INSTITUTUL POLITEHNIC "TRAIAN VUIA"
T I M I S O A R A
FACULTATEA DE ELECTROTEHNICA

Ing. Creţu Vladimir

SISTEME DE OPERARE TIMP-REAL PENTRU SISTEME DE CALCUL CU MICROPROCESOR

Teză de doctorat

CONDUCATOR STIINTIFIC

Prof.dr.ing. Alexandru Rogojan

Prof.dr.ing. Mircea Petrescu

1 9 8 4

Cuprins

P R E F A T A

Lucrarea de faţă se înscrie în preocupările actuale de proliferare a sistemelor de calcul bazate pe microprocesoare, de realizare a unor componente software sigure şi eficiente cu un efort redus de programare. Ea reprezintă continuarea activităţii depuse de autor pe parcursul a peste opt ani la Institutul de Tehnică de Calcul Timişoara, la Centrul de Calcul al I.P.T. şi la Catedra de Automatică şi calculatoare a Facultăţii de Electrotehnică, în domeniul sistemelor de operare timp-real. Pe cuprinsul acestei lucrări, autorul a abordat atît aspectele teoretice care definesc conceptul de timp-real cît şi aspectele practice legate de concepţia, proiectarea şi implementarea unor sisteme de operare timp-real pe sisteme de calcul cu microprocesor.

Pe toată durata elaborării tezei, am beneficiat de îndrumarea atentă, exigentă şi competentă a regretatului prof.dr.ing. Alexandru Rogojan, care cu înţelegere şi răbdare mi-a conturat domeniul, jalonîndu-mi activitatea. Acest lucru are o semnificaţie cu atît mai profundă întrucît profesorul Al.Rogojan şi-a adus o contribuţie esenţială la formarea personalităţii mele în calitate de dascăl, conducător de colectiv de muncă şi conducător de doctorat. Puternica sa personalitate, contribuţiile de cea mai înaltă valoare pe care şi le-a adus la dezvoltarea tehnicii româneşti de calcul, fac din profesorul Al.Rogojan o figură de neuitat pentru multe generaţii de ingineri, specialişti în calculatoare electronice Doresc ca aceste rînduri să fie considerate drept omagiu adus memoriei Domniei-Sale.

Aduc deosebite mulţumiri tovarăşului prof.dr.ing.M.Petrescu, care într-o situaţie de excepţie, a acceptat să supervizeze finalizarea tezei. Analiza atentă a materialului tezei, observaţiile şi recomandările formulate au fost elemente care au contribuit într-o manieră decisivă la etapa finală de elaborare, motiv pentru care îmi exprim întreaga stimă şi consideraţie faţă de persoana Domniei-Sale, împreună cu cele mai sincere mulţumiri.

Doresc să mulţumesc bunilor mei colegi şi prieteni Dorina şi Emil Petriu, pentru sprijinul constant, pentru discuţiile şi observaţiile care le-au formulat pe toată perioada elaborării tezei, pentru baza materială şi bibliografică pe care mi-au pus-o la dispoziţie.

Mulţumesc în mod deosebit prietenului Voicu Groza pentru maniera excelentă în care am colaborat în ceea ce priveşte partea de echipamente şi pentru întregul efort pe care l-a depus în vederea realizării şi testării sistemelor de calcul utilizate în experimentarea sistemelor de operare.

Mulţumesc de asemenea colegului şi prietenului Ioan Jurca pentru analiza atentă a materialului tezei, pentru observaţiile formulate, pentru sprijinul bibliografic şi pentru încurajările susţinute pe care mi le-a adresat.

Tin să mulţumesc colegilor Miranda şi Ionel Naforniţă pentru sprijinul moral pe care mi l-au acordat pe întreg parcursul elaborării tezei.

Datorez de asemenea mulţumiri colectivului Centrului de Calcul al I.P.T., în mod deosebit bunului meu prieten Alexandru Furtunescu pentru discuţiile purtate şi pentru sprijinul efectiv pe care mi l-a acordat din punct de vedere tehnic şi colegului Viorel Coifan, pentru contribuţia pe care şi-a adus-o la buna funcţionare a sistemelor de calcul.

Mulţumesc în final, tuturor acelora care într-o manieră sau alta mi-au permis să-mi finalizez în bune condiţiuni lucrarea, pentru înţelegerea şi îndemnurile constante cu care m-au încurajat.

Autorul,

1. Introducere

1.1. Microprocesorul, promotor al dezvoltării societăţii

Perioada istorică pe care o trăim este marcată de evenimente cu semnificaţie majoră în dezvoltarea actuală şi viitoare a umanităţii. Un astfel de eveniment îl constituie apariţia microprocesorului culme a inventivităţii şi geniului uman.

Potenţialul tehnic şi economic pe care îl reprezintă utilizarea microprocesoarelor (μP) în cele mai diferite domenii ale activităţii umane, este deosebit de important, ceea ce a determinat un interes sporit din partea industriei electronice moderne faţă de acest produs remarcabil al gîndirii umane. Acest interes s-a concretizat prin apariţia începînd din 1971, într-o succesiune din ce în ce mai accelerată a unor familii de μP cu caracteristici din ce în ce mai perfecţionate. Această dezvoltare spectaculoasă a fost favorizată în primul rînd de evoluţia explozivă a tehnologiilor de realizare a circuitelor integrate şi în al doilea rînd de progresele remarcabile realizate în concepţia şi proiectarea arhitecturilor hardware ale μ calculatoarelor.

Dimensiunile acestui eveniment, implicaţiile sale profunde în dezvoltarea social economică a ţării, au fost sesizate în întreaga lor complexitate de către conducerea superioară de partid şi de stat a ţării noastre. Astfel, în Raportul prezentat Congresului al XII-lea al partidului, tovarăşul Nicolae Ceauşescu secretarul general al partidului, preşedintele RSR, evidenţia principalele căi de modernizare a producţiei industriale făcînd referiri directe al automatizarea şi cibernetizarea proceselor de producţie, la dezvoltarea tehnicii de calcul, la utilizarea microprocesoarelor în economie.

"Productivitatea muncii în industrie va trebui să crească în cincinalul viitor într-un ritm mediu anual de 7 - 7,5%, contribuind cu circa 80% la realizarea sporului de producţie. Un rol însemnat vor avea în acest scop extinderea mecanizării şi automatizării, realizarea de linii complet automatizate, îndeosebi în turnătorii, forje, secţii de tratamente termice şi alte sectoare cu muncă grea, folosirea maşinilor agregat multifuncţionale, a roboţilor industriali şi microprocesoarelor ". [Ce 79] .

Pornind de la aceste indicaţii de excepţională valoare, în ţara noastră principalele documente programatice elaborate cu începere de la Congresul al XII-lea, conţin opţiunea fermă de a trece la utilizarea largă în economie a microprocesoarelor, la nivelul posibilităţilor oferite de acestea în scopul creşterii productivităţii muncii. Astfel, în "Programul privind îmbunătăţirea nivelului tehnic şi calitativ al produselor, reducerea consumurilor de materii prime, de combustibili şi energie şi valorificarea superioară a materiilor prime şi materialelor în perioada 1983-1985 şi pînă în 1990" elaborat cu prilejul plenarei CC al PCR din 14-15 noiembrie 1983 se precizează: "In producţia de componente electronice, accentul trebuie pus în toată această perioadă pe organizarea şi dezvoltarea producţiei de microelectronică, pe asimilarea de tehnologii de vîrf, care să permită trecerea la fabricaţia de microprocesoare" [** 83] .

Unul din domeniile în care sistemele de calcul bazate pe µP şi-au găