poate transmite trei tipuri de mesaje:

a). transfer de date normale sau de mesaje lungi cu randament sporit;
b). transfer de date expres, adică de mesaje scurte și, în același timp, urgente;
c). transfer de datagrame.

Nivelul SESIUNE asigură stabilirea unui canal virtual de legătură, de exemplu, în vederea unui dialog în care este implicat un sistem de recunoaștere de forme. Nivelul APLICAȚIE conține procedurile concrete de automatizare, ce fac obiectul a ceea ce se cheamă "*comandă numerică*".

Sistemul de operare al rețelei *MODIAC* este denumit *VNOS* (*VNOS: Virtual Network Operating System*). El este constituit prin reunirea sistemelor operative neomogene situate la nivel de nod. Cooperarea dintre diferite sisteme este asigurată printr-un sistem de extensii care permit interfațarea proceselor aplicative nu direct cu sistemul operativ, ci prin intermediul primitivelor din *VNOS*. Acesta este în măsură să deosebească cererea unui serviciu local pentru care sunt suficiente resursele aparținând nodului, de cererile care necesită transferul la distanță al datelor. În acest din urmă caz, este inițiat transferul între nodurile interesate.

Se menționează că atât *BRING*, cât și *MODIAC*, pot fi utilizate și în sisteme de comunicație ierarhizate, ele fiind capabile de a furniza date nivelului superior.

3.3.3. Soluția *TELWAY*

Una dintre cele mai reușite soluții pentru realizarea comunicației în sistemele de fabricație integrată prin calculator aparține firmei TELEMECANIQUE și a fost lansată sub sigla "*TELWAY*"[BORA'89].

Rețeaua *TELWAY* a fost, într-o primă instanță, utilizată pentru conectarea automatelor programabile din seria TSX în structuri ierarhizate pe trei nivele.

Principiul de comunicare al rețelei *TELWAY* este acela al "*căsuței poștale*", în care postul emitent depune un mesaj ce urmează a fi preluat de către postul destinaar. Din punct de vedere al interconectării lor în rețeaua *TELWAY*, automatele programabile TSX au arhitectura internă reprezentată în figura 3.3.3_1, în care se instanțiază un segment de rețea cuprinzând două automate.

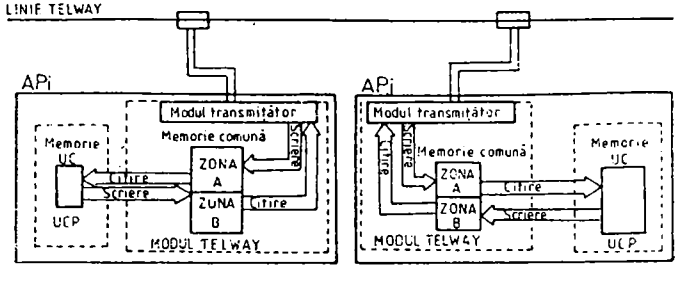


Fig. 3.3.3_1. Segment de rețea *TELWAY*, cuprinzând două automate programabile TSX.

Procedura de comunicație constă din următorii pași:

- un procesor "*master*" invită la emisie o stație, printr-un proces de tip "*polling*";
- stația interesată răspunde, emițându-și mesajul prin propriul modul *TELWAY*;
- toate stațiile citesc mesajul vehiculat în rețea, folosindu-se de memoria comună accesibilă prin modulele *TELWAY* proprii;
- sunt posibile, dependent de tipul mesajului, trei situații:

  a). dacă mesajul este de tip general, atunci toate stațiile îl preiau în memoria proprie;
  b). dacă mesajul este de tip parțial, atunci stațiile destinatare îl preiau în memoria proprie, iar celelalte îl ignoră.
  c). dacă mesajul este de tip particular, atunci doar una dintre stații are calitatea de destinatar și ea va fi singura care îl preia în memoria proprie.

Pentru a se asigura acest protocol, memoria comună rezidentă pe modulul *TELWAY* este constituită din două *zone*: *zona A*, în care se înscriu mesaje provenind de la rețea -mesaje care urmează să fie citite de către unitatea centrală în memoria proprie-, respectiv *zona B*, în care unitatea centrală înscrie mesajele produse de ea -mesaje ce urmează să fie vehiculate de modulul transmițător-.

Principalele avantaje ale rețelei *TELWAY* sunt:

- generare simplă a mesajelor, acestea prezentându-se drept blocuri de format fix;
- dialog transparent pentru utilizator, care nu trebuie să se procupe de procedurile de schimb;
- independență a duratei transmisiei mesajelor față de operațiile executate în slujba aplicației de către automatele programabile;
- capacitate de erijare a oricărui post ca "master";
- capacitate de a izola ușor un post avariat;
- capacitate a rețelei de a funcționa sub forma a două subrețele distincte, în cazul întreruperii liniei într-un punct oarecare.

Rețelele *TELWAY* pot fi omogene, adică formate din posturi de același tip (automate programabile, echipamente CNC, etc.), în număr de maximum 64, sau heterogene, adică formate dintr-o diversitate de tipuri de posturi (automate programabile, echipamente CNC, echipamente de conducere a roboților industriali, calculatoare de uz general, etc.), de asemenea în număr de maximum 64.

În cazul rețelelor omogene se folosește un singur canal de transmisie, iar în cazul rețelelor heterogene -două canale.

Protocolul de transmisie este de tipul *CSMA-CD* (*CSMA-CD: Carrier Sense Multiple Access-Collision Detection*), remarcabil prin cele surprinse în denumirea sa: asigurarea transmisiei cu acces multiplu și cu detectarea coliziunilor.

3.3.4. Soluția ETHERNET

Specificația *CSMA-CD*, menționată mai sus, a fost elaborată prin efortul comun al firmelor Digital Equipment Corporation, Intel, și Xerox, ea stând la baza standardului *ETHERNET*. În figura 3.3.4_1, este reprezentată arhitectura *ETHERNET*, evidențiindu-se faptul că ea corespunde primelor două nivele ale modelului *ISO-OSI*: nivelul FIZIC, respectiv nivelul LINIE [ROBU'95a].

Nivelul LINIE asigură două funcții principale: încadrarea mesajelor -care constă în stabilirea adresei, a secvenței de control de eroare și a delimitărilor-, respectiv gestiunea propriu-zisă a liniei. Formatul mesajelor este reprezentat în figura 3.3.4_2.

Nivelel LINIE gestionează semnalul purtătoare, furnizat de nivelul fizic, pentru a indica ocuparea/disponibilitatea canalului. Controlerul LINIE introduce o pauză între mesaje de 9.6 ms.

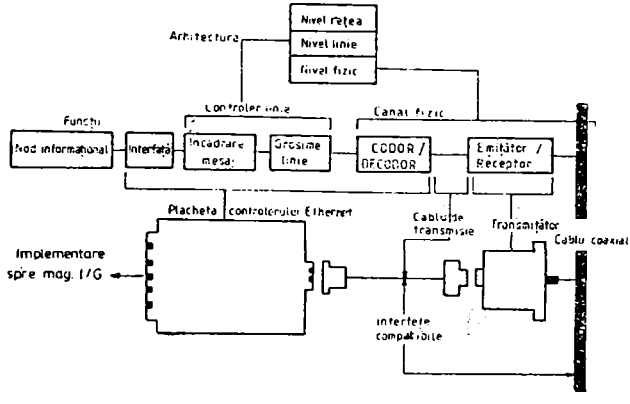


Fig. 3.3.4_1. Arhitectura ETHERNET.

Dacă două posturi doresc să intre simultan în emisie, controlerul detectează coliziunea, pe baza unui semnal generat de nivelul FIZIC. După detectarea unei coliziuni, emisia continuă cel puțin 32 biți, dar nu mai mult de 48, pentru ca detecția să fie posibilă la toate stațiile. O emisie întreruptă de o coliziune se reia de maximum 16 ori. Reluarea se face prin introducerea unor întârzieri aleatoare, acest lucru asigurând șanse egale de acces pentru toate stațiile. La recepția mesajului, delimitarea este dată de apariția purtătoarei.

Nivelul FIZIC conține cablul coaxial, transmițătorul, codorul/decodorul, și repetoarele. Cablul trebuie închis la capete cu adaptoare de impedanță. Transmițătorul permite cuplarea, prin separare galvanică, a controlerului de linie la cablul coaxial, el având, de asemenea, rolul de a indica eventualele coliziuni. Codorul execută o serializare sincronă a datelor, iar decodorul -o deserializare sincronă. Repetoarele permit extinderea rețelei, refăcând, în punctele de amplasare a lor, amplitudinea semnalelor. Este de menționat că, între două repetoare, nu poate exista decât un

singur traseu. În figura 3.3.4_3, se prezintă un exemplu de configurație *ETHERNET*, de tip extins.

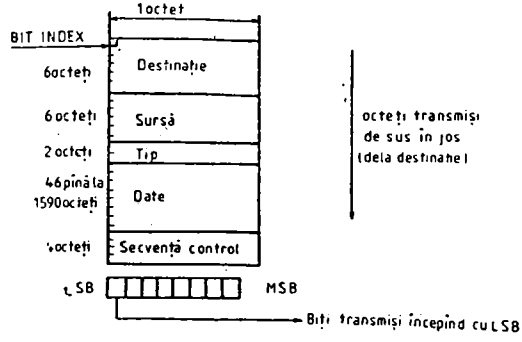


Fig. 3.3.4_2. Formatul mesajelor *ETHERNET*.

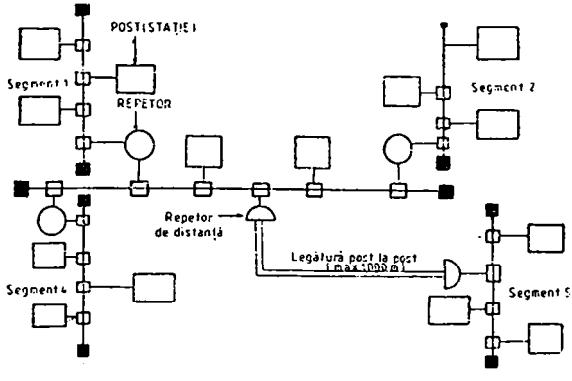


Fig. 3.3.4_3. Exemplu de configurație *ETHERNET*, extinsă.

Standardul *ETHERNET* și-a găsit numeroase aplicații în domeniul informaticii industriale, în general, și, în particular, în fabricația integrată prin calculator. Alături de avantajele sale, lesne de dedus din cele de mai sus, acest standard are, însă, și două dezavantaje majore, care ridică probleme deosebite în cazul aplicațiilor din domeniul fabricației integrată prin calculator.

Primul dezavantaj al standardului *ETHERNET* constă în faptul că nu asigură nici o posibilitate de discriminare, prin priorități distincte, între stațiile din rețea, chiar dacă -lesne de înțeles- unele dintre acestea sunt sau, cel puțin uneori, ajung în situația de a li se pretinde ca rapid să se informeze, să ia decizii și să comunice deciziile luate.

Al doilea dezavantaj al standardului *ETHERNET* consistă în aceea că nu permite alocarea unei benzi de frecvență anume pentru un anumit post, în scopul de a-i conferi facilități aparte, întrucât transmisia se face prin modulație în banda de frecvență de bază.

3.4. Tendințe privind rețelele informaționale din conducerea sistemelor flexibile de fabricație

Importanța rețelelor informaționale pentru conducerea sistemelor flexibile de fabricație -în mod deosebit, a celor ridicate la rangul de sisteme de fabricație integrată prin calculator-, a determinat marile concerne să se preocupe de promovarea unor specificații în domeniu demne de a fi acceptate drept standarde, ca deziderat, cel puțin pentru o perioadă de un deceniu. În principal, aceste specificații se referă la problematica comunicației, aceasta fiind, după cum se știe, elementul determinant pentru o rețea informațională.

Concernul Boeing a propus standardul ***TOP***, gândit să satisfacă, în același timp, cerințele aplicațiilor din sectorul tehnic și ale celor din sectoarele de birou (***TOP**: Technical and Office Protocol*) [CROW'86].

Concernul General Motors a propus standardul ***MAP*** (***MAP**: Manufacturing Automation Protocol*), respectând setul de specificații ale standardului ***IEEE 802***, referitor la rețelele informatice locale [CROW'86] [BABB'87]:

-IEEE 802-1: referitor la glosarul de termeni pentru rețele locale;
-IEEE 802-2: referitor la controlul liniei în rețelele locale;
-IEEE 802-3: referitor la metoda de acces *CSMA-CD*;
-IEEE 802-4: referitor la metoda de acces "*token passing*";
-IEEE 802-5: referitor la metoda de acces "*token ring*".

Ultimele variante ale specificațiilor ***MAP*** acoperă integral cele șapte nivele *ISO-OSI*, înglobând, în variantă redusă, protocoalele ***PROWAY***, deja mult răspândite în aplicațiile industriale [WALD'90]. Se scontează că, cel puțin în următorii ani, standardul ***MAP*** va rezista ca bază a dezvoltării rețelelor informaționale din sistemele de fabricație integrată prin calculator [BARB'92].

Analizând critic avantajele și dezavantajele specificațiilor *ETHERNET* și ***MAP***, se poate afirma că standardul *ETHERNET* este mai avantajos când se transmit blocuri lungi de date la intervale de timp necritice (de exemplu: în aplicațiile de gestiune de date de birou sau de laborator), în timp ce standardul ***MAP*** este adecvat când comunicațiile trebuie să fie în timp real, cu blocuri de date de lungime mică (cum este cazul în cele mai multe segmente ale aplicațiilor din domeniul fabricației integrate prin calculator).

Rețeaua informațională ***MAP*** este concepută ca un sistem ierarhic pe patru nivele, așa cum se arată în figura 3.4_1. Cele patru nivele sunt denumite:

- nivelul aparaturii;
- nivelul fabricației;

- nivelul conducerii uzinei;
- nivelul conducerii interuzinale.

La fiecare nivel, comunicația se asigură printr-o serie de echipamente și protocoale specifice.

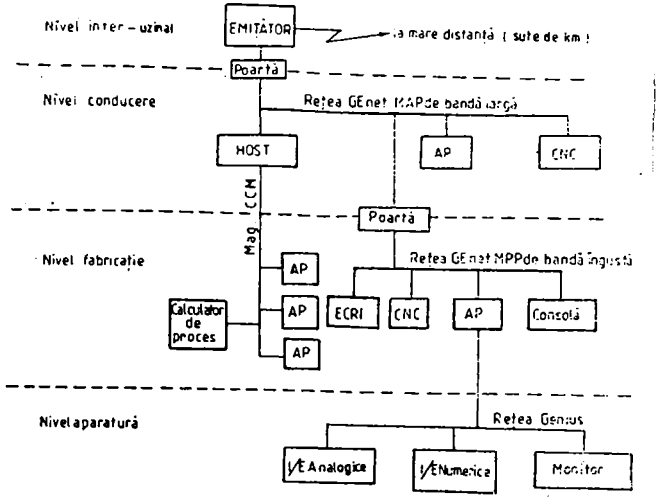


Fig. 3.4_1. Nivelele ierarhice ale rețelei ***MAP***.

La nivelul aparaturii, se folosește rețeaua *Genius*, care are la bază o familie de module inteligente de intrare/ieșire. Conectarea acestora în rețea este asigurată de către blocurile numite BCU, existente în structura fiecăruia. Suportul fizic de interconectare este linia bifilară torsadată ecranată, cu lungimea de maximum 3000 m. Numărul maxim al modulelor ce pot fi interconectate este de 32. Transmisia se face cu viteza de 150 KBauds.

La nivelul fabricației, se utilizează rețeaua *GEnet de bandă îngustă*. Pe această rețea, se vehiculează informații necesare conducerii proceselor, realizată cu echipamente CNC, echipamente de conducere a roboților industriali, automate programabile, etc. Conectarea diverselor echipamente în rețeaua *GEnet de bandă îngustă* este asigurată de către blocurile numite CCM, existente în structura fiecăruia dintre ele. Protocolul implementat de aceste blocuri (ANSI X 3.28) permite comunicații atât între echipamente de aceeași prioritate ("*peer-to-peer*"), cât și între echipamente ierarhizate ("*master-slave*"). Un bloc CCM posedă o interfață serială *RS-232*, pentru transmisii la distanțe de până la 15 m, și o interfață serială *RS-422*, pentru transmisii la distanțe de până la 1500 m, cu o viteză de 38.4 KBauds.

La nivelele conducerii uzinale și interuzinale, se folosește rețeaua *GEnet de bandă largă*. Aceasta este de tip "*token-bus*" și realizează transmisiile cu viteza de 10 MBauds. Interconectarea diverselor echipamente în rețeaua *GEnet de bandă largă* se realizează cu ajutorul blocurilor numite BIU, înglobate în fiecare dintre ele. Ca și blocurile CCM, blocurile BIU conțin două interfețe seriale: una *RS-232* și una *RS-422*. Protocolul de transmisie utilizat este HDLC.

Rețeaua *GEnet de bandă largă* dispune, de asemenea, de un modul numit TM, care asigură lucrul în frecvență înaltă, la 192.5 MHz.

Având în vedere caracteristicile standardului *MAP*, sub toate aspectele, acesta este, actualmente, unanim acceptat ca bază a concepției și implementării rețelelor informaționale din sistemele de fabricație integrată prin calculator.

Se apreciază că orientările spre standardizare, manifestate cu precădere în ultimul deceniu, vor deveni și mai pronunțate, nu numai în privința rețelelor din sistemele de fabricație integrată prin calculator, ci, în general, în domeniul informaticii distribuite, fiind de așteptat să se ajungă, astfel, la o adevărată tehnologie pentru implementarea fluxurilor informaționale [BARB'92].

4
MODELAREA SISTEMELOR ȘI SUBSISTEMELOR DE FABRICAȚIE FLEXIBILĂ

4.1. Rețelele Petri -principalul instrument de modelare în fabricația flexibilă

O *rețea Petri* este un model grafic din clasa grafurilor orientate, având două categorii de noduri, denumite, respectiv, "*poziții*" și "*tranziții*". Pozițiile modelează condiții și se reprezintă prin cercuri. Tranzițiile modelează evenimente și se reprezintă prin dreptunghiuri [BRAM'89].

Relațiile *poziții-tranziții*, respectiv *tranziții-poziții* se reprezintă prin arce care unesc simbolurile nodurilor în cauză. Evident, un eveniment corespunzător unei tranziții este efect al condițiilor pe care le modelează poziția/pozițiile din amonte, dar, în același timp, el modifică condițiile pentru producerea altor evenimente; din acest motiv, există arce atât de la poziții la tranziții, cât și de la tranziții la poziții [CĂLI'88].

În cercurile care reprezintă pozițiile, se înscriu *marcaje* corespunzătoare condițiilor. Aceste marcaje pot fi omogene, materializate, de exemplu, prin puncte -se vorbește, atunci, despre *rețele Petri ordinare*-, sau heterogene -caz în care se folosește termenul de *rețea Petri colorată*. Pentru fiecare poziție, este stabilit un marcaj maximal, ce reprezintă *capacitatea poziției* [BRAM'89].

Alăturat arcelor care leagă pozițiile de tranziții, se fac înscrisuri care indică -numai cantitativ, în cazul rețelelor Petri ordinare, respectiv cantitativ și calitativ, în cazul rețelelor Petri colorate-, marcajul necesar în amonte pentru ca tranziția să aibă loc. La fel se procedează referitor la arcele care leagă tranzițiile de poziții, de data aceasta pentru a indica marcajul injectat, prin execuția tranziției, în poziția/pozițiile din aval. Există convenția ca arcele din rețelele omogene alăturat cărora ar trebui a se înscrie cifra 1 să rămână neinscripționate [DAVI'89].

În figurile 4.1_1 și 4.1_2, se prezintă, pentru ilustrarea celor de mai sus, două exemple de rețele Petri, dintre care prima este ordinară, iar a doua -colorată.

Pentru ca o tranziție să se execute, este necesar și suficient, pe de o parte, ca toate arcele care intră în ea să fie satisfăcute -conform înscrisurilor pe care le poartă- de către marcajele pozițiilor în care își au originea, iar pe de altă parte, ca toate arcele care ies din ea să poată fi descărcate în pozițiile în care își au vârful, în limita capacităților acestora [BRAM'89].

Relațiile dintre condiții și evenimente, respectiv legăturile prin arce între poziții și tranziții, pot fi surprinse matematic prin intermediul a două matrici: matricea de incidență înainte, numită

matricea "*Pre*", respectiv matricea de incidență înapoi, numită matricea "*Post*". În aceste matrici, liniile corespund pozițiilor, iar coloanele -tranzițiilor. Pentru definirea acestor matrici, se consideră că inscripția asociată unui arc ce leagă o poziție "*i*" de o tranziție "*j*", indiferent în ce sens, este dată de o funcție l_{ij} [BRAM'89].

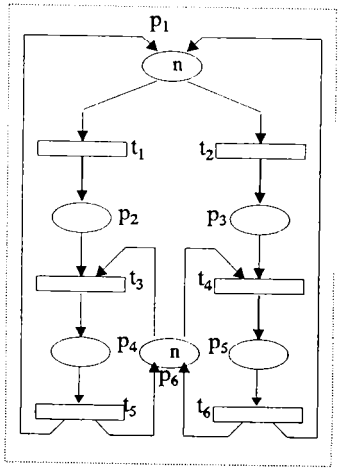


Fig. 4.1_1. Exemplu de rețea Petri ordinară.

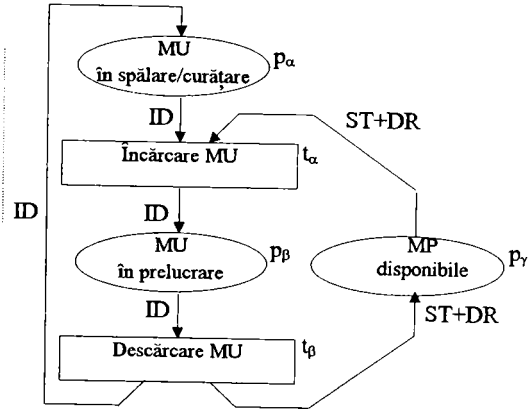


Fig. 4.1_2. Exemplu de rețea Petri colorată.

Un element al matricii *Pre* se definește astfel:

$$Pre[i][j] = \begin{cases} 0, & \text{daca de la pozitia } i \text{ nu pleaca arc la tranzitia } j \\ l_{ij}, & \text{daca de la pozitia } i \text{ pleaca arc la tranzitia } j \end{cases} \quad (4.1\text{-}1)$$

Un element al matricii *Post* se definește astfel:

$$Post[i][j] = \begin{cases} 0, & \text{daca la pozitia } i \text{ nu soseste arc de la tranzitia } j \\ l_{ij}, & \text{daca la pozitia } i \text{ soseste arc de la tranzitia } j \end{cases} \quad (4.1\text{-}2)$$

Dacă se introduc notațiile:

- P -pentru mulțimea pozițiilor;
- T -pentru mulțimea tranzițiilor;
- K -pentru vectorul capacităților pozițiilor în privința marcajului;
- M_0 -pentru vectorul marcajelor inițiale ale pozițiilor;

atunci rețeaua Petri se poate specifica prin sextuplul:

$$RP = (P, T, Pre, Post, K, M_0) \quad (4.1\text{-}3)$$

În locul matricilor *Pre* și *Post*, se poate folosi cuplul:

(F, T)

unde:

- F -este mulțimea arcelor (deci: $F \subseteq (P \times T) \cup (T \times P)$);
- L -este mulțimea marcajelor asociate arcelor,

obținându-se [CĂLI'88]:

$$RP = (P, T, F, L, K, M_0) \tag{4.1-4}$$

Se precizează că matricea diferență dintre matricea *Post* și matricea *Pre* se numește *matricea de incidență* a rețelei și se notează cu C.

$$C = Post - Pre \tag{4.1-5}$$

Până aici, nu s-a menționat nimic referitor la durata evenimentelor sau a intervalelor dintre evenimente. Implicit, s-a considerat că evenimentele au o durată infinit mică -deci, că se desfășoară instantaneu-, respectiv că duratele dintre ele sunt lipsite de relevanță [DAVI'89].

Totuși, în anumite situații, se impune ca în procesul de modelare să fie surprinse și aspecte temporale. Există două posibilități echivalente pentru a o face.

Prima posibilitate consistă în temporizarea pozițiilor. O poziție temporizată se caracterizează prin aceea că marcajul injectat în ea ca urmare a execuției unei tranziții din amonte devine efectiv abia după un timp, prestabilit. Mărimea acestuia va indica decalajul impus între evenimentul aferent tranziției din amonte și evenimentele aferente tranzițiilor din aval [CĂLI'88].

A doua posibilitate consistă în temporizarea tranzițiilor. O tranziție temporizată se caracterizează prin aceea că efectele execuției sale se fac simțite abia după un timp, prestabilit. Mărimea acestuia va indica decalajul impus între evenimentul aferent tranziției în cauză și evenimentele aferente tranzițiilor condiționate de pozițiile situate în aval [CĂLI'88].

Este lesne de înțeles că o rețea cu poziții temporizate poate fi transformată într-o rețea echivalentă, cu tranziții temporizate, și reciproc, întrucât o poziție temporizată poate fi înlocuită printr-o succesiune *poziție netemporizată-tranziție temporizată-poziție netemporizată*, după cum o tranziție temporizată se poate înlocui printr-o succesiune *tranziție netemporizată-poziție temporizată-tranziție netemporizată*.

O rețea Petri cu poziții temporizate, respectiv cu tranziții temporizate se numește *rețea Petri temporală* sau *rețea Petri temporizată*. În figura 4.1_3, se dă un exemplu de astfel de rețea, având temporizate tranzițiile.

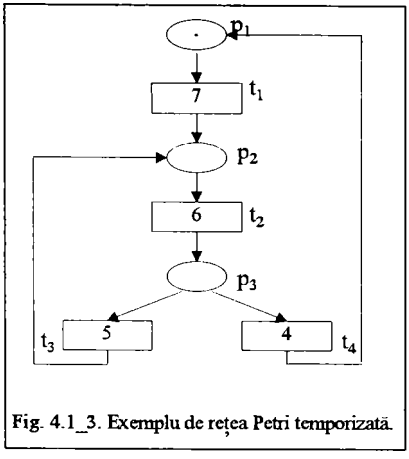


Fig. 4.1_3. Exemplu de rețea Petri temporizată.

Rețelele Petri, în diversele lor variante, reprezintă principalul instrument de modelare în domeniul sistemelor flexibile de fabricație [BEL'85], întrucât "trăirea" acestora se manifestă printr-o sumă de evenimente asincrone concurente, determinate de un ansamblu de condiționări. Asemenea evenimente pot fi, spre exemplu: prelucrările simultane ale unor piese pe diversele mașini-unelte, încărcarea/descărcarea pieselor pe/de pe mașinile-unelte cu ajutorul roboților de deservire, execuția unor operații de paletizare/ depaletizare, etc.

Se menționează că alte instrumente utilizabile -dar mai puțin utilizate, dintr-un inconvenient sau altul- pentru modelare în domeniul sistemelor flexibile de fabricație sunt: *sistemele de adunare a vectorilor*, *graful UCLA*, *graful GRAFCET* [AMAR'90] [AMAR'92] [BLAN'86] [BUZA'85] [DALL'86] [HILL'89].

Odată construită rețeaua de modelare a unui sistem sau subsistem, este necesar să se procedeze la analiza ei -ceea ce presupune efectuarea unor calcule specifice-, pentru a se evidenția proprietățile ce o caracterizează. În concret, plecând, de regulă, de la sextuplul:

$$RP = (P, T, Pre, Post, K, M_0)$$

se determină secvențele posibile de tranziții și evoluția succesivă a marcajelor, până la epuizarea tuturor stărilor posibile, verificându-se dacă apar blocaje (adică, dacă devine imposibilă continuarea evoluției, fără să fi fost epuizate toate stările posibile), dacă este obținut din nou marcajul inițial după trecerea prin toate stările -ceea ce ar corespunde unei funcționări repetitive-, respectiv dacă se ajunge într-o stare finală, corespunzătoare încheierii prevăzute a activităților stabilite pentru sistemul modelat.

Evoluția marcajelor pozițiilor unei rețele Petri se obține, pas cu pas, pe baza relației:

$$M_{t_i}^{+} = M_{t_i}^{-} + col_{t_i}(Post) - col_{t_i}(\Pr e) \qquad (4.1\text{-}6)$$

echivalentă cu:

$$M_{t_i}^{+} = M_{t_i}^{-} + col_{t_i}(C) \qquad (4.1\text{-}7)$$

unde:

- $M_{t_i}^{-}$ -este marcajul rețelei înainte de execuția tranziției t_i ;
- $M_{t_i}^{+}$ -este marcajul rețelei după eecuția tranziției t_i ;
- $col_{t_i}(\cdot)$-este coloana matricii "·" corespunzătoare tranziției t_i .

Evident:

$M_{t_i}^- = M_0$ -pentru o tranziție t_i ce decurge direct din M_0; (4.1-8)

$M_{t_i}^- = M_{pred(t_i)}^+$ pentru o tranziție t_i ce urmează tranziției $pred(t_i)$. (4.1-9)

Determinarea marcajelor accesibile și, implicit, a secvențelor de tranziții posibile, cu ajutorul relațiilor (4.1-6)...(4.1-9) reprezintă, chiar și în cazul unor rețele foarte simple, un travaliu însemnat. De aceea, au fost căutate soluții alternative. Una dintre acestea se caracterizează prin aceea că permite determinarea proprietăților unei rețele Petri pe baza unor mărimi care surprind aceste proprietăți, mărimi numite *invarianți*.

Se desemnează drept invariant un vector linie, I^T, care are proprietatea că înmulțirea sa cu vectorul oricărui marcaj accesibil conduce la același rezultat.

Înseamnă, deci, că:

$$I^T \cdot M_{t_i}^+ = I^T \cdot M_{t_i}^- \quad (4.1\text{-}10)$$

adică:

$$I^T (M_{t_i}^- + col_{t_i}(C)) = I^T \cdot M_{t_i}^- \quad (4.1\text{-}11)$$

$$I^T \cdot M_{t_i}^- + I^T \cdot col_{t_i}(C) = I^T \cdot M_{t_i}^- \quad (4.1\text{-}12)$$

ceea ce implică:

$$I^T \cdot col_{t_i}(C) = 0, \ \forall i = 1...n \quad (4.1\text{-}13)$$

sau, totuna:

$$I^T \cdot C = O \quad (4.1\text{-}14)$$

Relația (4.1-14) este cea care permite calculul invarianților.

În cele ce urmează, se vor folosi atât relațiile (4.1-6)...(4.1-9), cât și calculul de invarianți, în modelarea și analiza unor subsisteme tipice de fabricație flexibilă:

- subsistemul *o mașină-unealtă~un manipulator;*
- subsistemul *două mașini-unelte~un manipulator;*
- subsistemul *"n" mașini-unelte~un manipulator;*
- subsistemul cooperativ *două mașini-unelte~două manipulatoare;*
- subsistemul cooperativ *"n" mașini-unelte~"n" manipulatoare;*
- liniile de fabricație deschise ;
- liniile de fabricație închise.

4.2. Modelarea unui subsistem *o mașină-unealtă-un manipulator*

Se consideră că mașina-unealtă este deservită de către manipulator prin încărcare cu semifabricate, preluate dintr-un depozit de semifabricate, DSf, respectiv prin descărcare de piese finite, depuse într-un depozit de piese finite, DPf. Topologia subsistemului este prezentată în figura 4.2_1.

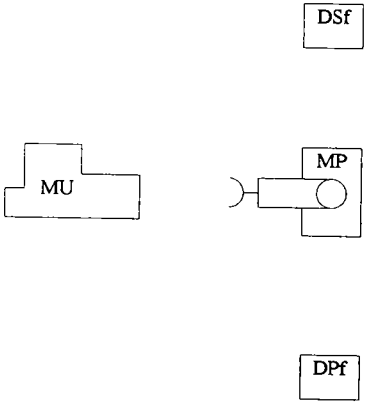


Fig. 4.2_1. Topologia subsistemului *o mașină-unealtă-un manipulator.*

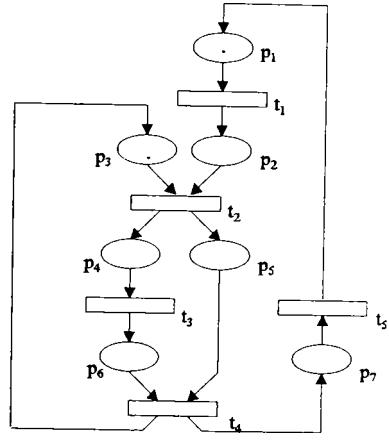


Fig. 4.2_2. Rețeaua Petri corespunzătoare subsistemului din fig. 4.2_1.

Un asemenea subsistem funcționează de o manieră ciclică, după cum se arată în continuare.

La început de ciclu, mașina-unealtă este liberă și manipulatorul la fel. Apoi, manipulatorul ia un semifabricat și îl încarcă în mașina-unealtă, condițiile necesitate de aceste operații -manipulator liber, respectiv manipulator cu semifabricat și mașină-unealtă liberă- fiind îndeplinite. Astfel, mașina-unealtă devine ocupată cu prelucrarea semifabricatului, iar manipulatorul ajunge din nou liber. Când prelucrarea se încheie, mașina-unealtă se cere descărcată. Descărcarea este realizată de către manipulator. Acesta preia piesa obținută prin prelucrare, eliberând astfel mașina-unealtă, cu prețul autoîncărcării sale. După aceea, el depune piesa în locul predestinat și redevine, astfel, liber, pregătit pentru un nou ciclu.

Rețeaua Petri, de tip "*condiții-evenimente*", care modelează un asemenea subsistem mașină-unealtă-manipulator -a se vedea figura 4.2_2- are șapte poziții -toate de capacitate unitară- și cinci tranziții:

- poziția p_1 este dedicată condiției *"manipulator liber"*;
- poziția p_2 este dedicată condiției "*manipulator încărcat cu semifabricat*";

- poziția p_3 este dedicată condiției "*mașină-unealtă liberă*";
- poziția p_4 este dedicată condiției "*mașină-unealtă încărcată*";
- poziția p_5 este dedicată condiției "*manipulator descărcat de semifabricat*";
- poziția p_6 este dedicată condiției "*mașină-unealtă gata de descărcare*";
- poziția p_7 este dedicată condiției "*manipulator încărcat cu piesă gata de descărcare*";
- tranziția t_1 corespunde evenimentului "*încărcare manipulator cu un semifabricat*";
- tranziția t_2 corespunde evenimentului "*încărcare mașină-unealtă*"
- tranziția t_3 corespunde evenimentului "*prelucrare*";
- tranziția t_4 corespunde evenimentului "*descărcare mașină-unealtă*";
- tranziția t_5 corespunde evenimentului "*descărcare manipulator de piesa obținută*".

Matricile *Pre, Post*, și *C* ale rețelei din figura 4.2_1 sunt:

$$\mathbf{Pre} = \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \left\| \begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} \end{array} \tag{4.2-1}$$

$$Post = \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \left\| \begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} \end{array} \tag{4.2-2}$$

$$C = \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \left\| \begin{array}{ccccc} -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} \end{array} \tag{4.2-3}$$

Ținând seamă că marcajul inițial este:

$$M_0 = \begin{Vmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{matrix} \tag{4.2-4}$$

pe baza relațiilor (4.1-7)...(4.1-9) va rezulta, succesiv:

$$1).\ M_0 \Rightarrow execution\ t_1; \tag{4.2-5}$$

Deci:

$$M_{t_1}^- = M_0 \tag{4.2-6}$$

$$\Rightarrow M_{t_1}^+ = M_{t_1}^- + col_{t_1}(C) = \begin{Vmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \tag{4.2-7}$$

$$2).\ M_{t_1}^+ \Rightarrow execution\ t_2; \tag{4.2-8}$$

Deci:

$$M_{t_2}^- = M_{t_1}^+ \tag{4.2-9}$$

$$\Rightarrow M_{t_2}^+ = M_{t_2}^- + col_{t_2}(C) = \begin{Vmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ -1 \\ -1 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \tag{4.2-10}$$

$$3).\ M_{t_2}^+ \Rightarrow execution\ t_3; \tag{4.2-11}$$

Deci:

$$M_{t_3}^- = M_{t_2}^+ \tag{4.2-12}$$

$$\Rightarrow M_{t_3}^+ = M_{t_3}^- + col_{t_3}(C) = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix} \tag{4.2-13}$$

4). $M_{t_3}^+ \Rightarrow$ *executie* t_4; (4.2-14)

Deci:

$$M_{t_4}^- = M_{t_3}^+ \tag{4.2-15}$$

$$\Rightarrow M_{t_4}^+ = M_{t_4}^- + col_{t_4}(C) = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -1 \\ -1 \\ 1 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \tag{4.2-16}$$

5). $M_{t_4}^+ \Rightarrow$ *executie* t_5; (4.2-17)

Deci:

$$M_{t_5}^- = M_{t_4}^+ \tag{4.2-18}$$

$$\Rightarrow M_{t_5}^+ = M_{t_5}^- + col_{t_5}(C) = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{Vmatrix} = \begin{Vmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \tag{4.2-19}$$

6). $M_{t_5}^+ = M_0 \Rightarrow$ *executie* t_1, *s.a.m.d* (4.2-20)

Așadar, în rețeaua în discuție, se execută toate tranzițiile, de manieră ciclică -așa cum se cuvine, ținând seamă de entitatea modelată-, în ordinea:

$$t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2 \cdots \qquad (4.2\text{-}21)$$

ceea ce arată corecta funcționare a acestei entități.

Este de remarcat faptul că rețeaua este *conservativă* [BRAM'89], în toate stările prin care ea evoluează numărul pozițiilor cu marcajul de valoare 1 fiind egal cu 2, la fel ca în starea inițială.

4.3. Modelarea unui subsistem *două mașini-unelte~un manipulator*

Se consideră că cele două mașini-unelte, situate una în stânga, MUSt, cealaltă în dreapta, MUDr, față de manipulator, MP, sunt deservite de către acesta prin încărcare cu semifabricate, preluate dintr-un depozit de semifabricate, DSf, respectiv prin descărcare de piese finite, depuse într-un depozit de piese finite, DPf. Topologia subsistemului și traseele pe care se poate deplasa dispozitivul de prehensiune al manipulatorului sunt prezentate în figura 4.3_1.

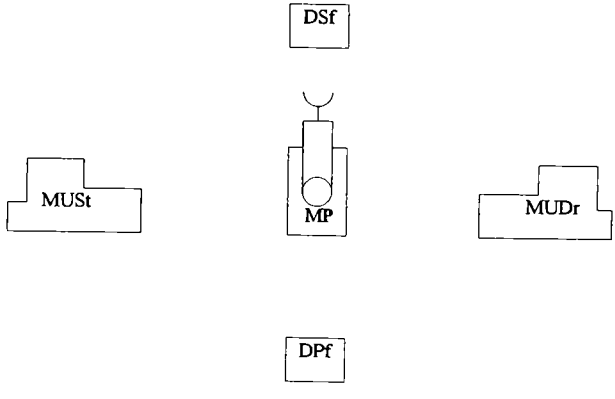


Fig. 4.3_1. Topologia subsistemului *două mașini-unelte~un manipulator*.

Manipulatorul din cadrul unui asemenea subsistem, ținut (este vorba despre manipulator), preferențial, în poziția situată la mijlocul distanței dintre cele două mașini-unelte -poziție numită *de așteptare*-, are de efectuat următoarele operații:

- Deplasare -descărcat- la depozitul de semifabricate, încărcare, și revenire în poziția de așteptare. Această operație o vom numi, generic, "*încărcare manipulator cu semifabricat*".

Condiția necesară execuției ei este ca manipulatorul să fie liber și, de asemenea, să fie liberă cel puțin una dintre mașinile-unelte.

- Deplasare -încărcat cu semifabricat- la mașina-unealtă din stânga, descărcare de semifabricat și, implicit, încărcarea acestei mașini-unelte, și revenire în poziția de așteptare. Această operație o vom numi, generic, "*încărcare mașină-unealtă stânga*". Condițiile necesare execuției ei sunt ca mașina-unealtă stânga să fie liberă și manipulatorul să fie încărcat cu semifabricat.
- Deplasare -încărcat cu semifabricat- la mașina-unealtă din dreapta, descărcare de semifabricat și, implicit, încărcarea acestei mașini-unelte, și revenire în poziția de așteptare. Această operație o vom numi, generic, "*încărcare mașină-unealtă dreapta*". Condițiile necesare execuției ei sunt ca mașina-unealtă dreapta să fie liberă și manipulatorul să fie încărcat cu semifabricat.
- Deplasare -descărcat- la mașina-unealtă din stânga, încărcare cu piesa obținută și, implicit, descărcarea acestei mașini-unelte, și deplasare la depozitul de piese. Această operație o vom numi, generic, "*descărcare mașină-unealtă stânga*". Condițiile necesare execuției ei sunt ca mașina-unealtă stânga să-și fi încheiat prelucrarea, ajungând gata de descărcare, și manipulatorul să fie liber.
- Deplasare -descărcat- la mașina-unealtă din dreapta, încărcare cu piesa obținută și, implicit, descărcarea acestei mașini-unelte, și deplasare la depozitul de piese. Această operație o vom numi, generic, "*descărcare mașină-unealtă dreapta*". Condițiile necesare execuției ei sunt ca mașina-unealtă dreapta să-și fi încheiat prelucrarea, ajungând gata de descărcare, și manipulatorul să fie liber.
- Descărcare piesă în depozitul de piese și revenire în poziția de așteptare. Această operație o vom numi, generic, "*descărcare manipulator de piesă*". Condiția necesară execuției ei este ca manipulatorul să fi ajuns gata de descărcare.

Mașinile-unelte au de efectuat următoarele operații:

- încărcare;
- prelucrare;
- descărcare.

Referitor la operațiile de încărcare, respectiv de descărcare, totul a fost precizat mai sus.

Operația de prelucrare pe o mașină-unealtă necesită condiția ca respectiva mașină să fie încărcată cu semifabricat.

Rețeaua Petri, de tip "*poziții-tranziții*", care modelează un asemenea subsistem *două mașini-unelte~un manipulator* -a se vedea figura 4.3_2- are nouă poziții -șapte de capacitate 1 și două de capacitate 3- și nouă tranziții:

- poziția p_1 este dedicată condiției "*manipulator liber*";
- poziția p_2 este dedicată condiției "*manipulator încărcat cu semifabricat*";
- poziția p_3 este dedicată condiției "*manipulator gata de descărcare piesă*";
- poziția p_4 este dedicată condiției "*mașină-unealtă stânga liberă*";

- poziția p_5 este dedicată condiției "*mașină-unealtă stânga încărcată*";
- poziția p_6 este dedicată condiției "*mașină-unealtă stânga gata de descărcare*";
- poziția p_7 este dedicată condiției "*mașină-unealtă dreapta liberă*";
- poziția p_8 este dedicată condiției "*mașină-unealtă dreapta încărcată*";
- poziția p_9 este dedicată condiției "*mașină-unealtă dreapta gata de descărcare*";

- tranziția t_1 corespunde evenimentului "*încărcare manipulator cu semifabricat grație mașinii-unelte stânga*";
- tranziția t_2 corespunde evenimentului "*încărcare manipulator cu semifabricat grație mașinii-unelte dreapta*";
- tranziția t_3 corespunde evenimentului "*descărcare manipulator de piesă*".
- tranziția t_4 corespunde evenimentului "*încărcare mașină-unealtă stânga*";
- tranziția t_5 corespunde evenimentului "*prelucrare pe mașina-unealtă stânga*";
- tranziția t_6 corespunde evenimentului "*descărcare mașină-unealtă stânga*";
- tranziția t_7 corespunde evenimentului "*încărcare mașină-unealtă dreapta*";
- tranziția t_8 corespunde evenimentului "*prelucrare pe mașina-unealtă dreapta*";
- tranziția t_9 corespunde evenimentului "*descărcare mașină-unealtă dreapta*".

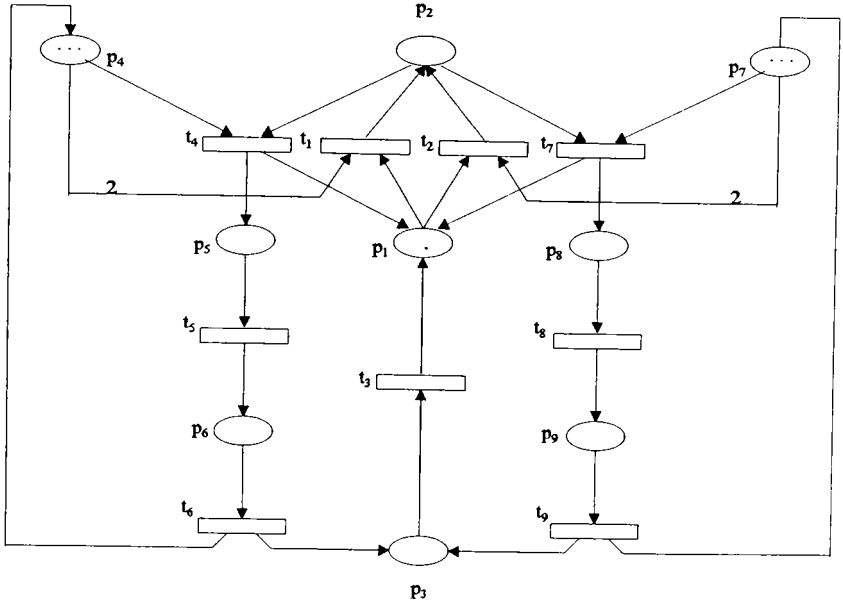


Fig. 4.3_2. Rețeaua Petri corespunzătoare subsistemului *două mașini-unelte-un manipulator.*

Matricile *Pre* și *Post* ale rețelei din figura 4.3_2 sunt:

$$Pre = \begin{array}{c} \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \\ \left\| \begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{array} \end{array} \tag{4.3-1}$$

$$Post = \begin{array}{c} \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \\ \left\| \begin{array}{ccccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{array} \end{array} \tag{4.3-2}$$

iar pentru matricea C se obține:

$$C = \begin{array}{c} \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \\ \left\| \begin{array}{ccccccccc} -1 & -1 & 1 & 1 & 0 & -1 & 1 & 0 & -1 \\ 1 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 \\ -2 & 0 & 0 & -1 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{array} \end{array} \tag{4.3-3}$$

Ținând seamă că marcajul inițial este:

$$M_0 = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{matrix} \qquad (4.3\text{-}4)$$

și aplicând relațiile (4.1-7)...(4.1-9), se ajunge la concluzia că rețeaua de modelare evoluează prin 24 stări (incluzând-o și pe cea inițială), pe care le parcurge așa cum se arată în figura 4.3_ 3 (în această figură, cu "M_i^j" s-a notat marcajul la care se ajunge prin execuția tranziției "t_i" într-un context anume, "j".

Figura 4.3_3 evidențiază faptul că, pentru fiecare stare a rețelei, există cel puțin câte o tranziție executabilă, ceea ce înseamnă că rețeaua este viabilă.

O altă modalitate de a analiza rețeaua Petri din figura 4.3_2 are la bază calculul invarianților. Ea va fi ilustrată în continuare.

Relația (4.1-14) se instanțiază prin:

$$\| x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9 \|.$$

$$\begin{Vmatrix} -1 & -1 & 1 & 1 & 0 & -1 & 1 & 0 & -1 \\ 1 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 \\ -2 & 0 & 0 & -1 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{Vmatrix} = \qquad (4.3\text{-}5)$$

$$= \| 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \|$$

de unde rezultă:

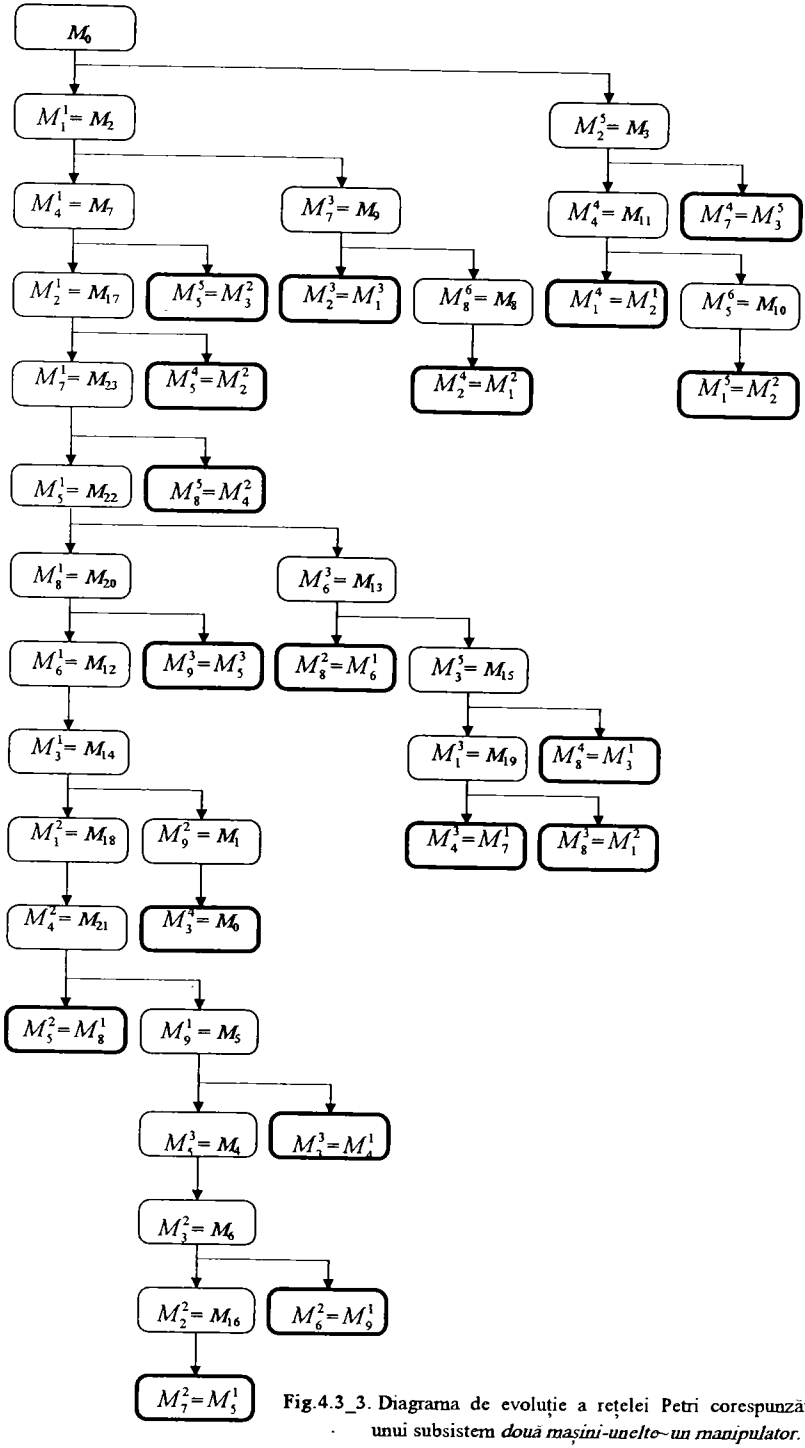


Fig.4.3_3. Diagrama de evoluție a rețelei Petri corespunzătoare unui subsistem *două mașini-unelte-un manipulator.*

$$\begin{cases} -x_1 + x_2 - 2x_4 = 0 \\ -x_1 + x_2 - 2x_7 = 0 \\ x_1 - x_3 = 0 \\ x_1 - x_2 - x_4 + x_5 = 0 \\ -x_5 + x_6 = 0 \\ -x_1 + x_3 + 3x_4 - x_6 = 0 \\ x_1 - x_2 - x_7 + x_8 = 0 \\ -x_8 + x_9 = 0 \\ -x_1 + x_3 + 3x_7 - x_9 = 0 \end{cases} \tag{4.3-6}$$

echivalent cu:

$$\begin{cases} x_3 = x_1 \\ x_5 = 3x_4 \\ x_6 = 3x_4 \\ x_8 = 3x_7 \\ x_9 = 3x_7 \\ -x_1 + x_2 - 2x_4 = 0 \\ -x_1 + x_2 - 2x_7 = 0 \\ x_1 - x_2 + 2x_4 = 0 \\ x_1 - x_2 + 2x_7 = 0 \end{cases} \tag{4.3-7}$$

Rangul matricii acestui sistem este 7. De altfel, ultimele 4 ecuații fac evidentă dubla nedeterminare. Se va obține:

$$\begin{cases} x_1 = p \\ x_2 = p + 2q \\ x_3 = p \\ x_4 = q \\ x_5 = 3q \\ x_6 = 3q \\ x_7 = q \\ x_8 = 3q \\ x_9 = 3q \end{cases} \tag{4.3-8}$$

Așadar:

$$I^{\mathrm{T}} = \| \quad p \quad p+2q \quad p \quad q \quad 3q \quad 3q \quad q \quad 3q \quad 3q \quad \| \tag{4.3-9}$$

Considerând, acum, un marcaj generic:

$$M = \begin{Vmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \end{Vmatrix} \tag{4.3-10}$$

pe baza proprietății specifică invarianților:

$$I^{\mathrm{T}} \cdot M = I^{\mathrm{T}} \cdot M_0 \tag{4.3-11}$$

se obține:

$$pm_1 + (p+2q)m_2 + pm_3 + qm_4 + 3qm_5 + 3qm_6 + qm_7 + 3qm_8 + 3qm_9 = p + 6q \tag{4.3-12}$$

Alegând:

$$q = 0 \tag{4.3-13}$$

pentru:

$$p \neq 0 \tag{4.3-14}$$

rezultă:

$$m_1 + m_2 + m_3 = 1 \tag{4.3-15}$$

ceea ce pune în evidență respectarea principiului excluderii mutuale cu referire la manipulator, acesta, într-adevăr, neputând satisface, la un moment, decât una dintre condițiile modelate prin pozițiile p_1, p_2, si p_3 ("*manipulator liber*", respectiv "*manipulator încărcat cu semifabricat*", respectiv "*manipulator gata de descărcare*").

Alegând:

$$p = q \neq 0 \tag{4.3-16}$$

rezultă:

$$m_1 + 3m_2 + m_3 + m_4 + 3m_5 + 3m_6 + m_7 + 3m_8 + 3m_9 = 7 \tag{4.3-17}$$

Dar, structura rețelei impune, în mod evident:

$$a).\ m_5 + m_6 + m_8 + m_9 = 0 \Rightarrow m_5 = m_6 = m_8 = m_9 = 0 \tag{4.3-18}$$

sau:

$$b).\ m_5 + m_6 + m_8 + m_9 = 1 \Rightarrow \begin{cases} m_5 + m_6 = 1 \\ m_8 + m_9 = 0 \end{cases} sau \begin{cases} m_5 + m_6 = 0 \\ m_8 + m_9 = 1 \end{cases} \tag{4.3-19}$$

sau:

$$c).\ m_5 + m_6 + m_8 + m_9 = 2 \Rightarrow \begin{cases} m_5 + m_6 = 1 \\ m_8 + m_9 = 1 \end{cases} \tag{4.3-20}$$

În *cazul a),* se va avea:

- dacă $m_2 = 0$, atunci $\begin{cases} m_1 + m_3 = 1 \Rightarrow (m_1 = 0,\ m_3 = 1)\ sau\ (m_1 = 1,\ m_3 = 0) \\ m_4 + m_7 = 6 \Rightarrow (m_4 = 3,\ m_7 = 3) \end{cases}$ (4.3-21)

ceea ce conduce la marcajele:

$$M_0 = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \tag{4.3-22}$$

$$M_1 = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 3 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \tag{4.3-23}$$

- dacă $m_2 = 1$, atunci $\begin{cases} m_1 + m_3 = 0 \Rightarrow (m_1 = 0,\ m_3 = 0) \\ m_4 + m_7 = 4 \Rightarrow (m_4 = 1,\ m_7 = 3)\ sau\ (m_4 = 3,\ m_7 = 1) \end{cases}$ (4.3-24)

ceea ce conduce la marcajele:

$$M_2 = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \qquad (4.3\text{-}25) \qquad M_3 = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \qquad (4.3\text{-}26)$$

În *cazul b),* se va avea:

- dacă $m_2 = 0$, atunci $\begin{cases} m_1 + m_3 = 1 \Rightarrow (m_1 = 0,\ m_3 = 1)\ sau\ (m_1 = 1,\ m_3 = 0) \\ m_4 + m_7 = 3 \Rightarrow (m_4 = 0,\ m_7 = 3)\ sau\ (m_4 = 1,\ m_7 = 2)\ sau \\ \qquad\qquad sau\ (m_4 = 2,\ m_7 = 1)\ sau\ (m_4 = 3,\ m_7 = 0) \end{cases}$ (4.3-27)

ceea ce, ținând seamă că:

$$m_5 + m_6 = 1 \Rightarrow m_4 \in \{0,\ 2\} \qquad (4.3\text{-}28)$$

iar:

$$m_8 + m_9 = 1 \Rightarrow m_7 \in \{0,\ 2\} \qquad (4.3\text{-}29)$$

în timp ce:

$$m_3 = 1 \Rightarrow m_4 = 3\ sau\ m_7 = 3 \qquad (4.3\text{-}30)$$

conduce la marcajele:

$$M_4 = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \qquad (4.3\text{-}31) \qquad M_5 = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \qquad (4.3\text{-}32)$$

$$M_6 = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \quad (4.3\text{-}33) \qquad M_7 = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \end{Vmatrix} \quad (4.3\text{-}34)$$

$$M_8 = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 2 \\ 0 \\ 1 \end{Vmatrix} \quad (4.3\text{-}35) \qquad M_9 = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 2 \\ 1 \\ 0 \end{Vmatrix} \quad (4.3\text{-}36)$$

$$M_{10} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \quad (4.3\text{-}37) \qquad M_{11} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \quad (4.3\text{-}38)$$

$$M_{12} = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \quad (4.3\text{-}39) \qquad M_{13} = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 3 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{Vmatrix} \quad (4.3\text{-}40)$$

$$M_{14} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \quad (4.3\text{-}41) \qquad M_{15} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{Vmatrix} \quad (4.3\text{-}42)$$

- dacă $m_2 = 1$, atunci $\begin{cases} m_1 + m_3 = 0 \Rightarrow (m_1 = 0,\ m_3 = 0) \\ m_4 + m_7 = 1 \Rightarrow (m_4 = 0,\ m_7 = 1)\ sau\ (m_4 = 1,\ m_7 = 0) \end{cases}$ (4.3-43)

ceea ce, ținând seamă că:

$$m_5 + m_6 = 1 \Rightarrow m_4 \in \{0,\ 2\} \quad (4.3\text{-}44)$$

iar:

$$m_8 + m_9 = 1 \Rightarrow m_7 \in \{0,\ 2\} \quad (4.3\text{-}45)$$

conduce la marcajele:

$$M_{16} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \quad (4.3\text{-}46) \qquad M_{17} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \quad (4.3\text{-}47)$$

$$M_{18} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \quad (4.3\text{-}48) \qquad M_{19} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{Vmatrix} \quad (4.3\text{-}49)$$

În *cazul c),* se va avea:

- dacă $m_2 = 0$, atunci $\begin{cases} m_1 + m_3 = 1 \Rightarrow (m_1 = 0,\ m_3 = 1)\ sau\ (m_1 = 1,\ m_3 = 0) \\ m_4 + m_7 = 0 \Rightarrow (m_4 = 0,\ m_7 = 0) \end{cases}$ (4.3-50)

ceea ce, ținând seamă că:

$$m_3 = 1 \Rightarrow m_4 = 3\ sau\ m_7 = 3 \tag{4.3-51}$$

conduce la marcajele:

$$M_{20} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \tag{4.3-52}$$

$$M_{21} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \tag{4.3-53}$$

$$M_{22} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{Vmatrix} \tag{4.3-54}$$

$$M_{23} = \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{Vmatrix} \tag{4.3-55}$$

- dacă $m_2 = 1$, atunci *(4.3-20) este o imposibilitate.* (4.3-56)

Analizând cele 24 marcaje obținute, se constată că ele sunt aceleași cu cele din diagrama reprezentată în figura 4.3_3. Existența a cel puțin unei tranziții executabilă corespunzător fiecăruia dintre ele face ca rețeaua să fie viabilă.

4.4. Modelarea unui subsistem *"n" mașini-unelte~un manipulator*

Un subsistem mai general și care are o însemnătate practică mai ridicată decât subsistemul ce a constituit obiectul subcapitolului precedent este cel constând din "*n*" mașini-unelte necooperante -cu $n \geq 3$- și un manipulator ce le deservește, într-o ordine oarecare. Un asemenea subsistem este reprezentat, pentru "*n*" neprecizat, în figura 4.4_1. Notațiile folosite au semnificațiile definite în subcapitolul precedent.

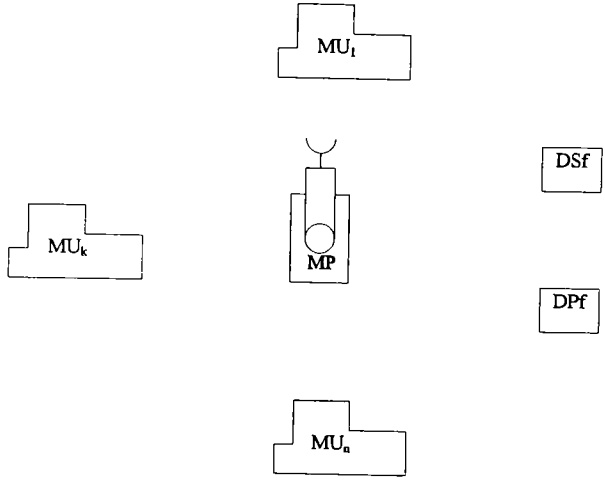


Fig. 4.4_1. Topologia subsistemului *"n" mașini-unelte~un manipulator.*

Rețeaua Petri care modelează subsistemul din figura 4.4_1 este reprezentată în figura 4.4_2. Se poate înțelege că, tot așa ca și în cazul particular cu două mașini unelte, pozițiile $p_1 \ldots p_3$ sunt referitoare la funcționarea manipulatorului. La fel, tranzițiile $t_1 \ldots t_{n+1}$. Condițiile care intervin în funcționarea unei mașini-unelte oarecare, "*k*", sunt modelate prin pozițiile p_{4k}, p_{4k+1}, p_{4k+2}, p_{4k+3}. Tranzițiile ce privesc mașina-unealtă "*k*" sunt: t_{n+3k-1}, t_{n+3k}, t_{n+3k+1}. Semnificațiile pozițiilor și tranzițiilor sunt:

- poziția p_1 este dedicată condiției "*manipulator liber*";
- poziția p_2 este dedicată condiției "*manipulator încărcat cu semifabricat*";
- poziția p_3 este dedicată condiției "*manipulator gata de descărcare piesă*";
- poziția p_4 este dedicată condiției "*mașină-unealtă 1 gata de încărcare cu semifabricat*"
- poziția p_5 este dedicată condiției "*mașină-unealtă 1 în solicitare de încărcare*";
- poziția p_6 este dedicată condiției "*mașină-unealtă 1 încărcată*";
- poziția p_7 este dedicată condiției "*mașină-unealtă 1 gata de descărcare*";

- ...
- poziția p_{4k} este dedicată condiției "*mașină-unealtă "k" gata de încărcare cu semifabricat*"
- poziția p_{4k+1} este dedicată condiției "*mașină-unealtă "k" în solicitare de încărcare*";
- poziția p_{4k+2} este dedicată condiției "*mașină-unealtă "k" încărcată*";
- poziția p_{4k+3} este dedicată condiției "*mașină-unealtă "k" gata de descărcare*";
- ...
- poziția p_{4n} este dedicată condiției "*mașină-unealtă "n" gata de încărcare cu semifabricat*"
- poziția p_{4n+1} este dedicată condiției "*mașină-unealtă "n" în solicitare de încărcare*";
- poziția p_{4n+2} este dedicată condiției "*mașină-unealtă "n" încărcată*";
- poziția p_{4n+3} este dedicată condiției "*mașină-unealtă "n" gata de descărcare*";

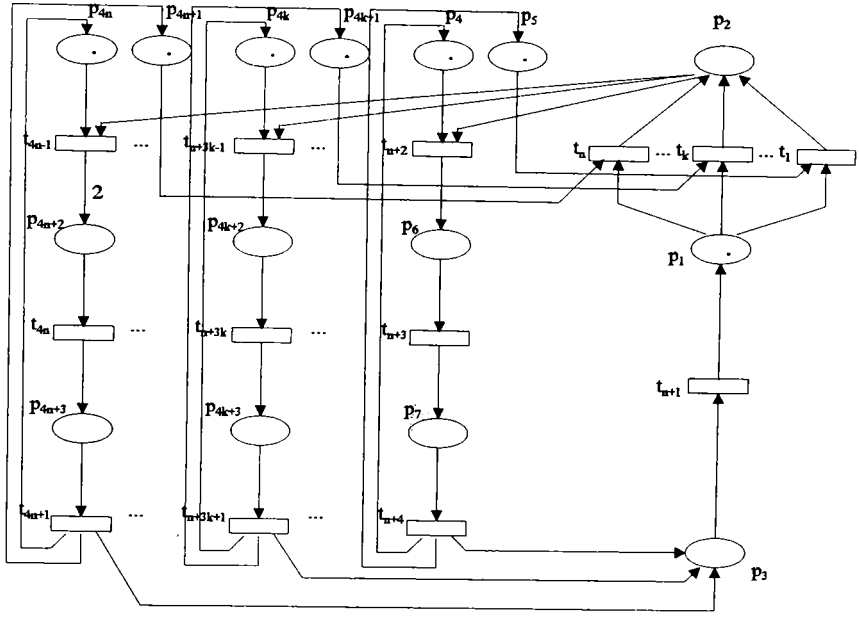


Fig. 4.4_2. Rețeaua Petri corespunzătoare subsistemului *"n" mașini-unelte-un manipulator.*

- tranziția t_1 corespunde evenimentului "*încărcare manipulator cu semifabricat grație mașinii-unelte 1*";
- ...
- tranziția t_k corespunde evenimentului "*încărcare manipulator cu semifabricat grație mașinii-unelte "k"*";
- ...
- tranziția t_n corespunde evenimentului "*încărcare manipulator cu semifabricat grație mașinii-unelte "n"*";

- tranziția t_{n+1} corespunde evenimentului "*descărcare manipulator de piesă*";

- tranziția t_{n+2} corespunde evenimentului "*încărcare mașină-unealtă 1*";
- tranziția t_{n+3} corespunde evenimentului "*prelucrare pe mașina-unealtă 1*";
- tranziția t_{n+4} corespunde evenimentului "*descărcare mașină-unealtă 1*";
- ...
- tranziția t_{n+3k-1} corespunde evenimentului "*încărcare mașină-unealtă "k"*;
- tranziția t_{n+3k} corespunde evenimentului "*prelucrare pe mașina-unealtă "k"*;
- tranziția t_{n+3k+1} corespunde evenimentului "*descărcare mașină-unealtă "k"*;
- ...
- tranziția t_{4n-1} corespunde evenimentului "*încărcare mașină-unealtă "n"*;
- tranziția t_{4n} corespunde evenimentului "*prelucrare pe mașina-unealtă "n"*;
- tranziția t_{4n+1} corespunde evenimentului "*descărcare mașină-unealtă "n"*.

Pentru un "*n*" fixat, evoluția rețelei se poate analiza așa cum s-a arătat în subcapitolul precedent. Bineînțeles, însă, travaliul necesitat crește considerabil, pe măsură ce crește valoarea lui "*n*".

4.5. Modelarea unui subsistem cooperativ *două mașini-unelte~două manipulatoare*

Se consideră un subsistem de tip cooperativ -de exemplu, de asamblare-, constituit din două mașini-unelte, MU1 și MU2, și din două manipulatoare, MP1 și MP2. Se va presupune că fiecare mașină-unealtă are rolul de a solidariza două semifabricate de tipuri diferite, Sf1 și Sf2, încărcate

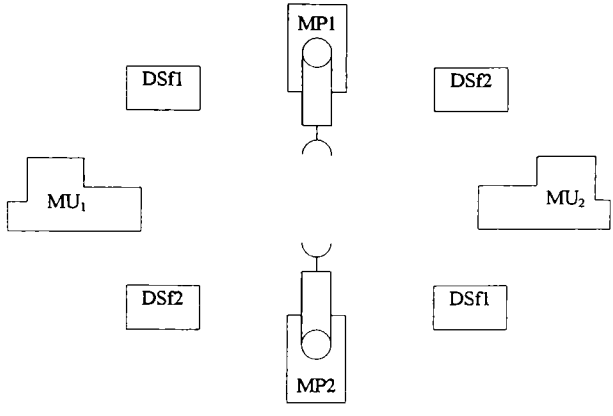


Fig. 4.5_1. Topologia subsistemului cooperativ *două mașini-unelte~două manipulatoare*

și menținute în poziția de solidarizare, în una dintre ele -este vorba despre mașinile-unelte-, de către cele două manipulatoare. De asemenea, se presupune că la nivelul mașinii-unelte are loc, premergător fiecărei operații de solidarizare, un proces de curățare-spălare. Semifabricatele de asamblat se găsesc separate, în funcție de tip, în depozite tampon, DSf1 și DSf2, situate, câte unul pentru fiecare tip, în zona de acces a fiecărui manipulator. Topologia subsistemului este prezentată în figura 4.5_1.

Un asemenea subsistem funcționează după cum se arată în continuare.

Inițial, manipulatoarele și mașinile-unelte sunt libere. Apoi, este posibil să fie deservită și să intre în lucru fie mașina-unealtă MU1, fie mașina-unealtă MU2.

În primul caz, se încarcă manipulatorul MP1 cu un semifabricat de tipul Sf1 și manipulatorul MU2 cu un semifabricat de tipul Sf2, și se efectuează operația de curățare-spălare a mașinii-unelte MU1. După aceea, manipulatoarele MP1 și MP2 aduc semifabricatele în această mașină-unealtă și le mențin în poziția de solidarizare. Are loc operația de solidarizare. La încheierea ei, produsul finit este evacuat, prin eliberarea lui de către cele două manipulatoare. Astfel, acestea, ca și mașina-unealtă MU1 devin din nou libere.

În al doilea caz, se încarcă manipulatorul MP2 cu un semifabricat de tipul Sf1 și manipulatorul MP1 cu un semifabricat de tipul Sf2 și se execută operația de curățare-spălare a mașinii-unelte MU2. După aceea, manipulatoarele MP1 și MP2 aduc semifabricatele în această mașină-unealtă și le mențin în poziția de solidarizare. Se continuă ca și în cazul precedent, în final, atât manipulatoarele, cât și mașina-unealtă MU2 reajungând libere.

Rețeaua Petri de tip "*condiții-evenimente*" care modelează subsistemul considerat -a se vedea figura 4.5_2- are 14 poziții și 8 tranziții:

- poziția p_1 este dedicată condiției "*manipulator MP1 liber*";
- poziția p_2 este dedicată condiției "*mașină-unealtă MU1 liberă*";
- poziția p_3 este dedicată condiției "*manipulator MP1 încărcat cu un semifabricat de tipul Sf1*";
- poziția p_4 este dedicată condiției "*mașină-unealtă MU1 curățată și spălată*";
- poziția p_5 este dedicată condiției "*manipulator MP2 încărcat cu un semifabricat de tipul Sf2*";
- poziția p_6 este dedicată condiției "*mașină-unealtă MU1 încărcată*";
- poziția p_7 este dedicată condiției "*mașină-unealtă MU1 gata de descărcare*";
- poziția p_8 este dedicată condiției "*manipulator MP2 liber*";
- poziția p_9 este dedicată condiției "*mașină-unealtă MU2 liberă*";
- poziția p_{10} este dedicată condiției "*manipulator MP2 încărcat cu un semifabricat de tipul Sf1*";
- poziția p_{11} este dedicată condiției "*mașină-unealtă MU2 curățată și spălată*";
- poziția p_{12} este dedicată condiției "*manipulator MP1 încărcat cu un semifabricat de tipul Sf2*";
- poziția p_{13} este dedicată condiției "*mașină-unealtă MU2 încărcată*";
- poziția p_{14} este dedicată condiției "*mașină-unealtă MU2 gata de descărcare*";

- tranziția t_1 corespunde evenimentului "*pregătire operare mașină-unealtă MU1*";
- tranziția t_2 corespunde evenimentului "*încărcare mașină-unealtă MU1*";
- tranziția t_3 corespunde evenimentului "*operare pe mașina-unealtă MU1*";
- tranziția t_4 corespunde evenimentului "*evacuare produs finit din mașina-unealtă MU1*";
- tranziția t_5 corespunde evenimentului "*pregătire operare mașină-unealtă MU2*";
- tranziția t_6 corespunde evenimentului "*încărcare mașină-unealtă MU2*";
- tranziția t_7 corespunde evenimentului "*operare pe mașina-unealtă MU2*";
- tranziția t_8 corespunde evenimentului "*evacuare produs finit din mașina-unealtă MU2.*

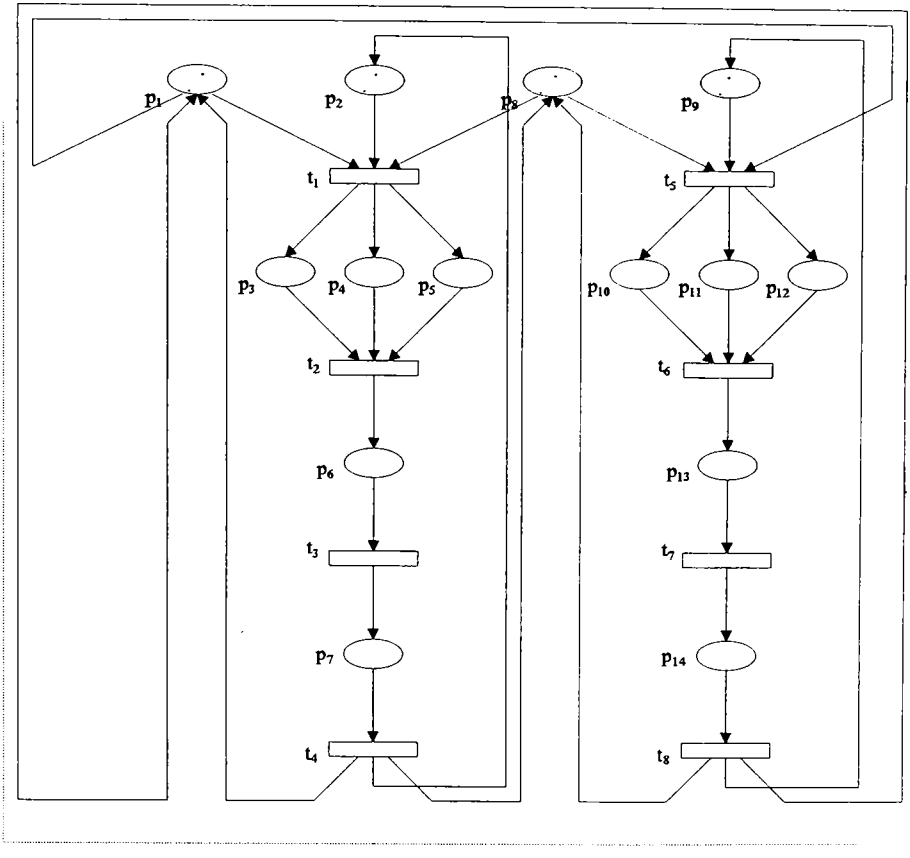


Fig. 4.5_2. Rețeaua Petri corespunzătoare subsistemului cooperativ *două mașini-unelte-două manipulatoare.*

Matricile *Pre* și *Post* ale rețelei din figura 4.5_2 sunt:

$$Pre = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \end{matrix} \\ \left\| \begin{matrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix} \right\| \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{matrix} \end{array}$$

(4.5-1)

$$Post = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \end{matrix} \\ \left\| \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{matrix} \right\| \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{matrix} \end{array}$$

(4.5-2)

iar pentru matricea C, se obține instanțierea (4.5-3).

$$C = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \end{matrix} \\ \left\| \begin{matrix} -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{matrix} \right\| \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{matrix} \end{array}$$

(4.5-3)

$$M_0 = \left\| \begin{matrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \right\| \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{matrix}$$

(4.5-4)

Ținând seamă că marcajul inițial este cel dat de relația (4.5-4) și aplicând relațiile (4.1-7)...(4.1-9), se ajunge la concluzia că rețeaua de modelare evoluează prin 7 stări, pe care le parcurge așa cum se arată în figura 4.5_3.

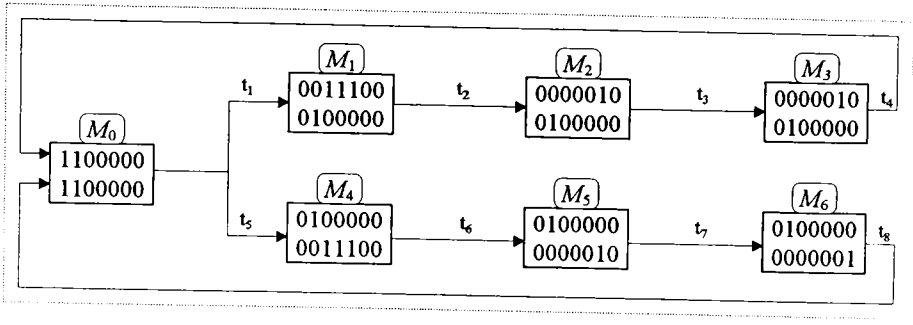


Fig. 4.5_3. Diagrama de evoluție a rețelei de modelare a subsistemului cooperativ *două mașini-unelte-două manipulatoare.*

Calculele sunt următoarele:

$$1).\ M_0 \Rightarrow \begin{cases} a).\ executie\ t_1 \\ b).\ executie\ t_5 \end{cases} \tag{4.5-5}$$

$$a).\ \Rightarrow M_{t_1}^- = M_0 \tag{4.5-6}$$

$$\Rightarrow M_{t_1}^+ = M_{t_1}^- + col_{t_1}(C) = \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \tag{4.5-7}$$

$$2).\ M_{t_1}^+ \Rightarrow executie\ t_2 \tag{4.5-8}$$

$$\Rightarrow M_{t_2}^- = M_{t_1}^+ \tag{4.5-9}$$

$$\Rightarrow M_{t_2}^{+} = M_{t_2}^{-} + col_{t_2}(C) = \begin{Vmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ -1 \\ -1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \quad (4.5\text{-}10)$$

3). $M_{t_2}^{+} \Rightarrow$ *executie* t_3 (4.5-11)

$$\Rightarrow M_{t_3}^{-} = M_{t_2}^{+} \quad (4.5\text{-}12)$$

$$\Rightarrow M_{t_3}^{+} = M_{t_3}^{-} + col_{t_3}(C) = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \quad (4.5\text{-}13)$$

4). $M_{t_3}^{+} \Rightarrow$ *executie* t_4 (4.5-14)

$$\Rightarrow M_{t_4}^{-} = M_{t_3}^{+} \quad (4.5\text{-}15)$$

$$\Rightarrow M_{t_4}^{+} = M_{t_4}^{-} + col_{t_4}(C) = \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = M_0 \qquad (4.5\text{-}16)$$

5). *ramura 1b).*

$$M_0 \Rightarrow \textit{executie } t_5 \qquad (4.5\text{-}17)$$

$$\Rightarrow M_{t_5}^{-} = M_0 \qquad (4.5\text{-}18)$$

$$\Rightarrow M_{t_5}^{+} = M_{t_5}^{-} + col_{t_5}(C) = \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} \qquad (4.5\text{-}19)$$

6). $M_{t_6}^{+} \Rightarrow \textit{executie } t_6$ (4.5-20)

$$\Rightarrow M_{t_6}^{-} = M_{t_5}^{+} \qquad (4.5\text{-}21)$$

$$\Rightarrow M_{t_6}^{+} = M_{t_6}^{-} + col_{t_6}(C) = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \\ 1 \\ 0 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{Vmatrix} \tag{4.5-22}$$

7). $M_{t_6}^{+} \Rightarrow$ *executie* t_7 (4.5-23)

$$\Rightarrow M_{t_7}^{-} = M_{t_6}^{+} \tag{4.5-24}$$

$$\Rightarrow M_{t_7}^{+} = M_{t_7}^{-} + col_{t_7}(C) = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{Vmatrix} = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} \tag{4.5-25}$$

8). $M_{t_7}^{+} \Rightarrow$ *executie* t_8 (4.5-26)

$$\Rightarrow M_{t_8}^{-} = M_{t_7}^{+} \tag{4.5-27}$$

$$\Rightarrow M_{t_8}^{+} = M_{t_8}^{-} + col_{t_8}(C) = \begin{Vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{Vmatrix} = \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = M_0 \qquad (4.5\text{-}28)$$

Rețeaua Petri din figura 4.5_2 se pretează, la fel de bine, a fi analizată prin tehnica calculului invarianților, după cum urmează.

Relația (4.5-2) se instanțiază prin:

$$\begin{Vmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} & x_{14} \end{Vmatrix} \cdot$$

$$\begin{Vmatrix} -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{Vmatrix} = \begin{Vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{Vmatrix} \qquad (4.5\text{-}29)$$

de unde rezultă:

$$\begin{cases} -x_1 - x_2 + x_3 + x_4 + x_5 - x_8 = 0 \\ -x_3 - x_4 - x_5 + x_6 = 0 \\ -x_6 + x_7 = 0 \\ x_1 + x_2 - x_7 + x_8 = 0 \\ -x_1 - x_8 - x_9 + x_{10} + x_{11} + x_{12} = 0 \\ -x_{10} - x_{11} - x_{12} + x_{13} = 0 \\ -x_{13} + x_{14} = 0 \\ x_1 + x_8 + x_9 - x_{14} = 0 \end{cases} \tag{4.5-30}$$

echivalent cu:

$$\begin{cases} x_1 + x_2 - x_6 + x_8 = 0 \\ x_3 + x_4 + x_5 - x_6 = 0 \\ -x_6 + x_7 = 0 \\ x_1 + x_8 + x_9 - x_{13} = 0 \\ x_{10} + x_{11} + x_{12} - x_{13} = 0 \\ -x_{13} + x_{14} = 0 \\ x_1 + x_2 - x_3 - x_4 - x_5 + x_8 = 0 \\ x_1 + x_8 + x_9 - x_{10} - x_{11} - x_{12} = 0 \end{cases} \tag{4.5-31}$$

Matricea sistemului (4.5-31) este:

$$A = \begin{array}{c} \begin{array}{cccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{array} \\ \left\| \begin{array}{cccccccccccccc} 1 & 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 1 & -1 & 0 & 0 \end{array} \right\| \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \end{array} \tag{4.5-32}$$

cu:

$$rang(A) = 6 \tag{4.5-33}$$

Se observă că determinantul obținut din elementele aflate la intersecția primelor 6 linii cu coloanele 2, 5, 7, 9, 12, 14 este:

$$\det\left(\begin{Vmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{Vmatrix}\right) = 1 \tag{4.5-34}$$

ceea ce înseamnă că necunoscutele x_2, x_5, x_7, x_9, x_{12}, $si\, x_{14}$ vor fi principale, iar celelalte -secundare.

Fie:

$$\begin{cases} x_1 = p \\ x_3 = q \\ x_4 = r \\ x_6 = s \\ x_8 = t \\ x_{10} = u \\ x_{11} = v \\ x_{13} = w \end{cases} \tag{4.5-35}$$

Rezultă:

$$\begin{cases} x_1 = p \\ x_2 = s - p - t \\ x_3 = q \\ x_4 = r \\ x_5 = s - q - r \\ x_6 = s \\ x_7 = s \\ x_8 = t \\ x_9 = w - p - t \\ x_{10} = u \\ x_{11} = v \\ x_{12} = w - u - v \\ x_{13} = w \\ x_{14} = w \end{cases} \tag{4.5-36}$$

Așadar:

$$I^T = \| p \quad s-p-t \quad q \quad r \quad s-q-r \quad s \quad s \quad t \quad w-p-t \quad u \quad v \quad w-u-v \quad w \quad w \| \tag{4.5-37}$$

Considerând, acum, un marcaj generic:

$$M = \begin{Vmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \end{Vmatrix} \tag{4.5-38}$$

pe baza proprietății:

$$I^T \cdot M = I^T \cdot M_0 \tag{4.5-39}$$

se obține:

$$\begin{aligned} &pm_1 + (s-p-t)m_2 + qm_3 + rm_4 + (s-q-r)m_5 + sm_6 + sm_7 + tm_8 + \\ &+ (w-p-t)m_9 + um_{10} + vm_{11} + (w-u-v)m_{12} + wm_{13} + wm_{14} = s + w - p - t \end{aligned} \tag{4.5-40}$$

Pentru:

$$\begin{cases} p = 1 \\ t = 1 \\ q = r = s = u = v = w = 0 \end{cases} \tag{4.5-41}$$

se obține:

$$I_1^T = \| 1 \quad -2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad -2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \| \tag{4.5-42}$$

și, pe baza lui:

$$m_1 - 2m_2 + m_8 - 2m_9 = -2 \tag{4.5-43}$$

cu soluțiile:

$$\begin{cases} m_1 = 1 \\ m_2 = 1 \\ m_8 = 1 \\ m_9 = 1 \end{cases} \quad sau \quad \begin{cases} m_1 = 0 \\ m_2 = 0 \\ m_8 = 0 \\ m_9 = 1 \end{cases} \quad sau \quad \begin{cases} m_1 = 0 \\ m_2 = 1 \\ m_8 = 0 \\ m_9 = 0 \end{cases} \tag{4.5-44}$$

(1a) (1b) (1c)

Pentru:

$$\begin{cases} q = 1 \\ r = 1 \\ p = s = t = u = v = w = 0 \end{cases} \tag{4.5-45}$$

se obține:

$$I_2^T = \| 0 \quad 0 \quad 1 \quad 1 \quad -2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \| \tag{4.5-46}$$

și, pe baza lui:

$$m_3 + m_4 - 2m_5 = 0 \tag{4.5-47}$$

cu soluțiile:

$$\begin{cases} m_3 = 1 \\ m_4 = 1 \\ m_5 = 1 \end{cases} \quad sau \quad \begin{cases} m_3 = 0 \\ m_4 = 0 \\ m_5 = 0 \end{cases} \tag{4.5-48}$$

(2a) (2b)

Pentru:

$$\begin{cases} u = 1 \\ v = 1 \\ p = q = r = s = t = w = 0 \end{cases} \tag{4.5-49}$$

se obține:

$$I_3^T = \| 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad -2 \quad 0 \quad 0 \| \tag{4.5-50}$$

și, pe baza lui:

$$m_{10} + m_{11} - 2m_{12} = 0 \tag{4.5-51}$$

cu soluțiile:

$$\begin{cases} m_{10} = 1 \\ m_{11} = 1 \\ m_{12} = 1 \end{cases} \quad sau \quad \begin{cases} m_{10} = 0 \\ m_{11} = 0 \\ m_{12} = 0 \end{cases} \tag{4.5-52}$$

(3a) (3b)

Pentru:

$$\begin{cases} q = 1 \\ r = 1 \\ s = 1 \\ p = t = u = v = w = 0 \end{cases} \tag{4.5-53}$$

se obține:

$$I_4^T = \| 0 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \| \tag{4.5-54}$$

și, pe baza lui:

$$m_2 + m_3 + m_4 - m_5 + m_6 + m_7 = 1 \tag{4.5-55}$$

cu soluțiile:

$$\begin{cases} m_2 = 1 \\ m_3 = 0 \\ m_4 = 0 \\ m_5 = 0 \\ m_6 = 0 \\ m_7 = 0 \end{cases} \quad sau \quad \begin{cases} m_2 = 0 \\ m_3 = 1 \\ m_4 = 1 \\ m_5 = 1 \\ m_6 = 0 \\ m_7 = 0 \end{cases} \quad sau \quad \begin{cases} m_2 = 0 \\ m_3 = 0 \\ m_4 = 0 \\ m_5 = 0 \\ m_6 = 1 \\ m_7 = 0 \end{cases} \quad sau \quad \begin{cases} m_2 = 0 \\ m_3 = 0 \\ m_4 = 0 \\ m_5 = 0 \\ m_6 = 0 \\ m_7 = 1 \end{cases} \tag{4.5-56}$$

(4a) (4b) (4c) (4d)

Pentru:

$$\begin{cases} u = 1 \\ v = 1 \\ w = 1 \\ p = q = r = s = t = 0 \end{cases} \tag{4.5-57}$$

se obține:

$$I_5^T = \| 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \| \tag{4.5-58}$$

și, pe baza lui:

$$m_9 + m_{10} + m_{11} - m_{12} + m_{13} + m_{14} = 1 \tag{4.5-59}$$

cu soluțiile:

$$\begin{cases} m_9 = 1 \\ m_{10} = 0 \\ m_{11} = 0 \\ m_{12} = 0 \\ m_{13} = 0 \\ m_{14} = 0 \end{cases} \quad sau \quad \begin{cases} m_9 = 0 \\ m_{10} = 1 \\ m_{11} = 1 \\ m_{12} = 1 \\ m_{13} = 0 \\ m_{14} = 0 \end{cases} \quad sau \quad \begin{cases} m_9 = 0 \\ m_{10} = 0 \\ m_{11} = 0 \\ m_{12} = 0 \\ m_{13} = 1 \\ m_{14} = 0 \end{cases} \quad sau \quad \begin{cases} m_9 = 0 \\ m_{10} = 0 \\ m_{11} = 0 \\ m_{12} = 0 \\ m_{13} = 0 \\ m_{14} = 1 \end{cases} \tag{4.5-60}$$

(5a) (5b) (5c) (5d)

Analizând, prin prisma compatibilității lor, subseturile de marcaje la care au condus cei 5 invarianți luați în considerare, rezultă că rețeaua Petri în discuție parcurge următoarele marcaje:

$$M_0 = \begin{Vmatrix} 1 & p_1 \\ 1 & p_2 \\ 0 & p_3 \\ 0 & p_4 \\ 0 & p_5 \\ 0 & p_6 \\ 0 & p_7 \\ 1 & p_8 \\ 1 & p_9 \\ 0 & p_{10} \\ 0 & p_{11} \\ 0 & p_{12} \\ 0 & p_{13} \\ 0 & p_{14} \end{Vmatrix}$$

(*implicat de* (1*a*), (4*a*), *si* (5*a*))

(4.5-61)

$$M_1 = \begin{Vmatrix} 0 & p_1 \\ 0 & p_2 \\ 1 & p_3 \\ 1 & p_4 \\ 1 & p_5 \\ 0 & p_6 \\ 0 & p_7 \\ 0 & p_8 \\ 1 & p_9 \\ 0 & p_{10} \\ 0 & p_{11} \\ 0 & p_{12} \\ 0 & p_{13} \\ 0 & p_{14} \end{Vmatrix}$$

(*implicat de* (1*b*), (4*b*), *si* (5*a*))

(4.5-62)

$$M_2 = \begin{Vmatrix} 0 & p_1 \\ 0 & p_2 \\ 0 & p_3 \\ 0 & p_4 \\ 0 & p_5 \\ 1 & p_6 \\ 0 & p_7 \\ 0 & p_8 \\ 1 & p_9 \\ 0 & p_{10} \\ 0 & p_{11} \\ 0 & p_{12} \\ 0 & p_{13} \\ 0 & p_{14} \end{Vmatrix}$$

(*implicat de* (1*b*), (4*c*), *si* (5*a*))

(4.5-63)

$$M_3 = \begin{Vmatrix} 0 & p_1 \\ 0 & p_2 \\ 0 & p_3 \\ 0 & p_4 \\ 0 & p_5 \\ 0 & p_6 \\ 1 & p_7 \\ 0 & p_8 \\ 1 & p_9 \\ 0 & p_{10} \\ 0 & p_{11} \\ 0 & p_{12} \\ 0 & p_{13} \\ 0 & p_{14} \end{Vmatrix}$$

(*implicat de* (1*b*), (4*d*), *si* (5*a*))

(4.5-64)

$$M_4 = \begin{Vmatrix} 0 & p_1 \\ 1 & p_2 \\ 0 & p_3 \\ 0 & p_4 \\ 0 & p_5 \\ 0 & p_6 \\ 0 & p_7 \\ 0 & p_8 \\ 0 & p_9 \\ 1 & p_{10} \\ 1 & p_{11} \\ 1 & p_{12} \\ 0 & p_{13} \\ 0 & p_{14} \end{Vmatrix}$$

(*implicat de* (1c), (4*a*), *si* (5*b*))

(4.5-65)

$$M_5 = \begin{Vmatrix} 0 & p_1 \\ 1 & p_2 \\ 0 & p_3 \\ 0 & p_4 \\ 0 & p_5 \\ 0 & p_6 \\ 0 & p_7 \\ 0 & p_8 \\ 0 & p_9 \\ 0 & p_{10} \\ 0 & p_{11} \\ 0 & p_{12} \\ 1 & p_{13} \\ 0 & p_{14} \end{Vmatrix}$$

(*implicat de* (1c), (4*a*), *si* (5c))

(4.5-66)

$$M_6 = \left\| \begin{matrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{matrix} \right\| \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{matrix}$$

(*implicat de* (1c), (4*a*), *si* (5*d*))

(4.5-67)

Evident, marcajele obținute cu ajutorul invarianților coincid cu cele la care s-a ajuns din aproape în aproape, pe baza relațiilor (4.1-7)...(4.1-9).

4.6. Modelarea unui subsistem cooperativ *"n" mașini-unelte~"n" manipulatoare* cu o rețea Petri colorată

Subsistemul din figura 4.5_1 poate fi generalizat, astfel încât să devină cu "*n*" mașini-unelte și "*n*" manipulatoare, așa cum se arată în figura 4.6_1.

Rețeaua Petri ordinară care modelează acest subsistem este prezentată în figura 4.6_2. Semnificațiile pozițiilor și tranzițiilor sale rezultă prin extrapolarea celor spuse relativ la rețeaua Petri din figura 4.5_2.

Rețeaua Petri ordinară din figura 4.6_2 prezintă neajunsul că are o capacitate redusă de genericitate. Ea reușește să modeleze subsistemul din figura 4.6_1 doar prin extindere până la exhaustiv. Mult mai concis poate fi modelat același subsistem, cu ajutorul unei rețele Petri colorată. Se procedează după cum se arată în continuare.

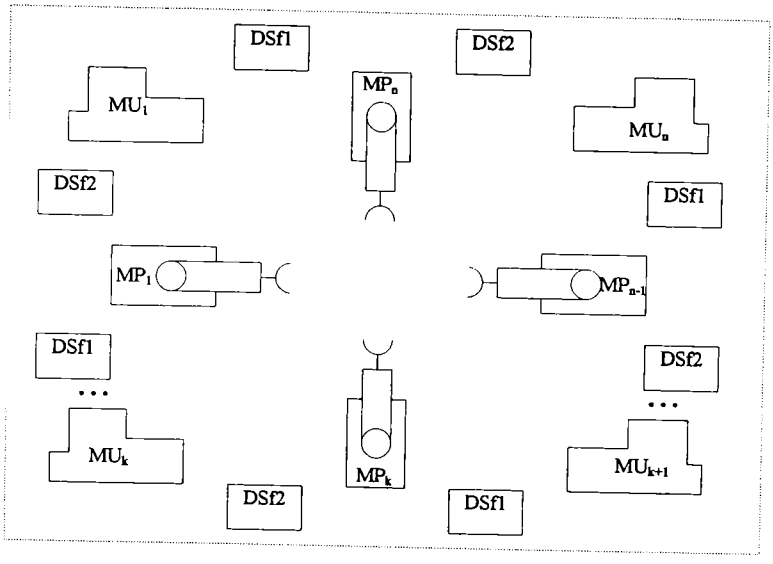


Fig. 4.6_1. Topologia subsistemului cooperativ *"n" mașini-unelte~ "n" manipulatoare.*

Fie:

$$MU = \{mu_1, mu_2, \ldots, mu_n\} \text{ - multimea mașinilor - unelte} \tag{4.6-1}$$

și:

$$MP = \{mp_1, mp_2, \ldots, mp_n\} \text{ - multimea manipulatoarelor} \tag{4.6-2}$$

Mulțimea *MU* reprezintă paleta de culori referitoare la entitatea "*mașină-unealtă*", iar mulțimea *MP*-paleta de culori referitoare la entitatea "*manipulator*".

Două considerente avem în vedere acum:

1) în cazul subsistemului în discuție, evenimentele sunt legate de mașinile-unelte (a se revedea semnificațiile tranzițiilor $t_1 \ldots t_8$ din rețeaua Petri din figura 5.5_2);
2) în cazul subsistemului în discuție, pentru o mașină-unealtă anume, prezintă interes doar cele două manipulatoare vecine: manipulatorul din stânga, respectiv manipulatorul din dreapta.

Plecând de la aceste considerente, introducem funcțiile:

$$ST: MU \rightarrow MP \tag{4.6-3}$$

$$DR: MU \rightarrow MP \tag{4.6-4}$$

$$ID: MU \rightarrow MU \tag{4.6-5}$$

care asigură:

$$ST(mu_1) = mp_n \quad ST(mu_2) = mp_1 \quad \vdots \quad ST(mu_n) = mp_{n-1} \tag{4.6-7}$$

respectiv:

$$DR(mu_1) = mp_1 \quad DR(mu_2) = mp_2 \quad \vdots \quad DR(mu_n) = mp_n \tag{4.6-8}$$

respectiv:

$$ID(mu_i) = mu_i, \ \forall i = 1...n \tag{4.6-9}$$

Folosindu-le, și, desigur, ținând seamă de specificațiile de funcționare a subsistemului de modelat, evidențiate în paragraful 5.5, ajungem la rețeaua Petri colorată reprezentată în figura 4.6_3.

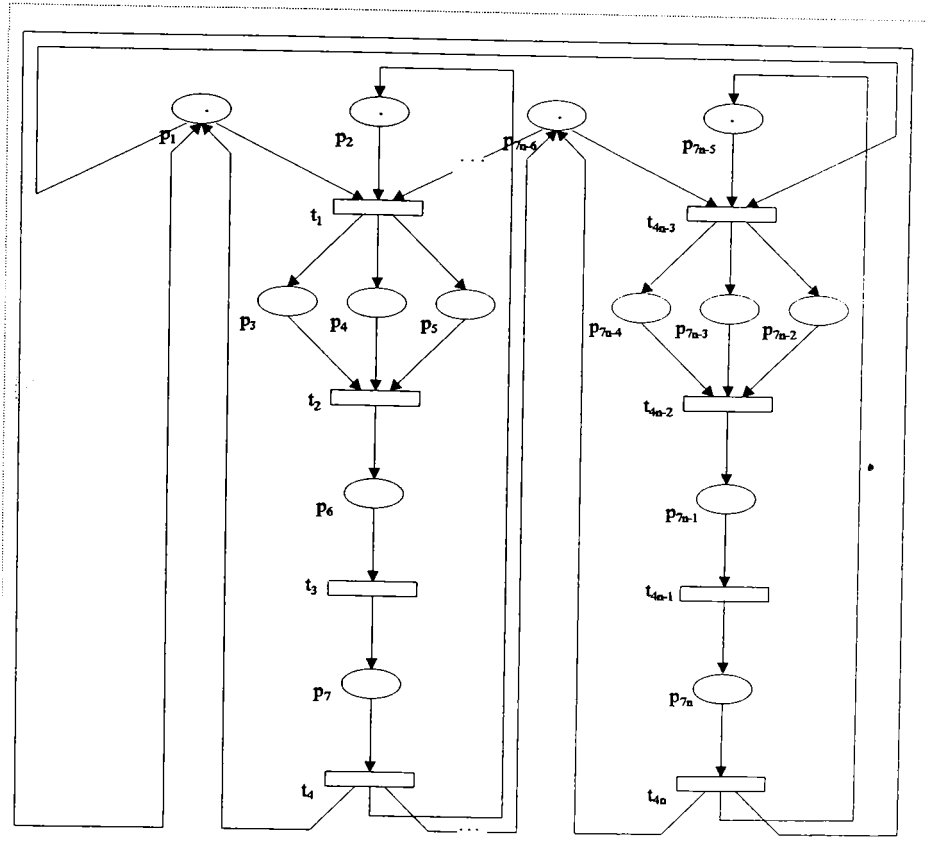


Fig. 4.6_2. Rețeaua Petri ordinară corespunzătoare subsistemului cooperativ *"n" mașini-unelte ~ "n" manipulatoare.*

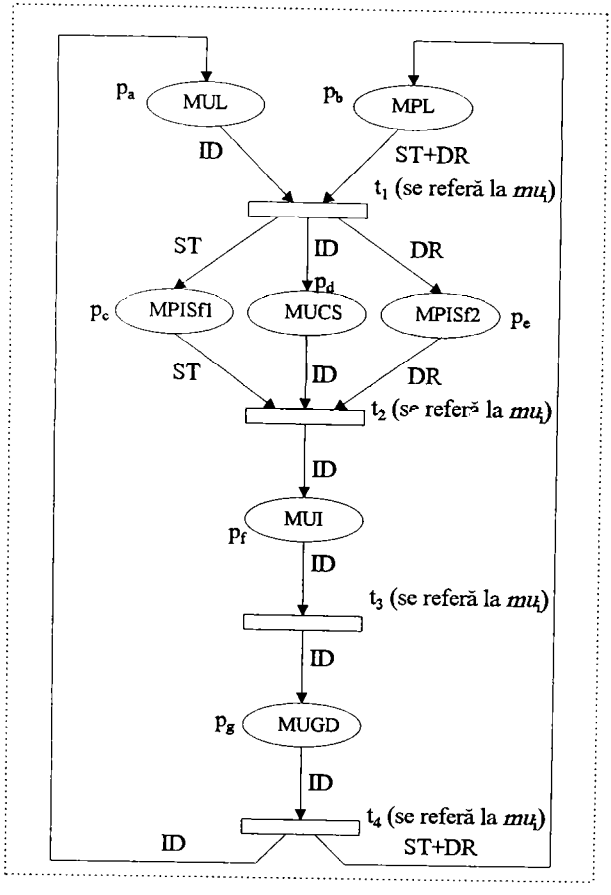


Fig. 4.6_3. Rețeaua Petri colorată corespunzătoare subsistemului *"n" mașini-unelte~ "n" manipulatoare.*

Pozițiile și tranzițiile acestei rețele au următoarele semnificații:

- poziția p_a modelează "*mașinile-unelte libere*, MUL" (*MUL: Mașini Unelte Libere*);
- poziția p_b modelează "*manipulatoarele libere*, MPL" (*MPL: ManiPulatoare Libere*);
- poziția p_c modelează "*manipulatoarele încărcate cu semifabricate de tipul Sf1*, MPÎSf1" (*MPÎSf1: ManiPulatoare Încărcate cu Semifabricate de tipul 1*);
- poziția p_d modelează "*mașinile-unelte curățate și spălate*, MUCS" (*MUCS: Mașini Unelte Curățate și Spălate*);
- poziția p_e modelează "*manipulatoarele încărcate cu semifabricate de tipul Sf2*, MPÎSf2" (*MPÎSf2: ManiPulatoare Încărcate cu Semifabricate de tipul 2*);
- poziția p_f modelează "*mașinile-unelte încărcate*, MUÎ" (*MUÎ: Mașini Unelte Încărcate*);

- poziția p_g modelează "*mașinile-unelte gata de descărcare*, MUGD" (*MUGD: Mașini Unelte Gata de Descărcare*);
- tranziția t_1 corespunde evenimentului "*pregătire operare mașină-unealtă* mu_i";
- tranziția t_2 corespunde evenimentului "*încărcare mașină-unealtă* mu_i";
- tranziția t_3 corespunde evenimentului "*operare pe mașina-unealtă* mu_i";
- tranziția t_4 corespunde evenimentului "*evacuare produs finit din mașina-unealtă* mu_i";

Evident, pozițiile p_a, p_d, p_f, și p_g pot avea drept marcaje, la un moment, oricare dintre părțile mulțimii MU, după cum pozițiile p_b, p_c, și p_e pot avea drept marcaje oricare dintre părțile mulțimii MP.

Pentru ca o tranziție t_j să se execute referitor la o mașină-unealtă mu_i, este necesar ca marcajul pozițiilor din amontele ei direct să cuprindă "*culorile*" date de funcțiile asociate arcelor ce duc de la aceste poziții la respectiva tranziție, când argumentul lor este mu_i.

De exemplu, pentru a se executa tranziția t_1 referitor la mașina-unealtă mu_4, este necesar ca marcajul lui p_a să conțină "*culoarea*":

$$ID(mu_4) = mu_4 \tag{4.6-10}$$

iar marcajul lui p_b -"*culorile*":

$$ST(mu_4) + DR(mu_4) = mp_3 + mp_4 = \{mp_3,\ mp_4\} \tag{4.6-11}$$

Notă:

Operatorul "+" introdus prin figura 4.6_3 are ca acțiune operația de reuniune a entităților pe care le găsește pe post de operanzi. Mai jos, se va folosi, de asemenea, operatorul "–", având ca acțiune operația de diferență a entităților pe care la găsește pe post de operanzi.

Matricea C a rețelei Petri din figura 4.6-3 este:

$$C = \begin{array}{c|cccc|c} & t_1 & t_2 & t_3 & t_4 & \\ \hline & -ID & 0 & 0 & ID & p_a \\ & -(ST+DR) & 0 & 0 & (ST+DR) & p_b \\ & ST & -ST & 0 & 0 & p_c \\ & ID & -ID & 0 & 0 & p_d \\ & DR & -DR & 0 & 0 & p_e \\ & 0 & ID & -ID & 0 & p_f \\ & 0 & 0 & ID & -ID & p_g \end{array} \tag{4.6-12}$$

Cu ajutorul ei, ținând seamă de marcajul inițial:

$$M_0 = \begin{Vmatrix} MU \\ MP \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \tag{4.6-13}$$

se va obține:

$$1).\ M_0 \Rightarrow \begin{cases} 1.1).\ executie\ t_1(mu_1) \\ 1.2).\ executie\ t_1(mu_2) \\ \vdots \end{cases} \tag{4.6-14}$$

$$1.1).\ executie\ t_1(mu_1) \Rightarrow M^-_{t_1(mu_1)} = M_0 \tag{4.6-15}$$

$$\Rightarrow M^+_{t_1(mu_1)} = M^-_{t_1(mu_1)} + col_{t_1}(C)(mu_1) =$$

$$= \begin{Vmatrix} MU \\ MP \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} -ID(mu_1) \\ -(ST(mu_1) + DR(mu_1)) \\ ST(mu_1) \\ ID(mu_1) \\ DR(mu_1) \\ 0 \\ 0 \end{Vmatrix} = \begin{Vmatrix} MU - mu_1 \\ MP - mp_0 - mp_1 \\ mp_0 \\ mu_1 \\ mp_1 \\ 0 \\ 0 \end{Vmatrix} \tag{4.6-16}$$

$$2).\ M^+_{t_1(mu_1)} \Rightarrow \begin{cases} 2.1).\ executie\ t_2(mu_1) \\ 2.2).\ executie\ t_1(mu_3) \\ \vdots \end{cases} \tag{4.6-17}$$

$$2.1).\ executie\ t_2(mu_1) \Rightarrow M^-_{t_2(mu_1)} = M^+_{t_1(mu_1)} \tag{4.6-18}$$

$$\Rightarrow M^+_{t_2(mu_1)} = M^-_{t_2(mu_1)} + col_{t_2}(C)(mu_1) =$$

$$= \begin{Vmatrix} MU - mu_1 \\ MP - mp_n - mp_1 \\ mp_n \\ mu_1 \\ mp_1 \\ 0 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ -ST(mu_1) \\ -ID(mu_1) \\ -DR(mu_1) \\ ID(mu_1) \\ 0 \end{Vmatrix} = \begin{Vmatrix} MU - mu_1 \\ MP - mp_n - mp_1 \\ 0 \\ 0 \\ 0 \\ mu_1 \\ 0 \end{Vmatrix} \tag{4.6-19}$$

$$3).\ M^+_{t_2(mu_1)} \Rightarrow \begin{cases} 3.1).\ executie\ t_3(mu_1) \\ 3.2).\ executie\ t_1(mu_3) \\ \vdots \end{cases} \tag{4.6-20}$$

$$3.1).\ executie\ t_3(mu_1) \Rightarrow M^-_{t_3(mu_1)} = M^+_{t_2(mu_1)} \tag{4.6-21}$$

$$\Rightarrow M^+_{t_3(mu_1)} = M^-_{t_3(mu_1)} + col_{t_3}(mu_1) =$$

$$= \begin{Vmatrix} MU - mu_1 \\ MP - mp_n - mp_1 \\ 0 \\ 0 \\ 0 \\ mu_1 \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -ID(mu_1) \\ ID(mu_1) \end{Vmatrix} = \begin{Vmatrix} MU - mu_1 \\ MP - mp_n - mp_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ mu_1 \end{Vmatrix} \tag{4.6-22}$$

$$4).\ M^+_{t_3(mu_1)} \Rightarrow \begin{cases} 4.1).\ executie\ t_4(mu_1) \\ 4.2).\ executie\ t_1(mu_3) \\ \vdots \end{cases} \tag{4.6-23}$$

$$4.1).\ executie\ t_4(mu_1) \Rightarrow M^-_{t_4(mu_1)} = M^+_{t_3(mu_1)} \tag{4.6-24}$$

$$\Rightarrow M^+_{t_4(mu_1)} = M^-_{t_4(mu_1)} + col_{t_4}(C)(mu_1) =$$

$$= \begin{Vmatrix} MU - mu_1 \\ MP - mp_n - mp_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ mu_1 \end{Vmatrix} + \begin{Vmatrix} ID(mu_1) \\ ST(mu_1) + DR(mu_1) \\ 0 \\ 0 \\ 0 \\ 0 \\ -ID(mu_1) \end{Vmatrix} = \begin{Vmatrix} MU \\ MP \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} = M_0 \qquad (4.6\text{-}25)$$

5). *ramura* 4.2).

$$executie\ t_1(mu_3) \Rightarrow M^-_{t_1(mu_3)} = M^+_{t_3(mu_1)} \qquad (4.6\text{-}26)$$

$$\Rightarrow M^+_{t_1(mu_3)} = M^-_{t_1(mu_3)} + col_{t_1}(C)(mu_3) =$$

$$= \begin{Vmatrix} MU - mu_1 \\ MP - mp_n - mp_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ mu_1 \end{Vmatrix} + \begin{Vmatrix} -ID(mu_3) \\ -(ST(mu_3) + DR(mu_3)) \\ ST(mu_3) \\ ID(mu_3) \\ DR(mu_3) \\ 0 \\ 0 \end{Vmatrix} =$$

$$= \begin{Vmatrix} MU - mu_1 - mu_3 \\ MP - mp_n - mp_1 - mp_2 - mp_1 \\ mp_2 \\ mu_3 \\ mp_3 \\ 0 \\ mu_1 \end{Vmatrix} \qquad (4.6\text{-}27)$$

$$6).\ M^+_{t_1(mu_3)} \Rightarrow \begin{cases} 6.1).\ executie\ t_4(mu_1) \\ 6.2).\ executie\ t_2(mu_3) \\ 6.3).\ executie\ t_1(mu_5) \\ \vdots \end{cases} \qquad (4.6\text{-}28)$$

7). *s.a.m.d.*

Rețeaua Petri colorată din figura 4.6_3 poate fi analizată, de asemenea, prin metoda invarianților. Invarianții se obțin rezolvând sistemul dat de relația (5.1-14), dacă se înlocuiește operația "$\cdot$" cu operația "$\circ$".

Rezultă:

$$\left\| x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \right\| \circ$$

$$\circ \begin{Vmatrix} -ID & 0 & 0 & ID \\ -(ST+DR) & 0 & 0 & (ST+DR) \\ ST & -ST & 0 & 0 \\ ID & -ID & 0 & 0 \\ DR & -DR & 0 & 0 \\ 0 & ID & -ID & 0 \\ 0 & 0 & ID & -ID \end{Vmatrix} =$$

$$= \left\| \begin{array}{cccc} 0 & 0 & 0 & 0 \end{array} \right\| \tag{4.6-29}$$

unde: $x_1, x_2, \ldots, x_7$ sunt funcții.

Deci:

$$\begin{cases} -x_1 \circ (ID) - x_2 \circ (ST) - x_2 \circ (DR) + x_3 \circ (ST) + x_4 \circ (ID) + x_5 \circ (DR) = 0 \\ -x_3 \circ (ST) - x_4 \circ (ID) - x_5 \circ (DR) + x_6 \circ (ID) = 0 \\ -x_6 \circ (ID) + x_7 \circ (ID) = 0 \\ x_1 \circ (ID) + x_2 \circ (ST) + x_2 \circ (DR) - x_7 \circ (ID) = 0 \end{cases} \tag{4.6-30}$$

de unde se obțin, de exemplu:

$$I_1^T = \left\| ID \quad 0 \quad 0 \quad ID \quad 0 \quad ID \quad ID \right\| \tag{4.6-31}$$

$$I_2^T = \left\| 0 \quad ID \quad ID \quad 0 \quad ID \quad 0 \quad 0 \right\| \tag{4.6-32}$$

Considerând două marcaje, M_0 și M, unde:

$$M = \begin{Vmatrix} MUL \\ MPL \\ MPISf1 \\ MUCS \\ MPISf2 \\ MUI \\ MUGD \end{Vmatrix} \tag{4.6-33}$$

și aplicând proprietatea definitorie a invarianților ($I^T \circ M = cons\tan t$) pentru instanțele I_1^T și I_2^T, se obține:

$$\begin{Vmatrix} ID & 0 & 0 & ID & 0 & ID & ID \end{Vmatrix} \circ \begin{Vmatrix} MUL \\ MPL \\ MPISf1 \\ MUCS \\ MPISf2 \\ MUI \\ MUGD \end{Vmatrix} =$$

$$= \begin{Vmatrix} ID & 0 & 0 & ID & 0 & ID & ID \end{Vmatrix} \circ \begin{Vmatrix} MU \\ MP \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \tag{4.6-34}$$

respectiv:

$$\begin{Vmatrix} 0 & ID & ID & 0 & ID & 0 & 0 \end{Vmatrix} \circ \begin{Vmatrix} MUL \\ MPL \\ MPISf1 \\ MUCS \\ MPISf2 \\ MUI \\ MUGD \end{Vmatrix} =$$

$$= \begin{Vmatrix} 0 & ID & ID & 0 & ID & 0 & 0 \end{Vmatrix} \circ \begin{Vmatrix} MU \\ MP \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Vmatrix} \tag{4.6-35}$$

adică:

$$MUL + MUCS + MUI + MUGD = MU \tag{4.6-36}$$

respectiv:

$$MPL + MPISf1 + MPISf2 = MP \tag{4.6-37}$$

conform așteptărilor.

Calculul invarianților în cazul rețelelor Petri colorate este tratat în [BRAM'89]. Metoda prezentată aici -în [BRAM'89]- nu conduce întotdeauna la toți invarianții semnificativi, dar, de cele mai multe ori, ea oferă posibilitatea determinării cel puțin a unui număr dintre aceștia, prin simpla inspectare a unor relații.

4.7. Modelarea liniilor flexibile cu rețele Petri continue

4.7.1. Preliminarii

În cazul unor sisteme complexe, cum sunt liniile flexibile de fabricație, numărul de marcaje accesibile rețelelor Petri discrete poate deveni atât de ridicat, încât să îngreuneze până la cote inacceptabile utilizarea lor. De aceea, într-un astfel de caz, sunt de preferat rețelele Petri continue [ZERH'92]. Evident, ele procedează la o aproximare; pentru că reflectă în continuu, prin mărimi medii, evoluția unor sisteme care, prin natura lor, sunt cu evenimente discrete. Se garantează, însă, că aproximarea nu scapă ceea ce este esențial în sistemele modelate [ALLA'88a] [ALLA'88b].

Două mărimi sunt definitorii pentru rețelele Petri continue: un marcaj continuu, exprimat printr-un număr real, asociat fiecărei poziții, respectiv o viteză asociată fiecărei tranziții,

reprezentând frecvența maximală de execuție a tranziției respective (transpare, de aici, filiația rețelelor Petri continue din rețelele Petri temporizate) [ZERH'92].

Dacă se introduc notațiile:

- U_j -pentru viteza maximă de execuție a tranziției t_j,
- v_j-pentru viteza curentă de execuție a tranziției t_j,
- $amont\ (t_j)$-pentru mulțimea pozițiilor din amontele tranziției t_j,
- $m\ (\cdot)$ -pentru marcajul poziției "·",

atunci se poate scrie că:

$$v_j = U_j \cdot \min_{p \in amont\,(t_j)} (m(p)) \tag{4.7.1-1}$$

Evoluția în timp a marcajului unei rețele este dat de relația matricială [ZERH'92]:

$$\frac{d}{dt} M = C \cdot v \tag{4.7.1-2}$$

unde:

- M-este vectorul marcajelor: $M^T = \| m_1 \quad m_2 \quad \dots \quad m_n \|$,
- C -este matricea de incidență,
- v este vectorul vitezelor de execuție a tranzițiilor: $v^T = \| v_1 \quad v_2 \qquad v_n \|$.

Sistemul de ecuații dat de relația (4.7-2) oferă, pentru fiecare poziție, variația marcajului în timp, ca rezultat a ceea ce intră și a ceea ce iese din ea. Evident, se presupune că vitezele de execuție a tranzițiilor sunt, toate, constante.

4.7.2. Modelarea unei stații de lucru

Se consideră stație de lucru ansamblul compus dintr-o mașină-unealtă și un stoc în amonte imediat de aceasta [ZERH'90a] [ZERH'90b]:

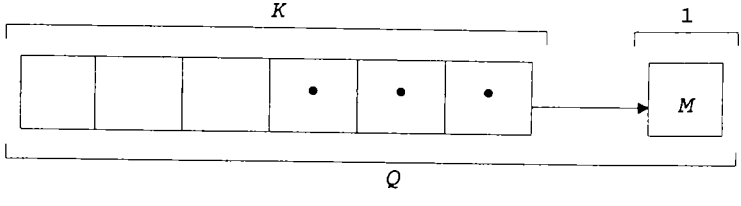


Fig. 4.7.2_1. Stația de lucru.

Parametrii stației -a se vedea figura 4.7.2_1- sunt:

- d : timpul de serviciu al stației, socotit pentru o piesă,
- K : capacitatea stocului din amonte,
- Q: capacitatea stației, cumulând capacitățile stocului și mașinii.

Rezultă, deci, că:

$$Q = K + 1 \tag{4.7.2-1}$$

Notă:

Se presupune că, întotdeauna, $K \geq 1$, *ceea ce implică* $Q \geq 2$. (4.7.2-2)

Rețeaua Petri continuă de modelare a unei stații "*i*" este cea din figura 4.7.2_2.

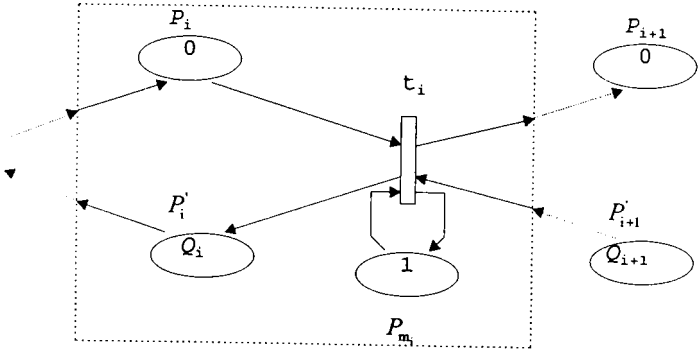


Fig. 4.7.2_2. Rețeaua Petri continuă de modelare a unei *stații de lucru*.

Notațiile folosite au următoarele semnificații:

- P_i reprezintă piesele prezente într-o stație (în stoc și pe mașină),
- P_i' reprezintă locurile disponibile într-o stație (în stoc și pe mașină),
- P_{m_i} reprezintă mașina-unealtă,
- t_i reprezintă operația efectuată în stație,
- d_i reprezintă temporizarea aferentă execuției tranziției t_i.

Așadar, se va avea [ZERH'92]:

$$U_i = 1/d_i \tag{4.7.2-3}$$

4.7.3. Modelarea liniilor de fabricație deschise

O linie de fabricație deschisă este un ansamblu de stații, în care fiecare stație efectuează un set de operații consecutive (reprezentând o fază a procesului de prelucrare) asupra fiecărei piese, piesele fiind nepaletizate, și care, în final, prin evacuarea produsului finit, se descarcă cu o unitate. Rezultă, deci, că într-o linie de fabricație deschisă, numărul pieselor nu este o constantă.

Fie linia de "*n+1*" stații reprezentată în figura 4.7.3_1 [DAVI'87a] [DAVI'87b].

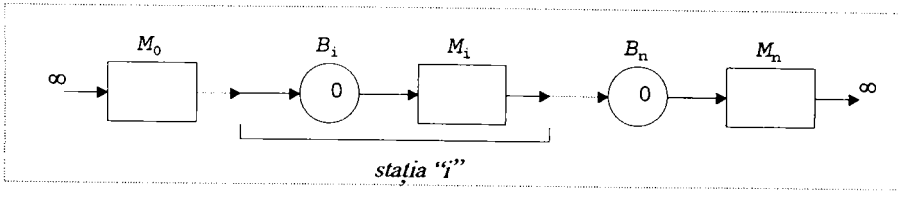


Fig. 4.7.3_1. Linie de fabricație deschisă din "*n+1*" stații.

Rețeaua Petri continuă corespunzătoare liniei din figura 4.7.3_1 este cea reprezentată în figura 4.7.3_2, referitor la care este de precizat că, în discordanță cu figura 4.7.2_2, nu mai surprinde și mașinile-unelte (pozițiile P_{m_i} lipsesc), presupunându-le subînțelese [ZERH'92].

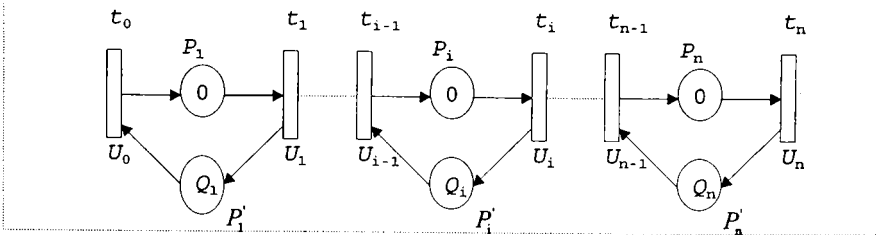


Fig. 4.7.3_2. Rețeaua Petri continuă corespunzătoare unei linii de fabricație deschisă.

U_0 este viteza de alimentare a liniei -debitul de intrare-, alimentare presupusă a se face dintr-un stoc de semifabricate infinit, așa cum figura 4.7.3_1 o sugerează.

Viteza de execuție a unei tranziții t_i, cu $i = 1...n-1$, depinde de marcajul pozițiilor din amontele ei, reprezentând, una (P_i), colecția de piese din stația precedentă, iar cealaltă (P_{i+1}'), capacitatea de stocare disponibilă în stația următoare:

$$v_i = U_i \cdot \min\ (m_i,\ m_{i+1}',\ 1) \qquad (4.7.3\text{-}1)$$

unde:

- m_i este marcajul curent al poziției P_i,
- m'_{i+1} este marcajul curent al poziției P'_{i+1}.

Evident:

$$m_i + m'_{i+1} = Q_i \tag{4.7.3-2}$$

Două stații sunt particulare în linia în discuție. Este vorba despre prima și despre ultima.

Prima stație are -așa cum a fost menționat- un stoc în amonte infinit, deci, niciodată vid. Va rezulta [ZERH'92]:

$$v_0 = U_0 \cdot \min\ (\infty, m'_1,\ 1) = U \cdot \min\ (m'_1,\ 1) \tag{4.7.3-3}$$

Ultima stație are un stoc în aval de capacitate infinită, deci, care nu se umple niciodată.

Va rezulta:

$$v_n = U_n \cdot \min\ (m_n,\ \infty,\ 1) = U_n \cdot \min\ (m_n,\ 1) \tag{4.7.3-4}$$

Evoluția liniei este dată de sistemul de ecuații exprimat, generic, prin:

$$\frac{d}{dt} m_i = \dot{m}_i = v_{i-1} - v_i,\ cu\ i = 1..n \tag{4.7.3-5}$$

sau, matricial, prin:

$$\dot{M} = \frac{d}{dt} M = C_n \cdot v \tag{4.7.3-6}$$

unde:

C_n-este submatricea din matricea de incidență care corespunde pozițiilor $p_1,\ldots,p_n$. Evident, se va avea:

$$C = \begin{array}{c} \begin{array}{cccccc} t_0 & t_1 & t_2 & \cdots & t_{n-1} & t_n \end{array} \\ \left\| \begin{array}{cccccc} 1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{array} \right\| \begin{array}{c} p_1 \\ p_2 \\ \vdots \\ p_n \\ p'_1 \\ p'_2 \\ \vdots \\ p'_n \end{array} \end{array} \tag{4.7.3-7}$$

și:

$$C_n = \begin{array}{c} \begin{matrix} t_0 & t_1 & t_2 & \cdots & t_{n-1} & t_n \end{matrix} \\ \left\| \begin{matrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 \end{matrix} \right\| \end{array} \begin{matrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{matrix} \qquad (4.7.3\text{-}8)$$

Relațiile (4.7.3-5)...(4.7.3-8) surprind comportamentul liniei atât în regim staționar, cât și în regim tranzitoriu.

Marcajul staționar al pozițiilor care modelează stocurile corespunde valorii medii a cumulului de piese în stoc. Este evidentă următoarea proprietate [ZERH'92]:

Proprietate:

Toate stocurile din amonte de stația cea mai lentă, al cărei indice îl notăm cu "s" -în cazul în care viteza minimă este avută de mai multe stații, atunci statutul de stație cea mai lentă este oferit primei dintre acestea, pe ordinea fluxului- ating saturația la nivelele:

$$m_i = Q_i - U_s / U_{i-1},\ unde\ i = 1...s \qquad (4.7.3\text{-}9)$$

în timp ce toate stocurile din aval de stația cea mai lentă sunt încărcate, în medie, la nivelele subunitare:

$$m_i = U_s / U_i,\ cu\ i = s+1...n \qquad (4.7.3\text{-}10)$$

Se are:

$$m_i = Q_i - U_s / U_{i-1},\ pentru\ i = 1...s \qquad (4.7.3\text{-}11)$$

și:

$$m_i = U_s / U_i,\ pentru\ i = s+1...n \qquad (4.7.3\text{-}12)$$

Într-adevăr, faptul că în regim staționar:

$$\dot{M} = 0 \qquad (4.7.3\text{-}13)$$

conduce la:

$$v_{i-1} = v_i = 0,\ \forall i = 1...n \qquad (4.7.3\text{-}14)$$

echivalent cu:

$$v_0 = v_1 = \ldots = v_n \overset{def}{=} U \tag{4.7.3-15}$$

Evident:

$$U \le U_s \tag{4.7.3-16}$$

Se va avea:

$$v_0 = U_0 \cdot \min\ (m_1', 1) \tag{4.7.3-17}$$

$$U = U_0 \cdot \min\ (m_1', 1) \tag{4.7.3-18}$$

Din:

$$U < U_0 \tag{4.7.3-19}$$

rezultă:

$$\min\ (m_1', 1) < 1 \Leftrightarrow m_1' < 1 \tag{4.7.3-20}$$

Așadar:

$$m_1' = U / U_0 \tag{4.7.3-21}$$

Dar:

$$m_1 + m_1' = Q_1 \tag{4.7.3-22}$$

Rezultă:

$$m_1 = Q_1 - m_1' = Q_1 - U / U_0 > 1 \tag{4.7.3-23}$$

Apoi:

$$v_1 = U_1 \cdot \min\ (m_1, m_2', 1) \tag{4.7.3-24}$$

ceea ce, ținând seamă de (4.7.3-23), conduce la:

$$v_1 = U_1 \cdot \min\ (m_2', 1) \tag{4.7.3-25}$$

Prin același raționament ca mai sus, se obține:

$$m_2' = U / U_1 < 1 \tag{4.7.3-26}$$

și:

$$m_2 = Q_2 - m_2' = Q_2 - U / U_1 > 1 \tag{4.7.3-27}$$

iar, în continuare:

$$m_i' = U / U_{i-1} < 1 \tag{4.7.3-28}$$

$$m_i = Q_i - m_i' = Q_i - U / U_{i-1} > 1 \tag{4.7.3-29}$$

pentru $i = 3...s$.

Pe de altă parte:

$$v_n = U_n \cdot \min\ (m_n,\ 1) \tag{4.7.3-30}$$

$$U = U_n \cdot \min\ (m_n,\ 1) \tag{4.7.3-31}$$

Din:

$$U < U_n \tag{4.7.3-32}$$

rezultă:

$$\min\ (m_n,\ 1) < 1 \Leftrightarrow m_n < 1 \tag{4.7.3-33}$$

Așadar:

$$m_n = U / U_n \tag{4.7.3-34}$$

și, implicit:

$$m_n' = Q_n - m_n = Q_n - U / U_n > 1 \tag{4.7.3-35}$$

Apoi:

$$v_{n-1} = U_{n-1} \cdot \min\ (m_{n-1},\ m_n',\ 1) \tag{4.7.3-36}$$

ceea ce, ținând seamă de (4.7.3-35), conduce la:

$$v_{n-1} = U_{n-1} \cdot \min\ (m_{n-1},\ 1) \tag{4.7.3-37}$$

Prin același raționament ca mai sus, se obține:

$$m_{n-1} = U / U_{n-1} < 1 \tag{4.7.3-38}$$

și:

$$m'_{n-1} = Q_{n-1} - m_{n-1} = Q_{n-1} - U / U_{n-1} > 1 \qquad (4.7.3\text{-}39)$$

iar, în continuare:

$$m_i = U / U_i < 1 \qquad (4.7.3\text{-}40)$$

$$m'_i = Q_i - m_i = Q_i - U / U_i > 1 \qquad (4.7.3\text{-}41)$$

pentru $i = n\text{-}2...s+1$.

Având în vedere că:

$$v_s = U_s \cdot \min\ (m_s,\ m'_{s+1},\ 1) \qquad (4.7.3\text{-}42)$$

și ținând seamă de (4.7.3-23), (4.7.3-27), (4.7.3-29), (4.7.3-35), (4.7.3-39), (4.7.3-41), se obține că:

$$v_s = U_s \qquad (4.7.3\text{-}43)$$

și, mai departe, că:

$$v_0 = ... = v_s = ... = v_n = U = U_s \qquad (4.7.3\text{-}44)$$

În consecință, se va avea:

$$m_i = Q_i - U_s / U_{i-1},\ pentru\ i = 1...s \qquad (4.7.3\text{-}45)$$

$$m_i = U_s / U_i,\ pentru\ i = s+1...n \qquad (4.7.3\text{-}46)$$

ceea ce ne-am propus să demonstrăm.

Pentru regimul tranzitoriu, se demonstrează următoarea [ZERH'92]:

Proprietate:

Toate stocurile au o evoluție crescătoare în timp.

Corolar:

Toate stocurile converg aperiodic către nivelele de regim staționar.

Spre exemplificare, vom aplica concluziile de mai sus rețelei de modelare a unei linii flexibile deschisă cu 3 stații, reprezentată în figura 4.7.3_3 [ZERH'92].

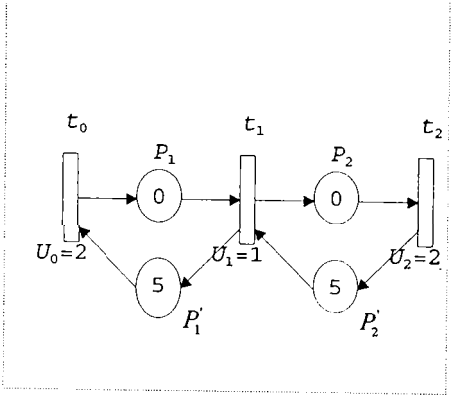


Fig. 4.7.3_3. Rețea Petri continuă corespunzătoare unei linii de fabricație deschisă -caz de studiu.

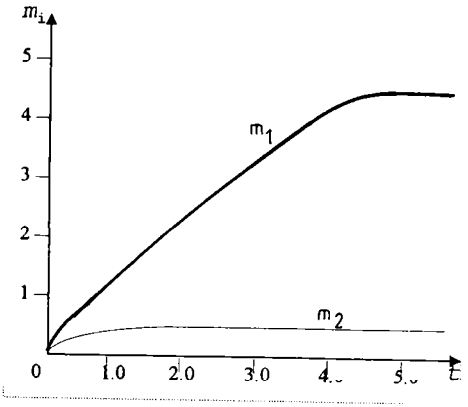


Fig. 4.7.3_4. Evoluția în timp a marcajelor m_1 și m_2 ale rețelei Petri continue din figura 4.7.3_3.

Evident:

$s = 1$
$U_s = U_1 = 1$
$v_0 = v_1 = v_2 = U_s = 1$
$m_1 = Q_1 - U / U_0 = 5 - 1/2 = 4.5$
$m_2 = U / U_2 = 1/2 = 0.5$

În figura 4.7.3_4, se dă o reprezentare a evoluției în timp a marcajelor m_1 și m_2.

4.7.4. Modelarea liniilor de fabricație închise

O linie de fabricație închisă este un ansamblu de stații în care fiecare stație efectuează un set de operații consecutive (reprezentând o fază a procesului de prelucrare) asupra fiecărei piese, piesele fiind paletizate, și care, în final, prin evacuarea produsului finit, nu se descarcă, ci își recirculă paleta devenită liberă (realimentând-o). Rezultă, deci, că într-o linie de fabricație închisă, numărul pieselor este o constantă: numărul paletelor [DAVI'87a] [DAVI'87b] [ZERH'92].

Fie linia de "n" stații reprezentată în figura 4.7.4_1.

Rețeaua Petri continuă corespunzătoare liniei din figura 4.7.4_1 este cea reprezentată în figura 4.7.4_2, referitor la care este de precizat că, la fel ca și figura 4.7.3_2, nu surprinde mașinile-unelte (pozițiile P_{m_i} lipsesc), presupunându-le subînțelese [ZERH'92].

Așa cum figura 4.7.4_2 sugerează, în starea inițială, toate paletele, în număr de "m", sunt la intrarea în linie, adică, în stocul primei stații, în timp ce stocurile tuturor celorlalte stații sunt vide.

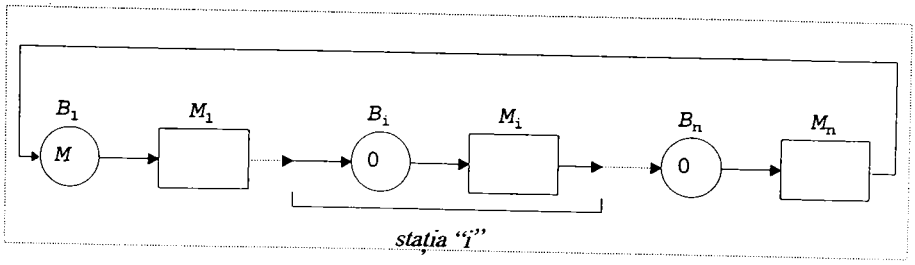


Fig. 4.7.4_1. Linie de fabricație închisă cu "*n*" stații.

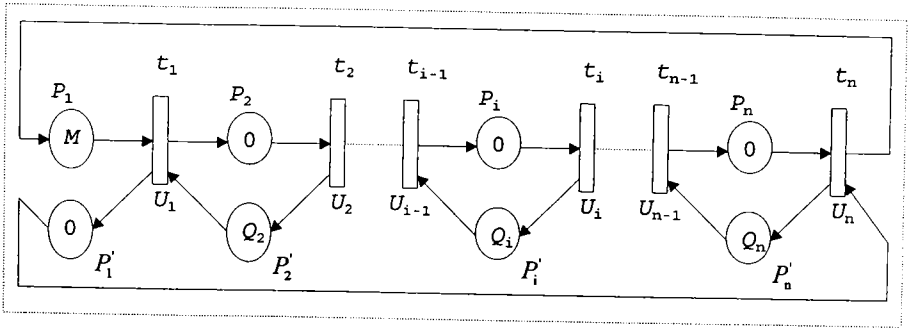


Fig. 4.7.4_2. Rețeaua Petri continuă corespunzătoare unei linii de fabricație închisă.

Dacă se admite ipoteza că stocurile tuturor stațiilor sunt suficient de mari în raport cu dinamica liniei, astfel încât niciunul dintre ele să nu ajungă să se umple, atunci rețeaua Petri continuă de modelare a unei linii de fabricație închisă se poate simplifica, prin eliminarea din ea a pozițiilor corespunzătoare capacităților de stocare disponibile (P_i), ajungând așa cum se arată în figura 4.7.4_3.

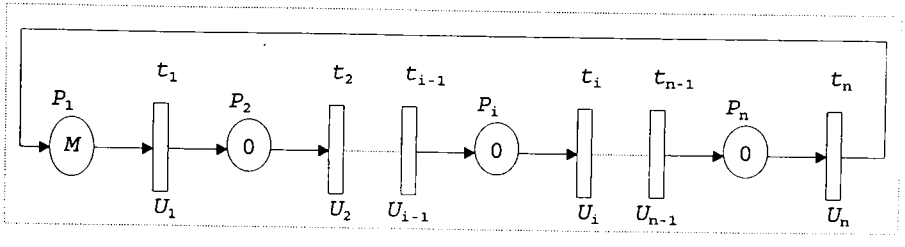


Fig. 4.7.4_3. Rețeaua Petri continuă corespunzătoare unei linii de fabricație închisă, în ipoteza că $m \le Q_i$, $\forall i = 1...n$.

Este evident faptul că, de la reprezentarea unei linii deschise se poate ajunge la reprezentarea unei linii închise -a se vedea figurile 4.7.3_1, respectiv 4.7.4_1-, prin simpla legare a ieșirii la intrare. De asemenea, rețeaua de modelare a liniei închise se poate obține din rețeaua

liniei deschise -a se vedea figurile 4.7.3_2, respectiv 4.7.4_2-, prin confundarea tranziției t_0 cu tranziția t_n.

În vederea determinării valorilor marcajelor în regim staționar, considerăm o permutare convenabilă a rețelei din figura 4.7.4_2, reprezentată în figura 4.7.4_4. Și de această dată, s-a folosit litera "*s*" ca indice referitor la stația cea mai lentă.

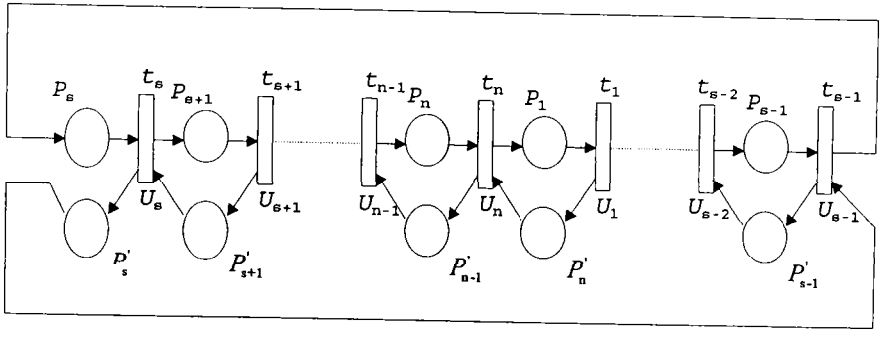


Fig. 4.7.4_4. O permutare a rețelei din figura 4.7.4_2.

Regimul staționar se caracterizează prin [ZERH'92]:

$$\dot{m}_i(t) = 0,\ i = 1...n \tag{4.7.4-1}$$

ceea ce înseamnă:

$$\begin{cases} \dot{m}_1(t) = v_n - v_1 = 0 \Rightarrow v_n = v_1 \\ \dot{m}_2(t) = v_1 - v_2 = 0 \Rightarrow v_1 = v_2 \\ \vdots \\ \dot{m}_n(t) = v_{n-1} - v_n = 0 \Rightarrow v_{n-1} = v_n \end{cases} \tag{4.7.4-2}$$

adică:

$$v_1 = v_2 = ... = v_n \overset{def}{=} U \tag{4.7.4-3}$$

Evident:

$$U \le U_s \tag{4.7.4-4}$$

Regimul staționar va fi considerat în două cazuri:

1) cazul atingerii saturației, caracterizat prin aceea că linia lucrează cu viteza stației cea mai lentă, adică:

$$U = U_s \tag{4.7.4-5}$$

2) cazul neatingerii saturației, caracterizat prin aceea că linia lucrează cu o viteză mai mică decât viteza stației cea mai lentă, adică:

$$U < U_s \tag{4.7.4-6}$$

Pentru ambele cazuri, vom presupune că niciunul dintre stocuri nu va ajunge să fie plin (de fapt, este suficient să se presupună că stocul stației cea mai lentă nu ajunge să fie plin, neumplerea celorlalte apărând ca o consecință). Presupunerea este echivalentă cu a considera că:

$$m_i' = \infty,\ \forall i = 1...n \tag{4.7.4-7}$$

În primul caz, se are:

$$v_{s-1} = U_{s-1} \cdot \min\ (m_{s-1},\ m_s',\ 1) = U \tag{4.7.4-8}$$

Din:

$$U \le U_{s-1} \tag{4.7.4-9}$$

rezultă:

$$\min\ (m_{s-1},\ m_s',\ 1) = m_{s-1} \Leftrightarrow m_{s-1} < 1 \tag{4.7.4-10}$$

Așadar:

$$m_{s-1} = U / U_{s-1} < 1 \tag{4.7.4-11}$$

În aceeași manieră, se ajunge la concluzia că:

$$m_i = U / U_i < 1,\ \forall i = 1...n,\ i \neq s \tag{4.7.4-12}$$

Apoi, cum:

$$m_1 + m_2 + ... + m_n = m \tag{4.7.4-13}$$

rezultă că:

$$m_s = m - \sum_{\substack{j=1..n \\ j \neq s}} U / U_j \tag{4.7.4-14}$$

iar din faptul că $m \geq n$, se obține că:

$$m_s \geq 1 \tag{4.7.4-15}$$

Prin urmare, relația:

$$v_s = U_s \cdot \min\ (m_s,\ m'_{s+1},\ 1) = U \tag{4.7.4-16}$$

conduce la:

$$v_s = U_s = U \tag{4.7.4-17}$$

Așadar:

$$m_i = U_s / U_i,\ \forall i = 1..n,\ i \neq s \tag{4.7.4-18}$$

$$m_s = m - U_s \sum_{\substack{j=1..n \\ j \neq s}} 1/U_j \tag{4.7.4-19}$$

În al doilea caz, se are:

$$v_s = U \cdot \min\ (m_s,\ m'_{s+1},\ 1) = U \tag{4.7.4-20}$$

Din:

$$U < U_s \tag{4.7.4-21}$$

rezultă:

$$\min\ (m_s,\ m'_{s+1},\ 1) = m_s \Leftrightarrow m_s < 1 \tag{4.7.4-22}$$

Așadar:

$$m_s = U / U_s < 1 \tag{4.7.4-23}$$

Apoi:

$$v_{s-1} = U_{s-1} \cdot \min\ (m_{s-1},\ m'_s,\ 1) = U \tag{4.7.4-24}$$

și, ca mai sus, din:

$$U < U_{s-1} \tag{4.7.4-25}$$

rezultă:

$$\min\ (m_{s-1},\ m_s^{'},\ 1) = m_{s-1} \Leftrightarrow m_{s-1} < 1 \qquad (4.7.4\text{-}26)$$

Așadar:

$$m_{s-1} = U / U_{s-1} < 1 \qquad (4.7.4\text{-}27)$$

În aceeași manieră, se ajunge la concluzia că:

$$m_i = U / U_i < 1,\ \forall i = 1...n \qquad (4.7.4\text{-}28)$$

Dar:

$$\sum_{j=1...n} m_j = m \qquad (4.7.4\text{-}29)$$

Rezultă că:

$$\sum_{j=1...n} U / U_j = m \qquad (4.7.4\text{-}30)$$

de unde:

$$U = \frac{m}{\sum_{j=1...n} 1/U_j} \qquad (4.7.4\text{-}31)$$

și, mai departe:

$$m_i = \frac{m}{U_i \sum_{j=1...n} 1/U_j} \qquad (4.7.4\text{-}32)$$

Definiție:

Se definește drept marcaj critic al unei linii de fabricație închisă marcajul inițial de cea mai mică valoare al poziției ce modelează stocul primei stații, pentru care linia funcționează la saturație.

Notând marcajul critic cu m_c, vom avea:

$$m_c = m_s \cdot U_s \sum_{j=1}^{n} 1/U_j \Big|_{m_s = 1} = U_s \sum_{j=1}^{n} 1/U_j \qquad (4.7.4\text{-}33)$$

Așadar, dacă $m \geq m_c$, atunci linia funcționează la saturație, iar dacă $m < m_c$, atunci saturația nu este atinsă.

Pentru regimul tranzitoriu, sunt valabile următoarea proprietate și următoarele corolare:

Proprietate:

Toate stocurile diferite de cel al primei stații au o evoluție crescătoare în timp.

Corolar 1:

Stocul primei stații are o evoluție descrescătoare în timp.

Corolar 2:

Toate stocurile converg aperiodic către nivelele de regim staționar.

Spre exemplificare, se vor aplica concluziile de mai sus rețelei de modelare a unei linii flexibile închisă cu 4 stații, reprezentată în figura 4.7.4_5.

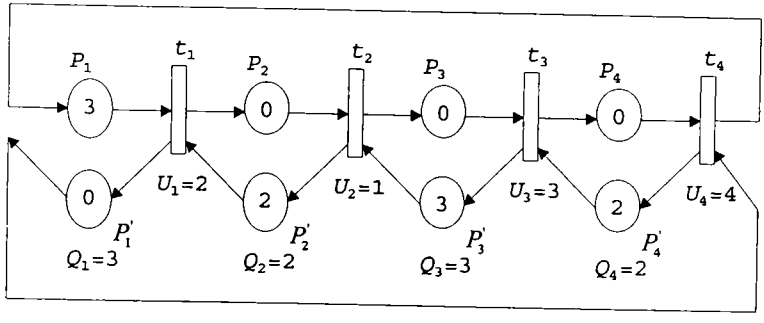


Fig. 4.7.4_5. Rețea Petri continuă corespunzătoare unei linii de fabricație închisă -caz de studiu.

Evident:

$s = 2$

$U_s = U_2 = 1$

Rezultă:

$m_c = 2.08$

ceea ce înseamnă că linia va funcționa la saturație ($m = 3 > m_c = 2.08$).

Se obțin:

$v_1 = v_2 = v_3 = v_4 = U_s = 1$

$m_1 = 1/2 = 0.5$

$m_2 = 23/12 = 1.92$

$m_3 = 1/3 = 0.33$

$m_4 = 1/4 = 0.25$

În figura 4.7.4_6, se dă o reprezentare a evoluției în timp a marcajelor $m_1,\ldots,m_4$ [ZERH'92].

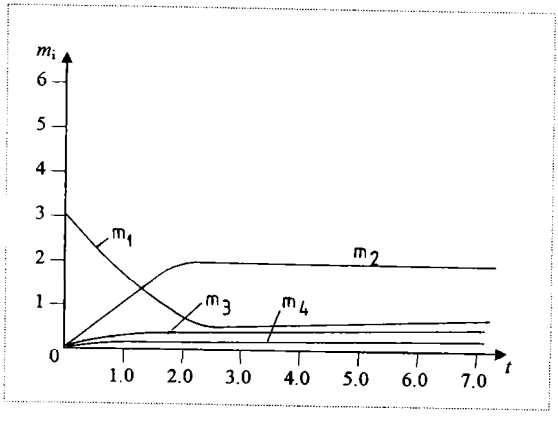


Fig. 4.7.4_6. Evoluția în timp a marcajelor $m_1,\ldots,m_4$ ale rețelei Petri continue din figura 4.7.4_5 .

5
ORDONANȚAREA OPERAȚIILOR TEHNOLOGICE ȘI SARCINILOR DE TRANSPORT ÎN SISTEMELE FLEXIBILE DE FABRICAȚIE

5.1. Aspecte introductive

După cum se cunoaște, ordonanțarea este acțiunea prin care, pentru un proces de fabricație dat și vizând un sistem de fabricație anume, se asigură distribuirea operațiilor tehnologice, respectiv a sarcinilor de transport, către celulele flexibile, respectiv către subsistemul de transport, astfel încât să fie respectate restricțiile de precedență și de compatibilitate impuse de procesul tehnologic și/sau de sistemul de fabricație, restricțiile de prelucrare/montaj specifice celulelor, respectiv restricțiile de transport, și, totodată, să fie maximizat gradul de utilizare a capabilităților resurselor [GRAV'81] [GUPT'89].

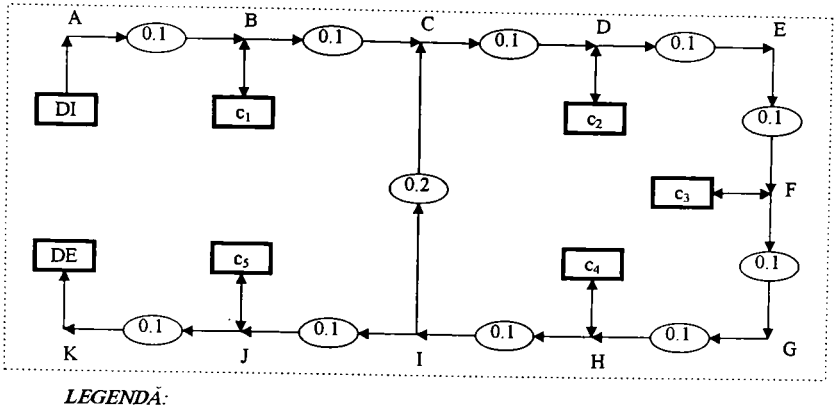
Din multitudinea aspectelor ce caracterizează un cuplu "*proces tehnologic-sistem de fabricație*", în vederea ordonanțării se iau în considerare, de regulă, doar următoarele [CARL'88] [BORA'89] :

1) Sistemul flexibil cuprinde un număr de celule flexibile, un depozit de piese de intrare, un depozit de piese de ieșire, și un subsistem de transport.
2) Fiecare celulă flexibilă poate executa o anumită gamă de operații tehnologice.
3) La nivelul fiecărei celule flexibile există un rezervor tampon pentru piesele ce intră în celulă și un rezervor tampon pentru piesele ce ies din celulă.
4) Există anumite posibilități de vehiculare a pieselor între celulele flexibile, respectiv de la depozitul de piese de la intrare la celule, și de la celule la depozitul de piese de la ieșire.
5) Există anumite posibilități de vehiculare de scule între celulele flexibile, respectiv de la un depozit central la celule, și invers.
6) *A*. Procesul de fabricație consistă în prelucrarea mai multor piese de același tip într-un anumit interval de timp.
   B. Procesul de fabricație consistă în prelucrarea mai multor piese de mai multe tipuri -care pot fi modele diferite ale aceluiași tip–, într-un anumit interval de timp.
7) Operațiile de executat trebuie să fie, între ele, într-o anumită relație de precedență.
8) Pentru fiecare operație, se cunosc celulele care o pot executa, durata execuției (incluzând și timpii necesari operațiilor auxiliare), și operațiile ce îi pot succede direct.
9) *A*. Sistemul de fabricație are un ritm fixat, ceea ce înseamnă că piesele avansează de la o celulă la alta doar la anumite momente de timp, prestabilite, în ideea unei ciclități.
   B. Sistemul de fabricație nu are un ritm fixat, ceea ce înseamnă că piesele avansează de la o celulă la alta în mod asincron, imediat ce o fază de prelucrare ia sfârșit.

Pentru realizarea ordonanțării, diverși algoritmi au fost imaginați [CARL'82a] [CARL'88]. Indiferent de principiile care stau la baza lor, ca și de performanțele pe care le pot atinge, acestor algoritmi li se cer, unanim, următoarele caracteristici:

1) Capacitate de adecvanță la un număr mare de procese de fabricație.
2) Capacitate de a trata diferite configurații de sisteme flexibile.
3) Capacitate de a trata un număr mare de subunități funcționale, de tipurile*): mașină-unealtă, robot industrial, celulă flexibilă, dispozitiv de transport, dispozitiv de stocare, dispozitiv de manipulare, etc.
4) Capacitate de a desemna, pentru execuția unei anumite operații tehnologice, o anumită celulă flexibilă, dintre mai multe echipotente, relativ la acea operație.
5) Capacitate de a stabili traseele de deplasare a pieselor aflate în fabricație între subunitățile tehnologice.
6) Capacitate de a trata stocarea în depozite tampon, cu considerarea restricțiilor privind capacitatea acestora.
7) Capacitate de a planifica transportul automat al unor anumite scule în cadrul sistemului.

Conform celor de mai sus, o vedere a unui algoritm de ordonanțare asupra unui sistem flexibil se prezintă, așa cum se exemplifică în figura 5.1_1.



LEGENDĂ:

DI: depozit de intrare; DE: depozit de ieșire; c_i: celula flexibilă "i"; x.y: durata operației de transport dintre punctele de alături notate cu litere majuscule.

Fig. 5.1_1. Ilustrarea vederii unui algoritm de ordonanțare asupra unui sistem de fabricație.

*) În cele ce urmează, se va aborda problema ordonanțării doar de la nivelul atelierului flexibil, deci cu referire exclusivă la celulele flexibile, respectiv la subsistemul de transport. Distribuția operațiilor în cadrul celulelor flexibile, către mașinile-unelte și roboții din componența lor, se face în aceeași manieră, dar, evident, ridică mai puține probleme.

Privitor la procesele tehnologice, vizibilitatea algoritmilor de ordonanțare se rezumă, maximal, la nivelul operațiilor sintetizate în tabelul din figura 5.1_2 [BORA'89].

NR. CRT.	OPERAȚIA	REMARCI
1.	Încărcare sculă în sistemul de transport.	Scula introdusă în sistemul de transport al sculelor poate proveni din depozitul central de scule sau dintr-un magazin de scule din cadrul unei celule flexibile.
2.	Transport sculă.	Scula este transportată cu sistemul de transport al sculelor către o celulă flexibilă sau către depozitul central de scule.
3.	Descărcare sculă din sistemul de transport al sculelor.	Scula eliminată din sistemul de transport al sculelor poate fi transportată în depozitul central de scule sau într-un magazin de scule din cadrul unei celule flexibile.
4.	Pornire celulă flexibilă.	Se pornesc componentele celulei flexibile și au loc procese de inițializare, de reglare, și de testare, în vederea începerii activității de fabricație.
5.	Încărcare piesă în sistemul de transport al pieselor.	Piesa introdusă în sistemul de transport al pieselor poate proveni din depozitul de piese de la intrare sau dintr-un rezervor tampon de ieșire al unei celule flexibile.
6.	Transport piesă.	Piesa este transportată, cu sistemul de transport al pieselor, către o celulă flexibilă sau către depozitul de piese de la ieșire.
7.	Descărcare piesă din sistemul de transport al pieselor.	Piesa eliminată din sistemul de transport al pieselor poate fi transportată în rezervorul tampon de intrare al unei celule flexibile sau în depozitul de piese de la ieșire.
8.	Încărcare piesă pe o mașină de lucru a unei celule flexibile.	Piesa este extrasă din rezervorul tampon de intrare, transferată, și apoi fixată, pe mașina de lucru din cadrul unei celule flexibile, care urmează să efectueze asupra ei o anumită fază de prelucrare.
9.	Prelucrare piesă pe o mașină de lucru a unei celule flexibile.	Asupra piesei se execută o fază tehnologică, constând în una sau mai multe operații.
10.	Descărcare piesă de pe o mașină de lucru a unei celule flexibile.	Piesa este extrasă de pe mașina de lucru pe care ea a fost în prelucrare și transferată în rezervorul tampon de ieșire.

Fig. 5.1_2. Operațiile proceselor tehnologice vizibile de la nivelul algoritmilor de ordonanțare.

Este de precizat că, în practică, operațiile 1, 2, și 3 intervin doar în situații speciale. Se întâmplă așa, întrucât majoritatea proceselor de fabricație utilizează scule aflate în magazinele de scule din cadrul celulelor flexibile. Drept urmare, aceste operații pot fi ascunse algoritmilor de ordonanțare.

De asemenea, nu este justificat ca algoritmii de ordonanțare (oricum, suficient de complecși și de laborioși) să trateze, în mod explicit, operațiile 4, 8, și 10, fiind mai avantajoasă subînțelegerea acestor operații pe lângă operația 9.

La fel, operațiile 5 și 7 pot fi subînțelese conex operației 6.

Ca o consecință a acestor remarci, în continuare, se vor avea în vedere, prin algoritmii de ordonanțare ce vor fi prezentați, doar operații de prelucrare și operații de transport, întregul proces de fabricație fiind considerat a consta doar în aceste două tipuri de operații.

5.2. Formularea problemei ordonanțării

Plecând de la cele de mai sus, problema ordonanțării poate fi formulată astfel*):

Se dau:

i_1). **operațiile în care consistă procesul tehnologic**, ca elemente ale unei mulțimi, "O":

$$O = \{o_1, o_2, \ldots, o_n\}$$

Remarcă:
În practică, de obicei, se consideră: fie $o_1 = 0$, $o_2 = 1$, ... , fie $o_1 = 1$, $o_2 = 2$, ...

i_2). **duratele operațiilor în care consistă procesul tehnologic**, ca valori ale unei funcții, "θ", numită "*funcție durată*", definită pe mulțimea operațiilor, și cu valori în R^+:

$$\theta: O \rightarrow R^+$$

i_3). **restricțiile de precedență** pentru operațiile procesului tehnologic [RUSU'90], exprimate prin așa-numitul "*digraf de precedență*", notat cu "G":

$$G = (O, V)$$

unde:

$O = \{o_1, o_2, \ldots, o_n\}$ este mulțimea nodurilor;

$V = \left\{ (o_i, o_j) \mid o_i, o_j \in O,\ o_i \text{ precede direct } o_j \right\}$ este mulțimea arcelor;

digraf în care fiecărui nod i se asociază valoarea luată de funcția durată pentru operația corespunzătoare nodului.

*) Se are în vedere doar cazul în care sistemul de fabricație este cu ritm fixat.

i_4). **celulele flexibile disponibile** în sistemul de fabricație, ca elemente ale unei mulțimi, "*P*":

$$P = \{p_0, p_1, \ldots, p_q, p_{q+1}\}$$

Observație:

Pentru facilitarea definirii restricțiilor de transport -a se vedea i_7)-, se vor asimila ca elemente ale mulțimii P, pe lângă celulele flexibile propriu-zise, etichetate cu "c_i", și depozitul de piese de la intrare, "DI", și depozitul de piese de la ieșire, "DE", astfel încât:

$p_0 = DI$
$p_i = c_i$, *pentru* $i = 1..q$
$p_{q+1} = DE$

În practică, de obicei, în locul etichetei "p_i", se preferă să se folosească doar indexul din cadrul ei, deci "*i*", astfel încât mulțimea *P* devine:

$$P = \{0, 1, \ldots, q+1\}$$

i_5). **restricțiile de prelucrabilitate** a operațiilor procesului tehnologic [ROBU'94g] [ROBU'94h], exprimate prin așa-numita "*funcție de prelucrabilitate*", notată cu "μ", definită pe mulțimea operațiilor, și cu valori în mulțimea "$\boldsymbol{P}$" a părților mulțimii celulelor flexibile:

$$\mu: O \rightarrow \boldsymbol{P}$$

sub forma:

$$\mu(x) = \{p_i^x, \ldots, p_{k_x}^x\}$$

unde:

$$x \in O,\ p_i^x \in P,\ i = 1...k_x$$

i_6). **restricțiile de compatibilitate** a operațiilor procesului tehnologic [ROBU'94g] [ROBU'94h], exprimate prin așa-numita "*funcție de incompatibilitate*", notată cu "ν", definită pe mulțimea operațiilor, și cu valori în mulțimea $\boldsymbol{O}$ a părților acesteia:

$$\nu: O \rightarrow \boldsymbol{O}$$

sub forma:

$$\nu(x) = \left\{ o_1^x, \ldots, o_{l_x}^x \right\}$$

unde:

$$x \in O,\ o_i^x \in O,\ i = 1..l_x$$

Observație:

Evident, "o_i^x" reprezintă operațiile incompatibile cu operația "x", în ceea ce privește efectuarea lor, în aceeași fază, pe o anumită celulă flexibilă.

i_7). restricțiile de transport, exprimate prin așa-numitul "*graf al fluxului pieselor*" [ROBU'94g] [ROBU'94h], notat cu "*F*":

$$F = (P,\ T)$$

unde:

$P = \left\{ p_0, p_1, \ldots, p_q, p_{q+1} \right\}$ este mulțimea nodurilor;

$T = \left\{ (p_i, p_j) \mid p_i, p_j \in P,\ \textit{de la } p_i \textit{ se poate ajunge la } p_j \right\}$ este mulțimea arcelor;

graf în care fiecărui arc i se asociază, ca lungime, durata operației de transfer al piesei din postul origine în postul țintă (prin termenul "*post*" se desemnează fie un depozit –de intrare sau de ieșire-, fie o celulă flexibilă).

i_8). ciclul sistemului de fabricație*), notat cu "δ", exprimat printr-o valoare din R^+, reprezentând perioada cu care piesele (sau produsele, după caz) părăsesc sistemul gata realizate [RUSU'90].

Se cere:

A) Să se stabilească, respectându-se *i_1)...i_8),* operațiile pe care le are de efectuat fiecare celulă flexibilă la nivelul unui ciclu al sistemului, astfel încât timpul de neutilizare a capacităților de producție să fie redus la minimum [RUSU'90] [[CHRY'91].

B) Să se stabilească sarcinile de transport specifice fiecărui ciclu al sistemului.

Rezolvarea problemei formulată se poate face apelând la algoritmi exacți sau la algoritmi euristici. Primii garantează optimalitatea soluțiilor oferite, dar reclamă timp de calcul și/sau spațiu de memorie uneori inacceptabil de mari [ACHU'82] [CARL'88]. Ceilalți pot fi

*) Unii autori folosesc, cu același înțeles, noțiunea de "*ritm al sistemului de fabricație*". Deși noi considerăm că, în context, termenul "*ciclu*" este mai adecvat decât termenul "*ritm*", îl vom folosi, uneori, și pe acesta din urmă.

satisfăcuți cu un spațiu de memorie rezonabil și rulați într-un timp tolerabil, dar nu garantează optimalitatea soluțiilor la care conduc [KIME'85] [DEWE'87].

Capacitatea memoriei și puterea de calcul, relativ reduse, care au caracterizat, până nu demult, calculatoarele de prețuri accesibile, au făcut ca problema ordonanțării să fie rezolvată, de regulă, cu ajutorul algoritmilor euristici [DEWE'87].

În ultimul timp, însă, datorită progreselor remarcabile cunoscute de domeniul tehnicii de calcul, a crescut interesul pentru algoritmii exacți [WHITE'90] [WIDM'91].

În continuarea acestui capitol, se vor prezenta algoritmi, atât exacți, cât și euristici, având la bază teoria și practica cercetării operaționale, precum și literatura de specialitate referitoare propriu-zis la problema ordonanțării în fabricația flexibilă, dar conținând elemente de originalitate care le conferă eficiență în aplicare, aplicabilitate largă, și capacitate de cuprindere ridicată.

5.3. Algoritmi pentru rezolvarea problemei ordonanțării

5.3.1. Definiții fundamentale

Definiția 5.3.1_1:

Se numește "fază" a procesului tehnologic o grupare de operații executabilă în cuanta de timp corespunzătoare unui ciclu al sistemului [ROBU'95b].

Dacă se notează cu "$\boldsymbol{Q}$" o partiție a procesului tehnologic în faze, presupunându-se că numărul fazelor este "m", și cu "Q_j" o fază oarecare, atunci, evident.

1). $\boldsymbol{Q} = \{Q_1, \ldots, Q_m\}$

2). $\forall j = 1...m \Rightarrow Q_j \subseteq O$

3). $\bigcup_{j=1...m} Q_j = O$

4). $\forall (i, j) \mid i = 1...m,\ j = 1...m,\ i \neq j \Rightarrow Q_i \cap Q_j = \varnothing$

5). $\exists x \in Q_r, \exists y \in Q_s, Q_r, Q_s \in \boldsymbol{Q} \mid (x, y) \in V \Rightarrow r \leq s$

6). $\forall j = 1...m \Rightarrow \sum_{x \in Q_j} \theta(x) \leq \delta$

Definiția 5.3.1_2:

Se numește "predecesor direct al unei operații" -fie aceasta "o"- o operație "x" care are proprietatea că $(x, o) \in V$ [RUSU'90] [ROBU'95b].

Fie $\pi(o)$ mulțimea predecesorilor direcți ai operației *o*. Rezultă:

$$\pi(o) = \{x \mid x \in O, (x,o) \in V\}$$

Definiția 5.3.1_3:

Se numește "predecesor direct al unei mulțimi de operații", inclusă în O -fie aceasta "H"- o operație "x" care are proprietatea că $\exists o \in H$ *astfel încât* $(x, o) \in V$ [ROBU'95b].

Fie $\pi(H)$ mulțimea predecesorilor direcți ai mulțimii *H*. Rezultă:

$$\pi(H) = \{x \mid x \in O, \{o \in H \mid (x,o) \in V\} \neq \varnothing\}$$

sau, echivalent [RUSU'90]:

$$\pi(H) = \bigcup_{o \in H} \pi(o)$$

Definiția 5.3.1_4:

Se numește "succesor direct al unei operații" -fie aceasta "o"- o operație "x" care are proprietatea că $(o, x) \in V$ [RUSU'90] [ROBU'95b].

Fie $\sigma(o)$ mulțimea succesorilor direcți ai operației *o*. Rezultă:

$$\sigma(o) = \{x \mid x \in O, (o, x) \in V\}$$

Definiția 5.3.1_5:

Se numește "succesor direct al unei mulțimi de operații", inclusă în O -fie aceasta "H"- o operație "x" care are proprietatea că $\exists o \in H$ *astfel încât* $(o, x) \in V$ [ROBU'95b].

Fie $\sigma(H)$ mulțimea succesorilor direcți ai mulțimii *H*. Rezultă:

$$\sigma(H) = \{x \mid x \in O, \{o \in H \mid (o,x) \in V\} \neq \varnothing\}$$

sau, echivalent [RUSU'90]:

$$\sigma(H) = \bigcup_{o \in H} \sigma(o)$$

Definiția 5.3.1_6:

Se numește "succesor direct exclusiv al unei operații" -fie aceasta "o"- o operație "x" care are proprietatea că $(u, x) \in V \Rightarrow u \equiv o$ [ROBU'95b].

Fie $\lambda(o)$ mulțimea succesorilor direcți exclusivi ai operației o. Rezultă:

$$\lambda(o) = \{x \mid x \in O,\ (u,x) \in V,\ cu\ u \in O \Rightarrow u \equiv o\}$$

Definiția 5.3.1_7:

Se numește "succesor direct exclusiv al unei mulțimi de operații", inclusă în O -fie aceasta "H"- o operație "x", $x \in O \setminus H$, *care are proprietatea că* $(o, x) \in V \Rightarrow o \in H$.

Fie $\lambda(H)$ mulțimea succesorilor direcți exclusivi ai mulțimii H. Rezultă [ROBU'95b]:

$$\lambda(H) = \{x \mid x \in O \setminus H,\ (o,x) \in V \Rightarrow o \in H\}$$

Definiția 5.3.1_8:

Se numește "grupare admisibilă de operații" o mulțime $X \subseteq O$, *pentru care* $\pi(X) \subseteq X$. *Mulțimea vidă se consideră grupare admisibilă* [RUSU'90].

Definiția 5.3.1_9:

Se numește "subciclu de prelucrare" corespunzător unei partiții a procesului tehnologic în faze intervalul de timp care începe odată cu ciclul sistemului și se termină odată cu faza de cea mai mare durată a respectivei partiții [ROBU'95b].

Definiția 5.3.1_10:

Se numește "subciclu de transport" corespunzător unei partiții a procesului tehnologic în faze, intervalul de timp care precede sfârșitul unui ciclu al sistemului, având durata egală cu a celei mai lungi operații de transport [ROBU'95b].

5.3.2. O metodă de rezolvare a problemei ordonanțării

Literatura de specialitate din ultimii ani este relativ bogată în abordări ale largii diversități de probleme de ordonanțare, o atenție aparte fiind acordată ordonanțării operațiilor tehnologice și sarcinilor de transport în fabricația flexibilă [CARL'88] [HENN'91].

Multe dintre aceste abordări se limitează, însă, la a fi doar valabile în principiu, în timp ce sub aspectul valorii de întrebuințare lasă mult de dorit [ERSC'83][CHU'89]. Este astfel, uneori pentru că se pleacă de la o punere a problemei ordonanțării ruptă de realitatea tehnică, alteori pentru că se dau soluții neimplementabile sau inacceptabil implementabile.

Abordarea de față, făcută de pe poziții pragmatice, propune o metodă de rezolvare a problemei ordonanțării ce se pretinde lipsită de neajunsuri de tipul celor evocate. După cum

se va putea aprecia, această metodă*) folosește algoritmi eficienți, atât prin prisma resurselor pe care le necesită pentru implementare, cât și în ceea ce privește timpul în care pot fi rulați.

Metoda rezidă în parcurgerea a trei etape [ROBU'95b]:

1) În prima etapă, se realizează partiționarea *echilibrată* a procesului tehnologic în faze, respectându-se restricțiile de precedență și de compatibilitate impuse de procesul tehnologic și/sau de sistemul de fabricație, restricțiile de prelucrare/montaj specifice celulelor, și, bineînțeles -însăși noțiunea de fază o implică-, restricțiile de ritm. Adoptând convenția ca pentru o partiție oarecare, "Q^i", cuprinzând "m_i" faze, fazele să se noteze cu "Q_k^i", unde $k = 1..m_i$, și atribuind indexul "$*$" partiției optime, se poate scrie că rezultatul parcurgerii primei etape este:

$$Q^* = \{Q_1^*, ..., Q_{m_*}^*\}$$

2) În a doua etapă, se identifică acoperirea optimă, sub aspectul duratelor operațiilor de transport, a fiecăreia dintre fazele procesului tehnologic, cu câte o celulă flexibilă. Acoperirile se apreciază după durata celei mai lungi operații de transport pe care o presupun. Este optimă acea acoperire pentru care durata celei mai lungi operații de transport este cea mai mică.

3) În a treia etapă, se reajustează dimensiunea ciclului sistemului -a cărei prescriere trebuie privită doar cu titlu orientativ-, astfel încât să devină:

$$\tilde{\delta} = \max_{i=1..m_*} (\textit{durata fazei "}i\textit{"} + \textit{durata operatiei de transport din celula fazei "}i\textit{" in celula fazei "}i+1\textit{"})$$

În continuare, metoda va fi detaliată prin prezentarea algoritmilor pe care etapele 1) și 2) îi presupun. O atenție mai mare va fi acordată partiționării echilibrate a procesului tehnologic în faze, referitor la aceasta propunându-se, în alternativă, 5 algoritmi.

5.3.3. Algoritm de partiționare echilibrată a procesului tehnologic în faze utilizând grupările admisibile și tehnica drumului critic

5.3.3.1. Macropașii algoritmului

La nivel macro, algoritmul pe care îl propunem [RUSU'90] consistă în următorii pași:

*) După cum a fost deja precizat (vezi paragraful 5.2), se are în vedere doar cazul în care sistemul de fabricație este cu ritm fixat.

r1). Se determină familia grupărilor admisibile de operații, asociată digrafului de precedență $G=(O,\ V)$. Fie "W" această familie:

$$W=\{X \mid X\subseteq O,\ \pi(X)\subseteq X\} \tag{5.3.3.1-1}$$

Prin convenție, se va considera că $\varnothing\in W$. Evident, un alt element al familiei W va fi însăși mulțimea O. Se notează cu X_0, ..., X_r mulțimile element ale familiei W, cu $X_0=\varnothing$ și $X_r=O$.

r2). Se construiește un digraf, "Z", numit "*graful grupărilor admisibile*":

$$Z=(W,\ Y) \tag{5.3.3.1-2}$$

unde:

$$Y=\left\{(X_i,\ X_j)\ \middle|\ X_i,\ X_j\in W,\ X_i\subset X_j,\ \underset{i\geq 1}{X_i}\Rightarrow\exists\ \underset{h\geq 0}{X_h}\in W \middle|\ (X_h,\ X_i)\in Y,\ \sum_{o\in X_j\setminus X_i}\theta(o)\leq\delta,\ \bigcap_{o\in X_j\setminus X_i}\mu(o)\neq\varnothing,\ \bigcup_{o\in X_j\setminus X_i}\left(\nu(o)\cap(X_j\setminus X_i)\right)=\varnothing\right\} \tag{5.3.3.1-3}$$

și se asociază fiecărui arc $(X_i,\ X_j)\in Y$ o lungime, notată cu "$l(X_i,\ X_j)$", dată de funcția:

$$l:\ Y\rightarrow R^+ \tag{5.3.3.1-4}$$

prin expresia:

$$l(X_i,\ X_j)=\delta-\sum_{o\in X_j\setminus X_i}\theta(o) \tag{5.3.3.1-5}$$

Prima condiție din relația (5.3.3.1-3) are menirea de a introduce în algoritm restricțiile de precedență. A doua condiție impune o cerință pe care am putea-o apela "*de filiație*", ea asigurând prezența în graf doar a arcelor ale căror surse au părinți, adică: au legături în amonte; evident, arcele ale căror surse nu au părinți nu prezintă nici un interes. A treia condiție introduce restricțiile de ritm, a patra -restricțiile de prelucrabilitate, iar a cincea -restricțiile de compatibilitate [ROBU'95b].

r3). Modul în care digraful Z a fost construit permite ca, pe baza lui, să fie determinate toate partițiile posibile ale procesului tehnologic în faze, o asemenea partiție corespunzând unui drum de la nodul X_0 la nodul X_r. O fază oarecare, "Q_k^i", a unei partiții "i" cuprinde operațiile rezultate ca diferență a mulțimilor X_k^i și X_{k-1}^i, corespunzătoare nodurilor situate în pozițiile "k", respectiv "k-1", pe traseul "i"[ROBU'95b].

Partiția optimă rezultă corespunzător drumului minim de la nodul X_0 la nodul X_r. Fie:

$$X_0^*, X_1^*, ..., X_{m_*}^*, \text{ cu } X_0^* = X_0 \text{ si } X_{m_*}^* = X_r \tag{5.3.3.1-6}$$

nodurile acestui drum și:

$$\boldsymbol{Q}^* = \{Q_1^*, ..., Q_{m_*}^*\} \tag{5.3.3.1-7}$$

partiția aferentă.

Evident:

$$Q_k^* = X_k^* \setminus X_{k-1}^* \tag{5.3.3.1-8}$$

Rezultă:

$$\sum_{k=1}^{m_*} I(X_k^*, X_{k-1}^*) = \sum_{k=1}^{m_*} \left(\delta - \sum_{o \in X_k^* \setminus X_{k-1}^*} \theta(o) \right) = m_* \cdot \delta - \sum_{k=1}^{m_*} \sum_{o \in Q_k^*} \theta(o) = m_* \cdot \delta - \sum_{o \in O} \theta(o) \tag{5.3.3.1-9}$$

Relația (5.3.3.1-9) arată că drumul minim dintre nodurile X_0 și X_r corespunde traseului care le unește trecând prin cât mai puține alte noduri, respectiv că partiția optimă este cea care consistă în cele mai puține faze.

În cazul în care în digraful Z există mai multe -de exemplu "v"- trasee de lungime minimă între X_0 și X_r se vor reține toate partițiile ce rezultă corespunzător lor, urmând ca selecția celei mai avantajoase dintre ele să se facă în etapa a doua a metodei. Pentru a adecva notațiile la eventualitatea unui asemenea caz, simbolul "$*$" se va înlocui cu simbolul "x", având dedicate valorile $1...v$.

5.3.3.2. Determinarea grupărilor admisibile de operații

Problema determinării grupărilor admisibile de operații este tratată detaliat în [RUSU'90], cu trimiteri la [GUPT'84]. În [RUSU'90], se prezintă un algoritm, de tip recursiv. Acest algoritm stă la baza a ceea ce urmează.

Fie "W_{k-1}" notația pentru familia grupărilor admisibile determinate într-un pas de calcul oarecare, "$k-1$", și "w_{k-1}" notația pentru cardinalul acestei familii:

$$w_{k-1} = |W_{k-1}| \tag{5.3.3.2-1}$$

Notăm cu "X_h^{k-1}", unde $h = 1...w_{k-1}$, grupările admisibile elemente ale familiei W_{k-1} :

$$W_{k-1} = \left\{ X_1^{k-1}, ..., X_{w_{k-1}}^{k-1} \middle| X_h^{k-1} \subseteq O,\ \pi\left(X_h^{k-1}\right) \subseteq X_h^{k-1},\ h = 1..w_{k-1} \right\} \qquad (5.3.3.2\text{-}2)$$

Fie:

$$E_h^{k-1} = \left\{ o \middle| o \in O,\ o \notin X_h^{k-1},\ \pi(o) = \varnothing \right\} \qquad (5.3.3.2\text{-}3)$$

mulțimea operațiilor fără predecesori direcți și neincluse în gruparea admisibilă X_h^{k-1}.

Conform celor precizate în 5.3.1, cu "$\lambda\left(X_h^{k-1}\right)$" desemnăm mulțimea succesorilor direcți exclusivi ai grupării admisibile X_h^{k-1}.

Fie "B_h^{k-1}" mulțimea operațiilor care sunt fie succesori direcți exclusivi ai grupării admisibile X_h^{k-1}, fie operații fără predecesori direcți, neincluse în X_h^{k-1}:

$$B_h^{k-1} = \lambda\left(X_h^{k-1}\right) \cup E_h^{k-1} \qquad (5.3.3.2\text{-}4)$$

Notăm cu "b_h^{k-1}" cardinalul mulțimii B_h^{k-1}:

$$b_h^{k-1} = \left| B_h^{k-1} \right| \qquad (5.3.3.2\text{-}5)$$

Fie "R_h^{k-1}" mulțimea părților mulțimii B_h^{k-1}. Evident:

$$\left| B_h^{k-1} \right| = 2^{b_h^{k-1}} - 1 \qquad (5.3.3.2\text{-}6)$$

și, deci:

$$R_h^{k-1} = \left\{ R_{h,i}^{k-1} \middle| R_{h,i}^{k-1} \subseteq B_h^{k-1},\ R_{h,i}^{k-1} \neq \varnothing,\ i = 1..(2^{b_h^{k-1}} - 1) \right\} \qquad (5.3.3.2\text{-}7)$$

În aceste condiții, după cum se demonstrează în [RUSU'90], se va avea:

$$W_k = \bigcup_{h=1..w_{k-1}} \left(X_h^{k-1} \cup R_h^{k-1} \right) \qquad (5.3.3.2\text{-}8)$$

pentru orice $k = 2...K$, unde:

$$K \text{ are proprietatea ca } W_K = O \qquad (5.3.3.2\text{-}9)$$

în timp ce:

$$W_0 = \{\varnothing\} \tag{5.3.3.2-10}$$

$$W_1 = \left\{ X_1^1, \ldots, X_{w_1}^1 \,\middle|\, X_h^1 \subseteq O,\ \pi\left(X_h^1\right) = \varnothing,\ h = 1..w_1 \right\} \tag{5.3.3.2-11}$$

iar familia grupărilor admisibile asociată digrafului de precedență $G = (O, V)$ se obține cu relația:

$$W = \bigcup_{k=0}^{K} W_k \tag{5.3.3.2-12}$$

Din cele mai de sus, rezultă că, într-o familie W_k, s-ar putea regăsi grupări admisibile prezente și în familia W_{k-1}. Cardinalul familiei W_k ar crește, atunci, nejustificat, conducând la creșterea numărului de pași în care ar urma să se determine W_{k+1}; evident fără ca din pașii în plus să poată rezulta vreo grupare admisibilă nouă. Având în vedere această observație, vom proceda la înlocuirea relației (5.3.3.2-12) cu relația:

$$W_k = \bigcup_{h=1..w_{k-1}} \left(\left(X_h^{k-1} \cup R_h^{k-1} \right) \setminus W_{k-1} \right) \tag{5.3.3.2-13}$$

Pe baza celor de mai sus, rezultă următorul algoritm de determinare a grupărilor admisibile:

1. $W_0 = \varnothing$;
2. $W_1 = \left\{ X_1^1, \ldots, X_{w_1}^1 \,\middle|\, X_h^1 \subseteq O,\ \pi\left(X_h^1\right) = \varnothing,\ h = 1..w_1 \right\}$;
3. $W = W_0 \cup W_1$;
4. $k = 2$;
5. *while* $\left(W_{k-1} \neq \varnothing\right)$ {
6. $\quad W_k = \varnothing$;
7. $\quad w_{k-1} = \left| W_{k-1} \right|$;
8. $\quad$ *for* $\left(h = 1;\ h \le w_{k-1};\ h{+}{+}\right)$ {
9. $\quad\quad E_h^{k-1} = \left\{ o \,\middle|\, o \in O,\ o \notin X_h^{k-1},\ \pi(o) = \varnothing \right\}$;
10. $\quad\quad \lambda\left(X_h^{k-1}\right)$;
11. $\quad\quad B_h^{k-1} = \lambda\left(X_h^{k-1}\right) \cup E_h^{k-1}$;
12. $\quad\quad b_h^{k-1} = \left| B_h^{k-1} \right|$;
13. $\quad\quad R_h^{k-1} = \left\{ R_{h,i}^{k-1} \,\middle|\, R_{h,i}^{k-1} \subseteq B_h^{k-1},\ R_{h,i}^{k-1} \neq \varnothing,\ i = 1..(2^{b_h^{k-1}} - 1) \right\}$;
14. $\quad\quad W_k = W_k \cup \left(X_h^{k-1} \cup R_h^{k-1} \right)$;
15. $\quad$ }

Fig. 5.3.3.2_1. Algoritm de determinare a grupărilor admisibile (partea 1/2).

```
16.     W_k = W_k \ W_{k-1} ;
17.     W = W ∪ W_k ;
18.     k++;
19. }
20. r = |W| - 1;
```

Fig. 5.3.3.2_1. Algoritm de determinare a grupărilor admisibile (partea 2/2).

5.3.3.3. Construcția grafului grupărilor admisibile

Digraful grupărilor admisibile -pe care l-am notat, reamintim, cu "Z"- se construiește considerând drept noduri grupările admisibile, $X_\bullet$, și trasând, în mulțimea acestora, arce orientate între acele perechi (X_i, X_j), care îndeplinesc condițiile:

a). $X_i \subset X_j$ (5.3.3.3-1)

b). $\underset{i\geq 1}{X_i} \Rightarrow \exists \underset{h\geq 0}{X_h} \in W \,|\, (X_h, X_i) \in Y$ (5.3.3.3-2)

c). $\sum_{o\in X_j \setminus X_i} \theta(o) \leq \delta$ (5.3.3.3-3)

d). $\bigcap_{o\in X_j \setminus X_i} \mu(o) \neq$ (5.3.3.3-4)

e). $\bigcup_{o\in X_j \setminus X_i} \left(\nu(o) \cap (X_j \setminus X_i)\right)$ (5.3.3.3-5)

reprezentând, respectiv [ROBU'95b]:

a). restricțiile de precedență;
b). cerința de filiație;
c). restricțiile de ritm;
d). restricțiile de prelucrabilitate;
e). restricțiile de compatibilitate.

În continuare, fiecărui arc $(X_i, X_j) \in Y$ i se asociază lungimea $l(X_i, X_j)$:

$$l(X_i, X_j) = \delta - \sum_{o\in X_j \setminus X_i} \theta(o) \qquad (5.3.3.3\text{-}6)$$

Evident, această lungime reprezintă timpul de neutilizare a capacităților de producție, socotit la nivelul unui ciclu, caracteristic fazei pe care o determină arcul (X_i, X_j).

Reprezentarea grafului grupărilor admisibile se poate face prin matricea de incidență, pe care o notăm cu "*Z[][]*". Numărul de linii, respectiv numărul de coloane ale tabloului *Z* vor fi egale cu cardinalul mulțimii *W*. Un element al tabloului cu indicele de linie egal cu "*i*" și indicele de coloană egal cu "*j*" va avea valoarea *l(i,j)*, dacă de la nodul X_i la nodul X_j există un arc, respectiv valoarea maximă reprezentabilă, pe care o vom desemna prin notația "*SUPREPR*", altfel.

5.3.3.4. Determinarea fazelor

Așa cum s-a arătat deja, fiecărui arc (X_i, X_j) îi corespunde o fază, cuprinzând operațiile rezultate ca diferență a mulțimilor X_j și X_i. Fazele optime se obțin corespunzător arcelor situate pe cel mai scurt drum de la nodul X_0 a nodul X_r.

Identificarea drumului minim dintre nodurile X_0 și X_r se poate face în diverse moduri, literatura de specialitate oferind un număr însemnat de algoritmi. Dintre aceștia, însă, o bună parte sunt nesatisfăcători pentru aplicația avută în vedere, deoarece își ating finalitatea printr-un număr mare de pași și necesită -fie pentru implementare, fie pentru rulare, fie pentru ambele- un volum important de memorie*). Un algoritm satisfăcător prin prisma aspectelor menționate este *algoritmul Floyd*. Propunem, însă, un algoritm de tip backtracking [ROBU'95b], care, pentru determinarea fazelor optime ale unui proces tehnologic, este, în mod sensibil, mai eficient (se cunoaște faptul că, în genere, algoritmii backtracking sunt cei mai eficienți algoritmi de explorare a grafurilor puțin dense; ori, graful grupărilor admisibile este un graf puțin dens).

Explorarea backtracking permite identificarea și reținerea tuturor traseelor existente între două noduri precizate ale unui graf. Desigur, reținerea se poate face selectiv, pe baza unui criteriu sau, eventual, a mai multora.

În algoritmul propus, de îndată ce un traseu de la nodul X_0 la nodul X_r este identificat, se compară lungimea sa cu lungimea traseului cel mai scurt de până atunci și, în funcție de rezultatul comparării, se ia una dintre deciziile:

*) Se are în vedere și faptul că simpla reprezentare a grafului grupărilor admisibile, în ipoteza că familia acestora este de cardinal cel mult egal cu 512, necesită, în cazul soluției bazată pe matricea de incidență, considerând că lungimea arcelor se exprimă pe 4 octeți (prin tipul *float*, în C), un volum de memorie de 512 x 512 x 4 octeți -adică: de 1 Mbytes-, ceea ce, evident, nu este puțin.

1) eliminarea din evidență a traseelor găsite în pașii anteriori ca fiind cele mai scurte, dacă traseul curent identificat este mai scurt decât ele, și reținerea, în locul lor, doar a acestui traseu;
2) reținerea în evidență, alături de traseele găsite în pașii anteriori ca fiind cele mai scurte, și a traseului curent identificat, dacă lungimea acestuia este egală cu a lor;
3) ignorarea traseului curent identificat, dacă lungimea sa este mai mare decât a celor găsite în pașii anteriori ca fiind cele mai scurte.

Pentru facilitarea exprimării exacte și concise a algoritmului, detașat de implementarea pe care el o va căpăta, se va considera că fiecare nod al grafului are asociată o structură informațională, denumită "*ELTR*", care îl poate caracteriza ca element al unui traseu și care cuprinde:

- un câmp, numit "*indulteltrt*", dedicat înregistrării indexului ultimului element al traseului tată;
- un câmp, numit "*indtrt*", dedicat înregistrării indexului traseului tată;
- un câmp, numit "*lungtr*", dedicat înregistrării distanței față de nodul origine;
- un câmp, numit "*indndeltr*", dedicat înregistrării indexului nodului corespunzător elementului de traseu în cauză;
- un câmp, reprezentând un tablou unidimensional cu "$r+1$" elemente (reamintim: "$r+1$" este cardinalul mulțimii grupărilor admisibile), numit "*nodtr[]*", dedicat înregistrării nodurilor traseului în cauză și a ordinii în care ele se succed, în convenția: un element de indice "j" corespunde nodului aferent grupării admisibile "X_j" și ia ca valoare indicele elementului corespunzător succesorului acestui nod.

Întru aceeași idee, se va conta pe existența unui tablou bidimensional de structuri de tipul *ELTR*, referit prin apelativul "*eltr[][]*", în cadrul căruia indicele de linie reprezintă indexul elementului de traseu corespunzător în cadrul traseului, iar indicele de coloană -indexul traseului.

Drumurile minime se vor oferi printr-un tablou de tablouri de același tip cu tabloul *nodtr[]* menționat mai sus. Acest tablou de tablouri va fi referit prin apelativul "*dm[][]*", în care indicele de linie reprezintă indexul drumului minim, în timp ce indicele de coloană desemnează tabloul prin care se definește traseul corespunzător drumului minim de respectivul index. Se face convenția ca situația -evident: anormală- în care între nodul corespunzător grupării admisibile X_0 și nodul corespunzător grupării admisibile X_r nu există nici un traseu să fie semnalată de algoritm prin returnarea valorii "-1" pentru indicele de linie aferent tabloului *dm[][]*.

Lungimea drumului curent de la nodul corespunzător grupării admisibile X_0 la nodul corespunzător grupării admisibile curente, "X_j", se notează cu "*ldc*", iar lungimea drumului minim de la nodul corespunzător grupării admisibile X_0 la nodul corespunzător grupării admisibile X_r se notează cu "*ldmin*".

Graful grupărilor admisibile se consideră reprezentat așa cum s-a arătat în paragraful 5.3.3.3, adică, prin matricea de incidență, *Z[][]*.

Având în vedere cele de mai sus, algoritmul backtracking pe care îl propunem se poate exprima prin pseudocodul*) din figura 5.3.3.4_1.

Observație:

În figura 5.3.3.4_1, în plus față de cele menționate, s-au folosit notațiile:

- *"NIMIC" pentru constanta cu toți biții la "1".*
- *"FALS" -pentru constanta nulă.*
- *"ADEVĂRAT" -pentru constanta "1".*

```
1.  ldmin = SUPREPR;
2.  k = -1;
3.  back = 0;
4.  m = 0;
5.  h = 0;
6.  t = 0;
7.  creat (n[0]);
8.  n[0] = -1;
9.  i = 0;
10. j = 1;
11. creat (eltr[0][0]);
12. p = &eltr[0][0];
13. p->indndeltr = 0;
14. p->indtrt = 0;
15. p->lungtr = 0;
16. p->indndulteltrt = 0;
17. p->nodtr[x] = NIMIC,  ∀x = 1...r;
18. p->nodtr[0] = 0;
19. while (j ≤ r || i ≠ 0) {
20.     if (j ≤ r) {
21.         if ((Z[i][j] ≠ SUPREPR) && (eltr[m][h].lungtr+Z[i][j] ≤ ldmin) ) {
22.             if (incautareramif) {
23.                 t++;
24.                 creat (n[t]);
25.                 n[t] = -1;
26.                 incautareramif = FALS;
27.             }
```

Fig. 5.3.3.4_1. Expresia pseudocod a algoritmului backtracking de determinare a traseelor de drum minim ale grafului grupărilor admisibile (partea 1/2).

*) Pseudocodul utilizat conține elemente de sintaxă preluate, cu semantica lor originală, din limbajul C. Motivația acestei opțiuni consistă în conciziunea și sugestivitatea ce caracterizează aceste elemente, precum și în adecvanța limbajului C pentru implementarea algoritmului.

```
28.             n[t]++;
29.             creat (eltr[n[t]][t]);
30.             p = &eltr[n[t]][t];
31.             p->indulteltrt = m;
32.             p->indtrt = h;
33.             p->lungtr = eltr[m][h].lungtr + Z[i][j];
34.             p->indndeltr = j;
35.             p->nodtr[x] = eltr[m][h].nodtr[x],  ∀x = 0...r /x ≠ i, x ≠ j;
36.             p->nodtr[i] = j;
37.             p->nodtr[j] = j;
38.             ldc = p->lungtr;
39.             m = n[t];
40.             h = t;
41.             i = j;
42.         }
43.         j++;
44.     }
45.     else {
46.         if(!incautareramif) {
47.             if (i ≡ r) {
48.                 if (ldc ≤ ldmin) {
49.                     if (ldc < ldmin) {
50.                         ldmin = ldc;
51.                         k = 0;
52.                     }
53.                     else {
54.                         k++;
55.                     }
56.                     creat (dm[k][x]);
57.                     dm[k][x] = p->nodtr[x], x = 0...r;
58.                 }
59.                 m = p->indulteltrt;
60.                 h = p->indtrt;
61.                 p = &eltr[m][h];
62.             }
63.             incautareramif = ADEVARAT;
64.         }
65.         else {
66.             p = &eltr[m][h];
67.         }
68.         m = p->indulteltrt;
69.         h = p->indtrt;
70.         i = eltr[m][h].indndeltr;
71.         j = p->indndeltr+1;
72.     }
73. }
```

Fig. 5.3.3.4_1. Expresia pseudocod a algoritmului backtracking de determinare a traseelor de drum minim ale grafului grupărilor admisibile (partea 2/2).

5.3.4. Algoritm de partiționare echilibrată a procesulu tehnologic în faze utilizând grupările admisibile ș programarea dinamică

5.3.4.1. Reducerea problemei partiționării echilibrate în faze l o problemă de programare dinamică

Din paragrafele precedente, transpare faptul că partiționarea echilibrată a operațiilo procesului tehnologic în faze poate fi asimilată cu un proces secvențial de decizii, rezulta prin descompunerea problemei inițiale în subprobleme [RUSU'90]. Cum mulțimea acestor este cel mult numărabilă, se deduce posibilitatea abordării problemei ordonanțării pe baz principiilor programării dinamice discrete. Principalele tehnici ale acesteia sunt sintetizate î [ZIDĂ'75].

În acord cu notațiile introduse deja, fie W mulțimea grupărilor admisibile de operații al mulțimii O, rezultate în condițiile restricțiilor de precedență impuse prin digrafu $G=(O, V)$, și δ ciclul sistemului.

Definind:

$$G(X)=(X, V(X)) \tag{5.3.4.1-1}$$

unde:

$$V(X)=\left\{(o_i, o_j) \,\middle|\, o_i, o_j \in X, (o_i, o_j) \in V\right\} \tag{5.3.4.1-2}$$

se creează premizele rezolvării problemei partiționării optime a operațiilor procesulu tehnologic în faze, într-o manieră recursivă, specifică programării dinamice, considerând-o succesiv, în ordinea creșterii cardinalilor, la nivelul fiecărei grupări admisibile.

Fie:

$$W=\left\{X_0, X_1, \ldots, X_r \,\middle|\, X_0=\varnothing, X_r=O, |X_i| \le |X_{i+1}|, \forall i=0\ldots r-1\right\} \tag{5.3.4.1-3}$$

Să presupunem că în momentul considerării problemei avută în vedere la nivelul une grupări admisibile "X_i", de cardinal "s", pentru grupările admisibile "X_k", de cardinal "s-l", problema este deja rezolvată, corespunzător lor, ajungându-se la fazele $Q_1^k, \ldots, Q_{m_k}^k$.

Având în vedere că gruparea admisibilă X_i este, cu cel puțin una dintre grupările X_k, într-o relație de genul:

$$X_i = X_k \cup \left\{o_{b_k}^i\right\},\ unde\ o_{b_k}^i \in X_i \setminus X_k \tag{5.3.4.1-4}$$

rezultă că:

$$\begin{aligned} Q_1^i &= Q_1^k \\ Q_{m_k-1}^i &= Q_{m_k-1}^k \end{aligned} \tag{5.3.4.1-5}$$

iar, în continuare, dacă expresia logică:

$$\left(\left(\theta\left(Q_{m_k}^k\right)+\theta\left(o_{b_k}^i\right)\le\delta\right)\&\&\left(\mu\left(Q_{m_k}^k\right)\cap\mu\left(o_{b_k}^i\right)\neq\varnothing\right)\&\&\left(\nu\left(o_{b_k}^i\right)\cap Q_{m_k}^k\equiv\varnothing\right)\right) \tag{5.3.4.1-6}$$

este adevărată, atunci:

$$m_i = m_k \tag{5.3.4.1-7}$$

$$Q_{m_i}^i = Q_{m_k}^k \cup \left\{o_{b_k}^i\right\} \tag{5.3.4.1-8}$$

altfel:

$$m_i = m_k + 1 \tag{5.3.4.1-9}$$

$$Q_{m_i-1}^i = Q_{m_k}^k \tag{5.3.4.1-10}$$

$$Q_{m_i}^i = \left\{o_{b_k}^i\right\} \tag{5.3.4.1-11}$$

"k" și *"h"* fiind, în cazul neunicității lor, cei pentru care, în plus, expresia:

$$\theta^*\left(X_i\right) = \begin{cases} \theta^*\left(X_k\right) - \theta\left(o_{b_k}^i\right) = \sum_{x=1}^{m_k}\left(\delta - \theta\left(Q_x\right)\right) - \theta\left(o_{b_k}^i\right),\ daca\ \theta\left(Q_{m_k}\right)+\theta\left(o_{b_k}^i\right)\le\delta \\ \theta^*\left(X_k\right)+\delta-\theta\left(o_{b_k}^i\right) = \sum_{x=1}^{m_k}\left(\delta-\theta\left(Q_x\right)\right)+\delta-\theta\left(o_{b_k}^i\right), daca\,\theta\left(Q_{m_k}\right)+\theta\left(o_{b_k}^i\right)>\delta \end{cases} \tag{5.3.4.1-12}$$

ia cea mai mică valoare.

5.3.4.2. Pașii algoritmului

Pe baza celor enunțate în paragraful precedent, se presupune următorul algoritm pentru rezolvarea problemei partiționării echilibrate a procesului tehnologic în faze:

```
1.  /*    = {X_0, X_1, ..., X_r}   */
2.  /*  X_0 = ∅     */
3.  /*  X_r = O  */
4.  /*  |X_i| ≤ |X_{i+1}|, ∀i = 0...r-1  */
5.  /*  X_i = {o_1^i, o_2^i, ..., o_{s_i}^i}   */
6.  θ*(X_0) = δ ;
7.  m_0 = 1;
8.  Q^0_{m_0} = ∅;
9.  θ(Q^0_{m_0}) = 0 ;
10. s = 0;
11. i = 1;
12. while (i ≤ r) {
13.    if(|X_i| > s) {
14.       K = i - 1 ;
15.       s = |X_i|;
16.    }
17.    θ*(X_i) = SUPREPR ;
18.    for (j = 1; j ≤ s; j++) {
19.       k = K ;
20.       while ((|X_k| ≡ s - 1) && (X_i \ {o_j^i} ≠ X_k)) {
21.          k--;
22.       }
23.       if(X_i \ {o_j^i} ≡ X_k) {
24.          alt = (θ(Q^k_{m_k}) + θ(o_j^i) ≤ δ) &&
25.                && (μ(Q^k_{m_k}) ∩ μ(o_j^i) ≠ ∅) && (ν(o_j^i) ∩ Q^k_{m_k} ≡ ∅);
```

Fig. 5.3.4.1_1. Expresia pseudocod a algoritmului de partiționare echilibrată a procesului tehnologic în faze după principii de programare dinamică (partea 1/2).

```
26.             if(alt) {
27.                 ε = −θ(o_j^i);
28.             }
29.             else {
30.                 ε = δ − θ(o_j^i);
31.             }
32.             if(θ*(X_i) > θ*(X_k) + ε) {
33.                 θ*(X_i) = θ*(X_k) + ε ;
34.                 h = j;
35.                 l = k;
36.                 stq = alt;
37.             }
38.         }
39.     }
40.     m_i = m_i;
41.     Q_1^i = Q_1^1, ..., Q_{m_i}^i = Q_{m_1}^1;
42.     if(stq) {
43.         Q_{m_i}^i = Q_{m_i}^i ∪ {o_h^i};
44.         (Q_{m_i}^i) = θ(Q_{m_1}^1) + θ(o_h^i);
45.     }
46.     else {
47.         m_i ++; ;
48.         Q_{m_i}^i = {o_k^i};
49.         θ(Q_{m_i}^i) = θ(o_k^i) ;
50.     }
51.     i++;
52. }
```

Fig. 5.3.4.1_1. Expresia pseudocod a algoritmului de partiționare echilibrată a procesului tehnologic în faze după principii de programare dinamică (partea 2/2).

O analiză a algoritmului evidențiază cerințele de memorie relativ mari pe care el le necesită, precum și eficiența sa remarcabilă în privința timpului de rulare.

5.3.5. Algoritm de partiționare echilibrată a procesului tehnologic în faze utilizând direct graful de precedență și tehnicile "*branch and bound*" și "*backtracking*"

5.3.5.1. Preliminarii

După cum transpare din paragraful 5.3.3.2, determinarea grupărilor admisibile de operații comportă un efort de calcul considerabil, însoțit de un necesar de memorie însemnat. Același lucru se poate afirma și relativ la algoritmii care, plecând de la aceste grupări admisibile, determină fazele optime ale procesului tehnologic, afirmația fiind valabilă indiferent dacă ei acționează pe baza identificării drumului minim -respectiv a drumurilor minime- în graful grupărilor admisibile sau pe principii de programare dinamică.

În consecință, s-au căutat soluții ale problemei partiționării care să nu necesite cunoașterea grupărilor admisibile, adică: soluții obtenabile direct din graful de precedență [BELL'82][BERR'86][CARL'89c][DEME'90][BRUC'91]. În cele ce urmează, vom prezenta o asemenea soluție, dataliind-o la nivel de pseudocod al algoritmului în care ea consistă. Acest algoritm, care este original, are la bază tehnicile *"branch and bound"* și "*backtracking*", binecunoscute în domeniul cercetării operaționale. În cadrul algoritmului, sunt considerate toate restricțiile avute în vedere la formularea problemei ordonanțării, adică: restricțiile de precedență, de prelucrabilitate, de compatibilitate, și de ritm [ROBU'94g].

5.3.5.2. Principiile partiționării

Definiția 5.3.5.2_1:

Dat fiind graful de precedență $G=(O,\ V)$, *o permutare* $p=(o_1,\ o_2,\ ...,\ o_n)$ *a mulțimii* O *se numește "permutare admisibilă" dacă pentru orice* $(x,\ y)\in V$, *cu* $x=o_r,\ y=o_s$, *rezultă* $r<s$ [ROBU'95b].

Notăm cu "$\boldsymbol{O}$" mulțimea permutărilor admisibile ale mulțimii O din $G=(O,\ V)$.

Este demonstrat că soluțiile problemei partiționării echilibrate a procesului tehnologic în faze corespund permutărilor admisibile ale mulțimii operațiilor [RUSU'90]. Pe această bază, rezultă posibilitatea de principiu de a determina cel puțin unele dintre soluții prin generarea mulțimii acestor permutări și supunerea lor la următorul algoritm:

```
1.  /* O = { p_1, p_2, ..., p_r | p_i = (o_1^i, o_2^i, ..., o_n^i | ∀(o_r^i, o_s^i) ∈ V ⇒ r < s) } */
2.  /* k: numarul partitiilor optime oferite de algoritm */
3.  k = 1;
4.  minf = SUPREPR;
5.  for (i=1; i≤r; i++) {
6.      m = 1;
7.      Q_m = ∅;
8.      for (j=1; i ≤ n; i++) {
9.          if (o_j^i este asignabil la Q_m^k) {
10.             Q_m^i = Q_m^i ∪ o_j^i;
11.         }
12.         else {
13.             m++;
14.             Q_m^i = o_j^i;
15.         }
16.     }
17.     if (m ≤ minf) {
18.         if (m < minf) {
19.             minf = m;
20.             k = 1;
21.         }
22.         else {
23.             k++;
24.         }
25.       Q^k = {Q_1^k, Q_2^k, ..., Q_m^k}
26.     }
27. }
```

Fig. 5.3.5.2_1. Algoritm de principiu pentru partiționarea echilibrată a procesului tehnologic în faze, pe baza mulțimii permutărilor admisibile.

Determinarea permutărilor admisibile plecând de la graful de precedență presupune arborizarea acestuia. În paragraful următor, se vor prezenta regulile pe baza cărora un graf orientat aciclic poate fi transformat într-o arborescență.

5.3.5.3. Arborizarea grafului de precedență

Fie:

$H = (Z, W)$ -arborescența corespunzătoare grafului $G = (O, V)$.

Pe mulțimea nodurilor ei, "*Z*", -noduri notate cu "$z_{k,h}$", unde "*k*" reprezintă nivelul p care ele se află, iar "*h*" este indexul în cadrul nivelului-, se definește o funcție "*f*", cu valor în mulțimea operațiilor procesului tehnologic, *0*. Se spune că funcția *f* atribuie fiecărui nod a arborescenței câte o marcă și, în acord, ea poartă numele de "*funcție de marcare*". Evident, c marcă reprezintă o operație sau, mai exact, indexul unei operații. Se precizează că se impun ca operațiile procesului tehnologic să se numeroteze începând cu *1*, în scopul rezervări valorii *0* pentru marcarea fictivă a nodului rădăcină. În aceste premize, regulile prin care s realizează arborizarea sunt următoarele [ROBU'94g]:

- Se creează nodul rădăcină al arborescenței, $z_{0,1}$, asociindu-i-se marca *0*. Deci, $f(z_{0,1}) = 0$.
- Se construiește mulțimea $\alpha(z_{0,1}) = \{o \mid o \in O,\ \pi(o) = \varnothing\}$. Dacă $\alpha(z_{0,1}) = \{o_1^{0,1}, \ldots, o_{s_{0,1}}^{0,1}\}$ atunci rezultă $|\gamma(z_{0,1})| = s_{0,1}$, cu $\gamma(z_{0,1}) = \{z_{1,1}, \ldots, z_{1,s_{0,1}}\}$ și $f(z_{1,1}) = o_1^{0,1}, \ldots, f(z_{1,s_{0,1}}) = o_{s_{0,1}}^{0,1}$
- Se construiesc mulțimile $\varphi(z_{1,1}) = \{f(z_{1,1})\}, \ldots, \varphi(z_{1,s_{0,1}}) = \{f(z_{1,s_{0,1}})\}$ și, corespunzător lor se determină mulțimile $\rho(z_{1,1}) = \{O \setminus \varphi(z_{1,1})\}, \ldots, \rho(z_{1,s_{0,1}}) = \{O \setminus \varphi(z_{1,s_{0,1}})\}$.
- Se construiește mulțimea $\alpha(z_{1,1}) = \{o \mid o \in \rho(z_{1,1}),\ \pi(o) \subseteq \varphi(z_{1,1})\}$. Dacă $\alpha(z_{1,1}) = \{o_1^{1,1}, \ldots, o_{s_{1,1}}^{1,1}\}$, atunci rezultă $|\gamma(z_{1,1})| = s_{1,1}$, cu $\gamma(z_{1,1}) = \{z_{2,1}, \ldots, z_{2,s_{1,1}}\}$, și $f(z_{2,1}) = o_1^{1,1}, \ldots, f(z_{2,s_{1,1}}) = o_{s_{1,1}}^{1,1}$.
- Se construiesc mulțimile $\varphi(z_{2,1}) = \{f(z_{1,1}),\ f(z_{2,1})\}, \ldots, \varphi(z_{2,s_{1,1}}) = \{f(z_{1,1}),\ f(z_{2,s_{1,1}})\}$ și, corespunzător lor, se determină mulțimile $\rho(z_{2,1}) = \{O \setminus \varphi(z_{2,1})\}$, $\rho(z_{2,s_{1,1}}) = \{O \setminus \varphi(z_{2,s_{1,1}})\}$.
- ș.a.m.d.

Un traseu al arborescenței se termină când se ajunge la $\alpha(\cdot) = \varnothing$, echivalent cu $\gamma(\cdot) = \varnothing$.

Remarcă:

Din modul în care $\varphi(\cdot)$ și $\rho(\cdot)$ au fost definite, rezultă că $\varphi(z_{k,h})$ reprezintă mulțimea ordonată a operațiilor atribuite ca mărci nodurilor arborescenței aflate pe traseul care unește nodul $z_{0,1}$ cu nodul $z_{k,h}$, iar $\rho(z_{k,h})$ -mulțimea operațiilor încă neatribuite ca mărci niciunuia dintre nodurile acestui traseu.

Fie $\{Z_0, Z_1, \ldots, Z_n\}$ partiția în nivele a mulțimii *Z*, unde:

$$Z_0 = \{z_{0,1}\}$$

$$Z_k = \bigcup_{z \in Z_{k-1}} \gamma(z),\ \forall k = 1...n$$

În continuare, vârfurilor $z_{k,h}$ ale arborescenței H, cu $z_{k,h} \in Z_k$, li se asociază submulțimi ale mulțimii permutărilor admisibile, $\boldsymbol{O}$, după cum urmează. Dacă $\varphi(z_{k,h}) = \{o_1^{k,h}, \ldots, o_k^{k,h}\}$, atunci submulțimea permutărilor admisibile asociată vârfului $z_{k,h}$ este:

$$S(z_{k,h}) = \{o_1^{k,h}, \ldots, o_k^{k,h}, y_{k+1}^{k,h}, \ldots, y_n^{k,h}\}$$

cu:

$$y_j^{k,h} \in \rho(z_{k,h}),\ \forall j = k+1..n;\ y_u \neq y_v,\ \forall u, v = k+1..n,\ u \neq v;\ (y_u, y_v) \in V \Rightarrow u < v$$

Pentru $k=n$, se definește $S(z_{n,h}) = \{o_1^{n,h}, \ldots, o_n^{n,h}\}$.

Evident, $S(z_{k,h})$ reprezintă familia tuturor permutărilor admisibile de prefix $\{o_1^{k,h}, \ldots, o_k^{k,h}\}$. Pentru $k=0$, se obține $S(z_{0,1}) = \boldsymbol{O}$.

5.3.5.4. Ramificarea și mărginirea

Algoritmul pe care îl propunem are abilitatea că, în cadrul procesului de construire a arborescenței corespunzătoare grafului de precedență, el ia în considerare, în fiecare etapă, doar alternativele care au șanse să conducă la permutările admisibile dătătoare de partiții optime. Această abilitate se asigură cu ajutorul unei funcții numită "*funcție cost*", definită pe mulțimea nodurilor spre care se pot realiza ramificații dintr-un nod al arborescenței deja creat, astfel încât ea să asocieze fiecărui nod argument marginea inferioară a mulțimii valorilor numărului de faze în care pot fi partiționate permutările admisibile al căror prefix se termină cu respectivul nod [ROBU'94g].

Fie "x" un nod argument. Dacă se notează cu $m_p(x)$ numărul minim de faze corespunzătoare prefixului terminat cu nodul "x" și cu $m_\rho(x)$ numărul minim de faze în care pot fi partiționate permutările posibile ale mulțimii $\rho(x)$, atunci funcția cost, pe care o notăm cu $c(\cdot)$, se poate exprima prin relația:

$$c(x) = m_p(x) + m_\rho(x) \qquad (5.3.5.4\text{-}1)$$

Termenul $m_p(x)$ se poate calcula recurent, așa cum se arată în continuare.

Se definește:

$$\tau(x) = \delta - \theta(Q_{m_p(x)}) \qquad (5.3.5.4\text{-}2)$$

disponibilul de timp al ultimei faze, $Q_{m_p(x)}$, corespunzătoare prefixului terminat cu "x", $x \in Z_k$, $k \in \{1, \ldots, n\}$. Pentru $x \in Z_0$, se va considera $m_p(x) = 1$ și, cum în acest caz $Q_{m_p(x)} = \varnothing$, va rezulta $\tau(x) = \delta$.

Se notează cu "*y*" unicul predecesor direct al nodului "*x*", presupunându-se cunoscuți $m_p(y)$ și $\tau(y)$.

Fie "*o*" marca asociată nodului *x*. Evident, $o = \rho(y) \setminus \rho(x)$. În aceste condiții, rezultă:

$$m_p(x) = \begin{cases} m_p(y), \text{ daca } \theta(o) \le \tau(y) \\ m_p(y)+1, \text{ daca } \theta(o) > \tau(y) \end{cases} \qquad (5.3.5.4\text{-}3)$$

și, totodată:

$$\tau(x) = \begin{cases} \tau(y) - \theta(o), \text{ daca } \theta(o) \le \tau(y) \\ \delta - \theta(o), \text{ daca } \theta(o) > \tau(y) \end{cases} \qquad (5.3.5.4\text{-}4)$$

Cunoscându-se durata totală a operațiilor cuprinse în $\rho(x)$, și anume: $\theta(\rho(x))$, termenul $m_\rho(x)$ poate fi determinat astfel:

$$m_\rho(x) = \begin{cases} 0, \text{ daca } \theta(\rho(x)) \le \tau(x) \\ (\theta(\rho(x)) - \tau(x))/\delta, \text{ daca } \theta(\rho(x)) > \tau(x) \end{cases} \qquad (5.3.5.4\text{-}5)$$

Evident, prima ramură a expresiei lui $m_\rho(x)$ corespunde cazului în care rezerva de timp a ultimei faze asociată prefixului este suficientă pentru ca toate operațiile din $\rho(x)$ să fie asignate acestei faze, iar a doua ramură -cazului contrar. În acesta din urmă, numărul minim de faze în care operațiile din $\rho(x)$ pot fi partiționate se obține, după cum expresia dată o sugerează, când $\theta(\rho(x)) - \tau(x)$ este multiplu de δ.

Analiza relațiilor (5.3.5.4-3), ..., (5.3.5.4-5) conduce la concluzia că $m_\rho(x)$ ia ca valori, când se aplică mulțimii nodurilor *x* ce descind din același nod *y*, pe cele date de relația:

$$m_\rho(x) = \begin{cases} 0, \text{ daca } \theta(o) \le \tau(y) \text{ si, mai mult: } \theta(\rho(y)) \le \tau(y) \\ (\theta(\rho(y) - \tau(y)))/\delta, \text{ daca } \theta(o) \le \tau(y), \text{ iar: } \theta(\rho(y)) > \tau(y) \\ 0, \text{ daca } \theta(o) > \tau(y) \text{ si, in plus: } \theta(\rho(y)) \le \delta \\ (\theta(\rho(y) - \delta)/\delta, \text{ daca } \theta(o) > \tau(y), \text{ iar: } \theta(\rho(y)) > \delta \end{cases} \qquad (5.3.5.4\text{-}7)$$

Mai departe, din (5.3.5.4-1), (5.3.5.4-3), și (5.3.5.4-6) se obține:

$$c(x) = \begin{cases} m_p(y), \text{ daca } \theta(o) \le \tau(y) \text{ si, mai mult: } \theta(\rho(y)) \le \tau(y) \\ m_p(y) + \dfrac{\theta(\rho(y))}{\delta} - \dfrac{\tau(y)}{\delta}, \text{ daca } \theta(o) \le \tau(y), \text{ iar: } \theta(\rho(y)) > \tau(y) \\ m_p(y)+1, \text{ daca } \theta(o) > \tau(y) \text{ si, in plus: } \theta(\rho(y)) \le \delta \\ m_p(y) + \dfrac{\theta(\rho(y))}{\delta}, \text{ daca } \theta(o) > \tau(y), \text{ iar: } \theta(\rho(y)) > \delta \end{cases} \qquad (5.3.5.4\text{-}7')$$

sau, mai nuanțat:

$$c(x)=\begin{cases} m_p(y),\ daca\ \theta(o)\le\tau(y)\ si\ \theta\big(\rho(y)\big)\le\tau(y) \\ m_p(y)+\dfrac{\theta\big(\rho(y)\big)}{\delta}-\dfrac{\tau(y)}{\delta},\ daca\ \theta(o)\le\tau(y)\ si\ \ \tau(y)\ <\theta\big(\rho(y)\big)\le\delta \\ m_p(y)+\dfrac{\theta\big(\rho(y)\big)}{\delta}-\dfrac{\tau(y)}{\delta},\ daca\ \theta(o)\le\tau(y)\ si\ \delta<\theta\big(\rho(y)\big) \\ -,\ daca\ \theta(o)>\tau(y)\ si\ \ \theta\big(\rho(y)\big)\le\tau(y) \\ m_p(y)+1,\ daca\ \theta(o)>\tau(y)\ si\ \ \tau(y)\ <\theta\big(\rho(y)\big)\le\delta \\ m_p(y)+\dfrac{\theta\big(\rho(y)\big)}{\delta},\ daca\ \theta(o)>\tau(y)\ si\ \ \delta<\theta\big(\rho(y)\big) \end{cases} \qquad (5.3.5.4\text{-}8)$$

Relația (5.3.5.4-8) relevă că, în procesul de construire a arborescenței, nu este necesar să se calculeze *funcția cost*, fiind suficient să se determine raportul în care se află durata operației atribuită ca marcă nodului argument, cu disponibilul de timp al ultimei faze corespunzătoare prefixului terminat cu nodul tată al nodului argument, adică: să se afle dacă operația marcă este asignabilă acestei faze sau nu [ROBU'94g].

Când se depistează existența atât a unor noduri argument pentru care $\theta(o)\le\tau(y)$, cât și a unor noduri argument pentru care $\theta(o)>\tau(y)$, acestea din urmă vor fi ignorate. Dacă se găsește fie că $\theta(o)\le\tau(y)$, fie că $\theta(o)>\tau(y)$, pentru totalitatea nodurilor argument, atunci toate acestea sunt luate, în continuare, în considerare.

Concluzia de mai sus este de o importanță practică deosebită, nu numai prin faptul că ea conduce la o micșorare a efortului de calcul implicat de algoritm, ci și prin aceea că indică un loc convenabil în care pot fi introduse restricțiile de prelucrabilitate și restricțiile de compatibilitate, ignorate până aici. Evident, acest loc este cel în care se determină dacă operația ce reprezintă marca nodului argument este sau nu asignabilă fazei curente, adică: ultimei faze corespunzătoare prefixului terminat cu nodul tată. Determinarea se face evaluând expresia logică:

$$\big(\theta(o)\le\tau(y)\big)\&\&\Big(\mu\big(Q_{m_p(y)}\big)\cap\mu(o)\ne\varnothing\Big)\&\&\Big(\nu(o)\cap Q_{m_p(y)}=\varnothing\Big) \qquad (5.3.5.4\text{-}9)$$

unde, reamintim:

- $\mu(\cdot)$ exprimă restricțiile de prelucrabilitate, fiind o funcție definită pe mulțimea operațiilor, O, și cu valori în mulțimea părților celulelor flexibile, $\boldsymbol{P}$, astfel încât să indice celulele flexibile apte să efectueze operația argument;
- $\nu(\cdot)$ exprimă restricțiile de compatibilitate, fiind o funcție definită pe mulțimea operațiilor, O, și cu valori în mulțimea părților operațiilor, $\boldsymbol{O}$, astfel încât să indice operațiile incompatibile, la nivel de fază, cu operația argument;

- $\mu\left(Q_{m_p(y)}\right) = \bigcap_{o \in Q_{m_p}(y)} \mu(o)$.

Dacă expresia (5.3.5.4-9) ia valoarea "*adevărat*", atunci operația *o* este asignabilă fazei $Q_{m_p(y)}$, altfel, nu este.

5.3.5.5. Pașii algoritmului

Algoritmul pe care îl propunem înglobează, sub o formă sau alta, toate elementele baleiate mai sus. Pentru exprimarea lui, se introduce o structură informațională asociabilă fiecărui nod al arborescenței, având numele "*NOD*" și cuprinzând [ROBU'94g]:

- un câmp, numit "*indnivt*", dedicat înregistrării indicelui nivelului tată din care descinde nodul în cauză;
- un câmp, numit "*umfp*", dedicat înregistrării ultimei mărci asignată fazei precedente;
- un câmp, numit "*umca*", dedicat înregistrării ultimei mărci curent asignată;
- un câmp, numit "*pfe*", dedicat înregistrării primei operații marcă, în ordinea lexicografică, corespunzătoare unui nod frate al nodului în cauză, egal cu acesta -prin prisma funcției cost-, adică, mai exact: sub aspectul asignabilității sau neasignabilității mărcilor lor la ultima fază corespunzătoare prefixului terminat cu nodul tată;
- un câmp, numit "*dtfc*", dedicat înregistrării disponibilului de timp al fazei curente, în urma creerii nodului în cauză;
- un câmp, numit "*lmca[]*", reprezentând un tablou unidimensional cu "*n+1*" elemente (reamintim: "*n*" este numărul de operații), dedicat implementării listei mărcilor curent asignate pe traseul pe care se situează nodul în cauză.

Se va presupune că se dispune de un tablou bidimensional de structuri de tipul *NOD*, tablou notat cu "*nod[][]*", în cadrul căruia indicele de linie arată nivelul pe care se află nodul în cadrul arborescenței, iar indicele de coloană -numărul de ordine al nodului în cadrul nivelului respectiv.

În acord cu cele expuse, algoritmul pe care îl propunem se prezintă, într-o exprimare de tip pseudocod, așa cum se arată în figura 5.3.5.5_1.

Stabilirea apartenenței sau nonapartenenței unei operații la mulțimea $\alpha(\cdot)$ se face cu algoritmul redat în figura 5.3.5.5_2.

Pentru determinarea asignabilității sau neasignabilității unei operații la faza curentă -a se vedea linia 18 din figura 5.3.5.5_1-, se propune algoritmul din figura 5.3.5.5_3.

```
1.  k[0] = 0, k[x] = -1, ∀x = 1...n;
2.  l = 0;
3.  s = 0;
4.  creat (nod[0][0]);
5.  p = &nod[0][0];
6.  p->indnivt = 0;
7.  p->umfp = 0;
8.  p->umca = 0;
9.  p->pfe = 0;`
10. p->dtfc = δ;
11. p->lmca[0] = 0, p->lmca[x] = NIMIC, ∀x=1...n;
12. j =1;
13. while (j ≤n || l ≠0) {
14.     cmin = 2;
15.     prifreg = n+1;
16.     while (j ≤n) {
17.         if (j ∈ ∝ (nod[l][s]) {
18.             if (j este asignabil la faza curenta) {
19.                 ccrt = 0;
20.             }
21.             else {
22.                 ccrt = 1;
23.             }
24.             if ((ccrt < cmin) || (ccrt ≡ cmin && prifreg ≡ n+1)) {
25.                 if (ccrt < cmin) {
26.                     jmem = j;
27.                     cmin = ccrt;
28.                     prifreg = n+1;
29.                 }
30.                 else {
31.                     prifreg = j;
32.                     if (cmin ≡ 0) {
33.                         break;
34.                     }
35.                 }
36.             }
37.         }
38.         j++;
39.     }
```

Fig. 5.3.5.5_1. Expresia pseudocod a algoritmului de partiționare echilibrată a procesului tehnologic în faze utilizând tehnicile "*branch and bound*" și "*backtracking*" (partea 1/2).

```
40.     if (cmin ≠ 2) {
41.         l++;
42.         k[l]++;
43.         creat (nod[l][k[l]]);
44.         p = &nod[l][k[l]];
45.         p->indnivt = s;
46.         p->umfp = nod[l-1][s].umfp;
47.         p->umca = jmem;
48.         p->pfe = prifreg;
49.         p->dtfc = nod[l-1][s].dtfc;
50.         p->lmca[x] = nod[l-1][s].lmca[x], ∀x = 1...n;
51.         p->lmca[nod[l-1][s].lmca[x], ∀x = 1...n;
52.         p->lmca[jmem] = jmem;
53.         if (cmin) ≡ 0) {
54.             p->dtfc -= θ(jmem);
55.         }
56.         else {
57.             p->dtfc = δ - θ(jmem);
58.             p->umfp = nod [l-1][s].umca;
59.         }
60.         s = k[l];
61.         j = 1;
62.     }
63.     else {
64.         l--;
65.         s = p->indnivt;
66.         j = p->pfe;
67.         p = &nod[l][s];
68.     }
69. }
```

Fig. 5.3.5.5_1. Expresia pseudocod a algoritmului de partiționare echilibrată a procesului tehnologic în faze utilizând tehnicile "*branch and bound*" și "*backtracking*" (partea 2/2).

```
1.  /* j: operația a cărei apartenență sau nonapartenență
2.                                          la α(nod[l][s]) prezintă interes */
3.  /*  p: pointerul către nod[l][s] */
4.  apart = FALSE;
```

Fig. 5.3.5.5_2. Cum se determină dacă o operație este sau nu este membru în mulțimea $\alpha(\cdot)$ (partea 1/2).

```
5.  if (p->lmca[j] ≡ NIMIC) {
6.      apart = TRUE;
7.      for (i = 1; i ≤ n; i++) {
8.          if (G[i][j] ≠ 0) {
9.              h = 0;
10.             while ((h ≠ p->lmca[h]) && (i ≠ p->lmca[h])) {
11.                 h = p->lmca[h];
12.             }
13.             if (h ≡ p->lmca[h]) {
14.                 apart = FALSE;  /* j ∉ α(nod[l][s]) */
15.                 break;
16.             }
17.         }
18.     }
19. }
```

Fig. 5.3.5.5_2. Cum se determină dacă o operație este sau nu este membru în mulțimea α(·) (partea 2/2).

```
1.  eas = FALSE /* j nu este asignabil la faza curentă */
2.  if (θ(j) ≤ p->dtfc) {
3.      eas = TRUE /* j este asignabil la faza curentă */
4.      μ = μ(j)
5.      h = p->umfp[h]
6.      while ((p->lmca[h] ≠ h) && (p->lmca[h] ∉ v(j)) &&
7.                                  (μ ∩ μ( p->lmca[h]) ≠ ∅)) {
8.          μ = μ ∩ μ( p->lmca[h]
9.          h = p->lmca[h]
10.     }
11      if (p->lmca[h] ≠ h) {
12.         eas = FALSE /* j nu este asignabil la faza curentă */
13.     }
14. }
```

Fig.5.3.5.5_3. Expresia pseudocod a algoritmului de determinare a asignabilității/nonasignabilității unei operații la faza curentă.

5.3.5.6. Concluzii

Algoritmul propus are ca avantaje faptele că este exact și, în cazurile reale, găsește soluția optimă luând în considerare doar o mică parte din arborescența ce ar corespunde grafului de precedență. Ori, asta înseamnă economie de memorie și de timp de calcul. De asemenea, este de remarcat valoarea de întrebuințare imediată pe care algoritmul o are, întrucât, ca și algoritmii prezentați în paragrafele precedente, abordează problema partiționării de pe poziții inginerești, cu considerarea, alături de restricțiile comune -de precedență și de ritm-, și a restricțiilor de prelucrabilitate și de compatibilitate.

5.3.6. Algoritm de partiționare echilibrată a procesului tehnologic în faze utilizând direct graful de precedență și tehnica euristică "*greedy*", aplicabil în timp real

5.3.6.1. Preliminarii

Algoritmii prezentați pentru partiționarea echilibrată a proceselor tehnologice în faze și alții, semnalați în literatură [ADAM'88][BAPT'91b][CHRE'91], exacți, ca și ei -unii chiar exhaustivi-, au drept caracteristici comune faptele că presupun un volum de memorie substanțial și că necesită un timp de rulare ridicat. Aplicarea lor în cazul unor procese tehnologice complexe, cu un număr de operații de ordinul sutelor, era, până nu demult, de neimaginat. Chiar și în prezent, doar calculatoarele superputernice pot da satisfacție la un astfel de proces. Lucrările fundamentale de algoritmică arată că nu se cunosc algoritmi exacți capabili să rezolve probleme de tipul celei avută în vedere într-un timp limitat polinomial de dimensiunea problemei [RUSU'90].

Drept consecință, în paralel cu algoritmii exacți, au fost elaborați algoritmi euristici [ERSC'81a] [RADH'86] [ROUB'87][DEWE'87] [WIDM'87b], care, cu prețul neasigurării certitudinii că soluțiile pe care le oferă sunt optime, dau satisfacție în ceea ce privește implementabilitatea pe calculatoare la îndemână, în condițiile unor timpi de rulare acceptabili. Experimentarea algoritmilor euristici pe un număr mare de cazuri tehnologice industriale a relevat concluzia că soluțiile la care ei conduc nu sunt considerabil distanțate de cele optime, ci, dimpotrivă, nu arareori, sunt chiar cele optime.

Literatura evidențiază că cele mai adecvate abordări euristice ale problemei partiționării echilibrate a proceselor tehnologice în faze sunt cele de tip "*greedy*" [RUSU'90]. În cele ce urmează, se va prezenta un algoritm rezultat dintr-o asemenea abordare, ținând seamă de toate restricțiile considerate până aici [ROBU'94h].

5.3.6.2. Principiile abordării "*greedy*"

Abordarea "*greedy*" are ca esență construirea rând pe rând a fazelor, prin asignarea câte unei operații la faza aflată în construcție, pe baza unei liste de asignare. Aceasta cuprinde operațiile asignabile la un moment. Reamintim: o operație este asignabilă la un moment dacă nu are nici un predecesor direct sau dacă are predecesori direcți, dar, la momentul respectiv, toți aceștia sunt deja asignați. Lista de asignare se menține ordonată descedent, pe baza unui punctaj dat de valoarea unei funcții, "*w*", de sorginte empirică, numită "*măsură de optimizare*":

$$w: O \to R^+$$

Procesul de construire a fazelor se derulează astfel:

- dacă în lista de asignare există operații care satisfac toate restricțiile -de prelucrabilitate, de compatibilitate, și de ritm-, relative la faza curent ultima, atunci se alege dintre ele aceea care este cea mai apropiată de capul listei și se asignează acestei faze; totodată, respectiva operație se elimină din listă.
- dacă în lista de asignare nu există nici o operație care satisface toate restricțiile -de prelucrabilitate, de compatibilitate, și de ritm-, relative la faza curent ultima, atunci se creează o nouă fază, inițializată cu operația aflată în cazul listei; totodată, respectiva operație se elimină din listă.

5.3.6.3. Măsurile de optimizare

Măsurile de optimizare considerate sunt cele pe care practica ordonanțării le-a calificat ca cele mai eficiente, și anume:

1). $w(o) = \theta(o)$

2). $w(o) = \theta(o) + \sum_{j \in \pi(o)} w(j)$

3). $w(o) = (\delta - \theta(o)) \ (\theta(o) + \sum_{j \in \pi(o)} w(j))$

4). $w(o) = \theta(o) \ (\theta(o) + \sum_{j \in \pi(o)} w(j))$

unde, reamintim:

$$\pi(o) = \{x \mid x \in O, (x, o) \in I^+\}$$

Corespunzător lor, se folosesc respectiv, sintagmele:

- regula candidatului cu timp maxim, "*RCTM*";
- regula candidatului cu pondere poziţională maximă, "*RCPPM*";
- regula candidatului cu deviaţie ponderată maximă, "*RCDPM*";
- regula candidatului cu timp ponderat maxim, "*RCTPM*".

5.3.6.4. Calculul măsurilor de optimizare

În cele ce urmează, se propune un algoritm pentru calculul măsurilor de optimizare, având înglobate, ca opţiuni, toate aceste patru reguli. Evident, regula *RCTM* corespunde cazului trivial. Pentru cazul celorlalte trei reguli, se procedează aşa cum se arată în continuare.

Se reprezintă graful de precedenţă al procesului tehnologic printr-un set de liste, fiecare corespunzând unui nod şi cuprinzând succesorii direcţi ai nodului respectiv. Fie "*lsd[x]*", cu *x=1...n*, apelativii formali ai acestor liste. Se ţine, într-un tablou "*npnp[x]*", cu *x=1...n*, evidenţa numărului predecesorilor direcţi încă neponderaţi ai fiecărei operaţii, iar într-o listă, "*lonp*", evidenţa operaţiilor încă neponderate. Evident, pentru început, numărul predecesorilor direcţi încă neponderaţi ai unei operaţii coincide cu cardinalul mulţimii predecesorilor direcţi ai respectivei operaţii, iar lista operaţiilor încă neponderate cuprinde toate operaţiile. Se desfăşoară un proces de calcul în două etape [ROBU'94h].

1) În prima etapă, pentru toate operaţiile, se fac măsurile de optimizare egale cu duratele.

2) Etapa a doua, cu conţinut pentru toate cele trei reguli diferite de *RCTM* şi numai pentru ele, ţine cât timp lista operaţiilor neponderate este nevidă şi consistă în trei subetape.

   - În prima subetapă, pentru operaţia aflată la rând -aceasta este prima din listă care nu are nici un predecesor direct încă neponderat-, se calculează ponderea poziţională şi se atribuie măsurii de optimizare.
   - Subetapa a doua este nevidă doar pentru regulile *RCDPM* şi *RCTPM*. În cadrul ei, se amendează valoarea măsurii de optimizare moştenită din subetapa întâi, pe baza formulelor specifice acestor două reguli.
   - În subetapa a treia, se elimină din listă operaţia tocmai ponderată şi apoi se micşorează cu o unitate numărul predecesorilor direcţi încă neponderaţi, pentru toţi succesorii ei direcţi. În acest fel, ţinând seamă că graful de precedenţă este aciclic, cel puţin o nouă operaţie ajunge fără nici un predecesor direct încă neponderat.

În vederea exprimării algoritmului la nivel de pseudocod, vom presupune că pentru operarea asupra listelor referite se dispune de următoarele funcţii: "*test()*", "*first()*", "*last()*", "*next()*", "*eliminate()*".

- Funcţia *test()* verifică dacă lista este vidă sau nevidă şi returnează o valoare nulă, în primul caz, şi nenulă, în cazul al doilea.

- Funcția *first(·)* indică operația aflată în capul listei.
- Funcția *last(·)* indică operația aflată în coada listei.
- Funcția *next(·)* indică operația aflată în listă imediat după cea curentă, atribuind, totodată, acelei operații, calitatea de nouă operație curentă.
- Funcția *eliminate(·, ·)* exclude din lista indicată de primul argument elementul specificat prin al doilea argument.

Notând măsurile de optimizare corespunzătoare celor patru reguli -*RCTM, RCPPM, RCDPM, RCTPM*- cu "*w[1][·]*", "*w[2][·]*", "*w[3][·]*", "*w[4][·]*", respectiv, algoritmul pentru calculul lor se prezintă așa cum se arată în figura 5.3.6.4_1.

```
1.  /* inițial: o ∈ lonp, ∀o ∈ O, și npnp[o] = card (π(o)), ∀o ∈ O */
2.  for (i = 1; i ≤ n; i++) {
3.      w[1][i] = θ(i);
4.      w[2][i] = θ(i);
5.      w[3][i] = θ(i);
6.      w[4][i] = θ(i);
7.  }
8.  while (test(lonp)) {
9.      o = first(lonp);
10.     while (npnp[o] ≠ 0) {
11.         o = next(lonp);
12.     }
13.     for (x = 1; x ≤ n; x++) {
14.         if (test(lsd[x])) {
15.             i = first(lsd[x]);
16.             k = last(lsd[x]);
17.             do {
18.                 j = i;
19.                 if (j ≡ o) {
20.                     w[2][o] += w[2][x];
21.                     w[3][o] += w[3][x];
22.                     w[4][o] += w[4][x];
23.                     break;
24.                 }
25.                 i = next(lsd[x]);
26.             } while (j ≠ k);
27.         }
28.     }
```

Fig. 5.3.6.4_1. Algoritmul de calcul al măsurilor de optimizare (partea 1/2).

```
29.      w[3][o] = (δ - θ(o)) / w[3][o];
30.      w[4][o] = (δ - θ(o)) / w[4][o];
31.      eliminate(lonp, o);
32.      if (test(lsd[o]) {
33.          i = first(lsd[o]);
34.          k = last(lsd[o]);
35.          do {
36.              j = i;
37.              npnp[j]--;
38.              i = next(lsd[o]);
39.          } while (j ≠ k);
40.      }
41.  }
```

Fig. 5.3.6.4_1. Algoritmul de calcul al măsurilor de optimizare (partea 2/2).

5.3.6.5. Pașii algoritmului

Algoritmul pe care îl propunem partiționează procesul tehnologic în faze folosind, succesiv, fiecare dintre cele patru reguli de ponderare a operațiilor, menționate. În final, el oferă partiția de cea mai bună eficiență pe care ansamblul celor patru reguli o poate garanta. Prin *"eficiență"*, înțelegem, aici, mărimea dată de expresia [ROBU'94h]:

$$e = 1 - \frac{tn}{m * \delta} \qquad (5.3.6.5\text{-}1)$$

unde:

"*tn*" este timpul de neutilizare a capacităților de lucru ale resurselor cumulat pe ansamblul fazelor;

iar:

"*m*" este numărul de faze.

Algoritmul de partiționare mizează pe aceeași reprezentare a grafului de precedență ca și algoritmul de calcul al măsurilor de optimizare. De asemenea, el utilizează un tablou, *npna[x]*, cu $x = 1...n$, pentru a ține evidența numărului predecesorilor direcți încă neasignați ai fiecărei operații. Suplimentar, sunt gestionate alte două liste: lista operațiilor neasignabile, "*lona*", respectiv lista operațiilor asignabile, "*loa*", numită, de asemenea, listă de asignare. Pentru început, numărul predecesorilor direcți încă neasignați ai unei operații se face să

coincidă cu cardinalul mulțimii predecesorilor direcți ai respectivei operații, iar în lista operațiilor neasignabile se includ toate operațiile, în ordinea descreșterii măsurilor lor de optimizare, date de regula curentă, în timp ce lista operațiilor asignabile este vidă.

În prima sa parte, algoritmul identifică în lista *lona* operațiile devenite fără nici un predecesor direct încă neasignat -adică: operațiile asignabile-, le elimină din această listă, și le inserează în lista *loa*, la locul corespunzător măsurii lor de optimizare (așa cum s-a menționat în paragraful 5.3.6.1., lista operațiilor asignabile este ordonată descendent, după măsura de optimizare).

În a doua parte a algoritmului, se caută în lista *loa*, de la cap spre coadă, o operație care satisface toate restricțiile -de prelucrabilitate, de compatibilitate, și de ritm-, relative la faza curent ultima. Când o asemenea operație este găsită, ea se asignează respectivei faze și a doua parte a algoritmului se încheie. Dacă nici o operație din lista *loa* nu satisface toate restricțiile relative la faza curent ultima, atunci se creează o nouă fază, inițializată cu operația aflată în capul listei, iar după aceea, a doua parte a algoritmului se încheie.

În partea a treia a algoritmului, se elimină din lista *loa* operația tocmai asignată, iar apoi se micșorează cu o unitate numărul predecesorilor direcți încă neasignați pentru toți succesorii direcți ai operației în cauză. În acest fel, ținând seamă de faptul că graful de precedență al procesului tehnologic este aciclic, cel puțin o operație ajunge fără nici un predecesor direct încă neasignat, adică: asignabilă.

Procesul se reia, începând cu acțiunea asupra listei *lona*, cât timp mai există operații neasignate. Când acestea se epuizează, se calculează, corespunzător partiției la care a condus regula curentă, timpul de neutilizare a capacităților de lucru ale resurselor, *tn*, și, cu ajutorul lui, eficiența partiției, *ep*. Dacă această eficiență este mai bună decât a tuturor partițiilor date de regulile precedente, respectiv, în cazul primei reguli, decât eficiența inițială, considerată nulă, atunci ea se reține, ca atare, în variabila *e*. Totodată, se reține fazele acestei partiții, sub numele "*Q[i]*", numărul fazelor, sub numele "*m*" (evident, $i = 1...m$), și faptul că regula curentă este, cel puțin temporar, cea mai bună; reținerea acestui fapt se face înregistrând indexul regulii în variabila "*rcmb*". Dacă regula curentă nu conduce la o partiție de o eficiență mai bună decât a tuturor celor care au precedat-o, atunci rezultatele aplicării ei se ignoră.

În vederea exprimării algoritmului la nivel de pseudocod, vom considera că, pentru operarea asupra listelor, se dispune de funcțiile clasice: "*test()*", "*first()*", "*last()*", "*next()*", "*eliminate()*", "*insert()*". Primele cinci au fost definite mai sus, iar:

- Funcția *insert(·, ·, ·)* adaugă listei indicată de primul argument elementul specificat prin al doilea argument, în poziția corespunzătoare măsurii de optimizare dată prin al treilea argument.

Figura 5.3.6.5_1 exprimă algoritmul în ton cu cele de mai sus.

```
1.  m = 1;
2.  Q[1] =∅;
3.  e = 0;
4.  for (x = 1; x ≤ 4; x++) {
5.      for (o = 1; o ≤ n; o++) {
6.          insert(lona, o, w[x][o]);
7.          npna[o] = card(π(o));
8.      }
9.      noa = 0;
10.     mp = 1;
11.     Qp[1] =∅;
12.     dtuf = δ;
13.     tn = 0;
14.     while (noa ≠ n) {
15.         i = first(lona);
16.         k = last(lona);
17.         do {
18.             j = i;
19.             if (npna[j] ≡ 0) {
20.                 eliminate(lona, j);
21.                 insert(loa, j, w[x][j]);
22.             }
23.             i = next(lona);
24.         } while (j ≠ k);
25.         i = first (loa);
26.         k = last (loa);
27.         o = i;
28.         stq = 0;
29.         do {
30.             j = i;
31.             if (j este asignabil la ultima faza) {
32.                 o = j;
33.                 stq = 1;
34.                 break;
35.             }
36.             i = next(loa);
37.         } while (j ≠ k);
38.         if (stq) {
39.             Qp[mp] = Qp[mp]∪{o};
40.             dtuf -= θ(o);
41.         }
```

Fig. 5.3.6.5_1. Expresia pseudocod a algoritmului de partiționare echilibrată a procesului tehnologic în faze utilizând direct graful de precedență și tehnica euristică "*greedy*" (partea 1/2).

```
42.          else {
43.             tn += dtuf;
44.             mp++;
45.             Qp[mp] = {o};
46.             dtuf = δ - θ(o);
47.          }
48.          eliminate(loa, o);
49.          if (test(lsd[o])) {
50.             i = first(lsd[o]);
51.             k = last(lsd[o]);
52.             do {
53.                j = i;
54.                npna[j]--;
55.                i = next(lsd[o]);
56.             } while (j ≠ k);
57.          }
58.          noa++;
59.       }
60.       tn += dtuf;
61.       ep = 1 - tn / (mp * δ);
62.       if (ep > e) {
63.          rcmb = x;
64.          e = ep;
65.          for (i = 1; i ≤ m; i++) {
66.             destroy Q[i];
67.          }
68.          m = mp;
69.          for (i = 1; i ≤ m; i++) {
70.             Q[i] = Qp[i];
71.          }
72.       }
73.    }
```

Fig. 5.3.6.5_1. Expresia pseudocod a algoritmului de partiționare echilibrată a procesului tehnologic în faze utilizând direct graful de precedență și tehnica euristică "*greedy*" (partea 2/2).

5.3.6.6. Concluzii

Algoritmul propus are ca avantaje faptele că oferă soluții care sunt fie optime, fie apropiate de cele optime și că necesită volum mic de memorie, în plus, caracterizându-se printr-un timp de rulare redus. Dintre toți algoritmii prezentați, acesta este singurul aplicabil în timp real. Ori, așa cum a fost precizat pe parcursul lucrării, tendințele în domeniu sunt

orientate înspre transferul în zona timpului real a rezolvării cât mai multora dintre sarcinile de conducere, dar, în orice caz, a celor privind ordonanțarea[ROUB'87]. De asemenea, este remarcabilă valoarea de întrebuințare imediată pe care algoritmul o are, conferită fiindu-i de considerarea tuturor restricțiilor de interes tehnic.

5.3.7. Algoritm de acoperire optimă a fazelor procesului de producție cu celule flexibile

5.3.7.1. Macropașii algoritmului

La nivel macro, algoritmul pe care îl propunem consistă în următorii pași [ROBU'95b]:

r_1). se determină, pentru fiecare dintre partițiile optime, mulțimile celulelor flexibile asignabile fazelor lor, mulțimi notate cu "C_k^x":

$$C_k^x = \mu\left(Q_k^x\right),\ k = 1...m_x,\ x = 1...v \tag{5.3.7.1-1}$$

unde:

$$\mu\left(Q_k^x\right) = \bigcap_{o \in Q_k^x} \mu(o) \tag{5.3.7.1-2}$$

și se consideră, prin definiție:

$$C_0^x = \{DI\},\ \forall x = 1...v \tag{5.3.7.1-3}$$

$$C_{m_x+1}^x = \{DE\},\ \forall x = 1...v \tag{5.3.7.1-4}$$

r_2). Se construiește, pentru fiecare valoare a lui x, un digraf pe care îl numim "*graful de acoperire*" și îl notăm cu "A^x":

$$A^x = \left(P,\ L^x\right) \tag{5.3.7.1-5}$$

unde, reamintim:

- P este mulțimea celulelor flexibile, incluzând depozitul de piese de la intrare și depozitul de piese de la ieșire $\left(P = \left\{p_0, p_1, \ldots, p_{q+1}\right\}\right)$;

iar:

$$L^x = \left\{ \left(p_i, p_j\right) \middle| p_i, p_j \in P;\ \left(p_i, p_j\right) \in T;\right.$$
$$\left. \exists\, \underset{k\geq 1}{k} \middle| p_i \in C_k^x \Rightarrow \exists\, p_h \in C_{k-1}^x \middle| \left(p_h, p_i\right) \in L^x;\ \exists\, \underset{k\geq 0}{k} \middle| p_i \in C_k^x,\ p_j \in C_{k+1}^x \right\} \quad (5.3.7.1\text{-}6)$$

atribuind arcelor aceleași orientări și aceleași lungimi pe care ele le au în graful fluxului pieselor, F (reamintim: $F = (P, T)$).

r3). Modul în care digrafurile A^x au fost construite permite determinarea acoperirii optime a mulțimii fazelor procesului tehnologic cu celule flexibile, o acoperire corespunzând unui drum care pleacă din nodul aferent depozitului de piese de la intrare, are ca punct terminus nodul aferent depozitului de piese de la ieșire, și trece prin câte un nod pentru fiecare fază, cu respectarea ordinii tehnologice în care fazele trebuie să se succeadă. Vom spune că un asemenea drum este un *"drum acoperitor"*. Evident, acoperirea optimă corespunde drumului pentru care cea mai mare distanță dintre două noduri consecutive este minimă.

În cazul în care există mai multe drumuri nediscriminabile sub acest aspect, acoperirea optimă se obține prin oprirea aleatoare sau după un anumit criteriu -obiectiv sau subiectiv- asupra unuia dintre ele.

Convenim să desemnăm această acoperire prin notația:

$$DI \rightarrow c_1^x \rightarrow \ \ldots\ \rightarrow c_{m_x}^x \rightarrow DE \quad (5.3.7.1\text{-}7)$$

5.3.7.2. Determinarea mulțimilor celulelor flexibile asignabile fazelor procesului tehnologic

Așa cum s-a arătat în paragraful precedent, mulțimile celulelor flexibile asignabile fazelor unei partiții x a procesului tehnologic se obțin cu relația:

$$C_k^x = \mu\left(Q_k^x\right),\ k = 1...m_x,\ x = 1...v \quad (5.3.7.2\text{-}1)$$

Pentru reprezentarea familiilor de asemenea mulțimi corespunzătoare câte unei partiții, se poate folosi -și facem uz de această posibilitate- câte un tablou bidimensional, "$S^x[][]$" (evident: $x = 1...v$). Un asemenea tablou -pe care îl numim "*tablou de asignabilitate*"- va avea un număr de linii egal cu numărul celulelor flexibile (considerăm incluse între acestea și depozitele de piese de la intrare, respectiv de la ieșire) și un număr de coloane egal cu numărul fazelor partiției căreia îi corespunde (reamintim: fazele *0* și m_x+1 sunt, prin definiție, asociate depozitelor *DI*, respectiv *DE*), adică: va fi cu $q+2$ linii (reamintim: numărul celulelor flexibile propriu-zise l-am notat cu q) și m_x+2 coloane. Un element al tabloului cu indicele de linie egal cu i și indicele de coloană egal cu j va avea valoarea *1*, dacă celula i este asignabilă fazei Q_j^x, iar valoarea *0*, altfel.

5.3.7.3. Construcția grafului de acoperire

După cum s-a văzut în paragraful precedent, graful de acoperire al unei partiții are drept noduri celulele flexibile, a căror mulțime (incluzând depozitele *DI* și *DE*) am notat-o cu *P*, și dispune de arce de la un nod p_i la un nod p_j, dacă:

$$a).\ (p_i, p_j) \in T \tag{5.3.7.3-1}$$

$$b).\ \exists\, k_{k\geq 1} \,|\, p_i \in C_k^x \Rightarrow \exists\, p_h \in C_{k-1}^x \,|\, (p_h, p_i) \in L^x \tag{5.3.7.3-2}$$

$$c)\ \exists\, k_{k\geq 0} \,|\, p_i \in C_k^x,\ p_j \in C_{k+1}^x \tag{5.3.7.3-3}$$

aceste condiții reprezentând, respectiv:

a). restricțiile de transport;
b). cerința de conectivitate în amonte;
c). cerința de conectivitate în aval.

Pentru reprezentarea grafurilor de acoperire, vom folosi câte un tablou bidimensional pătratic, "$A^x[][]$" (evident: $x = 1...v$), pe post de matrice de incidență. Tablourile $A^x[][]$ vor avea $q+2$ linii și $q+2$ coloane.

Un element al tabloului cu indicele de linie egal cu i și indicele de coloană egal cu j va avea valoarea *1*, dacă de la celula i se poate ajunge la celula j și, în plus, sunt satisfăcute cele două cerințe de conectivitate (respectiv doar cea de-a doua, dacă $i = 0$), iar valoarea *0*, altfel.

5.3.7.4. Determinarea drumurilor acoperitoare optime

Așa cum s-a arătat în paragraful 5.3.7.1, acoperirile optime ale fazelor procesului tehnologic cu stații de lucru se pot obține din digrafurile A^x, prin determinarea drumurilor acoperitoare optime pe care ele le conțin.

Algoritmul pe care îl propunem pentru determinarea drumurilor acoperitoare optime este de tip "*backtracking*". Pentru a-l defini, vom considera că fiecare nod al unui graf de acoperire are asociabilă o structură informațională care îl poate caracteriza ca element al unui traseu acoperitor. Denumim "*ETA*" această structură. Ea cuprinde:

- un câmp, numit "*indulteltrt*", dedicat înregistrării indexului ultimului element al traseului tată;
- un câmp, numit "*indtrt*", dedicat înregistrării indexului traseului tată;
- un câmp, numit "*indndeltr*", dedicat înregistrării indexului nodului corespunzător elementului de traseu în cauză;
- un câmp, numit "*fzasndeltr*", dedicat înregistrării fazei asignate nodului corespunzător elementului de traseu în cauză;
- un câmp, numit "*distmaxtr*", dedicat înregistrării distanței maxime dintre două noduri aflate pe traseul în cauză;
- un câmp, reprezentând un tablou unidimensional cu "$q+2$" elemente (reamintim: "$q+2$" este cardinalul mulțimii celulelor flexibile extinsă cu cele două depozite), numit "*nodtr[]*", dedicat înregistrării nodurilor traseului în cauză și a ordinii în care ele se succed, în convenția: un element de indice "*j*" corespunde nodului aferent celulei flexibile p_j și ia ca valoare indicele elementului corespunzător succesorului acestui nod. Elementele tabloului *nodtr[]* au, deasemenea, rolul de a arăta dacă celulele corespunzătoare lor sunt disponibile sau nu, conform convenției: dacă valoarea unui element este *NIMIC* (reamintim: *NIMIC* este variabila cu toți biții pe *1*), atunci celula aferentă este disponibilă, altfel -nu este. Inițial, tabloul va avea toate elementele la valoarea *NIMIC*.

Întru aceeași idee, se va conta pe existența unui tablou bidimensional de structuri de tipul *ETA*, referit prin apelativul "*eta[][]*", în cadrul căruia indicele de linie reprezintă indexul elementului de traseu aferent în cadrul traseului, iar indicele de coloană -indexul traseului.

Drumurile acoperitoare optime se vor oferi printr-un tablou de tablouri de același tip cu tabloul *nodtr[]* menționat mai sus. Acest tablou de tablouri va fi referit prin apelativul "*acopt[][]*", în care indicele de linie reprezintă indexul drumului optim, în timp ce indicele de coloană desemnează tabloul prin care se definește traseul corespunzător drumului optim de respectivul index.

Se face convenția ca situația -evident: anormală- în care între nodul corespunzător depozitului de piese de la intrare, p_0 ($p_0 = DI$) și nodul corespunzător depozitului de piese de la ieșire, p_{q+1} ($p_{q+1} = DE$) nu există nici un traseu acoperitor să fie semnalată de algoritm prin returnarea valorii "-1" pentru indicele de linie aferent tabloului *acopt[][]*.

Distanța maximă curentă dintre două noduri se notează cu "*distmaxcrt*", iar distanța maximă pe trasee complete (adică: pe drumuri acoperitoare) cea mai mică obținută până la un moment se notează cu "*distmaxmin*".

Ținând seamă de cele de mai sus, algoritmul se poate exprima prin pseudocodul din figura 5.3.7.4_1.

```
1.  distmaxcrt = 0;
2.  distmaxmin = SUPREPR;
3.  k = -1;
4.  incautareramif = 0;
5.  r = 0; /* r: o variabilă locală, auxiliară (fără legătură cu card(W)!) */
6.  h = 0;
7.  t = 0;
8.  creat (n[0]); /* n: variabilă tablou locală, auxiliară (fără legătură cu card(O))
9.  n[0] = -1;
10. f = 1;
11. i = 0;
12. j = 1;
13. creat (eta[0][0]);
14. p = &eta[0][0];
15. p->indulteltrt = 0;
16. p->indtrt = 0;
17. p->indndeltr = 0;
18. p->fzasndeltr = 0;
19. p->distmaxtr = 0;
20. p->nodtr[x] = NIMIC, ∀x = 1...q+1;
21. p->nodtr[0] = 0;
22. while (j ≤ q+1 || i ≠ 0) {
23.     if (f ≤ m+1 && j ≤ q+1) {
24.         if ((A[i][j] ≠ SUPREPR) && S[j][f] && (eta[r][h].nodtr[j] ≡ NIMIC)) {
25.             if (distmaxcrt < A[i][j]) {
26.                 distmaxcrt = A[i][j];
27.             }
28.             if (incautareramif) {
29.                 t++;
30.                 creat (n[t]);
31.                 n[t] = -1;
32.                 incautareramif = FALS;
33.             }
34.             n[t]++;
35.             creat (eta[n[t]][t]);
36.             p = &eta[n[t]][t];
37.             p->indulteltrt = r;
38.             p->indtrt = h;
39.             p->indndeltr = j;
40.             p->fzasndeltr = f;
41.             p->distmaxtr = distmaxcrt;
42.             p->nodtr[x] = eta[r][h].nodtr[x], ∀x = 0...r /x ≠ i x ≠ j;
43.             p->nodtr[i] = j;
44.             p->nodtr[j] = j;
```

Fig. 5.3.7.4_1. Expresia pseudocod a algoritmului de determinare a drumurilor acoperitoare optime (partea 1/2).

```
45.             f++;
46.             r = n[t];
47.             h = t;
48.             i = j;
49.             j = -1;
50.         }
51.         j++;
52.     }
53.     else {
54.         if(!incautareramif) {
55.             if (f > m+1) {
56.                 if (distmaxcrt ≤ distmaxmin) {
57.                     if (distmaxcrt < distmaxmin) {
58.                         distmaxmin = distmaxcrt;
59.                         if (k ≠ -1) {
60.                             destroy (acopt[y][x]), ∀y = 0...k, x = 0...q+1;
61.                         }
62.                         k = 0;
63.                     }
64.                     else {
65.                         k++;
66.                     }
67.                     creat (acopt[k][x]), x = 0...q+1;
68.                     acopt[k][x] = p->nodtr[x], x = 0...q+1;
69.                 }
70.                 f--;
71.                 r = p->indulteltrt;
72.                 h = p->indtrt;
73.                 destroy(*p);
74.                 p = &eta[r][h];
75.             }
76.             incautareramif = ADEVARAT;
77.         }
78.         else {
79.             p = &eta[r][h];
80.         }
81.         f--;
82.         r = p->indulteltrt;
83.         h = p->indtrt;
84.         i = eta[r][h].indndeltr;
85.         j = p->indndeltr+1;
86.         distmaxcrt = eta[r][h].distmaxtr;
87.     }
88. }
```

Fig. 5.3.7.4_1. Expresia pseudocod a algoritmului de determinare a drumurilor acoperitoare optime (partea 2/2).

5.4. Ilustrarea formalismului de punere și rezolvare a problemei ordonanțării prin aplicare la un caz de studiu

5.4.1. Ilustrarea punerii problemei ordonanțării

Se dau:

i_1). operațiile în care consistă procesul tehnologic:

$O = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

i_2). duratele operațiilor în care consistă procesul tehnologic:

$\theta(0) = 2.5$ $\quad$ $\theta(5) = 1.5$
$\theta(1) = 2$ $\quad$ $\theta(6) = 3.5$
$\theta(2) = 3$ $\quad$ $\theta(7) = 2.5$
$\theta(3) = 1.25$ $\quad$ $\theta(8) = 1$
$\theta(4) = 1$ $\quad$ $\theta(9) = 1.5$

Este convenabil ca duratele operațiilor să fie ținute în evidență cu ajutorul unui tablou unidimensional, așa cum se arată în figura 5.4_1.

0	2.5
1	2
2	3
3	1.25
4	1
5	1.5
6	3.5
7	2.5
8	1
9	1.5

D[]

Fig. 5.4_1. Tabloul duratelor operațiilor procesului tehnologic.

i_3). restricțiile de precedență pentru operațiile procesului tehnologic:

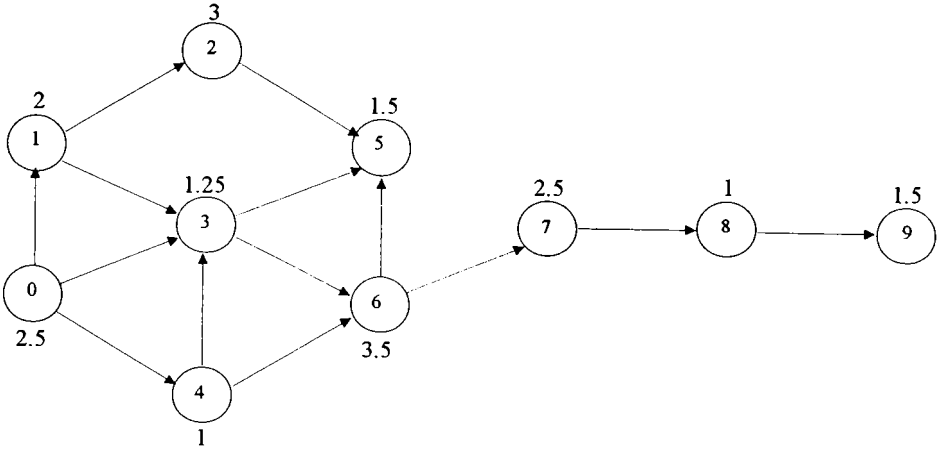


Fig. 5.4.1_2. Digraful de precedență a operațiilor procesului tehnologic.

Matricea de incidență corespunzătoare grafului de precedență este:

	0	1	2	3	4	5	6	7	8	9
0		1	1	1	1					
1			1	1						
2						1				
3						1	1			
4				1			1			
5							1			
6								1		
7									1	
8										1
9										

G[][]

Fig. 5.4.1_3. Matricea de incidență a grafului din figura 5.4.1_2.

i_4). **celulele flexibile disponibile** în sistemul de fabricație:

$P = \{0, 1, 2, 3, 4, 5, 6\}$

unde, reamintim:

- "*0*" reprezintă depozitul de piese de la intrare;
- "*i*", $1 \le i \le q$, reprezintă celula flexibilă propriu-zisă c_i;
- "*q+1*" reprezintă depozitul de piese de la ieșire.

i_5). restricțiile de prelucrabilitate a operațiilor procesului tehnologic:

$\mu(0) = \{1, 2, 3, 5\}$ $\quad$ $\mu(5) = \{2, 3, 4\}$
$\mu(1) = \{1, 3, 4\}$ $\quad$ $\mu(6) = \{2, 3, 4\}$
$\mu(2) = \{2, 5\}$ $\quad$ $\mu(7) = \{2, 4, 5\}$
$\mu(3) = \{1, 3, 4\}$ $\quad$ $\mu(8) = \{2, 3, 5\}$
$\mu(4) = \{1, 2, 3, 4\}$ $\quad$ $\mu(9) = \{2, 4, 5\}$

Pentru ținerea în evidență a restricțiilor de prelucrabilitate, poate fi folosit un tablou bidimensional, conținând un număr de linii egal cu numărul operațiilor și un număr de coloane egal cu numărul celulelor flexibile (incluzând depozitele). Se va adopta convenția ca un element al tabloului cu indicele de linie egal cu "*i*" și cu indicele de coloană egal cu "*j*" să aibă valoarea *1* dacă operația *i* este executabilă pe celula flexibilă *j*, iar valoarea *0* -altfel. Configurația valorilor elementelor tabloului -fie *M[][]* numele acestuia- va fi, în cazul restricțiilor de mai sus, următoarea:

	0	1	2	3	4	5	6
0		1	1	1		1	
1		1	1	1			
2			1			1	
3		1		1	1		
4		1	1	1	1		
5			1	1	1		
6			1	1	1		
7			1		1	1	
8			1	1		1	
9			1		1	1	

M[][]

Fig. 5.4.1_4. Tabloul restricțiilor de prelucrabilitate.

i_6). restricțiile de compatibilitate a operațiilor procesului tehnologic:

$v(0) = \{1\}$ $\quad$ $v(5) = \{4, 7\}$
$v(1) = \{0, 2\}$ $\quad$ $v(6) = \{7\}$
$v(2) = \{1, 3, 4\}$ $\quad$ $v(7) = \{5, 6\}$
$v(3) = \{2\}$ $\quad$ $v(8) = \varnothing$
$v(4) = \{2, 5\}$ $\quad$ $v(9) = \varnothing$

Pentru ținerea în evidență a restricțiilor de compatibilitate, poate fi folosit un tablou bidimensional pătratic, conținând un număr de linii, respectiv un număr de coloane, egale cu numărul operațiilor. Se va adopta convenția ca un element al tabloului cu indicele de linie egal cu "*i*" și cu indicele de coloană egal cu "*j*" să aibă valoarea *1* dacă operația *i* este incompatibilă cu operația *j*, iar valoarea *0* -altfel. Configurația valorilor elementelor tabloului -fie *N[][]* numele acestuia- va fi, în cazul de restricțiilor de mai sus, următoarea:

	0	1	2	3	4	5	6	7	8	9
0		1								
1	1		1							
2		1		1	1					
3			1							
4			1			1				
5					1			1		
6								1		
7						1	1			
8										
9										

N[][]

Fig. 5.4.1_5. Tabloul restricțiilor de compatibilitate.

i_7). **restricțiile de transport:**

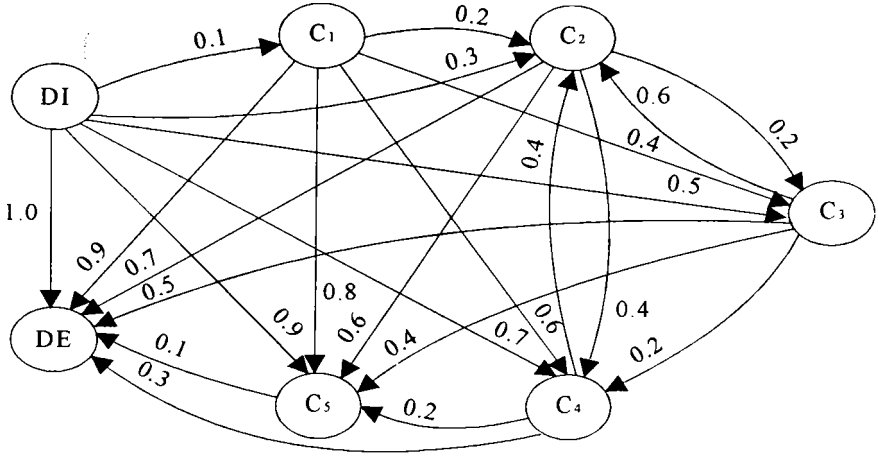


Fig. 5.4_1.6. Graful fluxului pieselor.

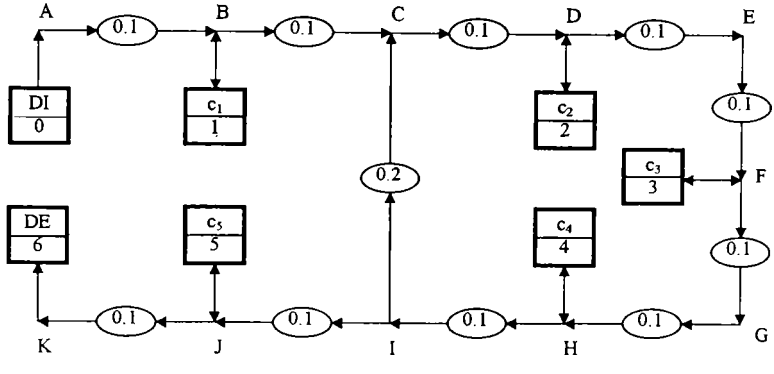
Pentru ținerea în evidență a restricțiilor de transport, poate fi folosit un tablou bidimensional pătratic, pe post de matrice de incidență a grafului fluxului de piese. Tabloul va avea un număr de linii, respectiv un număr de coloane, egale cu numărul operațiilor. Se va adopta convenția ca un element al tabloului cu indicele de linie egal cu "*i*" și cu indicele de coloană egal cu "*j*" să aibă ca valoare durata operației de transfer al piesei din postul *i* în postul *j*, dacă acest transfer este posibil, iar valoarea supremă reprezentabilă, *SUPREPR* -altfel. Configurația valorilor elementelor tabloului -fie *F[][]* numele acestuia- va fi, în cazul restricțiilor de mai sus, următoarea:

	0	1	2	3	4	5	6
0		0.1	0.3	0.5	0.7	0.9	1.0
1			0.2	0.4	0.6	0.8	0.9
2				0.2	0.4	0.6	0.7
3			0.6		0.2	0.4	0.5
4			0.4	0.6		0.2	0.3
5							0.1
6							

F[][]

Fig. 5.4.1_7. Tabloul restricțiilor de transport (matricea de incidență a grafului fluxului de piese).

Într-o reprezentare sinoptică, sistemul de fabricație considerat se prezintă așa cum se arată în figura 5.4.1_8.



LEGENDĂ:

DI: depozit de intrare; DE: depozit de ieșire; c_i: celula flexibilă "i"; x.y: durata operației de transport dintre punctele de alături notate cu litere majuscule.

Fig. 5.4.1_8. Reprezentare sinoptică a sistemului de fabricație.

i_8). ciclul sistemului de fabricație:

$$\delta = 5.00$$

Notă:

Ciclul sistemului de fabricație se exprimă în unități convenționale, presupuse subînțelese.

Se cere:

A) Să se stabilească, respectându-se i_1)...i_8), operațiile pe care le are de efectuat fiecare celulă flexibilă la nivelul unui ciclu al sistemului, astfel încât timpul de neutilizare a capacităților de producție să fie minim.

B) Să se stabilească sarcinile de transport specifice la nivel de ciclu al sistemului.

5.4.2. Ilustrarea rezolvării problemei ordonanțării

În cele ce urmează, se va efectua partiționarea echilibrată a procesului tehnologic în faze aplicând algoritmul bazat pe grupările admisibile și tehnica drumului critic, iar apoi, se vor aloca fazele obținute celulelor flexibile ale sistemului, în acest fel rezultând, implicit, și sarcinile subsistemului de transport.

r_1). Se determină familia grupărilor admisibile de operații, asociată digrafului de precedență $G = (O, V)$:

$$W = \{\varnothing,\ \{0\},\ \{0, 1\},\ \{0, 4\},\ \{0, 1, 2\},\ \{0, 1, 4\},\ \{0, 1, 2, 4\},\ \{0, 1, 3, 4\},\ \{0, 1, 2, 3, 4\},\{0, 1, 2, 3, 4, 5\},\{0, 1, 2, 3, 4, 5, 6\},\{0, 1, 2, 3, 4, 5, 6, 7\},\ \{0, 1, 2, 3, 4, 5, 6, 7, 8\},\ \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$

Așadar:

$X_0 = \varnothing$
$X_1 = \{0\}$
$X_2 = \{0, 1\}$
$X_3 = \{0, 4\}$
$X_4 = \{0, 1, 2\}$
$X_5 = \{0, 1, 4\}$
$X_6 = \{0, 1, 2, 4\}$
$X_7 = \{0, 1, 3, 4\}$
$X_8 = \{0, 1, 2, 3, 4\}$
$X_9 = \{0, 1, 2, 3, 4, 5\}$
$X_{10} = \{0, 1, 2, 3, 4, 5, 6\}$
$X_{11} = \{0, 1, 2, 3, 4, 5, 6, 7\}$
$X_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
$X_{13} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

r_2). Graful grupărilor admisibile, $Z = (W, Y)$, rezultă așa cum se arată în figura 5.4.1_9.

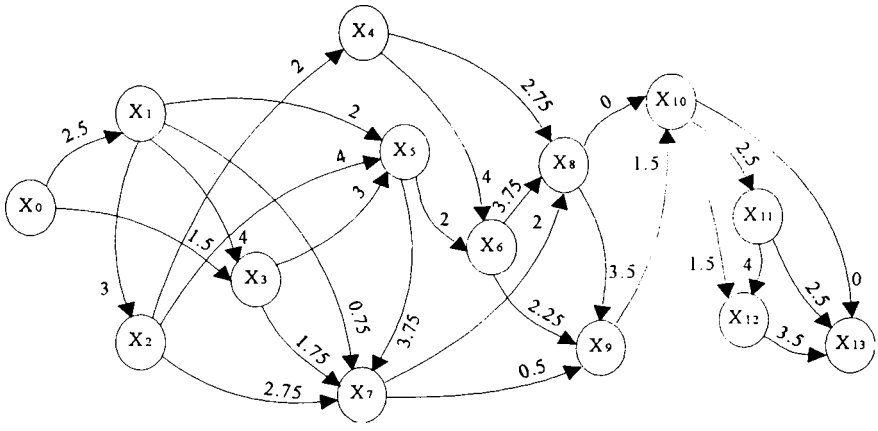


Fig. 5.4.2_1. Graful grupărilor admisibile.

Matricea de incidență a acestui graf -fie *Z[][]* numele ei- se prezintă așa cum se arată în figura 5.4.2_2.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0		2.5		1.5										
1			3	4		2		0.75						
2					2	4		2.75						
3						3		1.75						
4							4		2.75					
5							2	3.75						
6									3.75	2.25				
7									2	0.5				
8										3.5	0			
9											1.5			
10												2.5	1.5	0
11													4	2.5
12														3.5
13														

Z[][]

Fig. 5.4.2_2. Matricea de incidență a grafului grupărilor admisibile.

r3). Partițiile optime ce rezultă din graful $Z = (W, Y)$ sunt cele sintetizate în tabelul din figura 5.4.2_3.

x	DRUMUL MINIM	PARTIȚIA OPTIMĂ									
		Q_1^x	(Q_1^x)	Q_2^x	(Q_2^x)	Q_3^x	(Q_3^x)	Q_4^x	(Q_4^x)	Q_5^x	(Q_5^x)
1.	$X_0 \rightarrow X_1 \rightarrow X_4 \rightarrow X_8 \rightarrow X_{10} \rightarrow X_{13}$	{0}	2.5	{1, 3, 4}	4.25	{2}	3.0	{5, 6}	5.0	{7, 8, 9}	5.0
2.	$X_0 \rightarrow X_1 \rightarrow X_3 \rightarrow X_9 \rightarrow X_{10} \rightarrow X_{13}$	{0}	2.5	{1, 3, 4}	4.25	{2, 5}	4.5	{6}	5.0	{7, 8, 9}	5.0
3.	$X_0 \rightarrow X_3 \rightarrow X_7 \rightarrow X_8 \rightarrow X_{10} \rightarrow X_{13}$	{0, 4}	3.5	{1, 3}	3.25	{2}	3.0	{5, 6}	5.0	{7, 8, 9}	5.0
4.	$X_0 \rightarrow X_3 \rightarrow X_7 \rightarrow X_9 \rightarrow X_{10} \rightarrow X_{13}$	{0, 4}	3.5	{1, 3}	3.25	{2, 5}	4.5	{6}	5.0	{7, 8, 9}	5.0

Fig. 5.4.2_3. Partițiile optime ale procesului tehnologic în faze.

r_3). Mulțimile celulelor flexibile asignabile fazelor partițiilor optime sunt:

$C_0^1 = \{0\}$
$C_1^1 = \mu(Q_1^1) = \mu(0) = \{1, 2, 3, 5\}$
$C_2^1 = \mu(Q_2^1) = \mu(1) \cap \mu(3) \cap \mu(4) = \{1, 3, 4\} \cap \{1, 3, 4\} \cap \{1, 2, 3, 4\} = \{1, 3, 4\}$
$C_3^1 = \mu(Q_3^1) = \mu(2) = \{2, 5\}$
$C_4^1 = \mu(Q_4^1) = \mu(5) \cap \mu(6) = \{2, 3, 4\} \cap \{2, 3, 4\} = \{2, 3, 4\}$
$C_5^1 = \mu(Q_5^1) = \mu(7) \cap \mu(8) \cap \mu(9) = \{2, 4, 5\} \cap \{2, 3, 5\} \cap \{2, 4, 5\} = \{2, 5\}$
$C_6^1 = \{6\}$

Corespunzător, pentru prima partiție optimă, rezultă următoarea matrice de asignabilitate, S^1:

	0	1	2	3	4	5	6
0	1						
1		1	1				
2		1		1	1	1	
3		1	1		1		
4			1		1		
5		1		1		1	
6							1

$S^1[][]$

Fig. 5.4.2_4. Tabloul de asignabilitate corespunzător primei partiții optime.

$C_0^2 = \{0\}$
$C_1^2 = \mu(Q_1^2) = \mu(0) = \{1, 2, 3, 5\}$
$C_2^2 = \mu(Q_2^2) = \mu(1) \cap \mu(3) \cap \mu(4) = \{1, 3, 4\} \cap \{1, 3, 4\} \cap \{1, 2, 3, 4\} = \{1, 3, 4\}$

$C_3^2 = \mu(Q_3^2) = \mu(2) \cap \mu(5) = \{2, 5\} \cap \{2, 3, 4\} = \{2\}$
$C_4^2 = \mu(Q_4^2) = \mu(6) = \{2, 3, 4\}$
$C_5^2 = \mu(Q_5^2) = \mu(7) \cap \mu(8) \cap \mu(9) = \{2, 4, 5\} \cap \{2, 3, 5\} \cap \{2, 4, 5\} = \{2, 5\}$
$C_6^2 = \{6\}$

Corespunzător, pentru a doua partiție optimă, rezultă următoarea matrice de asignabilitate, S^2:

	0	1	2	3	4	5	6
0	1						
1		1	1				
2		1		1	1	1	
3		1	1		1		
4			1		1		
5		1				1	
6							1

$S^2[][]$

Fig. 5.4.2_5. Tabloul de asignabilitate corespunzător celei de-a doua partiții optime.

$C_0^3 = \{0\}$
$C_1^3 = \mu(Q_1^3) = \mu(0) \cap \mu(4) = \{1, 2, 3, 5\} \cap \{1, 2, 3, 4\} = \{1, 2, 3\}$
$C_2^3 = \mu(Q_2^3) = \mu(1) \cap \mu(3) = \{1, 3, 4\} \cap \{1, 3, 4\} = \{1, 3, 4\}$
$C_3^3 = \mu(Q_3^3) = \mu(2) = \{2, 5\}$
$C_4^3 = \mu(Q_4^3) = \mu(5) \cap \mu(6) = \{2, 3, 4\} \cap \{2, 3, 4\} = \{2, 3, 4\}$
$C_5^3 = \mu(Q_5^3) = \mu(7) \cap \mu(8) \cap \mu(9) = \{2, 4, 5\} \cap \{2, 3, 5\} \cap \{2, 4, 5\} = \{2, 5\}$
$C_6^3 = \{6\}$

Corespunzător, pentru a treia partiție optimă, rezultă următoarea matrice de asignabilitate, S^3:

	0	1	2	3	4	5	6
0	1						
1		1	1				
2		1		1	1	1	
3		1	1		1		
4			1		1		
5				1		1	
6							1

$S^3[][]$

Fig. 5.4.2_6. Tabloul de asignabilitate corespunzător celei de-a treia partiții optime.

$C_0^4 = \{0\}$

$C_1^4 = \mu(Q_1^4) = \mu(0) \cap \mu(4) = \{1, 2, 3, 5\} \cap \{1, 2, 3, 4\} = \{1, 2, 3\}$

$C_2^4 = \mu(Q_2^4) = \mu(1) \cap \mu(3) = \{1, 3, 4\} \cap \{1, 3, 4\} = \{1, 3, 4\}$

$C_3^4 = \mu(Q_3^4) = \mu(2) \cap \mu(5) = \{2, 5\} \cap \{2, 3, 4\} = \{2\}$

$C_4^4 = \mu(Q_4^4) = \mu(6) = \{2, 3, 4\}$

$C_5^4 = \mu(Q_5^4) = \mu(7) \cap \mu(8) \cap \mu(9) = \{2, 4, 5\} \cap \{2, 3, 5\} \cap \{2, 4, 5\} = \{2, 5\}$

$C_6^4 = \{6\}$

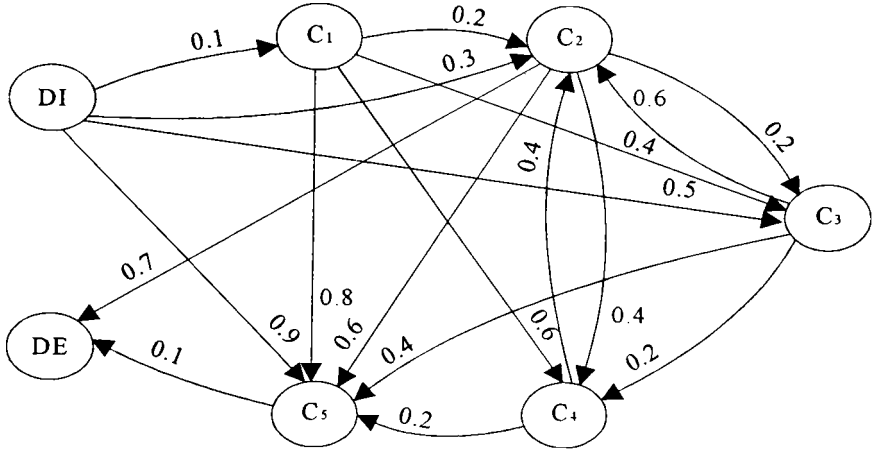
Corespunzător, pentru a patra partiție optimă, rezultă următoarea matrice de asignabilitate, S^4:

	0	1	2	3	4	5	6
0	1						
1		1	1				
2		1		1	1	1	
3		1	1		1		
4			1		1		
5		1		1		1	
6							1

$S^4[][]$

Fig. 5.4.2_7. Tabloul de asignabilitate corespunzător celei de-a patra partiții optime.

r_5). Grafurile de acoperire, $A^x = (P, L^x)$, și matricile lor de incidență, corespunzătoare celor patru partiții optime, vor fi următoarele:



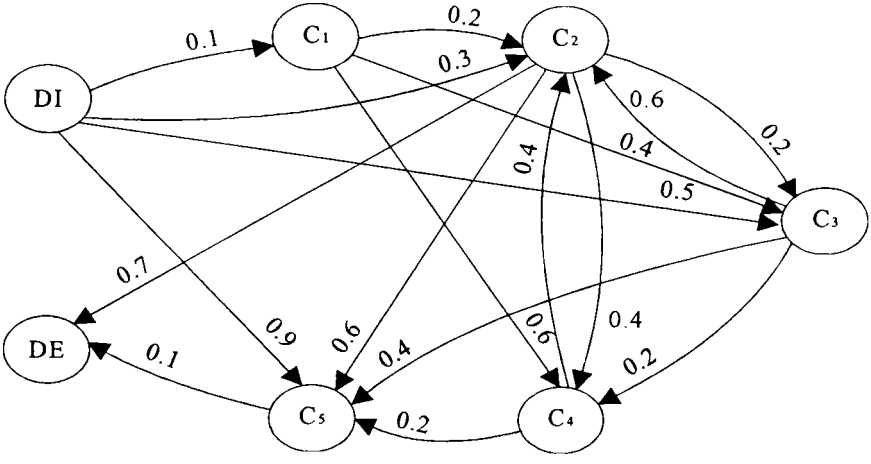
a)

	0	1	2	3	4	5	6
0		0.1	0.3	0.5		0.9	
1			0.2	0.4	0.6	0.8	
2				0.2	0.4	0.6	0.7
3			0.6		0.2	0.4	
4			0.4			0.2	
5							0.1
6							

$A^1[][]$

b)

Fig. 5.4.2_8. Graful de acoperire corespunzător primei partiții optime, A^1 -a)- și matricea sa de incidență, $A^1[][]$ -b)-.



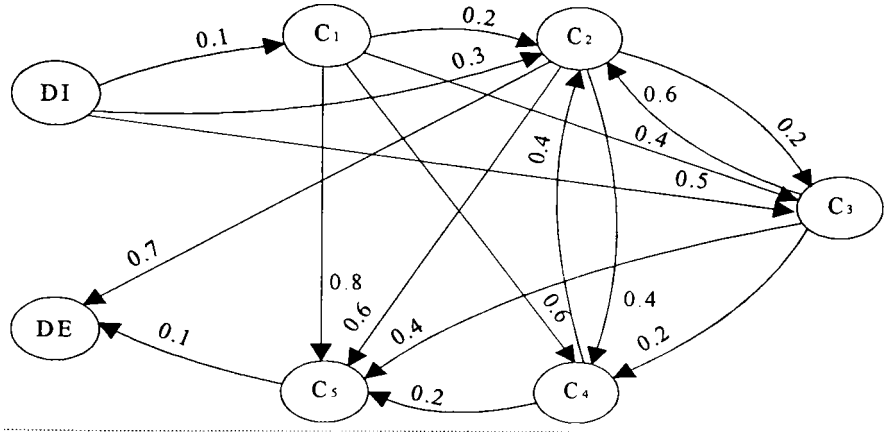
a)

	0	1	2	3	4	5	6
0		0.1	0.3	0.5		0.9	
1			0.2	0.4	0.6		
2				0.2	0.4	0.6	0.7
3			0.6		0.2	0.4	
4			0.4			0.2	
5							0.1
6							

$A^2[][]$

b)

Fig. 5.4.2_9. Graful de acoperire corespunzător celei de-a doua partiții optime, A^2 -a)- și matricea sa de incidență, $A^2[][]$ -b)-.



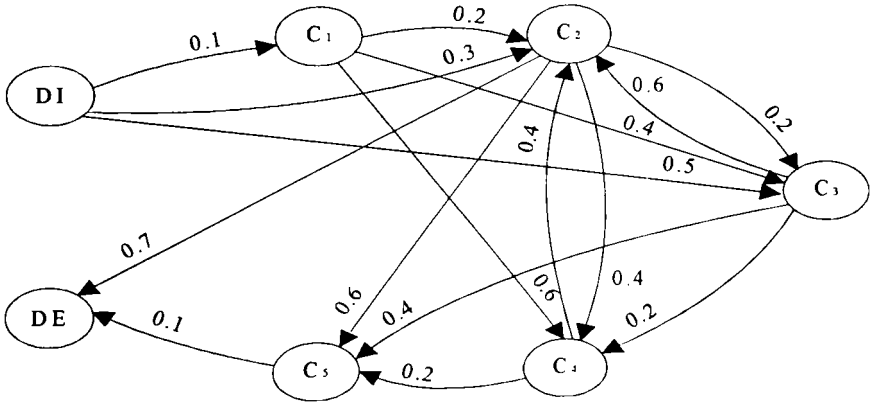
a)

	0	1	2	3	4	5	6
0		0.1	0.3	0.5			
1			0.2	0.4	0.6	0.8	
2				0.2	0.4	0.6	0.7
3			0.6		0.2	0.4	
4			0.4			0.2	
5							0.1
6							

$A^3[][]$

b)

Fig. 5.4.2_10. Graful de acoperire corespunzător celei de-a treia partiții optime, A^3 și matricea sa de incidență, $A^3[][]$.



a)

	0	1	2	3	4	5	6
0		0.1	0.3	0.5			
1			0.2	0.4	0.6		
2				0.2	0.4	0.6	0.7
3			0.6		0.2	0.4	
4			0.4			0.2	
5							0.1
6							

$A^4[][]$

b)

Fig. 5.4.2_11. Graful de acoperire corespunzător celei de-a patra partiții optime, A^4 -a)- și matricea sa de incidență, $A^4[][]$ -b)-.

r_6). Acoperirile optime ce rezultă sunt următoarele:

- din A^1:

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.4} c_3 \xrightarrow{0.6} c_2 \xrightarrow{0.4} c_4 \xrightarrow{0.2} c_5 \xrightarrow{0.1} DE$$

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.6} c_4 \xrightarrow{0.4} c_2 \xrightarrow{0.2} c_3 \xrightarrow{0.4} c_5 \xrightarrow{0.1} DE$$

- din A^2:

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.4} c_3 \xrightarrow{0.6} c_2 \xrightarrow{0.4} c_4 \xrightarrow{0.2} c_5 \xrightarrow{0.1} DE$$

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.6} c_4 \xrightarrow{0.4} c_2 \xrightarrow{0.2} c_3 \xrightarrow{0.4} c_5 \xrightarrow{0.1} DE$$

- din A^3:

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.4} c_3 \xrightarrow{0.6} c_2 \xrightarrow{0.4} c_4 \xrightarrow{0.2} c_5 \xrightarrow{0.1} DE$$

- din A^4:

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.4} c_3 \xrightarrow{0.6} c_2 \xrightarrow{0.4} c_4 \xrightarrow{0.2} c_5 \xrightarrow{0.1} DE$$

$$DI \xrightarrow{0.1} c_1 \xrightarrow{0.6} c_4 \xrightarrow{0.4} c_2 \xrightarrow{0.2} c_3 \xrightarrow{0.4} c_5 \xrightarrow{0.1} DE$$

Evident, doar două dintre aceste acoperiri sunt distincte. Presupunem că, în final, dintre acestea se alege, ca soluție a problemei ordonanțării, prima. Sinoptic, rezultatul obținut se poate reprezenta așa cum se arată în figura 5.4.2_12.

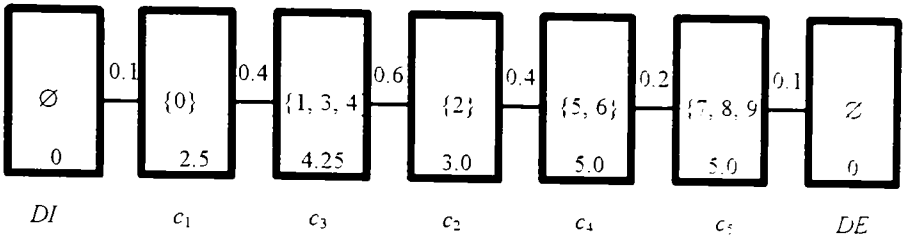


Fig. 5.4.2_12. Reprezentare sinoptică a rezultatului ordonanțării.

Din figura 3.4.2_12, rezultă, imediat, că ciclul de fabricație ajustat este:

$$\tilde{\delta} = \max((2.5+0.4),\ (4.25+0.6),\ (3.0+0.4),\ (5.0+0.2),\ (5.0+0.1)) = 5.2$$

Subciclul de prelucrare cuprinde intervalul [0.0, 5.0] din ciclul de fabricație ajustat, iar subciclul -de transport -intervalul [4.6, 5.2]. Există, așadar, o suprapunere a celor două subcicluri, în intervalul [4.6, 5.0]. Aceste aspecte sunt evidențiate clar de figura 5.4.2_13

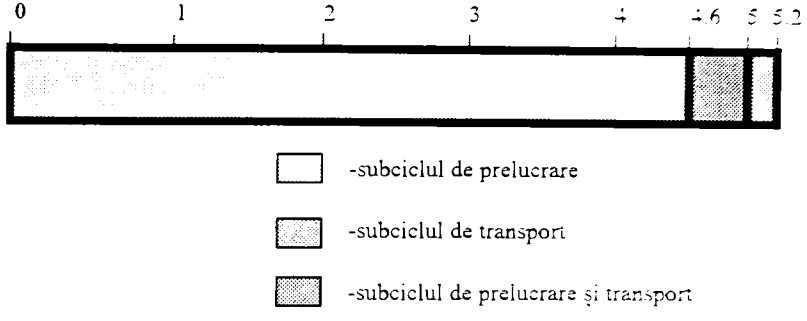


Fig. 5.4.2_13. Ciclul de fabricație și subciclurile sale.

Corespunzător acoperirii optime la care s-a ajuns, subsistemul de transport va avea sarcinile indicate în figura 3.4.2_14, figură în care se surprinde întreaga cronologie a operațiilor de prelucrare și de transport pentru cuplul sistem de fabricație-proces tehnologic considerat.

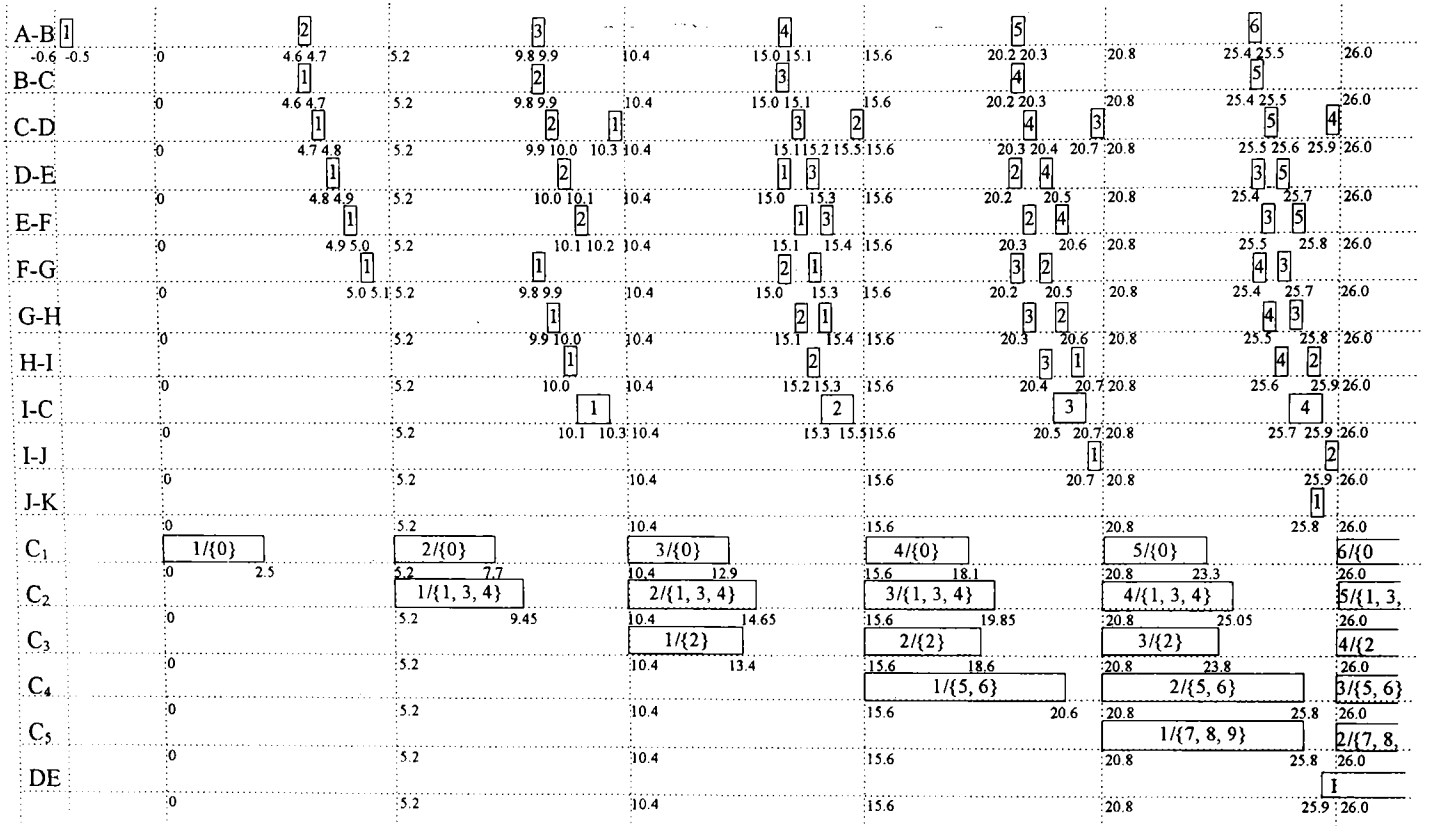


Fig. 5.4.2_14. Cronologia operaţiilor de prelucrare şi transport.

6
SOLUȚII PENTRU UN SUPORT DE PROGRAMARE CONCURENTĂ APLICABIL ÎN CONDUCEREA SISTEMELOR FLEXIBILE DE FABRICAȚIE

6.1. Aspecte introductive

Efortul de programare în fabricația flexibilă -mai ales în instanțierea ei cea mai evoluată: fabricația integrată prin calculator- este deosebit de consistent, atât la nivelele ierarhice inferioare, cât și la celelalte. La cele dintâi, pe de o parte întrucât constrângerile temporale și cele de volum de memorie disponibil sunt foarte severe, iar pe de altă parte întrucât acțiunile ce trebuic programate sunt procese concurente în timp real, cu tot ce acest lucru implică. La cele din urmă, întrucât anvergura problematicii atinge cote deosebit de ridicate.

În [BORA'89], chestiunea este fundamentat și in extenso tratată. În ceea ce ne privește, vis-a-vis de această chestiune, noi ne-am concentrat atenția doar spre găsirea unor soluții care să deschidă calea spre o cât mai bună eficiență a activității de dezvoltare a programelor din domeniul avut în vedere, cu precădere a celor corespunzătoare nivelelor ierarhice inferioare (0 până la 3). Mai exact, am elaborat un set de mecanisme de nucleu ce reprezintă, împreună, un executiv de timp real, grefat pe sistemul de operare *MS-DOS*. Acest executiv se prezintă drept extensie a mediilor de programare *TURBO C* și *TURBO C++*, ce le conferă capabilități de dezvoltare a aplicațiilor cu concurență de tip multitasking (în sensul plin al termenului, deci înglobând atributul "cooperativ") în timp real. Desigur, după cum se va vedea în cele ce urmează, păstrând toate facilitățile lor originale.

Se precizează că sintagma "timp real" este folosită în prezenta lucrare cu semantica ei restrânsă, ce trimite implicit la informatica tehnică [TSCH'90]. De asemenea, se face mențiunea că executivul rezultat pe baza soluțiilor ce vor fi prezentate satisface la fel de bine cerințele aplicațiilor din domeniul fabricației flexibile, ca și ale oricăror alte aplicații de informatică tehnică.

6.2. O soluție de comutare a proceselor

6.2.1. Preliminarii

Este cunoscut [ALLW'81] [BALT'84], comutarea proceselor se asigură de către o funcție de nucleu numită "*scheduler*".

Scheduler-ul cumulează următoarele acțiuni:

- salvarea contextului procesului ce iese din rulare;
- determinarea procesului care urmează să intre în rulare (*dispecerizarea*);
- restaurarea (eventual: instaurarea) contextului procesului ce urmează să intre în rulare și, implicit, introducerea în rulare a acestui proces.

În cele ce urmează, se prezintă, într-o abordare originală [ROBU'94a] [ROBU'95g], problematica comutării proceselor și o soluție pentru realizarea ei, într-o implementare *C*, sub sistemul de operare *MS-DOS*.

Se definește *contextul* unui proces, introducându-se noțiunile de "*context logic*" și "*context fizic*".

Se arată cum se pot face *salvarea* și *restaurarea contextului*.

Sunt evidențiate aspectele esențiale pentru implementarea *scheduler*-ului. Se propune un text *C* apt să-l întruchipeze.

6.2.2. Contextul proceselor

O accepțiune economică -sub aspect temporal- asupra contextului unui proces vede în el o entitate ce cuprinde [TSCH'90] [ROBU'94a]:

- starea fanioanelor;
- adresa instrucției cu care procesul urmează să-și înceapă sau să-și continue rularea;
- adresa curentă în stiva procesului;
- conținuturile registrelor procesorului diferite de cele implicate în găzduirea fanioanelor, pointarea instrucțiilor, și pointarea stivei;
- starea curentă a procesului;
- prioritatea procesului;
- timpul după care urmează să se producă deblocarea automată a procesului, în situațiile în care el este blocat cu "time-out".
- indicația că procesul a ajuns în situația de a fi deblocat automat, ca urmare a expirării timpului cât i s-a permis să rămână blocat.

Pentru a se conferi un plus de relevanță referirilor la informațiile ce definesc contextul proceselor, se propune gruparea lor în două categorii [ROBU'94a]. Pentru prima categorie, se introduce apelativul de "*context fizic*", iar pentru cea de a doua -apelativul de "*context logic*". Contextul fizic este definit în figura 6.2.2_1, iar contextul logic -în figura 6.2.2_2, pentru ambele avându-se în vedere un procesor de tipul *INTEL 80x86*.

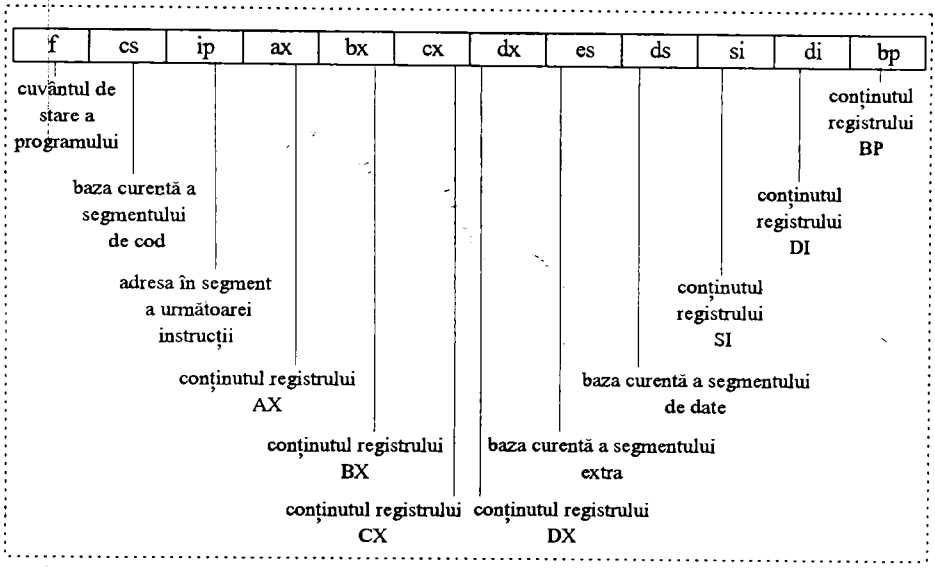


Fig. 6.2.2_1. Contextul fizic al unui proces.

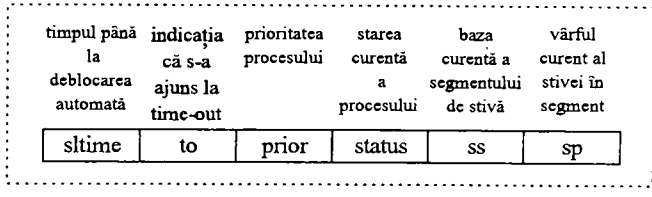


Fig. 6.2.2_2. Contextul logic al unui proces .

Pentru a se facilita operarea asupra informațiilor ce țin de *contextul logic*, respectiv de *contextul fizic*, corespunzător lor, se definesc două structuri, denumite "*CONTFIZ*", respectiv "*CONTLOG*". Aceste structuri fac obiectul figurilor 6.2.2_3, respectiv 6.7.2_4.

```
typedef struct {
        unsigned int bp, di, si, ds, es, dx, cx, bx, ax;
        union {
            TASK_ADR cs_ip;
            struct {
                unsigned int ip;
                unsigned int cs;
            } cai;
        } adr;
        unsigned int f;
    } CONTFIZ;
```

Fig. 6.2.2_3. Structura *CONTFIZ*

Este de precizat că *TASK_ADR* denominează o variabilă de tip definit, reprezentând un pointer către o funcție ce materializează un *proces* sau, altfel spus -mai concret-, un "*task*.".

```
typedef struct {
        unsigned int ss;
        unsigned int sp;
        unsigned char status;
        unsigned short prior;
        unsigned char to;
        unsigned int sltime;
      } CONTLOG;
```

Fig. 6.2.2_4. Structura *CONTLOG*.

Întrucât fiecare proces are asociată o structură de tipul *CONTLOG*, este firească gruparea într-un tablou a tuturor acestor structuri. Tabloului i se atribuie numele *_contlog[]*. Dimensiunea sa este, evident, egală cu numărul maxim anvizajat pentru procese.

6.2.3. Salvarea și restaurarea contextului proceselor

Având în vedere că, din punct de vedere al limbajului de programare, atât *procesele* -respectiv *task*-urile-, cât și *scheduler*-ul au statutul de *funcții*, rularea unui program concurent (multitasking) cuprinde apeluri ale funcției *scheduler* din câte o funcție *proces*, alternate cu apeluri ale câte unei funcții *proces* din funcția *scheduler*:

$$... \rightarrow proces\_i\_p \rightarrow scheduler \rightarrow proces\_j\_q \rightarrow ...$$

Remarcă:
Prin notația "proces_k_h" desemnează porțiunea "h" a procesului "k".

Apelul funcției *scheduler* dintr-o funcție *proces* poate fi fie programat în cadrul procesului, fie suprafortat din exterior, ca efect al unei întreruperi fizice. Evident, fiecare apel, indiferent de tipul lui, trebuie să fie însoțit de salvarea contextului procesului apelant.

Adoptând soluția ca și apelul programat să se facă prin intermediul unei întreruperi, de această dată -evident- de tip logic și ținând seamă de operațiile "*hardware*" pe care microprocesoarele din familia *INTEL 80x86* le execută la acceptarea de întreruperi -a se vedea figura 6.2.3_1-, rezultă că, în toate situațiile, la intrarea în funcția *scheduler*, în stiva procesului apelant se află deja primele două informații din contextul procesului: starea fanioanelor (conținutul registrului *F*) și adresa instrucției cu care procesul urmează să-și continue rularea (conținutul registrelor *CS* și *IP*), informații ce țin de contextul fizic.

Pentru a se realiza salvarea facilă a celorlalte informații din contextul fizic, la apelul funcției *scheduler*, se va adopta pentru această funcție tipul *interrupt*, cunoscut fiind faptul că, în

limbajul *C*, la intrarea în funcțiile de acest tip, se execută, ca prime instrucțiii-mașină, cele indicate în figura 6.2.3_2.

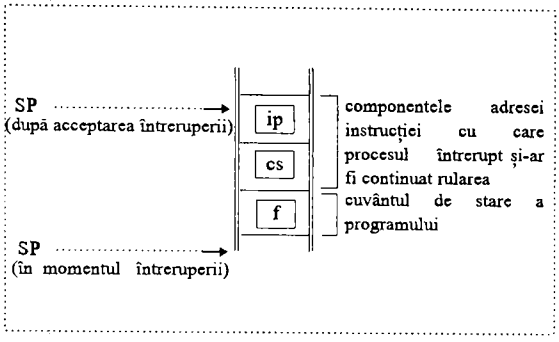


Fig. 6.2.3_1. Operațiile pe stivă ale microprocesorului *INTEL 80x86*, la întreruperi.

```
push ax
push bx
push cx
push dx
push es
push ds
push si
push di
push bp
mov bp, DGROUP
```

Fig. 6.2.3_2. Instrucțiile-mașină generate de compilatorul *TURBO C* în debutul funcțiilor de tipul *interrupt*.

Este de subînțeles, la părăsirea funcțiilor de tipul *interrupt*, se asigură descărcarea stivei, atât de informațiile încărcate în cadrul lor (*bp...ax*), cât și de cele depuse prin mecanismul de apel (*f, cs, ip*).

Conținuturile registrelor implicate în adresarea stivei -*SS* și *SP*-, reprezentând adresa curentă în stivă, se salvează în variabilele *ss*, respectiv *sp*, din componența structurii *CONTLOG* aferentă procesului în cauză. Evident, în aceeași structură, în variabilele *status, prior, to*, respectiv *sltime*, își au reședința informațiile privind starea procesului, prioritatea, ajungerea la "*time-out*", respectiv cea care arată timpul rămas până la situația de "*time-out*", situație în care procesul este deblocat automat.

6.2.4. Aspecte privind implementarea unui *scheduler*

După cum se cunoaște, sistemul de operare *MS-DOS* nu a fost gândit în ideea de reentranță a procedurilor sale. Din acest motiv, orice activare a mecanismului de comutare, suprafortată prin întreruperi fizice, trebuie să fie urmată de o verificare având ca scop detectarea situațiilor în care întreruperea a survenit în cadrul unei proceduri *MS-DOS*. In aceste situații, rutina de tratare a întreruperii va putea efectua orice operații specifice, în condiții de indivizibilitate, dar nu va da curs procesului de comutare. De exemplu, în cazul întreruperilor fizice de la ceasul de timp real, indiferent în ce condiții o întrerupere intervine, se va actualiza variabila *sltime* din contextul logic al fiecărui proces, iar, în continuare, nu se va da sau se va da curs unui proces de comutare, după cum întreruperea a intervenit în timp ce se rula o procedură *MS-DOS* sau în timp ce se rula altceva.

Se precizează că situațiile în care o întrerupere intervine în timpul rulării unei proceduri *MS-DOS* se disting prin faptul că registrul *CS* conține, în cazul lor, o valoare mai mică decât cea indicată de variabila sistem *_psp*.

Pe baza celor de mai sus, se propune, ca soluție de comutare a proceselor sau, altfel spus, ca soluție de implementare a *scheduler*-ului, textul *C* din figura 6.2.4_1.

```
1.  unsigned interrupt scheduler (CONTFIZ contfiz)
2.   {
3.    static CONTLOG *p;
4.    register unsigned short tsk;
5.    register unsigned int aux;
6.    aux=0;
7.    for (tsk = 0; tsk < MAX_TSK; tsk++) {
8.        p = &_contlog[tsk];
9.        if (p->status == BLOCAT && p->sltime) {
10.           if (! --p->sltime) {
11.               elim _ltb (tsk, p->prior);
12.               ins_lap (tsk, p->prior);
13.               p->status = EXEC;
14.               p->to=1;
15.           }
16.       }
17.   }
18.   if (contfiz.adr.cai.cs < _psp) {
19.       return;
20.   }
21.   p = &_contlog[_task_crt];
22.   p->sp = _SP;
23.   p->ss = _SS;
24.   _task_crt = urm_lap ();
25.   p = &_contlog[_task_crt];
26.   _SP = p->sp;
27.   _SS = p->ss;
28.   aux = p->to;
29.   return (aux);
30.  }
```

Fig. 6.2.4_1. Soluția de comutare propusă.
(textul funcției *scheduler*)

Se precizează că, variabila *_task_crt*, prezentă în textul funcției *scheduler*, este dedicată păstrării indexului taskului aflat curent în rulare, fiind una dintre variabilele cu vizibilitate globală pe care se mizează.

De asemenea, se face menţiunea că funcţiile *elim_lap()*, *ins_lap()*, şi *urm_lap()*, apelate, respectiv, în liniile *11*, *12*, şi *24* din figura 6.2.4_1, aparţin mecanismului de dispecerizare. Rolul lor este, respectiv, de a elimina un proces din rândul proceselor rulabile, de a insera un proces în rândul proceselor rulabile, şi de a identifica şi indica procesul care, în conformitate cu politica de dispecerizare, este îndrituit să obţină procesorul.

6.2.5. Concluzii

Mecanismul de comutare a taskurilor prezentat funcţionează cu succes în cadrul unui executiv de timp real, numit "*RTC86*", conceput şI realizat către autor [ROBU'94a]. Timpul în care el asigură comutare este de circa 60 μs, pe un procesor *INTEL 80386*, la 33 MHz. Se aminteşte că timpii de comutare pretinşi de domeniul informaticii industriale sunt inferiori pragului de 100 μs [TSCH'90].

6.3. O soluţie de dispecerizare a proceselor

6.3.1. Preliminarii

Eficienţa prelucrării concurente depinde, în bună măsură, de maniera în care este rezolvată problema dispecerizării proceselor [ALLW'81] [BALT'84] [STAN'90].

Un element de importanţă primordială în această problemă este politica după care se face dispecerizarea. O multitudine de politici pot fi imaginate; literatura de specialitate este destul de bogată în această privinţă [BALT'84] [STAN'90] [TSCH'90]. Practica, însă, a evidenţiat că, la performanţele tehnologiei informatice actuale, din această multitudine, politica bazată pe priorităţi neunivoce asigură cel mai bun compromis între gradul de raţionalitate a ordinii în care procesele sunt introduse în rulare şi preţul în timp procesor necesitat [TSCH'90] [ROBU'95g].

În cele ce urmează, se prezintă o soluţie de dispecerizare prin priorităţi neunivoce -adică: printr-o politică acţionând prin prioritizare şi rotaţie-, într-o implementare *C* originală [ROBU'94b] [ROBU'95g], sub sistemul de operare *MS-DOS*. Soluţia a fost adoptată având în vedere că, în aplicaţiile din informatica sistemelor de fabricaţie integrată prin calculator, ca, de altfel, şi în alte aplicaţii de informatică industrială, intervin, deopotrivă, activităţi de prioritate diferită şi activităţi echiprioritare, şi ţinând seamă de restricţiile temporale severe pe care aceste aplicaţii le impun.

Se oferă un model al dispecerizării.

Este abordată implementarea listei de aşteptare la procesor.

Sunt descrise funcțiile mecanismului de dispecerizare. Textele *C* ale acestor funcții sunt, inclusiv ele, redate.

6.3.2. Modelul dispecerizării bazate pe priorități neunivoce

Dispecerizarea prin priorități neunivoce se caracterizează prin aceea că fiecare proces are o anumită prioritate, diferită de sau identică cu a altora. În fenomenologia dispecerizării, primează prioritatea, procesele de aceeași prioritate fiind tratate prin rotație [TSCH'90]. Acestei dispecerizări i se poate asocia modelul din figura 6.3.2_1, model ce corespunde unei situații concrete, considerată spre exemplificare [ROBU'94b] [ROBU'95g].

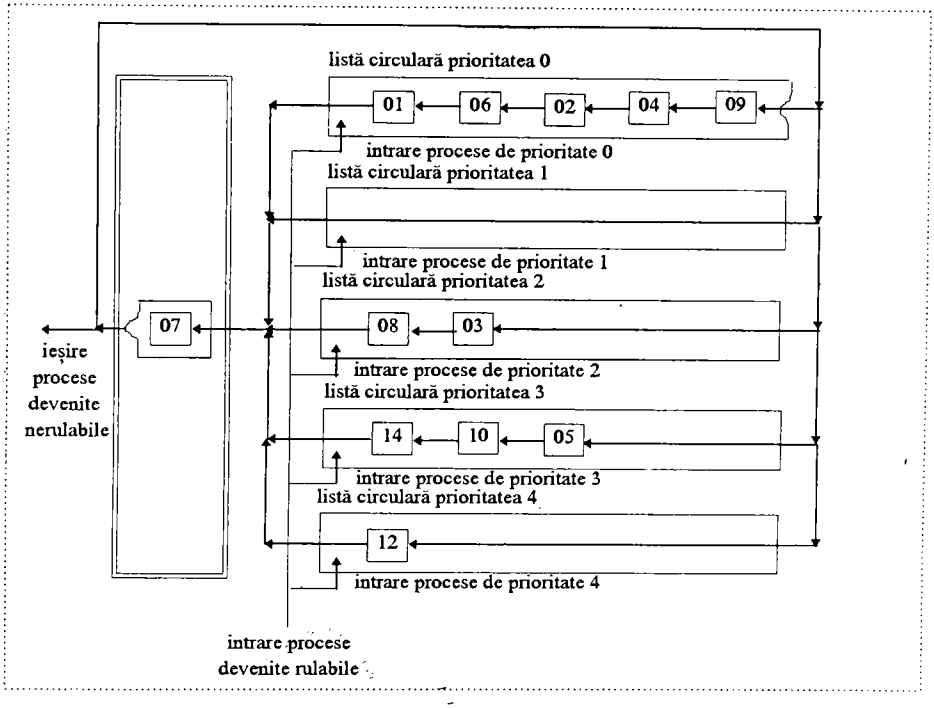


Fig. 6.3.2_1. Modelul dispecerizării prin priorități neunivoce.

În situația concretă surprinsă în model, se remarcă următoarele:

- lista de așteptare la procesor cuprinde cinci liste, corespunzătoare celor cinci nivele de prioritate aflate la dispoziția proceselor rulabile; lista de prioritate 1 este vidă;
- procesul 07, aparținând listei circulare a nivelului de prioritate 0, este în rulare, ocupând ultima poziție a acestei liste (s-a presupus că din momentul ultimei comutări, nici un proces mai prioritar decât procesul 07 nu a devenit rulabil);

- în ipoteza conservării conținutului listei de așteptare la procesor, prin procese de comutare succesive, vor intra în rulare procesele 01, 06, 02, 04, și 09, apoi din nou 07, 01, ș.a.m.d.;
- procesele de pe nivelul de prioritate 2 pot intra în rulare imediat după vidarea listei de prioritate 0, întrucât lista de prioritate 1 este vidă; dacă vidarea listei de prioritate 0 se produce, va intra în rulare mai întâi procesul 08, trecând, cu această ocazie, pe ultima poziție a listei ce-l conține; îi va urma procesul 03, etc.
- în general, procesele de pe un nivel de prioritate pot intra în rulare doar dacă toate listele corespunzătoare nivelelor de prioritate mai puternice sunt vide.

6.3.3. Lista de așteptare la procesor

Lista de așteptare la procesor, în care se înscriu toate procesele rulabile, reprezintă o înlănțuire de liste circulare, fiecare dintre acestea corespunzând unui grup de procese de o anumită prioritate; ordinea de succedere a listelor circulare este cea de slăbire a priorităților proceselor înscrise în ele.

O asemenea înlănțuire de liste, ca și listele însele, pot fi implementate cu ajutorul a două tablouri corelate: un tablou al proceselor și un tablou al priorităților [TSCH'90] [ROBU'94b] [ROBU'95g].

Tabloul proceselor implementează listele circulare. Un element al tabloului, împreună cu indicele său, reprezintă un nod al unei liste. Indicele are semnificația de index al procesului aferent nodului, iar elementul -semnificația de index al succesorului acestui proces. Considerându-se că în sistem pot fi maximum *MAX_TSK* procese, tabloul proceselor va avea *MAX_TSK* elemente, cu indicii *0...MAX_TSK-1*. Valoarea elementelor corespunzătoare proceselor neprinse în nici o listă va fi *MAX_TSK*.

Tabloul priorităților are *MAX_TSK+1* elemente, cu indicii *0...MAX_TSK*. Indicii *0...MAX_TSK-1* se pun în corespondență bijectivă cu prioritățile pe care le pot avea procesele. Elementele de tablou cu acești indici primesc ca valori indexurile proceselor aflate în coada listelor corespunzătoare priorităților respective. La fiecare schimbare a cozii unei liste, valoarea elementului asociat ei este actualizată. Dacă o listă de o anumită prioritate este vidă, valoarea elementului cu indicele egal cu respectiva prioritate va fi *MAX_TSK*.

Elementul de indice *MAX_TSK* are un rol aparte. Valoarea sa indică cel mai puternic nivel de prioritate a cărui listă circulară este nevidă, respectiv nivelul de prioritate al procesului aflat în rulare. Când toate listele circulare sunt vide, elementul de indice *MAX_TSK* are valoarea *MAX_TSK*.

Notând cu *_lap_t[]* tabloul proceselor și cu *_lap_p[]* tabloul priorităților, rezultă, evident, că valoarea expresiei:

_lap_p[_lap_p[MAX_TSK]]

repezintă indexul procesului aflat în rulare, iar valoarea expresiei:

_lap_t[_lap_p[_lap_p[MAX_TSK]]]

reprezintă indexul procesului din capul listei căreia îi aparține procesul aflat în rulare.

Presupunând *MAX_TSK = 15*, pentru situația din figura 6.3.2_1, tablourile *_lap_t* și *_lap_p* vor arăta astfel:

_lap_t[]: *MAX_TSK-1*

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
15	06	04	08	09	14	02	01	03	07	05	15	12	15	10

_lap_p[]: *MAX_TSK*

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
07	15	03	05	12	15	15	15	15	15	15	15	15	15	15	00

Fig. 6.3.3_1. Tablourile *_lap_t[]* și *_lap_p[]* corespunzătoare situației din figura 6.3.2_1.

6.3.4. Funcțiile mecanismului de dispecerizare

Mecanismul de dispecerizare propus cuprinde patru funcții [ALLW'81] [ROBU'94b] [ROBU'95g], cu ajutorul cărora se operează asupra listei de așteptare la procesor. Aceste funcții sunt prezentate sintetic în figura 6.3.4_1.

void ins_lap (usshort ind_tsk, usshort prior)
/* inserează în *lap* procesul */
/* cu indexul *ind_tsk* și prioritatea *prior* */

void elim_lap (usshort ind_tsk, usshort prior)
/* elimină din *lap* procesul cu indexul *ind_tsk* */

usshort urm_lap (void)
/* determină indexul procesului aflat în capul *lap* */

void init_lap (void)
/* inițializează lista de așteptare la procesor */

Notă:
"usshort" reprezintă prescurtare pentru "unsigned short".

Fig. 6.3.4_1. Funcțiile de gestionare a listei de așteptare la procesor.

Funcția *ins_lap ()*

Această funcție are, în primul rând, sarcina să verifice dacă lista circulară asupra căreia trebuie să opereze este vidă sau nu.

În cazul în care această listă este vidă, inserarea în ea a procesul cu indexul precizat prin argumentul *ind_tsk* se face prin înscrierea în elementul de indice *ind_tsk* din tabloul proceselor a valorii *ind_tsk*, întrucât procesul în cauză se va succede în respectiva listă pe el însuși, reprezentând, deopotrivă, capul și coada ei. Totodata, valoarea argumentului *ind_tsk* se înscrie, de astă dată cu semnificația de coadă de listă, în elementul din tabloul priorităților al cărui indice este egal cu valoarea argumentului *prior*.

În cazul în care lista nu este vidă, inserarea în ea a procesului cu indexul precizat prin argumentul *ind_tsk* presupune instalarea acestui proces în capul listei corespunzătoare priorității indicată de argumentul *prior*. Acest lucru se asigură prin copierea în elementul din tabloul proceselor cu indicele *ind_tsk* a indexului procesului surprins în postura de cap de listă, index aflat în elementul din același tablou corespunzător cozii listei în cauză, urmată de înscrierea valorii *ind_tsk* în acest element corespunzător cozii. Se reamintește că indicele elementului din tabloul proceselor asociat cozii unei liste de o anumită prioritate se găsește în tabloul priorităților, în elementul aferent priorității respective.

Revenim la cazul în care lista asupra căreia trebuie să opereze funcția *ins_lap ()* este găsită vidă, pentru a adăuga că, în acest caz, se impune, suplimentar, compararea prioritații procesului nou inserat cu valoarea înscrisă în elementul de indice *MAX_TSK* din tabloul priorităților (valoare care reprezintă cea mai puternică prioritate căreia îi corespunde o listă circulară nevidă) și, dacă este nevoie, înlocuirea acestei valori cu cea a argumentului *prior*.

Rezultă, așadar, pentru funcția *ins_lap ()*, textul din figura 6.3.4_2.

```
1.  void ins_lap (usshort ind_tsk, usshort prior)
2.    {
3.      if (_lap_p[prior] == MAX_TSK) {
4.          _lap_t[ind_tsk] = ind_tsk;
5.          _lap_p[prior] = ind_tsk;
6.          if (prior < _lap_p[MAX_TSK]) {
7.              _lap_p[MAX_TSK] = prior;
8.          }
9.      }
10.     else {
11.         _lap_t[ind_tsk] = _lap_t[_lap_p[prior]];
12.         _lap_t[_lap_p[prior]] = ind_tsk;
13.     }
14.   }
```

Fig. 6.3.4_2. Textul funcției *ins_lap ()*.

Funcția *elim_lap ()*

Această funcție trebuie, în primul rând, să efectueze identificarea în tabloul proceselor a elementului care conține indexul procesului de eliminat, index precizat prin argumentul *ind_tsk*.

Activitatea de căutare prin care se realizează această identificare se poate efectua analizând succesiv conținuturile elementelor tabloului respectiv, de exemplu, de la primul către ultimul [TSCH'90].

O soluție mai eficientă, în ceea ce privește timpul necesar găsirii elementului căutat, se poate obține prin limitarea căutării doar la nivelul subsetului de elemente ale tabloului proceselor, implicate în implementarea listei circulare în care trebuie să figureze procesul de eliminat [ROBU'94b] [ROBU'95g]. Dispunându-se, la intrarea în funcție, de prioritatea procesului, ca valoare a argumentului *prior*, prin citirea valorii înscrise în elementul din tabloul priorităților cu indicele egal cu această prioritate, se obține indexul procesului aflat în coada listei în cauză. Cu elementul din tabloul proceselor având indicele egal cu acest index, se poate începe procesul de căutare, ținând seamă că, valoarea înscrisă în el, ca de altfel și în celelalte cu care el este înlănțuit, reprezintă fie indexul procesului de eliminat, fie indicele elementului următor în care acest index s-ar putea găsi.

Odată identificat elementul care conține indexul procesului de eliminat, se impune tratarea diferită a cazului în care acest proces este coadă de listă, de cazul în care el nu este așa ceva. O comparare a valorii argumentului *ind_tsk* cu valoarea înscrisă în elementul din tabloul priorităților desemnat de argumentul *prior* pune în evidență cazul în speță: dacă cele două valori sunt identice, procesul de eliminat este coadă de listă, altfel, -nu.

În primul caz, dacă indicele elementului din tabloul proceselor în care a fost găsit indexul procesului de eliminat este egal cu acest index, înseamnă că acest proces era unicul din listă și, drept urmare, prin eliminarea lui, lista rămâne vidă. Eliminarea se efectuează, în această situație, prin înscrierea valorii *MAX_TSK* atât în elementul din tabloul proceselor care a conținut indexul procesului de eliminat, cât și în elementul din tabloul priorităților, corespunzător priorității acestui proces.

Tot în acest prim caz, dacă indicele elementului din tabloul proceselor în care a fost găsit indexul procesului de eliminat nu este egal cu acest index, înseamnă că acest proces nu este unicul în listă. Eliminarea presupune, în această situație, două acțiuni. Prima consistă în instalarea în coada listei a predecesorului procesului de eliminat, prin înscrierea indexului său, egal cu indicele elementului din tabloul proceselor în care a fost găsit indexul procesului de eliminat, în elementul din tabloului priorităților cu indicele precizat prin valoarea argumentului *prior*. A doua acțiune constă în copierea în elementul din tabloul proceselor în care a fost găsit indexul procesului de eliminat a indexului succesorului acestui proces, aflat în acest tablou în elementul de indice *ind_tsk*.

În al doilea caz, caracterizat prin faptul că procesul de eliminat nu este coadă de listă, eliminarea se efectuează prin simpla copiere în elementul din tabloul proceselor în care a fost

găsit indexul procesului de eliminat a indexului succesorului acestui proces, aflat, cum s-a precizat mai sus, în elementul acestui tablou cu indicele *ind_tsk*.

Revenim la cazul în care procesul de eliminat este unicul din lista circulară corespunzătoare priorității sale, pentru a adăuga că, în acest caz, se impune, suplimentar, să se verifice dacă nu cumva această prioritate era cea mai puternică dintre cele care aveau asociate liste circulare nevide.

Verificarea se efectuează prin compararea valorii argumentului *prior* cu valoarea înscrisă în elementul de indice *MAX_TSK* din tabloul priorităților. Egalitatea semnifică faptul că prioritatea procesului eliminat era cea mai puternică și conduce la necesitatea actualizării valorii elementului de indice *MAX_TSK*. Actualizarea se efectuează prin înscrierea în elementul în discuție a valorii celui mai mic indice din tabloul priorităților, care are asociat un element ce conține o valoare diferită de *MAX_TSK*.

Textul funcției *elim_lap ()*, scris în conformitate cu cele de mai sus, face obiectul figurii 6.3.4_3.

```
1.  void elim_lap (usshort ind_tsk, usshort prior)
2.    {
3.     register usshort j;
4      j = _lap_p[prior];
5.     if (j != MAX_TSK) {
6.        while ((_lap_t[j] != ind_tsk) && (_lap_t[j] != _lap_p[prior])) {
8.           j = _lap_t[j];
9.        }
10.       if (_lap_t[j] == ind_tsk) {
11.          if (ind_tsk == _lap_p[prior]) {
12.             if (j == ind_tsk) {
13.                _lap_t[ind_tsk] = MAX_TSK;
14.                _lap_p[prior] = MAX_TSK;
15.                if (_lap[MAX_TSK] == prior) {
16.                   j = prior;
17.                   do {
18.                      j++;
19.                   } while (_lap_p[j] == MAX_TSK);
20.                   _lap_p[MAX_TSK] = j;
21.                }
22.             }
23.             else {
24.                _lap_p[prior] = j;
25.                _lap_t[j] = _lap_t[ind_tsk];
26.                _lap_t[ind_tsk] = MAX_TSK;
27.             }
28.          }
```

Fig. 6.3.4_3. Textul funcției *elim_lap ()* (partea 1/2).

```
29.             else {
30.                 _lap_t[j] = _lap_t[ind_tsk];
31.                 _lap_t[ind_tsk] = MAX_TSK;
32.             }
33.         }
34.     }
35. }
```

Fig. 6.3.4_3. Textul funcției *elim_lap ()* (partea 2/2).

Funcția *urm_lap ()*

Această funcție are sarcina de a determina și indica succesorul procesului aflat în rulare. Acest succesor este procesul aflat curent în capul listei circulare cu cea mai puternică prioritate. Indexul său se află înscris în elementul din tabloul proceselor cu indicele dat de valoarea expresiei:

_lap_p[_lap_p[MAX_TSK]]

Totodată, funcția *urm_lap ()* asigura instalarea în coada listei circulare în cauză a procesului desemnat ca "următorul", înscriindu-i indexul în elementul din tabloul priorităților cu indicele *MAX_TSK*. Această operație este însoțită, implicit, de avansarea în cadrul listei, cu o poziție, și a celorlalte procese pe care ea le include.

Textul funcției *urm_lap ()* este redat în figura 6.3.4_4.

```
1.  usshort urm_lap (void)
2.    {
3.      _lap_p[_lap_p[MAX_TSK]] = _lap_t[_lap_p[_lap_p[MAX_TSK]]];
5.      return (_lap_p[_lap_p[MAX_TSK]]);
6.    }
```

Fig. 6.3.4_4. Textul funcției *urm_lap ()*.

Funcția *init_lap ()*

Această funcție are sarcina de a asigura inițializarea listei de așteptare la procesor, prin înscrierea valorii *MAX_TSK* în toate elementele tablourilor *_lap_t[]*, respectiv *_lap_p[]*.

Textul funcției *init_lap ()* este prezentat în figura 6.3.4_5.

```
1.  void init_lap (void)
2.    {
3.      register usshort i;
4.      for (I = 0; I < MAX_TSK; i++) {
5.          _lap_t[i] = MAX_TSK;
6.          _lap_p[i] = MAX_TSK;
7.      }
8.      _lap_p[MAX_TSK] = MAX_TSK;
9.    }
```

Fig. 6.3.4_5. Textul funcției *init_lap ()*.

6.3.5. Concluzii

Soluția prezentată a fost aplicată [ROBU'94b] în cadrul executivului de timp real RTC86, conceput și realizat de autor, dând deplină satisfacție. Simplitatea ce o caracterizează a asigurat eficiență procesului de dispecerizare. Este, în primul rând, ea cea care a permis încadrarea timpului de comutare la acest executiv în limita a 60 μs, pe un procesor INTEL 80386, la 33 MHz, după cum s-a menționat în concluziile subcapitolului precedent.

6.4. O soluție de excludere mutuală prin fanioane de excluziune active

6.4.1. Preliminarii

Consacrat, excluderea mutuală se asigură fie prin dezactivarea întreruperilor, fie cu ajutorul fanioanelor *TAS* (*TAS*: *Test And Set*), fie cu ajutorul semafoarelor [ALLW'81] [BALT'84] [TSCH'90].

Fiecare dintre aceste metode își are avantajele și dezavantajele sale [YOUN'82] [THOR'90] [ROBU'95g].

Metoda cea mai simplă -dezactivarea întreruperilor- are neajunsul de a implica riscul pierderii irecuperabile a unor cereri de întrerupere; în plus, această metodă nu este aplicabilă în cazul prelucrării concurente pe sisteme multiprocesor.

Metoda bazată pe fanioane *TAS*, caracterizată, de asemenea, prin simplitate, presupune irosirea unui cuantum important din capacitatea de prelucrare a procesorului în bucle de așteptare

activă și, în sistemele cu dispecerizare prin prioritizare, posibilitează apariția fenomenului de interblocare [CREȚ'84] [ELEȘ'91].

Metoda bazată pe semafoare este lipsită de dezavantajele celorlalte două dar, este mai complexă și mai mare consumator de timp [DIJK'65].

În cele ce urmează, se propune un mecanism original de excludere mutuală, cu o implementare *C*, sub sistemul de operare *MS-DOS*, care rezolvă problema accesului singular la resursele critice în aceeași manieră ca și mecanismele bazate pe semafoare, fiind, însă, mai simplu și, în cazul secțiunilor critice de scurtă durată, considerabil mai eficient [ROBU'94c].

Se definește noțiunea de fanion de excluziune activ, iar apoi se arată modul de implementare a fanioanelor propriu-zise.

Se pun în evidență funcțiile de gestiune a fanioanelor și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.4.2. Definirea mecanismului de excludere mutuală prin fanioane de excluziune active

Un *fanion de excluziune activ* este un ansamblu format dintr-un fanion *TAS* -pe care îl numim fanion de excluziune pasiv-, "*F*", și o coadă de așteptare, "*C*" [ROBU'94c] [ROBU'95g].

Fanionul de excluziune pasiv are rolul de a indica posibilitatea sau imposibilitatea inițierii de către un proces, la un moment, a secțiunii sale critice referitoare la o resursă pusă în corespondență cu fanionul.

Coada de așteptare servește înregistrării proceselor nesatisfăcute în tentativa lor de a-și demara secțiunea critică referitoare la resursa corespunzătoare fanionului.

Mecanismul de excludere mutuală prin fanioane de excluziune active este dezvoltat în jurul următoarei convenții:

- fanionul de excluziune este asociat resursei în cauză și numai ei; valoarea 0 a componentei *F* a fanionului arată că nici o secțiune critică referitoare la resursă nu este în derulare, iar valoarea 1 -contrariul;
- înainte de pătrunderea în secțiunea critică relativă la resursa în discuție, orice proces are obligația de a proceda la:
  1). efectuarea, în condiții de indivizibilitate, a operației *TAS* asupra fanionului.
  2). a). autoînscrierea în coada *C* a fanionului, *autoeliminarea din lista proceselor rulabile, autoblocarea și autoînscrierea în lista proceselor blocate, apelarea dispecerului,* și

repetarea, după deblocare, a obligației în curs de definire, începând cu 1) -toate acestea dacă operația *TAS* a găsit componenta *F* la 1.

b). inițierea secțiunii critice, dacă operația *TAS* a găsit componenta *F* la 0.

- la părăsirea secțiunii critice, orice proces are obligația de a proceda, printr-o secvență indivizibilă, la:

  1'). efectuarea operației *RES* (*RES*: *RESet*) asupra componentei *F* a fanionului.

  2'). *deblocarea* tuturor proceselor înscrise în coada *C* a fanionului și, implicit, vidarea acesteia, și *reînscrierea lor în lista proceselor rulabile*.

6.4.3. Aspecte privind implementarea mecanismului de excludere mutuală prin fanioane de excluziune active. Tipul de dată *FLAG*

Mecanismul de excludere mutuală prin fanioane de excluziune active are, ca elemente de bază, două funcții, corespunzătoare, una dintre ele, operațiilor 1) și 2), iar cealaltă, operațiilor 1') și 2'); se poate deduce ușor că aceste funcții vor fi plasate înaintea, respectiv la sfârșitul secțiunilor critice [ROBU'94c] [ROBU'95g].

Denumim aceste funcții "*_f_wait ()*", respectiv "*_f_signal ()*". Vom considera că, pentru efectuarea acțiunilor scrise cu caractere italice în textele corespunzătoare operațiilor 2), respectiv 2'), funcția *_f_wait ()* face apel la o funcție pe care o denumim "*sleep ()*", iar funcția *_f_signal ()* la o funcție pe care o denumim "*wakeup ()*"[ROBU'95g].

Pentru a oferi posibilitatea limitării duratelor intervalelor de timp în care procesele sunt ținute blocate, funcția *sleep ()* dispune de un argument, având numele *timeout*, prin care se poate specifica, în perioade ale ceasului de timp real, timpul după care procesul blocat să fie automat deblocat. Dacă, la apel, valoarea argumentului *timeout* este 0, atunci funcția *sleep ()* asigură o blocare pe termen nedefinit.

Pentru a se face posibilă implementarea funcției *sleep ()* conform celor de mai sus, este necesar ca în componența contextului logic al proceselor să fie prezente variabilele *sltime* și *to*, definite în paragraful 6.2.2 (a se revedea figurile 6.2.2_2 și 6.2.2_4). Variabila *sltime* are rolul de a prelua valoarea argumentului *timeout* și de a servi drept sediu de decrementare a acesteia, cât timp ea este nenulă, la fiecare intervenție a dispecerului cauzată de o întrerupere de la ceasul de timp real. Variabila *to* are rolul de a reține faptul că, prin decrementarea variabilei *sltime*, s-a ajuns la zero, apărând, deci, situația de "time-out".

Procesele blocate de funcția *sleep ()* vor fi înscrise într-o listă dedicată, *lista taskurilor blocate*, referită prin abrevierea *ltb*. Asupra acestei liste, se va opera cu ajutorul unor funcții clasice, având numele: *ins_ltb ()*, *elim_ltb ()*, *urm_ltb ()*, și *init_ltb ()* [APPE'84] [ROBU'94c]. Ele fac obiectul figurii 6.4.3_1

void ins_ltb (unsigned short ind_tsk)
/* inserează în *ltb* procesul cu indexul *ind_tsk* */

void elim_ltb (unsigned short ind_ltb)
/* elimină din *ltb* procesul cu indicele *ind_tsk* */

unsigned short urm_ltb (void)
/* determină indexul procesului aflat în capul *ltb* */

void init_ltb (void)
/* inițializează lista proceselor blocate */

Fig. 6.4.3_1. Funcțiile de gestionare a listei proceselor blocate.

Se menționează că, pentru ca apelantul funcției *sleep ()* să poată afla, la revenirea din aceasta, dacă deblocarea s-a făcut normal, adică: în limita de timp fixată prin argumentul *timeout*, sau forțat, la expirarea timpului, funcția returnează o valoare *unsigned int* nulă, în primul caz, respectiv nenulă, în al doilea; această valoare este generată, primar, de funcția *scheduler ()*.

Tinând seamă de cele de mai sus, funcțiile *sleep ()* și *wakeup ()* au implementările din figurile 6.4.3_2, respectiv 6.4.3_3.

```
1.  unsigned int sleep (unsigned short timeout)
2.    {
3.      CONTLOG *p;
4.      unsigned int aux;
5.      p = &_contlog[_task_crt];
6.      _lock_();
7.      p->status = BLOCAT;
8.      p->sltime = timeout;
9.      elim_lap (_task_crt);
10.     ins_ltb (_task_crt);
11.     aux = scheduler ();
12.     _unlock_();
13.     return (aux);
14.   }
```

Note:

1). "_task_crt" este o variabilă globală ce memorează indexul procesului aflat curent în rulare;

2). "_lock_()" salvează starea întreruperilor și apoi le dezactivează, iar "_unlock_()" reface starea găsită de _lock()_.

Fig. 6.4.3_2. Textul funcției *sleep ()*.

```
1.  void wakeup (unsigned short ind_tsk)
2.    {
3.     CONTLOG *p;
4.     p = &_contlog[ind_tsk];
5.     _lock_();
6.     if (p->status == BLOCAT) {
7.        p->status = EXEC;
8.        elim_ltb (ind_tsk);
9.        ins_lap (ind_tsk);
10.    }
11.    _unlock_();
12.   }
```

Fig. 6.4.3_3. Textul funcției *wakeup ()*.

Pentru implementarea mecanismului de excludere mutuală prin fanioane active, s-a definit un tip de dată, reprezentând fanionul propriu-zis, ca o structură cu numele *FLAG*, ce cuprinde:

- o variabilă de tipul *unsigned short*, reprezentând fanionul de excluziune pasiv; fie "*f*" numele acestei variabile;
- un tablou de tipul *unsigned short*, cu *MAX_TSK* elemente, dedicat a servi drept coadă a fanionului de excluziune activ; fie "*coada*" numele acestui tablou ("*MAX_TSK*" este numărul maxim al proceselor acceptate în sistem).
- o variabila de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista intrarea în și ieșirea din coada fanionului; fie "*pie*" numele acestei variabile.

Tipul de dată *FLAG* se introduce, deci, printr-o declarație de forma:

```
typedef struct {
        unsigned short f;
        unsigned short coada[MAX_TSK];
        unsigned short *pie;
      } FLAG;
```

Fig. 6.4.3_4. Tipul de dată *FLAG*.

Este firesc ca structurile *FLAG* să fie grupate sub forma unui tablou; acesta trebuie să fie de clasă *extern* [ROBU'95g]. Fie el de dimensiune *MAX_FLG* și cu numele "*_flg[]*".

Referirile la un fanion concret se fac cu ajutorul indicelui elementului prin care el este implementat, asociindu-se identificatori sugestivi fiecărui indice. Este practic ca valoarea unui asemenea identificator să fie stabilită printr-o funcție care asigură alocarea elementelor tabloului

de fanioane. O vom denumi "*f_creat ()*". Disocierea unui identificator de elementul cu care este pus în legătură prin funcția *f_creat ()* se face cu ajutorul unei funcții de dezalocare, denumită "*f_destroy ()*".

În fine, pentru inițializarea tabloului *_flg[]*, mecanismul de excludere mutuală prin fanioane de excluziune active dispune de o funcție denumită "*f_init ()*".

6.4.4. Funcțiile de operare asupra fanioanelor de excluziune active

Ansamblul celor cinci funcții prin care utilizatorul poate face referiri la fanioane este redat, sintetic, în figura 6.4.4_1.

void f_init (void)
/* inițializează tabloul de fanioane */

usshort f_creat (void)
/* creează un fanion, furnizând indicele elementului */
/* tabloului *_flg []* pe care îl alocă în acest sens */

void f_destroy (usshort ind_flg)
/* distruge un fanion, dezalocând elementul aferent */
/* din tabloul *_flg []*; argumentul *ind_flg* reprezintă */
/* indicele respectivului element */

unsigned f_wait (usshort ind_flg, usshort timeout)
/* asigură, dacă este cazul printr-o așteptare limitată */
/* superior de argumentul *timeout*, ca procesul */
/* curent să-și poată iniția secțiunea critică */
/* referitoare la resursa asociată fanionului cu */
/* indicele *ind_flg*, dacă așteptarea depășește limita, */
/* funcția returnează o valoare *unsigned* nenulă; */
/* altfel, valoarea returnată este nulă */

void f_signal (usshort ind_flg)
/* semnalează încheierea unei secțiuni critice */
/* referitoare la resursa asociată fanionului cu */
/* indicele *ind_flg*, deblocând toate procesele aflate */
/* în coada acestuia, și predă controlul dispecerului */

Notă:
"usshort" este prescurtare pentru "unsigned short".

Fig. 6.4.4_1. Funcțiile aferente mecanismului de excludere mutuală prin fanioane de excluziune active.

Funcția *f_init ()*

Această funcție are rolul de a inițializa tabloul de fanioane *_flg[]*, prin stabilirea valorii *NULL* pentru componentele *pie* ale tuturor elementelor sale.

Textul *C* al funcției *f_init ()* este următorul:

```
1.   void f_init (void)
2.     {
3.       register FLAG *fg;
4.       register unsigned j;
5.       fg = &_flg[0];
6.       for (j = 0; j < MAX_FLG; j++) {
7.           fg->pie = NULL;
8.           fg++;
9.       }
10.    }
```

Fig. 6.4.4_2. Textul funcției *f_init ()*.

Funcția *f_creat ()*

Funcția *f_creat ()* are rolul de a identifica un element liber al tabloului de fanioane *_flg[]* și de a-l aloca, returnând indicele-i.

De asemenea, funcției *f_creat ()* îi revine sarcina de a poziționa pointerul *pie* al fanionului pe care îl alocă pe primul element al cozii și de a inițializa la valoarea 0 componenta *f* a acestui fanion.

Textul funcției *f_creat ()* este redat în figura 6.4.4_3.

```
1.   usshort f_creat (void)
2.     {
3.       register FLAG *fg;
4.       register unsigned j;
5.       fg = &_flg[0];
6.       j = 0;
7.       _lock_ ();
```

Fig. 6.4.4_3. Textul funcției *f_creat ()* (partea 1/2).

```
8.      while ((fg->pie) && (j < MAX_FLG)) {
9.          fg++;
10.         j++;
11.         /*în ciclul while, s-a considerat ca semn */
12.         /*distinctiv pentru un fanion încă nealocat */
13.         /*valoarea NULL a componentei sale pie */
14.         /* (vezi funcțiile f_init () și f_destroy ()) */
15.         if (j < MAX_FLG) {
16.             fg->pie = &(fg->coada[0]);
17.             /*se pozitionează pie pe primul element al cozii */
19.             fg->f = 0;
20.         }
21.         else {
22.             /*se generează un mesaj de eroare */
23.         }
24.         _unlock_();
25.         return (j);
26.     }
```

Fig. 6.4.4_3. Textul funcției *f_creat ()* (partea 2/2).

Funcția *f_destroy ()*

Funcția *f_destroy ()* are rolul de a dezaloca elementul tabloului de fanioane *_flg []* cu indicele specificat prin argumentul ei, atribuind valoarea *NULL* componentei *pie* a acestui element.

Acțiunea de distrugere a funcției *f_destroy ()* se exercită doar dacă ea găsește coada fanionului vidă. Altfel, funcția se rezumă la generarea unui mesaj de eroare.

Textul funcției *f_destroy ()* este redat în figura 6.4.4_4.

```
1.  void f_destroy (usshort ind_flg)
2.    {
3.     register FLAG *fg;
4.     fg = &_flg[ind_flg];
5.     _lock_();
6.     if (fg->pie != &(fg->coada[0])) {
7.         /*eroare: coada fanionului este nevidă */
8.     }
9.     else {
10.            fg->pie = NULL;
11.        }
12.    _unlock_();
13.   }
```

Fig. 6.4.4_4. Textul funcției *f_destroy ()*.

Funcția *f_wait ()*

Funcția *f_wait ()* are rolul de a efectua operația *TAS* asupra componentei *f* a fanionului cu indicele *ind_flg* și de a permite procesului care o execută inițierea secțiunii sale critice, dacă această operație găsește componenta *f* la valoarea **0**, respectiv de a bloca acest proces și de a-l înscrie în coada fanionului, în caz contrar.

Operația de blocare a procesului în cauză se execută prin apelul funcției *sleep ()*. Acesteia i se pasează, în momentul apelului, valoarea argumentului *timeout*, prin care se precizează timpul limită cât procesul poate rămâne blocat. Dacă la expirarea acestui timp, procesul în cauză este încă blocat, se va produce deblocarea lui automată. O asemenea deblocare denotă o anomalie, care trebuie tratată în mod corespunzător de către utilizator. În sprijinul acestuia, funcția *f_wait ()* returnează o valoare *unsigned*, nenulă, în cazul în care anomalia s-a petrecut, respectiv nulă, altfel.

Textul funcției *f_wait()* este următorul:

```
1.  unsigned f_wait (usshort ind_flg, usshort timeout)
2.    {
3.     register FLAG *fg;
4.     unsigned aux;
5.     fg = &_flg[ind_flg];
6.     _lock_();
7.     while ((fg->f) &&( !aux)) {
8.        *fg->pie = _task_crt;
9.        fg->pie++;
10.          aux = sleep (timeout);
11.       }
12.       if (!aux) {
13.          fg->f = 1;
14.       }
15.       _unlock_();
16.       return (aux);
17.    }
```

Fig. 6.4.4_5. Textul funcției *f_wait ()*.

Funcția *f_signal ()*

Funcția *f_signal ()* are rolul de a efectua operația *RES* asupra componentei *f* a fanionului cu indicele *ind_flg* și de a debloca toate procesele aflate în coada acestui fanion. Deblocarea proceselor se asigură prin apelul funcției *wakeup ()*. Dacă nici un proces nu este găsit în coadă,

funcția *f_signal ()* asigură rămânerea în rulare a procesului care o execută, nemaifăcând apel la dispecer.

Textul functiei *f_signal ()* este redat în figura 6.4.4_6.

```
1.  void f_signal (usshort ind_flg)
2.    {
3.     register FLAG *fg;
4.     register usshort tsk;
5.     fg = &_flg[ind_flg];
6.     _lock_0;
7.     while (fg->pie != &(fg->coada[0])) {
8.        fg->pie--;
9.        tsk = *(fg->pie);
10.       wakeup (tsk);
11.    }
12.    _unlock_0;
13.   }
```

Fig. 6.4.4_6. Textul funcției *f_signal ()*.

6.4.5. Concluzii

Mecanismul de excludere mutuală propus este avantajos mai ales în cazul secțiunilor critice cu durata mai mică decât perioada de comutare a proceselor și referitoare la resurse critice puternic disputate.

Într-un asemenea caz, presupunând o dispecerizare prin rotație, dacă un proces își inițiază o secțiune critică referitoare la o resursă cu puțin înaintea momentului unei comutări, atunci, cu ocazia comutării, el va pierde controlul asupra procesorului, împiedicând, însă, alte procese să acceseze resursa în cauză. Drept urmare, acestea se vor bloca, rând pe rând. La un moment, reprimind controlul, primul proces își va încheia, relativ rapid, secțiunea critică antamată mai devreme.

În ipotezele considerate, ar fi avantajos ca toate procesele blocate la resursa respectivă să fie deblocate deodată, pentru că, în mod normal, cuantumul de timp ce va fi alocat fiecăruia dintre ele pentru o repriză de rulare va fi suficient pentru parcurgerea completă a secțiunii critice a fiecăruia, într-o singură astfel de repriză. Ori, niciunul dintre mecanismele consacrate de excludere mutuală nu asigură acest lucru [PERE'90]. Mecanismul propus, însă, o face.

În fine, considerăm demn de a fi remarcat faptul că, pentru implementarea mecanismului propus, s-a adoptat o soluție bazată pe principii moderne de programare, ce definesc ceea ce a primit numele de programare încapsulată.

6.5. O soluție de excludere mutuală și sincronizare prin semafoare

6.5.1. Preliminarii

După cum se cunoaște, semafoarele, introduse în 1965, de către olandezul E.W.Dijkstra [DIJK'65], joacă un rol de primă importanță în rezolvarea problemelor de excludere mutuală și de sincronizare în programarea concurentă.

Un semafor este un ansamblu format dintr-o variabilă întreagă, "*I*", și o coadă de așteptare, "*C*". Variabila servește la a se stabili dacă, la un moment, un proces poate trece de semafor sau nu; în cazul din urmă, procesul devine blocat. Se consideră că semaforul este "pe verde" atunci când variabila este pozitivă și "pe roșu" atunci când variabila este negativă. ţn primul caz, modulul variabilei arată câte procese vor putea trece de semafor -găsindu-l "pe verde"- fără a fi supuse unei blocări. ţn al doilea caz, modulul variabilei arată câte procese au încercat să treacă de semafor și, nereușind, întrucât l-au găsit "pe roșu", au devenit blocate. Coada servește înregistrării proceselor devenite blocate ca urmare a încercării de a trece de semafor.

În momentul creerii semaforului, variabilei i se atribuie o valoare inițială nenegativă, iar coada este vidă [ANDL'78] [BALT'84]. Pe parcursul utilizării unui semafor, variabila sa este supusă exclusiv unor operații de incrementare și decrementare. De aceea, ea poartă denumirea de "*contor*"[ROBU'95g]. În ceea ce privește coada semafoarelor, se precizează că, de obicei, ea este gestionată fie după principiul *FIFO* (*FIFO: First In, First Out*), fie pe bază de priorități [THOR'90]. Mai frecvent, este adoptat principiul *FIFO*.

Realizarea excluderii mutuale sau a sincronizării cu ajutorul semafoarelor are la bază două primitive, numite *P*, respectiv *V*[DIJK'65].

Primitiva *P* introduce următoarele operații în cadrul procesului care o execută:

1) Decrementare contor: $I = I-1$;
2) Dacă $I < 0$: eliminare proces dintre procesele rulabile, blocare proces, înscriere proces în coada de așteptare, comutare.

Primitiva *V* introduce următoarele operații în cadrul procesului care o execută:

1) Incrementare contor: $I = I+1$;
2) Dacă $I \leq 0$: eliminare din coadă și deblocare proces aflat în capul cozii, introducere proces deblocat în rândul proceselor rulabile, comutare.

În cele ce urmează, se propune un mecanism original de excludere mutuală și sincronizare prin semafoare, cu o implementare *C*, sub sistemul de operare *MS-DOS* [ROBU'94d].

Este introdus un tip de dată dedicat, numit *SEMAPHORE*.

Sunt puse în evidență funcțiile de gestiune a semafoarelor și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.5.2. Tipul de dată *SEMAPHORE*

Anatomia și fiziologia unui mecanism de excludere mutuală și sincronizare bazat pe semafoare se profilează odată cu stabilirea statutului acordat semaforului propriu-zis [JACK'80] [TSCH'90]. Pentru mecanismul pe care îl propunem, s-a adoptat ca semaforul să fie o structură, cu numele *SEMAPHORE*, cuprinzând [ROBU'94d] [ROBU'95g]:

- 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*;
- o variabilă de tipul *short*, având rolul de a păstra valoarea inițială a contorului, cu scopul de a se permite a stabili dacă, la un moment, semaforul este în folosină sau nu; s-a dat numele "*valincont*" acestei variabile, ca abreviere de la *valoare inițiala contor*.

Tipul de dată *SEMAPHORE* se introduce, deci, așa cum se arată în figura 6.5.2_1.

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

Fig. 6.5.2_1. Tipul de dată *SEMAPHORE*.

Este firesc ca toate semafoarele de care se dispune să fie grupate sub forma unui tablou. Dacă admitem că numărul maxim de semafoare este *MAX_SEM*, atunci acest tablou, cu necesitate de clasă *extern* -fie *_sem[]* numele lui-, va fi declarat astfel:

```
extern SEMAPHORE _sem[MAX_SEM];
```

Fig. 6.5.2_2. Declararea tabloului de semafoare *_sem[]*.

Evident, un semafor concret va fi exploatat cu ajutorul indicelui său în cadrul tabloului de semafoare, introducând un indentificator sugestiv pentru fiecare indice.

6.5.3. Funcțiile de operare asupra semafoarelor

Dată fiind importanța și delicatețea semafoarelor într-un sistem cu prelucrare concurentă, se impune ca utilizatorul să poată face referiri la ele doar prin intermediul unor funcții de nucleu [APPE'84] [THOR'90]. Clasic, acestea sunt cele prezentate, sintetic, în figura 6.5.3_1 [ROBU'94d] [ROBU'95g].

```
void s_init (void)
/* inițializează tabloul de semafoare prin */
/* atribuirea valorii NULL componentelor */
/* pi ale elementelor sale */

usshort s_creat (short valinit)
/* alocă un element al tabloului _sem[], */
/* atribuie componentelor contor și valincont */
/* ale a elementului alocat valoarea valinit */
/* și furnizează indicele acestui element */

void s_destroy (usshort ind_sem)
/* dezalocă elementul tabloului _sem[] cu */
/* indicele ind_sem, punându-l la dispoziția */
/* funcției s_creat () */

usshort s_wait (usshort ind_sem, usshort timeout)
/* efectuează funcția P asupra semaforului */
/* cu indicele ind_sem și înregistrează */
/* valoarea argumentului timeout ca timp */
/* limită în care procesul curent trebuie să */
/* treacă de semafor; returnează o valoare */
/* nulă dacă trecerea se produce înaintea */
/* expirării timpului, altfel -una nenulă */

void s_signal (usshort ind_sem)
/* efectuează funcția V asupra semaforului */
/* cu indicele ind_sem */
```

Fig. 6.5.3_1. Funcțiile relative la semafoare.

Funcția *s_init ()*

Această funcție are rolul de a inițializa tabloul de semafoare, prin stabilirea valorii *NULL* pentru componetele *pi* ale tuturor elementelor sale.

Textul funcției *s_init ()* este redat în figura 6.5.3_2.

```
1.  void s_init (void)
2.    {
3.     register SEMAPHORE *s;
4.     register unsigned j;
5.     s = &_sem[0];
6.     for (j = 0; j<MAX_SEM; j++) {
7.        s->pi = NULL;
8.        s++;
9.     }
10.     }
```

Fig. 6.5.3_2. Textul funcției *s_init ()*.

Funcția *s_creat ()*

Funcția *s_creat ()* are rolul de a identifica un element liber al tabloului de semafoare *_sem[]* și de a-l aloca, atribuind componentelor sale *contor* și *valincont* valoarea precizată prin argumentul *valinit*; indicele elementului este făcut cunoscut apelantului prin returnare. În plus, acestei funcții îi revine și sarcina de a pregăti semaforul alocat pentru acțiunile pe care funcțiile *s_wait ()* și *s_signal ()* le vor intreprinde asupra sa. Achitarea acestei sarcini comportă poziționarea pointerilor *pi* și *pe* din componența tipului de dată *SEMAPHORE*, către primul element al cozii.

Rezultă pentru funcția *s_creat ()* textul din figura 6.5.3_3.

```
1.  usshort s_creat (short valinit)
2.    {
3.     register SEMAPHORE *s;
4.     register unsigned n;
5.     s = &_sem[0];
6.     n = 0;
```

Fig. 6.5.3_3. Textul funcției *s_creat ()* (partea 1/2).

```
7.      _lock_();
8.      while ((s->pi) && (n<MAX_SEM)) {
9.          s++;
10.         n++;
11.     }
12.     /*în ciclul while, s-a considerat că un */
13.     /*semafor încă nealocat se distinge prin */
14.     /*valoarea NULL a componentei sale pi */
15.     /* (a se vedea s_init () și s_destroy ()) */
16.     if (n < MAX_SEM) {
17.         s->contor = valinit;
18.         s->valincont = valinit;
19.         s->pi = &(s->coada[0]);
20.         s->pe = &(s->coada[0]);
21.     }
22.     else {
23.         /* eșec: nici un semafor nu a fost găsit liber */
25.     }
26.     _unlock_();
27.     return (n);
28.     /*se furnizează indicele semaforului alocat */
29. }
```

Fig. 6.5.3_3. Textul funcției *s_creat ()* (partea 2/2).

Funcția *s_destroy ()*

Funcția *s_destroy ()* are rolul de a dezaloca elementul tabloului de semafoare *_sem[]* cu indicele specificat prin valoarea argumentului ei. Dezalocarea se realizează prin readucerea componentei *pi* a acestui element la valoarea inițială *NULL*. In urma acțiunii funcției *s_destroy ()*, elementul în cauză este lăsat la dispoziția funcției *s_creat ()*, în vederea unei noi alocări.

Se precizează că regulile care guvernează lucrul cu semafoare interzic distrugerea unui semafor aflat în folosință. Fidelă fiind acestor reguli și ținând seamă că variabila *contor* este cea care arată dacă, la un moment, un semafor este în folosință sau nu (când variabila *contor* este la o altă valoare decât cea la care a fost inițializată, atunci semaforul este în folosință, altfel, nu este), funcția *s_destroy ()* va efectua, mai întâi, o verificare asupra acestei variabile și va proceda la distrugere doar dacă ea o permite. Altfel, acțiunea funcției se rezumă la generarea unui mesaj de eroare.

Textul funcției *s_destroy ()* este dat în figura 6.5.3_4.

```
1.  void s_destroy (usshort ind_sem)
2.    {
3.     register SEMAPHORE *s;
4.     s = &_sem[ind_sem];
5.     _lock_();
6.     if (s->contor != s->valincont) {
7.     /*eroare: se încearcă distrugerea */
8.     /*unui semafor aflat în folosință */
9.     }
10.       else {
11.          s->pi = NULL;
12.          /*se atribuie valoarea NULL componentei */
13.          /*pi a semaforului care se distruge */
14.       }
15.       _unlock_();
16.    }
```

Fig. 6.5.3_4. Textul funcției s_destroy ().

Funcția *s_wait ()*

Funcția *s_wait ()* are rolul de a materializa operațiile care definesc primitiva *P*, cu referire la semaforul precizat prin primul său argument. O parte dintre aceste operații, și anume cele de scoatere a procesului curent dintre procesele rulabile și de blocare a acestui proces, sunt asigurate prin apelul funcției *sleep ()*.

Cel de-al doilea argument al funcției *s_wait ()* este destinat transmiterii lui la funcția *sleep ()*, pentru ca aceasta să asigure limitarea timpului cât procesul în cauză poate rămâne blocat la semafor. Ajungerea unui proces pus în așteptare la un semafor în situația deblocării automate reprezintă o anomalie care trebuie tratată în mod corespunzător de către utilizator. În sprijinul acestuia, funcția *s_wait ()* returnează o valoare *unsigned short*, nenulă, în cazul în care anomalia s-a petrecut, și nulă, altfel (reamintim: funcția *scheduler ()* este cea care generează primar această valoare; funcția *s_wait ()* o preia de la *scheduler ()* prin intermediul funcției *sleep ()*).

Legat de operația de înscriere a procesului curent în coada de așteptare la semafor, se precizează că ea trebuie să se efectueze circular, prin repoziționarea pointerului *pi* pe primul element al tabloului care implementează coada, după ce el a servit înscrierii unui proces în ultimul element al acestui tablou.

Conform celor de mai sus, funcția *s_wait ()* se poate implementa prin textul *C* redat în figura 6.5.3_5.

```
1.  usshort s_wait (usshort ind_sem, usshort timeout)
2.    {
3.     register SEMAPHORE *s;
4.     register unsigned aux;
5.     s = &_sem[ind_sem];
6.     aux = 0;
7.     /*se inițializează variabila aux pentru a */
8.     /*indica ieșirea normală din funcția s_wait */
9.     _lock_();
10.       if (-- s->contor < 0) {
11.        *s->pi = _task_crt;
12.        /*se înscrie procesul curent în */
13.        /*coada de așteptare la semafor */
14.        if (s->pi == &(s->coada[MAX_TSK-1])) {
15.        s->pi = &(s->coada[0]);
16.        /*se actualizează valoarea pointerului */
17.        /*de intrare în coadă, astfel încât să */
18.        /*indice primul element al tabloului */
19.        }
20.        else {
21.           s->pi++;
22.        /*se actualizează valoarea pointerului */
23.        /*de intrare în coadă, astfel încât să */
24.        /*indice următorul element al tabloului */
25.        }
26.        aux = sleep (timeout);
27.        /*se elimină procesul curent dintre */
28.        /*procesele rulabile și se autoblochează; */
29.        /*se returnează o valoare nenulă în caz */
30.        /*de "timeout" și nulă, altfel */
31.        }
32.        _unlock_();
33.        return (aux);
34.      }
```

Fig. 6.5.3_5. Textul funcției *s_wait ()*.

Este lesne de înțeles că, dacă în urma decrementării contorului semaforului, valoarea acestuia este negativă, funcția *s_wait ()* se va derula în două reprize. Prima repriză va cuprinde liniile [1...26], iar a doua repriză liniile [26...34]. Asta înseamnă că o parte a funcției *sleep ()* -linia 26- se va rula în repriza întâi, iar cealaltă parte -în repriza a doua. Între cele două reprize, va avea loc rularea altor procese, printre care, în mod normal, trebuie să fie și cele care au executat înaintea procesului curent funcția *s_wait ()* cu referire la același semafor. Pentru a atrage atenția asupra acestor aspecte a fost supraimprimată linia 26 din figura 6.5.3_5.

Funcția *s_signal ()*

Funcția *s_signal ()* are rolul de a materializa operațiile care definesc primitiva *V*, cu referire la semaforul precizat prin argumentul său. O parte dintre aceste operații, și anume cele de deblocare a procesului aflat în capul cozii semaforului și de înscriere a acestui proces în rândul proceselor rulabile, sunt asigurate prin apelul funcției *wakeup ()*.

Legat de operația de determinare a procesului aflat în capul cozii semaforului și de eliminare a lui din această coadă, se precizează că ea trebuie să se efectueze circular, prin poziționarea pointerului *pe* către primul element al tabloului care implementează coada, după ce el a servit extragerii unui proces înscris în ultimul element al acestui tablou.

Rezultă următorul text pentru funcția *s_signal ()*:

```
1.  void s_signal (usshort ind_sem)
2.    {
3.     register SEMAPHORE *s;
4.     register usshort tsk;
5.     sem = &_sem [ind_sem];
6.     _lock_();
7.     if(++ s->contor <= 0) {
8.        tsk = *s->pe;
9.        /*se determină procesul aflat */
10.       /*în capul cozii de așteptare */
11.       if (s->pe == &(s->coada[MAX_TSK-1])) {
12.          s->pe = &(s->coada[0]);
13.          /*se actualizează valoarea pointerului */
14.          /*de ieșire  din coadă, astfel încât să */
15.          /*indice  primul  element  al tabloului */
16.       }
17.       else {
18.          s->pe++;
19.          /*se actualizează  valoarea pointerului */
20.          /*de ieșire  din coadă, astfel încât să */
21.          /*indice următorul element al tabloului */
22.       }
23.       wakeup (tsk);
24.       /*se deblochează procesul găsit în capul */
25.       /*cozii de așteptare la semafor și se */
26.       /*înscrie printre procesele rulabile */
27.    }
28.    _unlock_();
29.  }
```

Fig. 6.5.3_6. Textul funcției *s_signal ()*.

Se face observația că apelul funcției *wakeup ()* va provoca o acțiune de comutare, prin care este posibil ca procesul curent să piardă controlul asupra procesorului, chiar dacă el rămâne în continuare rulabil, în favoarea procesului proaspăt deblocat cu ajutorul acestei funcții. Rezultă, prin urmare, o posibilă fragmentare a rulării funcției *s_signal ()*, în două reprize, în cazul în care, în urma incrementării contorului semaforului, valoarea sa este negativă sau nulă, indicând că cel puțin un proces se află în coada de așteptare la semafor. Prima repriză va cuprinde liniile [1...23], iar a doua repriză -liniile [23...29]. Aceasta înseamnă că o parte a funcției *wakeup ()* -linia 23- se va rula în repriza întâi, iar cealaltă parte -în repriza a doua. Între cele două reprize, poate avea loc rularea altor procese. Pentru a atrage atenția asupra acestor aspecte a fost supraimprimată linia 23 din figura 6.5.3_6.

6.5.4. Concluzii

Soluția propusă conduce la un mecanism de excludere mutuală și de sincronizare ușor de utilizat și, totodată, slab consumator de timp procesor.

6.6. O soluție de excludere mutuală prin blocuri resursă

6.6.1. Preliminarii

După cum se cunoaște, un neajuns al semafoarelor, aparent minor, dar, în practică, sursă a numeroase erori, greu de depistat în aplicațiile complexe, este că, la nivelul execuției programelor, nu există garanția delimitării corecte a secțiunilor critice prin funcțiile ce implementează primitivele *P*, respectiv *V*.

De exemplu, o dublă eroare de programare ca cea surprinsă în figura 6.6.1_1, nu va putea fi semnalată la compilare, dar nici în timpul rulării, conducând la un comportament al programului greu de explicat.

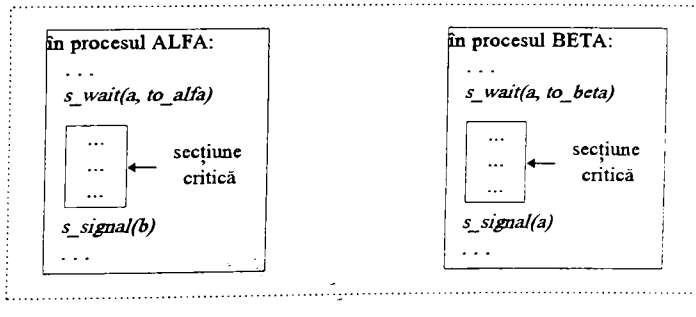


Fig. 6.6.1_1. Exemplu de utilizare greșită a primitivelor *P* și *V*.

Pentru a se putea semnala, în timpul rulării, existența unor situații ca cea din figura 6.6.1_1, soluția este să se extindă accepțiunea consacrată a noțiunii de semafor [DIJK'65], cu o componentă în care să se înregistreze indexul procesului ce, executând primitiva *P* asupra semaforului, reușește să-și inițieze secțiunea critică [TSCH'90] [ROBU'95g]. O asemenea extindere transformă semaforul într-un "*bloc resursă*". Componenta specifică a blocului resursă oferă primitivei *V* posibilitatea de a o consulta și folosi pentru condiționarea execuției sale propriu-zise de concordanța dintre indexul procesului în care această primitivă este programată și valoarea pe care ea -componenta- o conține înregistrată.

Se menționează că se obișnuiește ca procesul al cărui index este înregistrat în componenta specifică a unui bloc resursă să fie considerat "*proprietarul*" curent al "*resursei*" în cauză la un moment, și că, pentru perechea de primitive *P* și *V* aferente unui bloc resursă este folosit termenul "*gestionar de resursă*".

În continuare, se propune un mecanism original de excludere mutuală prin blocuri resursă, cu o implementare *C*, sub sistemul de operare *MS-DOS* [ROBU'94e].

Este introdus un tip de dată dedicat, numit *RESOURCE*.

Sunt puse în evidență funcțiile de gestiune a blocurilor resursă și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.6.2. Tipul de dată *RESOURCE*

Atât la nivel conceptual, cât și în ceea ce privește implementarea, un mecanism de excludere mutuală prin blocuri resursă este amprentat de statutul pe care îl are blocul resursă propriu-zis. În mecanismul propus, blocul resursă este o structură, cu numele *RESOURCE*, cuprinzând [ROBU'94e] [ROBU'95g]:

- un tablou de tipul *unsigned short*, cu *MAX_TSK* (numărul maxim al proceselor acceptabile în sistem) elemente, destinat a servi drept coadă de așteptare la blocul resursă; s-a ales numele "*coada[]*" pentru acest tablou;
- o variabilă de tipul *short*, reprezentând contorul blocului resursă; s-a ales numele "*contor*" pentru această variabilă;
- o variabilă de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista intrarea în coada blocului resursă; s-a ales numele "*pi*" pentru această variabilă;
- o variabilă de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista ieșirea din coada blocului resursă; s-a ales numele "*pe*" pentru această variabilă;
- o variabilă de tipul *unsigned short*, dedicată înregistrării indexului proprietarului curent al resursei căreia îi corespunde blocul resursă; s-a ales numele "*prop*" pentru această variabilă.

Tipul de dată *RESOURCE* se va introduce, deci, așa cum se arată în figura 6.6.2_1.

```
typedef struct {
        usshort coada [MAX_TSK];
        short  contor;
        usshort *pi;
        usshort *pe;
        usshort prop;
      } RESOURCE;
```

Fig. 6.6.2_1. Tipul de dată *RESOURCE*.

Este firesc ca toate blocurile resursă disponibile într-un sistem să fie grupate sub forma unui tablou. Dăm numele *_res[]* acestui tablou. Dacă admitem că numărul maxim de blocuri resursă este *MAX_RES*, atunci tabloul *_res[]*, care, din motive lesne de înțeles, trebuie să fie de clasă *extern*, va fi declarat așa cum se arată în figura 6.6.2_2.

```
extern RESOURCE  _res[MAX_RES];
```

Fig. 6.6.2_2. Declararea tabloului de blocuri resursă *_res[]*.

Pe baza celor de mai sus, un bloc resursă concret va fi exploatat cu ajutorul indicelui său în cadrul tabloului de blocuri resursă, introducând câte un identificator sugestiv, pentru fiecare indice.

6.6.3. Funcțiile de operare asupra blocurilor resursă

Dată fiind importanța și delicatețea blocurilor resursă într-un sistem cu prelucrare concurentă, se impune ca utilizatorul să poată face referiri la ele doar prin intermediul unor funcții ale sistemului. Clasic [ALLW'81][APPE'84], acestea sunt cele prezentate, sintetic, în figura 6.6.3_1.

```
void  r_init (void)
/* inițializează tabloul de blocuri */
/* resursă, prin atribuirea valorii NULL */
/* componentelor pi ale elementelor sale */

usshort r_creat (void)
/* alocă un element al tabloului _res[], */
/* atribuie componentei contor a elementului */
/* alocat valoarea 1 și furnizează indicele-i */

void r_destroy (usshort ind_res)
/* dezalocă elementul tabloului _res[] */
/* cu indicele ind_res, repunându-l la */
/* dispoziția funcției r_creat () */
```

Fig. 6.6.3_1. Funcțiile de exploatare a blocurilor resursă (partea 1/2).

```
usshort r_wait (usshort ind_res, usshort timeout)
/* efectuează operația P asupra blocului */
/* resursă cu indicele ind_res și reține */
/* valoarea argumentului timeout ca timp */
/* limită în care procesul curent poate */
/* rămâne blocat la blocul resursă; */
/* returnează o valoare nulă, dacă deblocarea */
/* se produce înaintea expirării */
/* timpului, altfel -una nenulă; înregistrează */
/* în câmpul prop al blocului */
/* resursă ind_res valoarea indexului */
/* procesului curent, când acesta este */
/* autorizat să-și continue rularea */

void r_signal (usshort ind_res)
/* efectuează operația V asupra blocului */
/* resursă cu indicele ind_res, dacă */
/* proprietarul resursei corespondente */
/* acestuia este procesul curent, altfel */
/* semnalează eroare */
```

Fig. 6.6.3_1. Funcțiile de exploatare a blocurilor resursă (partea 2/2).

Funcția *r_init ()*

Această funcție are rolul de a inițializa tabloul de blocuri resursă. Inițializarea constă în stabilirea valorii *NULL* pentru componetele *pi* ale tuturor elementelor tabloului.

Textul funcției *r_init ()* este redat în figura 6.6.3_2.

```
1.  void r_init (void)
2.    {
3.      register RESOURCE *r;
4.      register unsigned j;
5.      r = &_res[0];
6.      for (j = 0; j < MAX_RES; j++) {
7.          r->pi = NULL;
8.          r++;
9.      }
10.     }
```

Fig. 6.6.3_2. Textul funcției *r_init ()*.

Funcția *r_creat ()*

Din punct de vedere al utilizatorului, funcția *r_creat ()* are rolul de a identifica un element liber al tabloului de blocuri resursă *_res[]* și de a-l aloca, atribuind componentei sale *contor* valoarea 1; indicele elementului este făcut cunoscut apelantului prin returnare. In plus, acestei funcții îi revine și sarcina de a pregăti blocul resursă alocat pentru acțiunile pe care funcțiile *r_wait ()* și *r_signal ()* le vor intreprinde asupra sa. Achitarea acestei sarcini comportă poziționarea pointerilor *pi* și *pe* din componența tipului de dată *RESOURCE*, către primul element al cozii.

Rezultă, pentru funcția *r_creat ()*, textul din figura 6.6.3_3.

```
1.  usshort r_creat (void)
2.    {
3.     register RESOURCE *r;
4.     register unsigned n;
5.     r = &_res[0];
6.     n = 0;
7.     _lock_();
8.     while ((r->pi) && (n < MAX_RES)) {
9.        r++;
10.          n++;
11.       }
12.       /*în ciclul while, s-a considerat că un bloc */
13.       /*resursă încă nealocat se distinge prin */
14.       /*valoarea NULL a componentei sale pi */
15.       /*  (a se vedea r_init () și r_destroy ())  */
16.       if (n < MAX_RES) {
17.       r->contor = 1;
18.       r->pi = &(r-> coada[0]);
19.       r->pe = &(r-> coada[0]);
20.    }
21.    else {
22.       /*eșec: nici un bloc resursă nu a fost găsit liber */
23.    }
24.    _unlock_();
25.    return (n);
26.    /*se furnizează indicele blocului resursă alocat */
27.  }
```

Fig. 6.6.3_3. Textul funcției *r_creat ()*.

Funcția *r_destroy ()*

Din punct de vedere al utilizatorului, funcția *r_destroy ()* are rolul de a dezaloca elementul tabloului de blocuri resursă *_res[]* cu indicele specificat prin valoarea argumentului ei. Dezalocarea se realizează prin readucerea componentei *pi* a acestui element la valoarea inițială *NULL*. In urma acțiunii funcției *r_destroy ()*, elementul în cauză este lăsat la dispoziția funcției *r_creat ()*, în vederea unei noi alocări.

Se precizează că regulile care guvernează lucrul cu blocuri resursă interzic distrugerea unui bloc a cărui coadă este nevidă. Fidelă fiind acestei reguli, funcția *r_destroy ()* va efectua o verificare a cozii blocului resursă asupra căruia planează perspectiva distrugerii și îl va distruge doar dacă coada sa este vidă. Altfel, acțiunea funcției se va rezuma la generarea unui mesaj de eroare.

Textul funcției *r_destroy ()* este redat în figura 6.6.3_4.

```
1.  void r_destroy (usshort ind_res)
2.    {
3.     register RESOURCE *s;
4.     r = &_res[ind_res];
5.     _lock_();
6.     if (r->contor != 1) {
7.        /*eroare: se încearcă distrugerea unui bloc */
8.        /*resursă a cărui  coadă este nevidă */
9.     }
10.    else {
11.       r->pi = NULL;
12.       /*se atribuie valoarea NULL componentei */
13.       /* pi a blocului resursă care se distruge */
14.    }
15.    _unlock_();
16.  }
```

Fig. 6.6.3_4. Textul funcției *r_destroy ()*.

Funcția *r_wait ()*

Funcția *r_wait ()* are rolul de a materializa operațiile care definesc primitiva *P*, cu referire la blocul resursă precizat prin primul său argument. O parte dintre aceste operații, și anume: cele de scoatere a procesului curent dintre procesele rulabile și de blocare a acestui proces, sunt asigurate prin apelul funcției *sleep ()*.

Cel de-al doilea argument al funcției *r_wait ()* este destinat transmiterii lui la funcția *sleep ()*, pentru ca aceasta să asigure limitarea timpului cât procesul în cauză poate rămâne blocat la blocul resursă. Ajungerea unui proces pus în așteptare la un bloc resursă în situația deblocării automate reprezintă o anomalie care trebuie tratată în mod corespunzător de către utilizator. În sprijinul acestuia, funcția *r_wait ()* returnează o valoare *unsigned short*, nenulă, în cazul în care anomalia s-a petrecut, și nulă, altfel. Evident, funcția *scheduler ()* este cea care generează primar această valoare; funcția *r_wait ()* o preia de la *scheduler ()* prin intermediul funcției *sleep ()*.

Legat de operația de înscriere a procesului curent în coada de așteptare la blocul resursă, se precizează că ea trebuie să se efectueze circular, prin repoziționarea pointerului *pi* pe primul element al tabloului care implementează coada, după ce el a servit înscrierii în ultimul element.

Rezultă că funcția *r_wait ()* se poate implementa prin textul *C* redat în figura 6.6.3_5.

```
1.  usshort r_wait (usshort ind_res, usshort timeout)
2.    {
3.     register RESOURCE *r;
4.     register usshort aux;
5.     r = &_res[ind_res];
6.     aux = 0;
7.     /*se inițializează variabila aux pentru a */
8.     /*indica ieșirea normală din funcția r_wait ()*/
9.     _lock_();
10.       if (--r->contor < 0) {
11.          *r->pi = _task_crt;
12.          /*se înscrie procesul curent în */
13.          /*coada de așteptare la resursă */
14.          if (r->pi == &(r->coada[MAX_TSK-1])) {
15.             r->pi = &(r->coada[0]);
16.             /*se actualizează valoarea pointerului */
17.             /*de intrare în coadă, astfel încât să */
18.             /*indice primul element al tabloului */
19.          }
20.          else {
21.             r->pi++;
22.             /* se actualizează valoarea pointerului */
23.             /*de intrare în coadă, astfel încât să */
24.             /*indice următorul element al tabloului */
25.          }
26.          aux = sleep (timeout);
27.          /*se elimină procesul curent dintre */
28.          /*procesele rulabile și se autoblochează; */
29.          /*se returnează o valoare nenulă în caz */
30.          /*de "timeout" și nulă altfel */
31.       }
```

Fig. 6.6.3_5. Textul funcției *r_wait ()*.

```
32.         r->prop = _task_crt;
33.         _unlock_();
34.         return (aux);
35.       }
```

Fig. 6.6.3_5. Textul funcției *r_wait ()*.

Este lesne de înțeles că, dacă în urma decrementării contorului blocului resursă, valoarea acestuia este negativă, funcția *r_wait ()* se va derula în două reprize. Prima repriză va cuprinde liniile [1...26], iar a doua repriză -liniile [26...35]. Asta înseamnă că o parte a funcției *sleep ()* -linia 26- se va rula în repriza întâi, iar cealaltă parte -în repriza a doua. Între cele două reprize, va avea loc rularea altor procese, printre care, în mod normal, trebuie să fie și cele care au executat înaintea procesului curent funcția *r_wait ()* cu referire la același bloc resursă. Pentru a atrage atenția asupra acestor aspecte a fost supraimprimată linia 26 din figura 6.6.3_5.

Funcția *r_signal ()*

Funcția *r_signal ()* trebuie să debuteze cu consultarea câmpului *prop* al blocului resursă cu indicele *ind_res*. Dacă valoarea înscrisă în acest câmp nu coincide cu indexul procesului curent, acțiunea funcției *r_signal ()* se limitează la generarea unui mesaj de eroare. Altfel, funcția *r_signal ()* efectuează, în continuare, operațiile care definesc primitiva *V*, cu referire la blocul resursă precizat prin argumentul său. O parte dintre aceste operații, și anume: cele de deblocare a procesului aflat în capul cozii blocului resursă și de înscriere a acestui proces în rândul proceselor rulabile, sunt asigurate prin apelul funcției *wakeup ()*.

Legat de operația de determinare a procesului aflat în capul cozii blocului resursă și de eliminare a lui din această coadă, se precizează că ea trebuie să se efectueze circular, prin poziționarea pointerului *pe* către primul element al tabloului care implementează coada, după ce el a servit extragerii unui proces înscris în ultimul element al acestui tablou.

Rezultă, pentru funcția *r_signal ()*, textul din figura 6.6.3_6.

Se face observația că apelul funcției *wakeup ()* va provoca o acțiune de comutare, prin care este posibil ca procesul curent să piardă controlul asupra procesorului, chiar dacă el rămâne în continuare rulabil, în favoarea procesului proaspăt deblocat cu ajutorul acestei funcții. Rezultă, prin urmare, o posibilă fragmentare a rulării funcției *r_signal ()*, în două reprize, în cazul în care, în urma incrementării contorului blocului resursă, valoarea sa este negativă sau nulă, indicând că cel puțin un proces se află în coada de așteptare la resursă. Prima repriză va cuprinde liniile [1...28], iar a doua repriză liniile [28...35]. Aceasta înseamnă că o parte a funcției *wakeup ()* -linia 28- se va rula în repriza întâi, iar cealaltă parte -în repriza a doua. Între cele două reprize, poate avea loc rularea altor procese. Pentru a atrage atenția asupra acestor aspecte a fost supraimprimată linia 28 din figura 6.6.3_6.

```
1.  void r_signal (usshort ind_res)
2.    {
3.     register RESOURCE *r;
4.     register usshort tsk;
5.     r = &_res[ind_res];
6.     _lock_();
7.     if (r->prop != _task_crt) {
8.        /*eroare: procesul curent nu */
9.        /*figurează ca proprietar */
10.    }
11.    else {
12.       if(++ r->contor <= 0) {
13.          tsk = *r->pe;
14.          /*se determină procesul aflat */
15.          /*în capul cozii de așteptare */
16.          if (r->pe == &(r->coada[MAX_TSK-1])) {
17.             r->pe = &(r->coada[0]);
18.             /*se actualizează valoarea pointerului */
19.             /*de ieșire din coadă, astfel încât să */
20.             /*indice primul element al tabloului */
21.          }
22.          else {
23.             r->pe++;
24.             /*se actualizează valoarea pointerului */
25.             /*de ieșire din coadă, astfel încât să */
26.             /*indice următorul element al tabloului */
27.          }
28.          wakeup (tsk);
29.          /*se deblochează procesul găsit în capul */
30.          /* cozii de așteptare la semafor și se */
31.          /*înscrie printre procesele rulabile */
32.       }
33.    }
34.    _unlock_();
35. }
```

Fig. 6.6.3_6. Textul funcției *r_signal ()* (partea 2/2).

6.6.4. Concluzii

Soluția propusă conduce la un mecanism de excludere mutuală ușor de utilizat și slab consumator de timp procesor, capabil, totodată, să detecteze și să semnaleze, în timpul execuției, o anumită clasă de erori, relativ frecvente în practică.

6.7. O soluție de sincronizare prin blocuri eveniment

6.7.1. Preliminarii

După cum se cunoaște, din punct de vedere conceptual [BALT'84], un bloc eveniment este un ansamblu format dintr-o variabilă booleană "*B*" și 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. Coada servește înregistrării proceselor aflate, la un moment, în așteptarea evenimentului aferent blocului. Evident, la crearea blocului eveniment, se asigură pentru variabilă valoarea, iar coada se videază.

Când un proces ajunge în stadiul în care necesită sincronizarea cu un eveniment, el se autoblochează și se înscrie în coada de așteptare a blocului asociat evenimentului respectiv. Când evenimentul se produce, toate procesele înscrise în coada de așteptare 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*; prezența ei în definiția blocului eveniment este bazată exclusiv pe considerente conceptuale [ROBU'95g].

În continuare, se propune un mecanism original de sincronizare prin blocuri eveniment, cu o implementare *C*, sub sistemul de operare *MS-DOS* [ROBU'94f].

Se arată modul de implementare a blocurilor eveniment propriu-zise.

Se pun în evidență funcțiile mecanismului de sincronizare prin blocuri eveniment și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.7.2. Tipul de dată *EVENT*

Pentru mecanismul propus, s-a adoptat ca blocul eveniment să fie o structură, cu numele *EVENT*, 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, având rolul de a asista intrarea în și ieșirea din coada blocului eveniment; s-a ales numele "*pie*" pentru această variabilă.

Introducerea acestei structuri se poate face așa cum se arată în figura 6.7.2_1.

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

Fig. 6.7.2_1. Tipul de dată *EVENT*.

Oferta de blocuri eveniment se face printr-un tablou de clasă *extern*, care le grupează pe toate. S-a dat numele "*_evn[]*" acestui tablou. Dacă admitem că numărul maxim de blocuri eveniment este *MAX_EVN*, atunci el se declară așa cum se arată în figura 6.7.2_2.

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

Fig. 6.7.2_2. Declararea tabloului de blocuri eveniment *_evn []*.

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

6.7.3. Funcțiile de operare asupra blocurilor eveniment

În acord cu principiile programării încapsulate, referirile utilizatorului la blocurile eveniment se fac posibile doar prin intermediul unui set de funcții dedicate. Ansamblul acestora, prezentat, sintetic în figura 6.7.3_1, împreună cu blocurile eveniment propriu-zise, constituie mecanismul de sincronizare propus.

```
void e_init (void)
/* inițializează 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-i */

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

Fig. 6.7.3_1. Funcțiile de operare asupra blocurilor eveniment (partea 1/2).

```
usshort e_wait (usshort ind_evn, usshort timeout)
        /* blochează procesul curent și-l înscrie */
    /* în coada blocului eveniment cu indicele ind_evn, */
        /* reține valoarea argumentului timeout */
          /* ca timp limită cât procesul curent */
        /* poate rămâne 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 */
```

Fig. 6.7.3_1. Funcțiile de operare asupra blocurilor eveniment (partea 2/2).

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 6.7.3_2.

```
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.   }
```

Fig. 6.7.3_2. 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, furnizând 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 6.7.3_3.

```
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->coada[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.    }
```

Fig. 6.7.3_3. 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, procedând la distrugere doar dacă aceasta este vidă. Altfel, acțiunea funcției se rezumă la generarea unui mesaj de eroare.

Textul funcției *e_destroy ()* face obiectul figurii 6.7.3_4.

```
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])) {
7.           /*eroare: coada este nevidă */
8.       }
9.       else {
10.          e->pie = NULL;
11.      }
12.      _unlock_();
13.    }
```

Fig. 6.7.3_4. Textul funcției *e_destroy ()*.

Funcția *e_wait ()*

Funcția *e_wait ()* are rolul de a bloca procesul care o execută și de a-l înscrie în coada blocului eveniment cu indicele specificat prin argumentul *ind_evn*. Blocarea se poate menține cel mult cât arată argumentul *timeout*. Operația de blocare se efectuează prin apelul funcției *sleep ()*. De asemenea, prin funcția *sleep()* se realizează și limitarea timpului cât procesul poate rămâne blocat; cel de-al doilea argument al funcției *e_wait ()* este, de fapt, destinat funcției *sleep ()*. Neproducerea evenimentului în intervalul stabilit prin acest argument va conduce la deblocarea automată a procesului în cauză. O asemenea deblocare reprezintă o anomalie care trebuie tratată în mod corespunzător de către utilizator. ţn sprijinul acestuia, funcția *e_wait ()* returnează o valoare *unsigned short* nenulă, în cazul în care anomalia s-a petrecut, și nulă, altfel. După cum a fost deja precizat, această valoare se generează, primar, la nivelul *scheduler*-ului, fiind, apoi, preluată de către funcția *sleep ()* și returnată funcției *e_wait ()*.

În conformitate cu cele de mai sus, funcția *e_wait ()* a fost implementată prin textul din figura 6.7.3_5.

```
1.   usshort e_wait (usshort ind_evn, usshort timeout)
2.     {
3.       register EVENT *e;
4.       register usshort aux;
5.       e = &_evn[ind_evn];
```

Fig. 6.7.3_5. Textul funcției *e_wait ()* (partea 1/2).

```
6.      _lock_();
7.      *e->pie = _task_crt;
8.      e->pie++;
9.      aux = sleep (timeout);
10.     _unlock_();
11.     return (aux);
12.   }
```

Fig. 6.7.3_5. Textul funcției *e_wait ()* (partea 2/2).

Funcția *e_signal ()*

Funcția *e_signal ()* are rolul de a debloca toate procesele aflate în coada blocului eveniment cu indicele specificat prin argumentul său *ind_evn*. Deblocarea proceselor se asigură prin apelul funcției *wakeup ()*. Dacă nici un proces nu așteaptă evenimentul în cauză, funcția *e_signal ()* nu are nici un efect.

Textul funcției *e_signal ()* este redat în figura 6.7.3_6.

```
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.   }
```

Fig. 6.7.3_6. Textul funcției *e_signal ()*.

Se poate înțelege, din cele de mai sus, că procesele care utilizează pentru sincronizare mecanismul blocurilor eveniment trebuie să execute, când ajung în punctul în care se impune sincronizarea, funcția *e_wait ()*, cu referire la blocul eveniment asociat evenimentului în cauză. Semnalarea evenimentului se va asigura prin funcția *e_signal ()*, cu referire la același bloc eveniment. Funcția *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 6.7.3_7 ilustrează

cele de mai sus, pentru cazul sincronizării a două procese, "*A*" și "*B*", cu un eveniment "*e*", de tip concret.

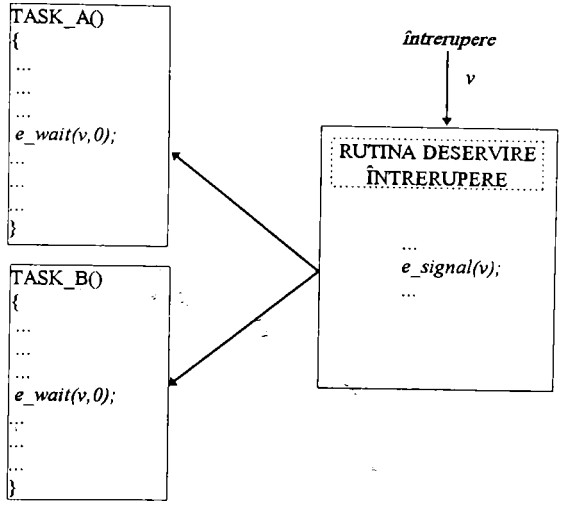


Fig. 6.7.3_6. Sincronizarea proceselor A și B cu evenimentul legat de întreruperea *e*.

6.7.4. Concluzii

Soluția propusă conduce la un mecanism de sincronizare ușor de utilizat și slab consumator de timp procesor.

6.8. O soluție de sincronizare prin blocuri multieveniment

6.8.1. Preliminarii

După cum se cunoaște [ALLW'81] [BALT'84], din punct de vedere conceptual, un bloc multieveniment este un ansamblu format dintr-o mulțime de variabile booleene, o funcție logică definită pe această mulțime sau pe o submulțime a ei, și o coadă.

Fiecare variabilă booleană corespunde unui eveniment. Mulțimea variabilelor booleene reprezintă, prin urmare, o mulțime de evenimente. Implicit, funcția logică va fi o funcție de evenimente. Valoarea logică 1 a unei variabile semnifică faptul că evenimentul asociat ei s-a produs, iar valoarea 0-contrariul.

Funcția logică din cadrul unui bloc multieveniment poate fi fie *ȘI*, fie *SAU*.

Sincronizarea prin blocuri multieveniment constă în așteptarea, respectiv semnalarea, valorii logice 1 a unei funcții de evenimente sau, cu alte cuvinte, în așteptarea, respectiv semnalarea, unei anumite configurații a stadiilor în care se află un număr de evenimente.

Procesele interesate de o asemenea configurație de evenimente trebuie, mai întâi, să o specifice, iar apoi să se autoblocheze și să se înscrie, în așteptarea ei, în coada blocului multieveniment în cauză.

Semnalarea apariției unei configurații de evenimente așteptată este însoțită de deblocarea tuturor proceselor aflate în coada corespunzătoare ei.

În continuare,se propune un mecanism original de sincronizare prin blocuri multieveniment, cu o implementare *C*, sub sistemul de operare *MS-DOS* [ROBU'95c] [ROBU'95g].

Se introduce un tip de dată specific, numit "*MULTIEVENT*".

Se pun în evidență funcțiile mecanismului de sincronizare prin blocuri multieveniment și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.8.2. Tipul de dată *MULTIEVENT*

Se reprezintă mulțimea evenimentelor printr-o variabilă de un tip întreg fără semn, instituindu-se convenția că fiecare bit al acestei variabile corespunde unui eveniment [TSCH'90].

Submulțimea pe care este definită funcția logică se precizează printr-o mască. Aceasta este tot o variabilă de un tip întreg fără semn, același adoptat pentru mulțimea evenimentelor, având biți de valoare 1 în toate pozițiile corespunzătoare unor evenimente care aparțin submulțimii și biți de valoare 0 în celelalte. Evident, când funcția este definită pe întreaga mulțime a evenimentelor, toți biții măștii au valoarea 1.

Specificarea unei configurații de evenimente care reprezintă o condiție de sincronizare se face indicând masca potrivită, tipul funcției logice, și valorile pe care trebuie să le aibă biții corespunzători evenimentelor respective, pentru ca această funcție să ia valoarea 1. Denumim aceste valori "*valori active*".

Evident, o funcție logică de tip *ȘI* va lua valoarea 1 doar atunci când toate valorile indicate vor fi atinse deodată, iar o funcție logică de tip *SAU* -doar atunci când cel puțin una dintre aceste valori va fi atinsă. În alți termeni, însă, având în vedere una dintre teoremele lui de Morgan, se poate afirma că o funcție logică de tip *SAU* ia valoarea 1 doar atunci când nu toate negatele valorilor indicate vor fi atinse deodată.

Se definește drept "*configurație de referință pentru funcția ȘI*" o mulțime de același cardinal cu mulțimea evenimentelor, obținută prin aplicarea măștii asupra submulțimii pe care este definită funcția logică (aplicarea măștii constă în efectuarea unei funcții logice *ȘI* între mască și submulțimea completată cu elemente de valori oarecari, până la atingerea cardinalului în cauză).

Se definește drept "*configurație de referință pentru funcția SAU*" o mulțime de același cardinal cu mulțimea evenimentelor, obținută prin aplicarea măștii asupra submulțimii pe care este definită funcția logică, după negarea valorilor elementelor acestei mulțimi.

O configurație oarecare de evenimente va reprezenta o "*condiție de sincronizare definită printr-o funcție ȘI*" dacă, aplicându-i masca, se obține o valoare identică cu cea a configurației de referință corespunzătoare respectivei condiții de sincronizare.

O configurație oarecare de evenimente va reprezenta o "*condiție de sincronizare definită printr-o funcție SAU*" dacă, aplicându-i masca, se obține o valoare diferită de cea a configurației de referință corespunzătoare respectivei condiții de sincronizare.

Pentru ilustrarea celor de mai sus, se consideră, în continuare, două exemple, presupunând că mulțimea evenimentelor este de cardinal 8 și că evenimentele poartă indici egali cu rangul biților prin care se reprezintă.

Exemplul 1).

Se consideră drept condiție de sincronizare atingerea deodată a valorilor 1, 0, 1, 1 de către evenimentele 0, 2, 3, și 6, respectiv.

Această condiție de sincronizare se va specifica astfel:

masca: 01001101
funcția logică: *ȘI*
valorile active: *1**10*1

Notă:
Cu "" s-au marcat biții ale căror valori sunt indiferente.*

Configurația de referință corespunzătoare condiției de sincronizare considerate se obține astfel:

configurația de referință = 01001101 & *1**10*1 = 01001001

Configurațiile oarecari:

00001001 (00001001 & 01001101 = 00001001 ≠ 01001001)
01001101 (01001101 & 01001101 = 01001101 ≠ 01001001)
etc.

nu reprezintă condiția de sincronizare.

Configurațiile oarecari:

01001001 (01001001 & 01001101 = 01001001 ≡ 01001001)
11001001 (11001001 & 01001101 = 01001001 ≡ 01001001)
11111001 (11111001 & 01001101 = 01001001 ≡ 01001001)
etc.

reprezintă condiția de sincronizare.

Exemplul 2).

Se consideră drept condiție de sincronizare atingerea la un moment a cel puțin uneia dintre valorile 1, 0, 1, 1 de către evenimentele 0, 2, 3, și 6, respectiv.

Această condiție de sincronizare se va specifica astfel:

masca: 01001101
funcția logică: *SAU*
valorile active: *1**10*1

Configurația de referință corespunzătoare condiției de sincronizare considerate se obține astfel:

configurația de referință = 01001101 & (~*1**10*1) = 00000100

Configurațiile oarecari:

00100110 (00100110 & 01001101 = 00000100 ≡ 00000100)
10110110 (10110110 & 01001101 = 00000100 ≡ 00000100)
etc.

nu reprezintă condiția de sincronizare.

Configurațiile oarecari:

01000100 (01000100 è 01001101 = 01000100 ≠ 00000100)
11001001 (11001001 è 01001101 = 00001100 ≠ 00000100)
etc.

reprezintă condiția de sincronizare.

Pentru implementarea mecanismului de sincronizare prin blocuri multieveniment, se definește, mai întâi, tipul de dată *FOR_ME* (*FOR_ME* provine de la *FOR MultiEvent*) ca o structură cuprinzând:

- o variabilă de tipul *unsigned short*, care reprezintă, prin biții săi, mulțimea evenimentelor, și în care se înregistrează configurațiile de referință; fie "*evntval*" numele acestei variabile.
- o variabilă de tipul *unsigned short*, reprezentând o mască prin care se desemnează submulțimea evenimentelor vizate de funcția logică; fie "*mask*" numele acestei variabile. Se precizează că această variabilă are și rolul de a indica, prin valoarea sa nulă, respectiv nenulă, absența, respectiv prezența cel puțin a unui proces în coada blocului multieveniment;
- o variabilă de tipul *unsigned short*, dedicată codificării operației efectuate de funcția logică asupra evenimentelor; fie "*oper*" numele acestei variabile.

Tipul de dată *FOR_ME* se introduce, deci, așa cum se arată în figura 6.8.2_1.

```
typedef struct {
        unsigned short evntval;
        unsigned short mask;
        unsigned short oper;
    } FOR_ME;
```

Fig. 6.8.2_1. Tipul de dată *FOR_ME*.

Un bloc multieveniment trebuie să cuprindă câte o structură *FOR_ME* pentru fiecare proces. Este firesc, atunci, ca toate structurile *FOR_ME* din cadrul aceluiași bloc multieveniment să fie grupate sub forma unui tablou. S-a dat numele "*for_me[]*" acestui tablou.

Alături de un tablou *for_me[]*, în compoziția unui bloc multieveniment trebuie introdusă și o variabilă de tipul *unsigned short*, care să servească drept indicator de alocare a blocului. S-a adoptat convenția ca valoarea nulă a acestei variabile să corespundă situației când blocul nu este alocat, iar o valoare nenulă -situației când blocul este alocat. S-a dat numele "*nfree*" acestei variabile.

Se poate conchide, acum, că un bloc multieveniment va fi o structură formată din tabloul *for_me[]* și variabila *nfree*.

Corespunzător acestei structuri, s-a introdus, așa cum se arată în figura 6.8.2_2, un tip de dată cu numele "*MULTIEVENT*".

```
typedef struct {
        FOR_ME for_me[MAX_TSK];
        unsigned short nfree;
      } MULTIEVENT;
```

Fig. 6.8.2_2. Tipul de dată *MULTIEVENT*.

Cum într-un sistem există mai multe blocuri multieveniment -presupunem *MAX_ME* numărul acestora-, vom proceda la gruparea lor într-un tablou. Admițând pentru acest tablou -din motive lesne de înțeles, de clasă *extern*- numele *_me[]*, el va fi introdus prin următoarea declarație:

```
extern MULTIEVENT _me[MAX_ME];
```

Fig. 6.8.2_3. Declararea tabloului de blocuri multieveniment *_me[]*.

6.8.3. Funcțiile de operare asupra blocurilor multieveniment

Referirile utilizatorului la blocurile multieveniment se pot face doar prin intermediul unor funcții dedicate. Ansamblul acestora, prezentat sintetic în figura 6.8.3_1, împreună cu blocurile multieveniment propriu-zise, constituie mecanismul de sincronizare prin blocuri multieveniment [APPE'84] [TSCH'90].

```
void me_init (void)
/* inițializează tabloul de blocuri multieveniment */
/* prin atribuirea valorii 0 componentelor pic */
/* ale elementelor sale */

usshort me_creat (void)
/* alocă un element al tabloului _me[] */
/* și furnizează indicele-i */

void me_destroy (usshort ind_me)
/* dezalocă elementul tabloului _me[] */
/* cu indicele ind_me, repunându-l la */
/* dispoziția funcției me_creat () */
```

Fig. 6.8.3_1. Funcțiile de operare asupra blocurilor multieveniment (partea 1/2).

```
usshort me_wait (usshort ind_me, usshort evv,
                 usshort msk, usshort op, usshort timeout)
   /* blochează procesul curent și-l înscrie */
   /* în  coada blocului multieveniment cu */
   /* indicele ind_me, reținând în structura */
   /* for_me a acestui bloc, corespunzătoare */
   /* procesului în cauză, argumentul msk; */
   /* reține, de  asemenea în această structură, */
   /* argumentele evv și op  și înregistrează */
   /* valoarea argumentului timeout  ca timp limită */
   /*  cât procesul curent poate rămâne blocat */
   /* la blocul  multieveniment; returnează o valoare nulă, /
   /* dacă deblocarea se produce înaintea expirării */
   /* timpului, altfel -una nenulă; */

void me_signal (usshort ind_me, usshort evv)
   /* deblochează  toate  taskurile  aflate în */
   /* coada blocului  multieveniment  cu indicele ind_me, */
   /* satisfăcute  de  configurația de  evenimente */
   /* precizată prin argumentul evv */
```

Fig. 6.8.3_1. Funcțiile de operare asupra blocurilor multieveniment (partea 2/2).

Funcția *me_init ()*

Această funcție are rolul de a inițializa tabloul de blocuri multieveniment, prin stabilirea valorii 0 pentru componentele *nfree* ale tuturor elementelor acestuia.

Textul funcției *me_init ()* este următorul:

```
1.  void me_init (void)
2.    {
3.     register MULTIEVENT *me;
4.     register usshort j;
5.     me = &_me[0];
6.     for (j = 0; j < MAX_ME; j++) {
7.        me->nfree = 0;
8.        me++;
9.     }
10.   }
```

Fig. 6.8.3_2. Textul funcției *me_init ()*.

Funcția *me_creat ()*

Funcția *me_creat ()* are rolul de a identifica un element liber al tabloului de blocuri multieveniment *_me[]* și de a-l aloca, furnizând indicele-i. De asemenea, funcției *me_creat ()* îi revine sarcina de a forța valoarea **0** pentru toate componentele *mask* ale blocului multieveniment pe care îl alocă, vidând, astfel, coada acestui bloc. Se reamintește că o valoare nenulă a unei componente *mask*, dincolo de aceea că desemnează submulțimea de evenimente pe care este definită funcția logică de care este interesat procesul pus în corespondență cu acea componentă, semnifică și faptul că respectivul proces se află în coada blocului multieveniment în cauză.

Textul funcției *me_creat ()* este redat în figura 6.8.3_3.

```
1.  usshort me_creat ()
2.    {
3.     register MULTIEVENT *me;
4.     register usshort j;
5.     usshort i;
6.     me = &_me[0];
7.     j = 0;
8.     _lock_();
9.     while ((me->nfree) && (j<MAX_ME)) {
10.       me++;
11.       j++;
12.    }
13.    if (j < MAX_ME) {
14.       me->nfree = 1;
15.       for (I = 0; I < MAX_TSK; i++) {
16.          me->for_me[i].mask = 0;
17.       }
18.    }
19.    else {
20.       /*eșec: nici un bloc multieveniment nu este liber */
21.    }
22.    _unlock_();
23.    return (j);
24.  }
```

Fig. 6.8.3_3. Textul funcției *me_creat ()*.

Funcția *me_destroy ()*

Funcția *me_destroy ()* are rolul de a dezaloca elementul tabloului de blocuri multieveniment cu indicele specificat prin valoarea argumentului ei, atribuind valoarea **0** componentei *nfree* a acestui element. În acest fel, elementul în cauză este lăsat la dispoziția funcției *me_creat ()*, în vederea unei noi alocări.

Se precizează că funcția *me_destroy ()* efectuează o verificare a cozii blocului multieveniment vizat a fi distrus și procedează la distrugere doar dacă coada este vidă. Altfel, acțiunea funcției se rezumă la generarea unui mesaj de eroare.

Textul funcției *me_destroy ()* este redat în figura 6.8.3_4.

```
1.  void me_destroy ()
2.    {
3.     register MULTIEVENT *me;
4.     register usshort j;
5.     me = &_me[ind_me];
6.     j = 0;
7.     _lock_();
8.     while ((me->for_me[j].mask == 0) && (j < MAX_TSK)) {
9.        j++;
10.    }
11.    if (j<MAX_TSK) {
12.       /*eroare: coada este nevidă */
13.    }
14.    else {
15.       me->nfree = 0;
16.    }
17.    _unlock_();
18.   }
```

Fig. 6.8.3_4. Textul funcției *me_destroy ()*.

Funcția *me_wait ()*

Funcția *me_wait ()* are rolul de a bloca procesul care o execută și de a-l înscrie în coada blocului multieveniment cu indicele *ind_me*, reținând în structura *for_me* a acestui bloc, corespunzătoare respectivului proces, argumentul *msk*.

De asemenea, funcția *me_wait ()* reține, în aceeași structură, argumentul *op*, apoi determină configurația de referință corespunzătoare condiției de sincronizare specificată prin argumentele *msk*, *op* și *evv*, și o înscrie în componenta *evntval* a structurii în cauză.

Operația de blocare a procesului se asigură prin apelul unei funcții *sleep ()*. Acesteia îi este destinat argumentul *timeout*. Cu ajutorul lui, ea limitează timpul cât procesul poate rămâne blocat la blocul multieveniment. Dacă procesul va ajunge să fie deblocat automat, ca urmare a expirării timpului acordat prin argumentul *timeout* pentru apariția condiției de sincronizare, funcția *me_wait ()* va proceda la radierea lui din coada blocului multieveniment în cauză și va returna o valoare *usshort* nenulă. Altfel, funcția returnează o valoare nulă.

Pentru facilitarea determinării configurației de referință, s-a codificat operatorul logic *ȘI* prin valoarea hexazecimală *0x0000*, iar operatorul logic *SAU* prin valoarea hexazecimală *0xFFFF*, ambele de tipul *usshort*. Argumentul *op* are, deci, la un moment, una dintre aceste două valori. Rezultă că, aplicând operatorul *SAU EXCLUSIV* (^) între valoarea acestui argument și valoarea argumentului *evv*, se obține, ca rezultat, complementul valorii *evv*, în cazul operației *SAU*, respectiv chiar această valoare, în cazul operației *ȘI*.

Pentru a nu se pretinde utilizatorului cunoașterea acestei codificări, se pun la dispoziția lui simbolii *_AND_*, respectiv *_OR_*, definiți cu ajutorul preprocesorului, după cum urmează:

```
# define _AND_ 0x0000;
# define _OR_  0xFFFF;
```

Fig. 6.8.3_5. Definirea simbolilor *_AND_* și *_OR_*.

Textul funcției *me_wait ()* este redat în figura 6.8.3_6.

```
1.  usshort me_wait (usshort ind_me, usshort evv,
2.                          usshort msk, usshort op, usshort timeout)
3.    {
4.     register FOR_ME *fm;
5.     register usshort aux;
6.     fm=&_me[ind_me].for_me[task_crt];
7.     _lock_();
8.     fm->evntval=msk&(evv^op);
9.     /* se determină și se reține configurația de referință */
10.    fm->mask=msk;
11.    fm->oper=op;
12.    aux = sleep (timeout)
13.    if (aux) {
14.       /* o valoare nenulă returnată de funcția sleep () */
15.       /* semnifică faptul că procesul a fost deblocat automat, */
16.       /* expirând timpul limită în care el a fost autorizat */
17.       /* să aștepte; într-o asemenea situație, are loc radierea */
18.       /* procesului din coada blocului multieveniment în cauză */
19.       /* și se generează un mesaj specific de eroare */
20.       fm->msk=0;
21.    }
22.    _unlock_();
23.    return (aux);
24.  }
```

Fig. 6.8.3_6. Textul funcției *me_wait ()*.

Funcția *me_signal ()*

Această funcție are rolul de a debloca toate procesele aflate în coada blocului multieveniment cu indicele *ind_me*, pentru care configurația de evenimente precizată prin argumentul *evv* reprezintă condiția de sincronizare așteptată.

Deblocarea proceselor se asigură prin apelul unei funcții numită *wakeup ()*. Cu ocazia deblocării, procesele sunt radiate din coada blocului multieveniment în cauză, prin înscrierea valorii **0** în componenta *mask* a structurii *for_me* corespunzătoare lor.

Dacă nici un proces nu este satisfăcut de configurația de evenimente precizată prin *evv*, funcția *me_signal ()* rămâne fără nici un efect.

Textul funcției *me_signal ()* este redat în figura 6.8.3_7.

```
1.  void me_signal (usshort ind_me, usshort evv)
2.    {
3.     register FOR_ME *fm;
4.     register usshort tsk;
5.     usshort aux;
6.     fm = &_me[ind_me].for_me[0];
7.     _lock_();
8.     for (tsk = 0; tsk < MAX_TSK; tsk++) {
9.        if (fm->mask) {
10.          aux = (fm->evntval == ((fm->mask) & evv));
11.          if ((((! (fm->oper)) && (aux)) || ((fm->oper) && (! aux))) {
12.             wakeup (tsk);
13.             fm->mask = 0;
14.          }
15.       }
16.       fm++;
17.    }.
18.    _unlock_();
19.  }
```

Fig. 6.8.3_7. Textul funcției *me_signal ()*.

6.8.4. Concluzii

Mecanismul propus face posibilă sincronizarea interprocese și sincronizarea proceselor cu evenimente externe într-o manieră comodă pentru utilizator și deosebit de eficace. Este ușor, dacă se dispune de un asemenea mecanism, să se realizeze sincronizări cu diverse condiții referitoare la ambianța sistemului de programe.

6.9. O soluție de sincronizare prin blocuri *rendez-vous*

6.9.1. Preliminarii

După cum se cunoaște [PERE'90] [TSCH'90], în anumite aplicații, intervin situații 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, și abia după aceea să poată din nou avansa. Se poate considera că, în aceste situații, există un punct în spațiul multidimensional al acțiunilor respectivelor procese în care ele au stabilit o *"întîlnire"*, după care, din nou, drumurile lor se despart.

Situațiile de tipul menționat ridică o problemă de sincronizare specifică, ce se se rezolvă printr-un mecanism dezvoltat în jurul conceptului de bloc *"rendez-vous"* [EVEN'80] [TSCH'90].

Din punct de vedere principial, un bloc *rendez-vous* este un ansamblu format dintr-o variabilă întreagă cu rol de contor și o coadă de așteptare [ROBU'95d] [ROBU'95g].

Contorul indică, inițial, numărul proceselor care trebuie să se întîlnească, iar curent, numărul proceselor care încă nu au ajuns la întîlnire.

Coada servește înregistrării proceselor care au ajuns deja la întîlnire, blocîndu-se, cu această ocazie, în așteptarea celorlalte.

De fiecare dată cînd un proces ajunge la întîlnire, valoarea contorului se decrementează cu o unitate. Atingerea, prin decrementări succesive, a valorii 0, semnifică faptul că întîlnirea a avut loc în formație completă și determină deblocarea tuturor proceselor care au participat la ea. Evident, deblocarea se efectuează pe baza conținutului cozii.

Se impune unui bloc *rendez-vous* să asigure și posibilitatea fixării unui timp limită în care procesele trebuie să sosească în punctul de întîlnire, pentru a se considera că aceasta a avut loc [ROBU'95d] [ROBU'95g]. Dacă cel puțin un proces nu reușește să se încadreze în timpul fixat, întîlnirea se contramandează și toate procesele care au ajuns la ea sunt deblocate.

Pentru a se asigura ca procesele vizate pentru o întâlnire să-și poată continua activitatea în cunoștință de cauză, în ceea ce privește reușita sau eșecul respectivei întâlniri, este necesar ca la deblocare să li se furnizeze o informație de stare referitoare la cele două alternative posibile.

În continuare, se propune un mecanism de sincronizare prin blocuri *rendez-vous,* cu o implementare *C*, sub sistemul de operare *MS-DOS*

Se arată modul de implementare a blocurilor *rendez-vous* propriu-zise.

Se pun în evidență funcțiile mecanismului de sincronizare prin blocuri *rendez-vous* și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.9.2. Tipul de dată *RENDEZ_VOUS*

În cadrul mecanismul în discuție, la nivelul concret, un bloc *rendez-vous* este reprezentat printr-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 "*coada[]*" pentru acest tablou;
- o variabilă de tipul *pointer* către un întreg scurt fără semn, având rolul de a asista intrarea în, respectiv ieșirea 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 așa cum se arată în figura 6.9.2_1.

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

Fig. 6.9.2_1. 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 înțeles, 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:

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

Fig. 6.9.2_2. Declararea tabloului de blocuri *rendez-vous*, *_rv[]*.

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

6.9.3. Funcțiile de operare asupra blocurilor *rendez-vous*

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

void rv_init (void)
/* inițializează tabloul de blocuri *rendez-vous*, */
/* prin atribuirea valorii 0 componentelor *contor* */
/* ale tuturor elementelor sale */

usshort rv_creat (usshort ntv, usshort timeout)
/* alocă un element al tabloului *_rv []*, */
/* atribuie componentei *contor* a elementului alocat */
/* valoarea specificată prin argumentului *ntv* ,*/
/*reține valoarea argumentului *timeout* și */
/ * furnizează indicele respectivului element */

void rv_destroy (usshort ind_rv)
/* dezalocă elementul tabloului *_rv []* , cu indicele *ind_rv*, */
/* repunându-l la dispoziția funcției *rv_creat ()* */

usshort rv_signal (usshort ind_rv)
/* anunță sosirea la *rendez-vous* a procesului curent; */
/* dacă nu toate celelalte procese vizate sunt sosite, */
/* atunci procesul curent se blochează; */
/* altfel, are loc deblocarea tuturor proceselor blocate */
/* în așteptarea întâlnirii; returnează o valoare nulă, */
/* dacă întâlnirea a reușit, altfel, -una nenulă */

Fig. 6.9.3_1. Funcțiile de operare asupra blocurilor *rendez-vous*.

Funcția *rv_init ()*

Această funcție are rolul de a inițializa tabloul de blocuri *rendez_vous*, prin atribuirea valorii 0 componentelor *contor* ale tuturor elementelor sale.

Textul funcției *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.   }
```

Fig. 6.9.3_2. Textul funcției *rv_init ()*.

Funcția *rv_creat ()*

Funcția *rv_creat ()* are rolul de a identifica un element liber al tabloului de blocuri *rendez_vous*, *_rv []*, și de a-l aloca, furnizând indicele-i. De asemenea, funcției *rv_creat ()* îi revin sarcinile de a poziționa pointerul *pie* al blocului pe care îl alocă pe primul element al cozii, de a înregistra argumentele *ntv* și *timeout* în componentele *contor*, respectiv *timeout* ale acestui bloc, și de a forța la 0 componenta *status*.

Textul funcției *rv_creat ()* este redat în figura 6.9.3_3.

```
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 0 a componentei sale */
15.    /* contor, stabilită prin funcțiile rv_init (), */
16.    /* rv_destroy () și, de asemenea, rv_signal () */
```

Fig. 6.9.3_3. Textul funcției *rv_creat ()* (partea 1/2).

```
17.     if (j<MAX_RV) {
18.         rv->contor=ntv;
19.         rv->pie=&(rv->coada[0]);
20.         /* se poziționează 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 nu este liber */
27.     }
28.     _unlock_();
29.     return (j);
30. }
```

Fig. 6.9.3_3. Textul funcției *rv_creat ()* (partea 2/2).

Funcția *rv_destroy ()*

Funcția *rv_destroy ()* are rolul de a dezaloca elementul tabloului de blocuri *rendez_vous* cu indicele specificat prin valoarea argumentului ei, atribuind valoarea **0** componentei *contor* a acestui element. În acest fel, elementul în cauză este lăsat la dispoziția funcției *rv_creat ()*, în vederea unei noi alocări.

Pentru evitarea unor anomalii, funcția *rv_destroy ()* efectuează o verificare a cozii blocului *rendez_vous* vizat a fi distrus, procedând la distrugere doar dacă coada este vidă. Altfel, acțiunea funcției se rezumă la generarea unui mesaj de eroare.

Textul funcției *rv_destroy ()* este redat în figura 6.9.3_4.

```
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.  }
```

Fig. 6.9.3_4. Textul funcției *rv_destroy ()*.

Funcția *rv_signal ()*

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

În primul rând, funcția verifică, uzând de componenta *contor* a blocului *rendez-vous*, dacă nu cumva toate procesele vizate pentru *rendez-vous* și-au anunțat deja sosirea. În caz afirmativ, se poziționează la valoarea 0 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 conținutului cozii. Deblocarea se efectuează prin apelul unei funcții numită *wakeup ()*, repetat de câte ori este necesar. În caz negativ, taskul în care se execută funcția *rv_signal ()* este blocat și înscris în coada blocului *rendez-vous* cu indicele *ind_rv*. Operația de blocare se efectuează prin apelul funcției *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 situații, ce pot fi deosebite prin valorile pe care le au componentele *contor* și *status* ale blocului, imediat după deblocare.

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

În a doua situație, caracterizată prin valoarea nenulă a contorului, deblocarea este rezultatul expirării timpului afectat pentru realizarea întâlnirii înainte ca toate procesele să ajungă în punctul prevăzut. În această situație, unul dintre procesele care au fost blocate la blocul *rendez-vous* -firesc, primul- va fi reintrodus în rulare prin deblocare automată. Acest proces va poziționa, mai întâi, la o valoare nenulă componenta *status* a blocului *rendez-vous* în cauză, va aduce componenta *contor* la 0, iar apoi va debloca, pe baza cozii, toate celelalte procese blocate în așteptarea întâlnirii. Deblocarea se efectuează prin apelul funcției *wakeup ()*, repetat de câte ori este necesar. Componenta *contor* a blocului este adusă la 0 de către procesul deblocat în mod automat, pentru a se asigura pentru procesele pe care el le deblochează posibilitatea de a se autolocaliza în situația a treia.

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

Textul funcției *rv_signal ()* este redat în figura 6.9.3_5.

```
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 în  coada */
15.          /*blocului  rendez-vous: cazul întâlnirii reușite, */
16.          /*subcazul procesului  sosit ultimul */
17.       }
18.    }
19.    else {
20.       *rv->pie=_task_crt;
21.       rv->pie++;
22.       sleep (rv->timeout);
23.       /*procesul curent se blochează în așteptarea întâlnirii */
24.       If (rv->contor) {
25.          rv->status = i;
26.          rv->contor = 0;
27.          while (rv->pie != &(rv->coada[0])) {
28.             rv->pie--;
29.             tsk=*rv->pie;
30.             wakeup (tsk);
31.             /*sunt deblocate toate procesele găsite în coada */
32.             /*blocului  rendez-vous: cazul întâlnirii eșuate, */
33.             /*subcazul procesului deblocat  primul */
34.          }
35.       }
36.    }
37.    _unlock_();
38.    return (rv->status);
39.    /*în  afară  de procesul sosit ultimul -cazul întâlnirii reușite-, */
40.    /*respectiv  de procesul deblocat primul, în mod automat, */
41.    /*din  motiv de "time-out" -cazul  întâlnirii  eșuate-, toate */
42.    /*celelalte vor parcurge doar liniile: 1...8, 19...24 și 37...38 */
43. }
```

Fig. 6.9.3_5 Textul funcției *rv_signal ()*.

6.9.4. Concluzii

Mecanismul propus face posibilă sincronizarea interprocese -evident: în cazurile în care se impune ca aceasta să fie de tipul *rendez-vous*-, în condițiile unui consum relativ redus de timp procesor.

6.10. O soluție de comunicare prin conductă

6.10.1. Preliminarii

După cum se cunoaște [TSCH'90] [ELEŞ'91], o conductă este un tampon circular gestionat după principiul *FIFO*, asociat, la un moment, unei perechi de procese, dintre care unul are exclusiv rolul de producător, iar celălalt -exclusiv rolul de consumator. Producătorul depune datele în tampon, octet cu octet, iar consumatorul le extrage, de asemenea octet cu octet, indiferent de ce tip ele sunt.

Mecanismul de comunicare prin conductă trebuie să asigure blocarea procesului producător, când acesta ajunge în fața unei operații de scriere iar tamponul este plin, respectiv blocarea procesului consumator, când acesta ajunge în fața unei operații de citire, iar tamponul este vid [BALT'84].

Nici un fel de restricții privind lungimea datelor vehiculate nu intervin la comunicarea prin conductă; datele care nu încap, în întregul lor, în tampon, sunt emise și recepționate fragmentar [TSCH'90].

În continuare, se propune un mecanism de comunicare prin conductă, cu o implementare *C*, sub sistemul de operare *MS-DOS*.

Se arată modul de implementare a conductelor propriu-zise.

Se pun în evidență funcțiile mecanismului de comunicare prin conductă și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.10.2. Tipul de dată *PIPE*

În mecanismul în discuție, la nivelul concret, conducta este o structură, cu numele *PIPE*, cuprinzând:

- 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 valore nenulă, pe de o parte, faptul că procesul producător s-a blocat, găsind conducta plină, iar pe de altă parte, numărul octeților 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 octeților ce au mai rămas de citit;
- o variabilă de tipul *unsigned int*, denumită "*nocumul*", destinată înregistării numărului de octeți cumulați 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 octeți, desemnată prin identificatorul *DIM_COND*, tipul de dată *PIPE* se va declara așa cum se arată în figura 6.10.2_1.

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

Fig. 6.10.2_1. Tipul de dată *PIPE*.

Ansamblul conductelor este grupat într-un tablou de structuri de tipul *PIPE*, denumit *_pipe[]*, cu dimensiunea *MAX_PIPE*. Din motive lesne de înțeles, tabloul *_pipe[]* este de clasă extern:

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

Fig. 6.10.2_2. Declararea tabloului de structuri conductă *_pipe[]*.

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

6.10.3. Funcțiile de operare asupra structurilor conductă

În conformitate cu principiile programării încapsulate [APPE'84], referirile utilizatorului la o structură conductă sunt posibile doar prin intermediul unui set de funcții dedicate. Ansamblul acestora, prezentat sintetic în figura 6.10.3_1, împreună cu structurile conductă propriu-zise, constituie mecanismul de comunicare prin conductă [ROBU'95e] [ROBU'95g].

void p_init (void)
/* inițializează tabloul *_pipe []*, prin */
/* atribuirea valorii *NULL* componentelor */
/* *ps* și *pc*, respectiv a valorii *0xFF* */
/* componentelor *itp* și *itc* ale tuturor */
/* elementelor sale */

usshort p_creat (usshort prod, usshort cons)
/* alocă proceselor *prod*, cu calitatea de */
/* producător, și *cons*, cu calitatea de */
/* consumator, un element al tabloului *_pipe[]* */
/* și returnează indicele-i */

void p_destroy (usshort ind_p)
/* dezalocă elementul tabloului *_pipe[]* */
/* cu indicele *ind_p*, repunându-l la */
/* dispoziția funcției *p_creat ()* */

*void p_send (usshort ind_p, void *pdata, usint ldata)*
/* depune în tamponul structurii conductă */
/* cu indicele *ind_p* cei *ldata* octeți ai */
/* datei pointate de *pdata* */

*void p_receive (usshort ind_p, void *pdata, usint ldata)*
/* extrage din tamponul structurii conductă */
/* cu indicele *ind_p* un număr de *ldata* */
/* octeți și îi atribuie datei pointate de *pdata* */

Fig. 6.10.3_1. Funcțiile mecanismului de comunicare prin conductă.

Funcția *p_init ()*

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

Textul funcției *p_init ()* este redat în figura 6.10.3_2.

```
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.   }
```

Fig. 6.10.3_2. Textul funcției *p_init ()*.

Funcția *p_creat ()*

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

Textul funcției *p_creat ()* este redat în figura 6.10.3_3.

```
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 nu este liberă */
14.    }
```

Fig. 6.10.3_3. Textul funcției *p_creat ()* (partea 1/2).

```
15.     else {
16.         p->itp = prod;
17.         p->itc = cons;
18.         p->rs = 0;
19.         p->rc = 0;
20.         p->nocumul = 0;
21.         p>ps = &(p->conducta[0]);
22.         p->pc = &(p->conducta[0]);
23.     }
24.     _unlock_();
25.     return (n);
26. }
```

Fig. 6.10.3_3. Textul funcției *p_creat ()* (partea 2/2).

Funcția *p_destroy ()*

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

Textul funcției *p_destroy ()* este redat în figura 6.10.3_4.

```
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.    }
```

Fig. 610.3_4. Textul funcției *p_destroy ()*.

Funcția *p_send ()*

Funcția *p_send ()* asigură depunerea în tamponul structurii conductă cu indicele precizat prin argumentul *ind_p* a unui număr de *ldata* octeți 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 funcției, el va fi convertit în *pointer de caracter fără semn* și utilizat ca atare.

Când funcția *p_send ()* constată că tamponul este plin, ea procedează astfel:

- înregistrează în componenta *rs* a stucturii conductă numărul de octeți pe care îi mai are de transmis, pentru a-l face cunoscut funcției *p_receive ()*, prevăzută în procesul consumator;
- verifică dacă procesul consumator este blocat în interiorul funcției *p_receive ()*, ca urmare a faptului că a găsit tamponul vid, 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 producător), lăsându-l în așteptare pasivă a deblocării de către procesul consumator. Procesul consumator asigură deblocarea producătorului, dacă îl găsește blocat la tampon plin, cu ocazia execuției funcției *p_receive ()* prevăzută în cadrul lui, atunci când în tampon este creat suficient spațiu liber pentru depunerea tuturor octeților de transmis rămași restanți sau, oricum, atunci când această funcție găsește tamponul vid sau termină de preluat o dată.

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

Se precizează că funcția *p_send ()* realizează blocarea, respectiv deblocarea proceselor, făcând apel la funcțiile *sleep ()*, respectiv *wakeup ()*. Argumentul *timeout* al funcției *p_receive()* este destinat funcției *sleep ()* (reamintim: dacă argumentul *timeout* este nul, atunci blocarea prin *sleep ()* se face pe timp nelimitat; dacă argumentul *timeout* 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 a sa).

Textul fucției *p_send ()* este redat în figura 6.10.3_5.

```
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 asociat la conductă */
9.      }
10.     else {
11.         _lock_();
12.         q=(uschar *)pdata;
13.         nos=ldata;
```

Fig. 6.10.3_5. Textul functiei *p_send ()* (partea 1/2).

```
14.          while (nos) {
15.             if (p->nocumul<DIM_COND){
16.                *p->ps=*q++;
17.                if (p->ps==&(p->conducta[DIM_COND-1])) {
18.                   p->ps=&(p->conducta[0]);
19.                }
20.                else {
21.                   p->ps++;
22.                }
23.                nos--;
24.                p->nocumul++;
25.             }
26.             else {
27.                p->rs=nos;
28.                if (p->rc) {
29.                   p->rc=0;
30.                   wakeup (p->itc);
31.                }
32.                while (p->rs){
33.                   sleep (0);
34.                }
35.             }
36.          }
37.          if (p->rc) {
38.             p->rc=0;
39.             wakeup (p->itc);
40.          }
41.          _unlock_();
42.       }
43.    }
```

Fig. 6.10.3_5. Textul funcției *p_send ()* (partea 2/2).

Funcția *p_receive ()*

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

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

Pe parcursul extragerii octeților din tampon, funcția *p_receive ()* verifică, utilizând componenta *rs* a structurii conductă, dacă procesul producător este blocat în interiorul funcției *p_send ()*, ca urmare a faptului că a găsit tamponul plin. În caz afirmativ, când se ajunge ca în tampon să fie suficient spațiu liber pentru depunerea tuturor octeților rămași restanți, funcția *p_receive ()* anunță procesul producător asupra acestui lucru și îl deblochează, înregistrând în componenta *rc* a structurii conductă numărul octeților neextrași ai datei curente și blocând procesul în care ea se execută (procesul consumator).

Când funcția *p_receive ()* constată că tamponul este vid, procedează astfel:

- înregistrează în componenta *rc* a structurii conductă numărul de octeți pe care îi mai are de recepționat;
- 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-lîn 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 figura 6.10.3_6.

```
1.  p_receive (usshort ind_p, void *pdata, usint 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 asociat la conductă */
10.     }
11.     else {
12.         _lock_();
13.         q = (uschar *)pdata;
14.         noc = ldata;
```

Fig. 6.10.3_6. Textul funcției *p_receive ()* (partea 1/2).

```
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 && (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.  }
```

Fig. 6.10.3_6. Textul funcției *p_receive ()* (partea 2/2).

6.10.4. Concluzii

Mecanismul propus face posibilă comunicarea interprocese sigur, versatil și eficient. Timpul procesor consumat de mecanism este relativ scăzut.

6.11. O soluție de comunicare prin cutie poștală

6.11.1. Preliminarii

După cum este cunoscut [BALT'84], o cutie poștală reprezintă o structură informațională al cărei element central este un tampon circular (gestionat după principiul *FIFO*), în care orice proces poate scrie informații de un anumit format, invariabil, denumite mesaje, accesibile în citire oricărui alt proces.

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

Comunicarea prin cutie poștală trebuie să se efectueze cu blocarea proceselor producătoare, când acestea ajung în fața unei operații de scriere iar tamponul este plin, respectiv cu blocarea proceselor consumatoare, când acestea ajung în fața unei operații de citire iar tamponul este vid. Scrierea în și citirea din tampon reprezintă secțiuni critice.

Este evident, comunicarea prin cutie poștală se derulează după modelul producătorului și al consumatorului [BALT'84]. Operațiile de sincronizare care fac obiectul acestui model sunt intrinseci mecanismului, împreună cu operațiile de depunere, respectiv de extragere a mesajelor în/din tampon. Așadar, mecanismul asigură degrevarea utilizatorului de sarcina aplicării explicite a modelului producătorului și consumatorului.

În continuare, se propune un mecanism de comunicare prin cutie poștală, cu o implementare *C*, sub sistemul de operare *MS-DOS*.

Se arată modul de implementare a cutiilor poștale propriu-zise.

Se pun în evidență funcțiile mecanismului de comunicare prin cutie poștală și principiile care au stat la baza implementării lor. Textul *C* al acestor funcții este, și el, redat.

6.11.2. Tipul de dată MAILBOX

Pentru implementarea mecanismului de comunicare prin cutie poștală, 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 definește, deci, așa cum se arată în figura 6.11.2_1.

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

Fig. 6.11.2_1. Tipul de dată *MESAJ*.

În vederea înglobării în mecanismul de comunicare prin cutie poștală a regulilor care definesc modelul producătorului și al consumatorului, 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 6.11.2_2.

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

Fig. 6.11.2_2. Tipul de dată *MAILBOX*.

Fie "*_mb[]*" un tablou ce grupează totalitatea structurilor *MAILBOX* de care dispune mecanismul. Considerând dimensiunea acestui tablou specificată prin identificatorul *MAX_MB*, declararea sa se face așa cum se arată în figura 6.11.2_3.

extern MAILBOX _mb[MAX_MB];

Fig. 6.11.2_3. 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.

6.11.3. Funcțiile de operare asupra cutiilor poștale

Comunicarea prin cutie poștală este instrumentată cu ajutorul setului de funcții prezentate, grupat, în figura 6.11.3_1.

void ***mb_init*** *(void)*
/* inițializează tabloul *_mb []* prin */
/* atribuirea valorii *NULL* componentelor *ps* */
/* ale elementelor sale */

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

void ***mb_destroy*** *(usshort ind_mb)*
/* dezalocă elementul tabloului *_mb []* */
/*cu indicele *ind_mb*, repunându-l*/
/* la dispoziția funcției *mb_creat ()* */

void ***mb_send*** *(usshort ind_mb, void *pdata, usint ldata)*
/* depune în cutia poștală cu indicele *ind_mb* */
/* mesajul corespunzător */
/* datei pointate de argumentul *pdata*, */
/* având lungimea, în octeți, */
/* specificată de argumentul *ldata* */

usshort ***mb_receive*** *(usshort ind_mb, void *pdata)*
/* extrage din cutia poștală cu indicele *ind_mb* */
/* mesajul următor, atribuind */
/* valoarea variabilei pe care acesta o reprezintă */
/* variabilei pointate de argumentul *pdata*, */
/* și returnează indexul procesului */
/* producător */

Fig. 6.11.3_1. Funcțiile mecanismului de comunicare prin cutie poștală.

Funcția *mb_init ()*

Această funcție are rolul de a inițializa tabloul de structuri cutie poștală, prin stabilirea valorii *NULL* pentru componentele *ps* ale tuturor elementelor sale.

Textul funcției *mb_init ()* este redat în figura 6.11.3_2.

```
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.    }
```

Fig. 6.11.3_2. Textul funcției *mb_init ()*.

Funcția *mb_creat ()*

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

De asemenea, funcției *mb_creat ()* îi revine sarcina de a inițializa semafoarele din componența structurii cutie poștală alocată, în conformitate cu rolul lor în cadrul mecanismului de comunicare, și anume:

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

În plus, funcția *mb_creat ()* asigură inițializarea componentelor *ps* și *pc* ale cutiei poștale alocată la valoarea de pointare a elementului **0** al componentei *buf []* a acesteia.

Textul funcției *mb_creat ()* este redat în figura 6.11.3_3.

```
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 nu este liberă */
14.   }
15.   else {
16.      _mb[n].s1.contor = 1;
17.      _mb[n].s1.pi = &(_mb[n].s1.coada[0]);
18.      _mb[n].s1.pe = &(_mb[n].s1.coada[0]);
19.      _mb[n].s2.contor = 0;
20.      _mb[n].s2.pi = &(_mb[n].s2.coada[0]);
21.      _mb[n].s2.pe = &(_mb[n].s2.coada[0]);
22.      _mb[n].s3.contor = DIM_CP;
23.      _mb[n].s3.pi = &(_mb[n].s3.coada[0]);
24.      _mb[n].s3.pe = &(_mb[n].s3.coada[0]);
25.      p->ps = &(p->buf[0]);
26.      p->pc = &(p->buf[0]);
27.   }
28.   _unlock_();
29.   return (n);
30. }
```

Fig. 6.11.3_3. Textul funcției *mb_creat ()*.

Funcția *mb_destroy ()*

Funcția *mb_destroy ()* are rolul de a dezaloca elementul tabloului de structuri cutie poștală 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.    }
```

Fig. 6.11.3_4. Textul funcției *mb_destroy ()*.

Funcția *mb_send ()*

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

În conformitate cu modelul producătorului și al consumatorului, depunerea va fi precedată de o secvență de instrucții indivizibilă, care execută operațiile ce definesc primitiva *P*, asupra semafoarelor *s3* și *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* și 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* și *s1*; se semnalează, astfel, faptul că un nou mesaj a fost depus în cutia poștală, respectiv că secțiunea critică s-a încheiat.

Se precizează că funcția *mb_send ()* realizează blocarea, respectiv deblocarea proceselor. făcând apel la funcțiile *sleep ()*, respectiv *wakeup ()*. Argumentul *timeout* al funcției *mb_send ()* este destinat funcției *sleep ()*. 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 ca urmare a execuției funcției *mb_send ()* -situație de "time-out"-, se produce o deblocare automată.

Textul funcției *mb_send ()* este redat în figura 6.11.3_5.

```
/*************************************************/
/*      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, void *pdata, usint ldata)
2.    {
3.     register usshort tsk;
4.     SEMAPHORE *p1, *p2, *p3;
5.     p1 = _mb[ind_mb].s1;
6.     p2 = &_mb[ind_mb].s2;
7.     p3 = &_mb[ind_mb].s3;
8.     _lock_();
9.     if (--p3->contor < 0) {
10.        *p3->pi = _task_crt;
11.        if (p3->pi == &(p3->coada[MAX_TSK-1])) {
12.           p3->pi = &(p3->coada[0]);
13.        }
14.        else {
15.           p3->pi++;
16.        }
17.        sleep (0);
18.    }
```

Fig. 6.11.3_5. Textul funcției *mb_send ()* (partea 1/2).

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

Fig. 6.11.3_5. Textul funcției *mb_send ()* (partea 2/2).

Funcția *mb_receive ()*

Funcția *mb_recevie ()* 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.

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.

În final, o nouă secvență de instrucții indivizibilă execută operațiile ce definesc primitiva *V*, asupra semafoarelor *s3* și *s1*; se semnalează, astfel, faptul că un nou mesaj a fost extras din cutia poștală, respectiv că secțiunea critică s-a încheiat.

```
/*************************************************/
/*    funcția mb_receive () 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];                     */
/*                                                      */
/*************************************************/
```

Fig. 6.11.3_6. Textul funcției *mb_receive ()* (partea 1/3).

```
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.    }
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 ==  &(_mb[ind_mb].buf[DIM_CP-1])) {
40.        _mb[ind_mb].pc = &(_mb[ind_mb].buf[0];
41.    }
42.    else {
43.        _mb[ind_mb].pc++;
44.    }
```

Fig. 6.11.3_6. Textul funcției *mb_receive ()* (partea 2/3).

```
45.     if (++p3->contor <= 0) {
46.         tsk = *p3->pe;
47.         if (p3->pe == &(p3->coada[MAX_TSK-1])) {
48.             p3->pe = &(p3->coada[0]);
49.         }
50.         else {
51.             p3->pe++;
52.         }
53.         wakeup (tsk);
54.     }
55.     if (++p1->contor <= 0) {
56.         tsk = *p1->pe;
57.         if (p1->pe == &(p1->coada[MAX_TSK-1])) {
58.             p1->pe = &(p1->coada[0]);
59.         }
60.         else {
61.             p1->pe++;
62.         }
63.         wakeup (tsk);
64.     }
65.     _unlock_();
66.     return (indprd);
67. }
```

Fig. 6.11.3_6. Textul funcției *mb_receive ()* (partea 3/3).

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

Textul funcției *mb_receive ()* este redat în figura 6.11.3_6.

6.11.4. Concluzii

Mecanismul propus face posibilă comunicarea interprocese într-o manieră simplă și eficientă. Timpul procesor consumat de mecanism este relativ scăzut. Mecanismul lasă, însă, în sarcina utilizatorului, grija de a nu modifica data ai cărei parametri se transmit printr-o funcție *mb_send ()*, până după ce ea a fost preluată printr-o funcție *mb_receive ()*.

7
CONCLUZII GENERALE

Capitolele 1-6 ale lucrării pot fi privite ca reprezentând patru părți.

Partea întâi, subsumând capitolele 1-3, este dedicată evidențierii importanței acordată actualmente în lume fabricației ntegrate prin calculator și punctării principiilor după care aceasta este, în prezent, abordată; de asemenea, ea ocazionează un excurs asupra sistemelor flexibile -temelie a fabricației integrate prin calculator-, vizând, dominant, sistematizarea considerării structurii acestora -inclusiv a sistemelor lor de conducere- și consemnarea celor mai importante soluții industriale -actuale și de perspectivă-, privitoare la rețelele informaționale aferente. Principalele concluzii ce pot fi desprinse din această primă parte sunt:

◊ industria actuală parcurge un proces de redefinire, având ca punct de plecare conceptul de fabricație integrată prin calculator;
◊ la temelia conceptului de fabricație integrată prin calculator stă ideea de sistem flexibil de fabricație;
◊ sistemele flexibile de fabricație sunt, din punct de vedere structural, sisteme ierarhizate multinivel;
◊ conducerea sistemelor flexibile de fabricație se realizează cu sisteme informatice distribuite multinivel;
◊ complexitatea problematicii aferente fabricației integrate prin calculator impune respectarea unor specificații promovate drept standarde, de către Comunitatea Europeană, ISO, ANSI, sau unele dintre marile companii, cum sunt: Boeing, General Motors, ș.a.
◊ toate specificațiile dedicate domeniului fabricației integrate prin calculator respectă preceptul de sistem deschis;
◊ cel puțin încă timp de câțiva ani, rețelele informatice standard pentru domeniul fabricației integrate prin calculator vor fi cele de specificație MAP.

Această parte fiind planificată să reprezinte, în esența sa, un studiu bibliografic, la elaborare nu ne-am propus să venim cu contribuții originale. Totuși, capitolul 3, în special prin paragrafele sale 3.1 și 3.2, este, în bună măsură, purtător al unor puncte de vedere proprii.

Partea a doua, suprapusă capitolului 4, se vrea menită să pună și să rezolve probleme privind modelarea unor subsisteme și sisteme flexibile de fabricație, într-o abordare pragmatică, originală, folosind rețele Petri: ordinare ("condiții-evenimente", respectiv "poziții-tranziții"), colorate, și continue. Subsistemele și sistemele considerate sunt, din punct de vedere generic, cazuri de studiu prezente în bibliografie. Lucrarea de față le definește, însă, în manieră originală, astfel încât ele să corespundă cât mai fidel, atât sub aspect structural, cât și din punct de vedere comportamental, unor entități regăsibile în realitatea tehnică. Contribuțiile autorului prezente în această parte sunt:

- definirea unui subsistem de fabricație format dintr-o mașină-unealtă și un manipulator;

- construcția rețelei Petri de tip condiții-evenimente corespunzătoare subsistemului *"o mașină-unealtă~un manipulator"*;
- analiza rețelei Petri aferentă subsistemului *"o mașină-unealtă~un manipulator"*;
- definirea unui subsistem de fabricație format din două mașini-unelte și un manipulator;
- construcția rețelei Petri de tip poziții-tranziții corespunzătoare subsistemului *"două mașini-unelte~un manipulator"*;
- analiza rețelei Petri aferentă subsistemului *"două mașini-unelte~un manipulator"*, din aproape în aproape și cu ajutorul invarianților;
- definirea unui subsistem de fabricație format din "*n*" mașini-unelte și un manipulator;
- construcția rețelei Petri de tip poziții-tranziții corespunzătoare subsistemului *"n mașini-unelte~un manipulator"*;
- definirea unui subsistem de fabricație de tip cooperativ, format din două mașini-unelte și două manipulatoare;
- construcția rețelei Petri de tip condiții-evenimente corespunzătoare subsistemului cooperativ *"două mașini-unelte~două manipulatoare"*;
- analiza rețelei Petri aferentă subsistemului *"două mașini-unelte~ două manipulatoare"*, din aproape în aproape și cu ajutorul invarianților;
- definirea unui subsistem de fabricație de tip cooperativ, format din "*n*" mașini-unelte și "*n*" manipulatoare;
- construcția rețelei Petri de tip colorată corespunzătoare subsistemului cooperativ *"n mașini-unelte~n manipulatoare"*;
- analiza rețelei Petri aferentă subsistemului *"n mașini-unelte~n manipulatoare"*, din aproape în aproape și cu ajutorul invarianților;
- modelarea liniilor de fabricație deschise cu rețele Petri continue -deducerea expresiilor stocurilor tampon medii;
- modelarea liniilor de fabricație închise cu rețele Petri continue -deducerea expresiilor stocurilor tampon medii;

Partea a treia, reprezentată de capitolul 5, acoperă un segment din problematica conducerii sistemelor de fabricație cu integrare prin calculator, și anume: segmentul referitor la ordonanțarea operațiilor tehnologice și a sarcinilor de transport. Contribuțiile autorului cuprinse în această parte sunt:

- o manieră de formulare a problemei ordonanțării, ținând seamă de restricții de orientare pragmatică: de precedență, de prelucrabilitate, de compatibilitate, de ritm, respectiv de transport;
- o metodă de rezolvare a problemei ordonanțării, fondată pe disocierea acestei probleme în două subprobleme: una referitoare la partiționarea operațiilor proceselor tehnologice în faze, iar cealaltă referitoare la acoperirea fazelor cu celule de fabricație;

iar corespunzător acestora:

- un algoritm de partiționare echilibrată a proceselor tehnologice în faze, cu respectarea restricțiilor de precedență, de prelucrabilitate, de compatibilitate, și de ritm, utilizând grupările

admisibile și tehnica drumului critic (algoritmul propus este redat detaliat, în exprimare pseudocod);

- expresia pseudocod a unui algoritm de partiționare echilibrată a proceselor tehnologice în faze, cu respectarea restricțiilor de precedență, de prelucrabilitate, de compatibilitate, și de ritm, acționând după principii de programare dinamică (algoritmul în sine este preluat din literatură și modificat încât să înglobeze restricțiile menționate);
- un algoritm de partiționare echilibrată a proceselor tehnologice în faze, cu respectarea restricțiilor de precedență, de prelucrabilitate, de compatibilitate, și de ritm, utilizând direct graful de precedență și tehnicile "*branch and bound*" și "*backtracking*";
- un algoritm de partiționare echilibrată a proceselor tehnologice în faze, cu respectarea restricțiilor de precedență, de prelucrabilitate, de compatibilitate, și de ritm, utilizând direct graful de precedență și tehnica euristică "*greedy*", aplicabil în timp real;
- un algoritm de acoperire optimă a fazelor procesului tehnologic cu celule flexibile.

Remarcă:

Toate aceste contribuții -ne referim la cele cuprinse în partea a treia a lucrării- au fost valorificate într-un produs-program, elaborat de colaboratorii autorului Zsolt HAAG și Adrian DUNU, studenți UTT, anul V -Automatică și Informatică Industrială.

Partea a patra este cvasiintegral dedicată prezentării unui set de soluții propuse de autor -deci reprezentând contribuții proprii-, cu menirea de a fi apte a sta la baza generării unui suport de programare, în regim concurent și în timp real, a aplicațiilor din domeniul fabricației integrate prin calculator -suport utilizabil, volens nolens, și în alte aplicații-. Din respectivul set fac parte:

- o soluție de comutare a proceselor;
- o soluție de dispecerizare a proceselor;
- o soluție de excludere mutuală prin fanioane de excluziune active;
- o soluție de excludere mutuală prin semafoare;
- o soluție de excludere mutuală prin blocuri resursă;
- o soluție de sincronizare pri blocuri eveniment;
- o soluție de sincronizare prin blocuri multieveniment;
- o soluție de sincronizare prin blocuri rendez-vous;
- o soluție de comunicare prin conducte;
- o soluție de comunicare prin cutii poștale.

Remarcă:

Toate aceste soluții -sunt avute în vedere cele cuprinse în partea a patra a lucrării- au fost valorificate într-un produs-program, elaborat de autor sub numele RTC86, produs reprezentând un executiv de timp real grefat pe sistemul de operare MS-DOS și apărând pentru utilizator ca extensie a mediilor de programare TURBO C și BORLAND C(++).

În final, facem mențiunea că o parte a rezultatelor obținute de autor în activitatea de elaborare a prezentei teze fac obiectul a 12 lucrări științifice, publicate în 1994 -acestea sunt în număr de 8-, respectiv acceptate pentru publicare în 1995 -acestea sunt în număr de 4-.

BIBLIOGRAFIE:

[ACHU'82] **Achugbue, J.O., Chin, F.Y.**: "*Complexity and Solution of Some Three-Stage Flow Scheduling Problem*", Mathematical Operational Research, no. 7, pp. 532-544, 1982.

[ADAM'88] **Adams, J., Balas, E., ...**: "*The Shifting Bottleneck Procedure for Job Shop Scheduling*", Management Science, no. 34, pp. 391-401, 1988.

[AHO'87] **Aho, A., Hopcroft, J., ...**: "*Structures des donnees et alghorithmes*", InterEditions, Paris, 1987.

[ALLA'88a] **Alla, H., David, R.**:"*Modelling of Production Systems by Continuous Petri Nets*", 3th International Conference on CAD/CAM, CARS&FOF88, Detroit, August, 1988.

[ALLA'88b] **Alla, H., David, R.**: "*Modelisation de production/Gestion, par Reseaux de Petri Continus*", Congres AFCET Automatique, Grenoble, Octobre, 1988.

[ANDL'78] **Andler, S.**: "*Syncronization Primitives and the Verification of Concurent Programs*", IRIA and CMU Symposium of Operating Systems -Theory and Practice, Rocquencourt, 1978.

[ALLW'81] **Allworth, S.T.**: *"Introduction to Real-Time Software Design"*, Macmillan Press Ltd., London, 1981.

[AMAR'90] **Amar, S., Castelain, E.**: *"Modelisation des moyens de production par langages orientes objet en vue de la conception de la commande d'un SPF"*, Congres CIM90, Proceedings, pp. 323-331, Bordeaux, 1990.

[AMAR'92] **Amar, S., Craye, E., ...**: *"Une methode hierarchique de specification et de prototypage des systemes de production flexibles"*, RAIRO/APII, vol. 26, pp. 483-514, 1992.

[AMAL'89] **Amaldi, E., Mayoraz, E., ...**: "*Apprentissage dans les reseaux de Hopfield*", Proceedings of the International Conference on Artificiel Neural Networks, EPFL, Lausanne, 1989.

[APPE'84] **Appelbe, W.F., Rown, A.P.**: *"Encapsulation Constructs in Systems Programming Languages"*, ACM Tranzaction on Programming Languages and Systems, Vol. 6, Nr. 2, 1984.

[ARCH'84] **Achier, G., Serieyx, H.**: "*L'entreprise du toisieme type*", Le Seuil, Paris, 1984.

[ASHT'89] **Ashton, J.E., Cook Jr., F.W.**: "*Time to Reform Job-Shop Manufacturing*", Harward Bussines Review, pp. 106-111, Mar.-Apr. 1989.

[AXSA'84] **Axsater, S., Jonsson, H.**: "*Aggregation and Disaggregation in Hierarchical Production Planning*", EJOUR, no.17, pp.338-350, 1984.

[BABB'87] **Babb, M.**: "*MAP 3.0 -Reaching Out for Stability*", Control Engineering, Oct., 1987.

[BAKE'74] **Baker, K.R.**: "*Introduction to Sequencing and Scheduling*", John Wiley and Sons Editions, 1974.

[BALT'84] **Baltac V., Davidoviciu, A., ...**: "*Sisteme interactive și limbaje conversaționale. Utilizare și proiectare*", Editura Tehnică, București, 1984.

[BAPT'91a] **Baptiste, P., Favrel, J.**: "*Representation of a maximal set of feasible routings in a real time control environment: a PQR trees approach*", Proceedings of the Eighth International Conference on Systems Engineering, Coventry, 1991.

[BAPT'91b] **Baptiste, P., Cho, C.H., ...**:"*Une caracterisation analitique des ordonnancements admissibles sous contraintes heterogenes en flow-shop*", RAIRO/APII, no. 25, pp. 87-102, Paris, 1991.

[BARB'92] **Barbier, F., Jaulent, P.**: "*Techniques orientees objet et CIM*", Eyrolles, Paris, 1992.

[BEL '85] **Bel, G., Dubois, D.**: "*Modelisation et simulation des systemes automatises de production*", RAIRO/APII, no. 19/1, 1985.

[BELL'82] **Bellman, R., Esogbue, A.O., ...**: "*Mathematical Aspects of Scheduling and Applications*", Pergamon Press Editions, Oxford, 1982.

[BERL'88] **Berlioux, P., Biyard, Ph. ...**: "*Algorithmique. Structures de donnees et algorithmes de recherche*", Dunod, Paris, 1988.

[BERR'86] **Berrada, M., Stecke, K.E.**: "*A Branch and Bound Approach for Machine Load Balancing in Flexible Manufacturing Systems*", Management Science, vol. 32, pp. 1316-1335, 1986.

[BIEG'90] **Biegel, J.E., Davern, J.J.**: "*Genetic Algorithms and Job Shop Scheduling*", Computer Industrial Engineering, no. 19, pp. 81-91, 1990.

[BLAN'86] **Blanchard, M.**: *"Comprendre, maitriser et appliquer le GRAFCET"*, Editions Copadues, Toulouse, 1986.

[BORA'89] **Borangiu, Th., Dobrescu, R., ...**: "*Conducerea multiprocesor în timp real a structurilor flexibile de fabricație*", Editura Tehnică, București, 1989.

[BOUR'87] **Bourjault, A., Chappe, D., ...**: "*Elaboration automatique des gammes d'assemblage a l'aide de Reseaux de Petri*", RAIRO/APII, no. 21, pp. 323-342, 1987.

[BOUR'88] **Bourrieres, J.P., Chevillard, A.**: "*Modelisation des convoyeurs a transfert libre a l'aide de Reseaux de Petri*", RAPA, no. 4, pp. 87-100, 1988.

[BRAM'89] **Brams, G. W.**: "*Reseaux de Petri: Theorie et analyse*", Editions Masson, Paris, 1983.

[BRUC'91] **Brucker, P., Jurish, B., ...**: "*A Branch and Bound Algorithm for the Job Shop Scheduling*", Applications of Mathematics, 1991.

[BUZA'85] **Buzacott, J.A., Shanthikumar, J.G.**: "*Models for Understanding Flexible Manufacturing Systems*", IEEE Tranzactions, no. 12, pp. 339-350, 1980.

[BUZA'85] **Buzacott, J.A., Yao, D.D.**: "*Queueing Models for Flexible Manufacturing Station*", European Journal of Operational Research, no. 19, pp. 233, 252, 1985.

[CARL'82a] **Carlier, J., Chretienne, P.**: "*Une domaine tres ouvert: les problemes des ordonnancements*", RAIRO/APII, no. 12, pp. 175-217, 1982.

[CARL'82b] **Carlier, J., Chretienne, P., ...**: "*Modelling Scheduling Problems with Timed Petri Nets*", 4th European Workshop on Theory and Applications of Petri Nets, Actes, pp. 84-103, 1982.

[CARL'88] **Carlier, J., Chretienne, P.**: "*Problemes d'ordonnancement: Modelisation, Algorithmes et Complexite*", Editions Masson, Paris, 1988.

[CARL'89a] **Carlier, J., Chretienne, P., ...**: "*Timed Petri Nets Schedules*", Advances in Petri Nets 1988, Lecture Notes in Computer Science, Springer Verlag, January, 1989.

[CARL'89b] **Carlier, J., Pinson, E.**: "*A Branch and Bound Method for Solving the Job Shop Problem*", Management Science, no.35/2, pp. 164-176, February, 1989.

[CARL'89c] **Carlier, J., Pinson, E.**: "*An Algorithm for Solving the Job-Shop Problem*", Management Science, no. 35/2, pp. 164-176, Feb. 1989.

[CĂLI'88] Călin, S., Popescu, Th., ...: "*Conducerea adaptivă și flexibilă a proceselor industriale*", Editura Tehnică, București, 1988.

[COHE'85] **Cohen, G., Dubois, O., ...**: "*A Linear System Theoretic View Discrete Event Processes and its Use for Performance Evaluation in Manufacturing*", IEEE Tranzactions on Automatic Control, March, 1985.

[CHRE'91] **Chretienne, P.**: "*The Cyclic Scheduling Problem with Deadlines*", Applications of Mathematics, no. 30, pp. 109-123, 1991.

[CHRY'91] **Cryssolouris, G., Dicke, K., ...**: "*An Approach to Short Interval Scheduling for Discrete Parts Manufacturing*", International Journal of Computer Integrated Manufacturing, no. 4/3, pp. 157-168, 1991.

[CHU '89] **Chu, C., Portmann, M.C.**: "*Minimization de la somme de retards pour les problemes d'ordonnancement a une machine*", Rapport de Recherche INRIA no. 1023, Avril, 1989.

[CREȚ'84] Crețu, V.: "*Sisteme de operare timp real pentru sisteme cu microprocesor*", Teză de doctorat, Institutul Politehnic "Traian Vuia" din Timișoara, 1984.

[CROW'86] **Crowder, R.**: "*The MAP/TOP Specification Update*", Control Engineering, Oct., 1986.

[DALL'86] **Dallery, Y.**: "*On Modelling Flexible Manufacturing Systems Using Closed Queueing Networks*", Large Scale Systems, no. 11, pp. 109-119, 1986.

[DALL'87] **Dallery, Y.**: "*Approximate Analysis of General Open Queueing Networks with Restricted Capacity*", Performance Evaluation, no. 11, pp. 209-222, 1987.

[DABĂ'86] Davidoviciu, A., Bărbat, B.: *"Limbaje de programare pentru sisteme în timp real"*, Editura Tehnică, București, 1986.

[DAVI'87a] **David, R., Alla, H.**: "*Continuous Petri Nets*", 8th European Workshop on Application and Theory of Petri Nets, Proceedings, pp. 275-294, Saragosse, Juin, 1987.

[DAVI'87b] **David, R., Alla, H.**: "*Autonomous and Timed Continuous Petri Nets*", 11th International Conference on Application and Theory of Petri Nets, Proceedings, pp. 367-386, Paris, June, 1987.

[DAVI'89] **David, R., Alla, H.**: "*Du GRAFCET aux Reseaux de Petri*", Editions Hermes, Paris, 1989.

[DEME'90] **Demeulemeester, E., Herroelen, W.**: "*A Branch and Bound Procedure for the Multiple Constrained Resource Project Scheduling Problem*", EURO WG-PMS, Compiegne, 1990.

[DEWE'87] **De Werra, D.**: "*Design and Operation of Flexible Manufacturing Systems: the Kingdom of Heuristic Methods*", RAIRO/Recherche Operationelle, no. 21, pp. 365-382, 1987.

[DEWE'88] **De Werra, D., Widmer, M.**: "*Les ateliers flexibles*", Output, no. 1, pp. 36-41, 1988.

[DIJK'65] **Dijkstra, E.W.**: "*Solution of a Problem in Concurrent Programming Control*", Communications ACM, Nr. 8, pp. 569, 1965.

[ELEȘ'91] Eleș, P., Ciocârlie H.: *"Programarea concurentă în limbaje de nivel înalt"*, Editura Științifică, București, 1991.

[ERSC'81a] **Erschler, J., Roubellat, F., ...**: "*Une approche pour l'ordonnancement en temps reel d'attelier*", Congres AFCET Automatique, 1981.

[ERSC'81b] **Erschler, J., Ancelin, B., ...**: "*A Simulation Tool for Manufacturing Systems Design and Control Aid*", 4th IFAC-IFORS Symposium Large Scale Systems, Zurich, 1981.

[ERSC'81c] **Erschler, J., Lopez, P., ...**: "*Raisonnement temporel sous contraintes de ressource et problemes d'ordonnancement*", Revue d'intelligence artificielle", no. 5, pp. 7-32.

[ERSC'83] **Erschler, J., Fontan, G., ...**: "*New Dominance Concepts for Scheduling N Jobs on a Single Machine with Ready Times and Due Dates*", Operational Research, no. 1/3, 1983.

[EVEN'80] **Eventoff, M., ...**: "*The Rendez-Vous and Monitor Concepts: Is There an Efficiency Difference?*", SIGPLAN Notices, Nr. 15/11, Nov. 1980, pp. 156-165.

[FALK'91] **Falkenauer, E., Bouffouix, S.**: "*A Genetic Algorithm for JOB Shop*", Proceedings of the 1991 IEEE International Conference on Robotics and Automation, pp. 824-829, Sacramento, April, 1991.

[FDID'86] **Fdida, S., Pujolle, G., ...**: "*Reseaux de files d'attente avec semaphores*", T.S.I., pp. 187-196, 1986.

[FERR'91] **Ferreira, J. F., Ribeiro, B., ...**: "*Selection of Manufacturing Cells for Group Technology*", IMACS MCTS'91, Proceedings, vol. 1, pp. 675-681, Lille, 1991.

[FREN'82] **French, S.**: "*Sequencing and Scheduling*", John Wiley and Sons Editions, 1982.

[FROM'84] **Froment, B., Lesage, J.J.**: "*Productique: les techniques de l'usinage flexible*", Dunod, Paris, 1984.

[GASN'88] **Gasnier, B., Vercauter, C.**: "*An Interactive Tool for Computer Aided Manufacturing Sequence Specification*", Congres IFAC88, Proceedings, pp. 297-300, Swansea, 1988.

[GASP'91] **Gasperoni, F., Schwiegelsohn, U.**: "*Efficient Algorithms for Cyclic Scheduling*", IBM Tecnic Report, RC 17068, August, 1991.

[GLOV'89] **Glover, F.**: "*TABU Search: Part I*", ORSA Journal on Computing, vol. 1, pp. 190-206, 1989.

[GLOV'89] **Glover, F.**: "*TABU Search: Part II*", ORSA Journal on Computing, vol. 2, pp. 4-32, 1990.

[GRAB'80] **Grabowski, J.**: "*On Two Machine Scheduling with Release and Due Dates to Minimize Maximum Lateness*", Operational Research, no.17, 1980.

[GRAB'82] **Grabowski, J.**: "*A New Algorithm of Solving the Flow Shop Problem*", Operational Research in Progress, Reidel, Dordrecht, pp. 55-75, 1982.

[GRAV'81] **Graves, S. C.**: "*A Review of Production Scheduling*", Operational Research, no. 29, pp. 646-675, 1981.

[GUPT'89] **Gupta, Y.P., Gupta, M.C., ...**: "*A Review of Scheduling Rules in Flexible Manufacturing Systems*", International Journal of CIM, no. 2/6, pp. 356-377, 1989.

[GURA'86] **Guran, M., Filip, F.G.**: "*Sisteme ierarhizate în timp real cu prelucrare distribuită a datelor*", Editura Tehnică, București, 1986.

[HAMA'86] **Hamachi, S., Kieffer, J.P.**: "*Les progiciels de la gestion de production*", Editions de l'Usine nouvelle, Paris, 1986.

[HENN'92] **Hennet, J.C., Passos, C.A.S., ...**: "*Optimal routing of products in a queueing network representing a FMS*", ECC91, vol. 3, pp. 2107-2112, 1991.

[HERT'90] **Hertz, A., De Werra, D.**: "*The Tabu Search Metaheuristic: How to Used It*", Annals of Mathematics and Artificial Intelligence, no. 1, pp. 111-121, 1990.

[HILL'89] **Hillion, H.P., Proth, J.M.**: "*Performance Evaluation of Job Shop Systems Using Timed Event Graph*", IEEE Tranzactions on Automatic Control, vol. AC-34, January, 1989.

[HOAR'74] **Hoare, C.**: "*Monitors: An Operating System Structuring Concept*", Communications ACM, Nr. 17/10, pp. 549-557, Oct. 1974.

[HOIT'90] **Hoitomt, D.J., Luh, P.B.**: "*Job-Shop Scheduling with Simple Precedence Constraints*", in: Proceedings of the Automatic Control Conference, pp. 1-6, San Diego, CA, May 1990.

[HOIT'92] **Hoitomt, D.J., Luh, P.B.**: "*Scheduling Jobs with Simple Precedence Constraints on Parallel Machines*", Dynamics of Discrete Event Systems, Y.C.Ho, Ed. Piscataway, NJ: IEEE Press, pp. 271-277, 1992.

[HOIT'93] **Hoitomt, D.J., Luh, P.B.**: "*A Practical Approach to Job-Shop Scheduling Problems*", IEEE Transactions on Robotics and Automation, no. 9/1, pp.1-13, February, 1993.

[HUVE'92] **Huvenoit, B., Craye, E., ...**: "*Elaboration de la commande des cellules de production flexibles dans l'industrie manufacturiere*", Conferences Canadiennes sur l'Automatisation Industrielle, pp. 621-625, Montreal, 1992.

[HWAN'89] **Hvang, J.J., Chow, Y.C., ...**: "*Scheduling Precedence Graphs in Systems with Interprocessor Communication Times*", SIAM Journal of Computers, no. 18, pp. 244-257, April, 1989.

[JACK'80] **Jackson, K., Harte, H.F.**: "*The Achievement of Well-Structured Software in Real- Time Applications*", MASCOT Suppliers Asociation, London, 1980.

[KIME'85] **Kimenia, J., Gershwin, S.B.**: "*Flow Optimization in Flexible Manufacturing Systems*", International Journal of Production Research, no. 23/1, pp. 81-96, 1985.

[KUSI'85a] **Kusiak, A., Vanneli, A., ...**: "*Grouping Problem in Scheduling Flexible Manufacturing Systems*", Robotica, no. 3, pp. 245-252, 1985.

[KUSI'85b] **Kusiak, A.**: "*Efficient Implementation of Johnson's Scheduling Algorithm*", IEEE Tranzactions on Automatic Control, no. 18, pp. 215-216, 1986.

[KUSI'85c] **Kusiak, A., Chen, M.**: "*Expert Systems for Planning and Scheduling Manufacturing Systems*", EJOR, no. 34, pp. 113-130, 1988.

[KUSI'85d] **Kusiak, A.**: "*Flexible manufacturing systems: a structural approach*", International Journal of Production Research, no. 23, 1985.

[KUSI'90] **Kusiak, A.**: "*Intelligent Manufacturing Systems*", Englewood Cliffs, NJ: Prentice-Hall, 1990.

[LEGA'92] **Le Gall, A., Roubellat, F.**: "*Caracterisation d'un ensemble d'ordonnancements avec contraintes de resources de type cumulatif*", RAIRO/APII, no. 26, pp. 483-514, 1992.

[LEIS'90] **Leisten, R.**: "*Flowshop Problems with Limited Buffer Storage*", International Journal of Production Research, no. 18/11, pp. 2085-2100, 1990.

[MINO'86] **Minoux, M.**: "*Programmation mathematique: theorie et algorithmes*", tome 1, Editions Dunod, Paris, 1986.

[MOTT'89] **Mottet, Y., Widmer, M.**: "*De la commande du client a la gestion en temps reel: le chemin vers les systemes de production integres*", Output, no. 12, pp. 85-92, 1989.

[MUNI'91] **Munier, A.**: "*Resolution d'un probleme d'ordonnancement cyclique a iterations independantes et contraintes de ressource*", Operational Research, no. 25/2, pp. 161-182, 1991.

[OHMA'85] **Ohmae, K.**: "*La triade*", Flammarion, Paris, 1985.

[PERE'90] **Perez, J.P.**: "*Systemes Temps Reel -Methodes de Specification et de Conception,* Dunod, Paris, 1990.

[PORT'88] **Portmann, M.C.**: "*Methodes de decomposition spatiales et temporales en ordonnancement de la production*", RAIRO/APII, no. 22/5, pp.439-451, 1988.

[QUIN'89] **Quinton, P., Robert, Y.**: "*Algorithmes et architectures systoliques*", Etudes et Recherches en Informatique, Editions Masson, Paris, 1989.

[PROU'87] **Proust, C., Drogou, M., ...**: "*Une heuristique pour le probleme d'ordonnancement statique de type n/m/flow-shop avec prise en compte des temps de montage et de demontage d'outils*", 2 -eme Conference Internationale de Systemes de Production, INRIA, pp. 125-141, Paris, 1987.

[RADH'86] **Radharaman, R**: "*A Heuristic Algorithm for Group Scheduling*", Proceedings of the International Industrial Engineering Conference, pp. 229-236, 1986.

[ROUB'87] **Roubellat, F., Thomas, V.**: "*Une methode et un logiciel pour l'ordonnancement en temps reel d'attelier*", Seconde Conference Internationale des Systemes de Production, Proceedings, pp. 87-101, Paris, Avril, 1987.

[ROBU'88a] Robu, N.: "*Sistem informatic de proces cu multitasking*", Lucrările celui de Al V -lea Simpozion Național de Teoria Sistemelor, Vol. IV -Sisteme de Calcul, Craiova, pp.133-138, 1988.

[ROBU'88b] Robu, N.: "*RTEI80: Executiv de timp real dedicat dezvoltării sistemelor informatice de proces cu prelucrare concurentă pe microprocesorul INTEL 8080*", Lucrările celui de Al V -lea Simpozion Național de Teoria Sistemelor, Vol. IV -Sisteme de Calcul, Craiova, pp. 139-145, 1988.

[ROBU'94a] Robu, N.: "*Mecanism de comutare a taskurilor într-un executiv de timp real*", Revista Română de Informatică și Automatică, nr. 1, pp. 79-82, București, 1994.

[ROBU'94b] Robu, N.: "*Mecanism de dispecerizare a taskurilor într-un executiv de timp real*", Revista Română de Informatică și Automatică, nr. 1, pp. 83-86, București, 1994.

[ROBU'94c] Robu, N.: "*Mecanism de excludere mutuală prin fanioane de excluziune active*", Revista Română de Informatică și Automatică, nr. 2, pp. ??-??, București, 1994.

[ROBU'94d] Robu, N.: "*Mecanism de excludere mutuală și sincronizare bazat pe semafoare*", Revista Română de Informatică și Automatică, nr. 3, pp. ??-??, București, 1994.

[ROBU'94e] Robu, N.: "*Mecanism de excludere mutuală prin blocuri resursă*", Revista Română de Informatică și Automatică, nr. 4, pp. ??-??, București, 1994.

[ROBU'94f] Robu, N.: "*Mecanism de sincronizare prin blocuri eveniment*", Revista Română de Informatică și Automatică, nr. 4, pp. ??-??, București, 1994.

[ROBU'94g] Robu, N.: "*Backtracking Algorithm Using the Branch and Bound Technique for the Balanced Partitioning of Technological Processes into Phases*", Proceedings of International Conference on Technical Informatics, vol. 3, pp. 239-249, November, 1994.

[ROBU'94h] Robu, N.: "*Greedy Heuristic Algorithm for the Balanced Partitioning of Technological Processes into Phases*", Proceedings of International Conference on Technical Informatics, vol. 3, pp. 239-249, November, 1994.

[ROBU'95a] Robu, N.: "*Stadiul actual și tendințele de evoluție în fabricația integrată prin calculator*", Referat de doctorat nr. 1, 1995.

[ROBU'95b] Robu, N.: "*Contribuții privind integrarea prin calculator a fabricației*", Referat de doctorat nr. 2, 1995.

[ROBU'95c] Robu, N.: "*Mecanism de sincronizare prin blocuri multieveniment*", Revista Română de Informatică și Automatică, nr. 1, pp. ??-??, București, 1995.

[ROBU'95d] Robu, N.: "*Mecanism de sincronizare prin blocuri rendez-vous*", Revista Română de Informatică și Automatică, nr. 1, pp. ??-??, București, 1995.

[ROBU'95e] **Robu, N.**: "*Mecanism de comunicare prin cutie poștală*", Revista Română de Informatică și Automatică, nr. 2, pp. ??-??, București, 1995.

[ROBU'95f] **Robu, N.**: "*Mecanism de comunicare prin conductă*", Revista Română de Informatică și Automatică, nr. 2, pp. ??-??, București, 1995.

[ROBU'95g] **Robu, N.**: "*Mecanismele programării concurente în timp real. Definire și implementare*", Editura Helicon, Timișoara, 1995.

[ROUB'88] **Roubellat, F., Thomas, V.**:"*Une methode et un logiciel pour l'ordonnancement en temps reel d'ateliers*", RAIRO/APII, no. 22, pp. 419-438, 1988.

[RUSU'90] **Rusu, C., Brudaru, O.**: "*Proiectarea liniilor de fabricație flexibilă*", Editura Tehnică, București, 1990.

[SHAL'85] **Shalev-Oren, S., Seidmann, A.,...**: "*Analysis of Flexible Manufacturing Systems with Priority Scheduling: PMVA*", Annals of Operational Research, vol. 3, pp. 115-139, 1985.

[SEDG'91] **Sedgewick, R.**: "*Algorithmes en langage C*", InterEditions, Paris, 1991.

[SEKI'83] **Sekine, K.**: "*Kanban: gestion de production a zero stock*", Editions Hommes et Techniques, Paris, 1983.

[SIMP'78] **Simpson, H.R., Jackson, K.**: "*Process Syncronization in MASCOT*", The Computer Journal, no. 22/4, pp. 332-345, 1978.

[SLOW'89] **Slowinski, R., Weglarz, J.**: "*Advances in Project Scheduling*", Elsevier Science, Amsterdam, April 1989.

[SOLO'88] **Solot, P., Bastos, J.M., ...**: *"MULTIQ: a Queueing Model for FMSs with Several Pallet Types*", Journal of the Operational Research Society, vol. 39, pp. 811-821, 1988.

[STAN'88] **Stankovic, J.A.**: "*Real Time Computing Systems: The Next Generation*", COINS Technical Report 88-06, University of Massachusetts at Amherst, 1988.

[STAN'88] **Stankovici, J.A., Ramamritham, K.**: "*The Design of the Spring Kernel*", COINS Technical Report 88-85, University of Massachusetts at Amherst, 1988.

[STEC'85] **Stecke, K.E., Morin, T.L.**: "*The Optimality of Balancing Workloads in Certain Types of Flexible Manufacturing*", European Journal of Operational Research, no. 20/1, pp. 68-82, 1985.

[TAYL'80] **Taylor, B.J.**: "*A Method for Expressing Functional Requirements of Real-Time Systems*", IFAC/IFIP Workshop on Real-Time Programming, Leibnitz, pp. 111-120, April 1980.

[TOGU'90] **Toguyeni, A.K.A., Craye, E., ...**: "*A Method of Temporal Analysis to Perform On-line Diagnosis in the Context of Flexible Manufacturing System*", IECON90 Symposium, Proceedings, vol. 1, pp. 445-450, Pacific Grove, California, 1990.

[THOR'90] **Thorin, M.**: *"Parallelisme: Genie Logiciel Temps Reel"*, Dunod, Paris, 1990.

[TSCH'90] **Tschirhart, D.**: "*Commande en Temps Reel*", Dunod, Paris, 1990.

[VINO'85] **Vinod, B., Solberg, J.J.**: "*The Optimal Design of Flexible Manufacturing Systems, International Journal of Production Research*", no. 23/6, pp. 1141-1151, 1985.

[WALD'90] **Waldner, J.B.**: "*CIM: les nouvelles perspectives de la production*", Dunod, Paris, 1990.

[WHITE'90] **White Jr., K.P.**: "*Advances in the Theory and Practice of Scheduling*", in: Advances in Industrial Systems, Control and Dynamic Systems, no. 37, pp. 115-158, C.T. Leondes, Ed. San Diego, CA: Academic, 1990.

[WIDM'87a] **Widmer, M.**: "*Evaluation des performances d'un atelier flexible en tenant compte des pannes et des maintenances*", RAIRO/Recherche Operationelle no. 21, pp. 137-151, 1987.

[WIDM'87b] **Widmer, M., Hertz, A.**: "*A New Heuristic Method for the Flow Shop Sequencing Problem*", European Journal of Operational Research, vol. 41, pp. 186-193, 1989.

[WIDM'91] **Widmer, M.**: "*Modeles mathematiques pour une gestion efficace des ateliers flexibles*", Presses Politechniques et Universitaires Romandes, Paris, 1991.

[WIRT'77] **Wirth, N.**: "*Toward a Discipline of Real-Time Programming,* Communications ACM, no. 20/8, pp.577-583, Aug., 1977.

[WIRT'82] **Wirth, N.**: "*Programming in MODULA-2*", Springer-Verlag, Berlin, Heidelberg, New York, 1982.

[YOUN'82] **Young, St.**: *"Real-Time Languages: Design and Development"*, Ellis Horwood Ltd., Chichester, 1982.

[ZERH'90a] **Zerhouni, H., Alla, H.**: "*Dynamic Analysis of Manufacturing Systems Using Continuous Petri Nets*", IEEE International Conference on Robotics and Automation, Cincinnati, May, 1990.

[ZERH'90b] **Zerhouni, H., Alla, H.**: "*Le RdP continus: un outil pour l'analyse dynamique des systemes de production*", Colloque International Productique & Integrations, CIM90, Bordeaux, Juin, 1990.

[ZERH'92] **Zerhouni, H., Alla, H.**: "*Sur l'analyse des lignes de fabrication par reseaux de Petri continus",* RAIRO/APII, vol. 26, pp. 253-276, 1992.

[****'80] ***: "*The Ethernet: a Local Area Network Data Link Layer and Physical Layer Specification*", DEC, Sept., 1980.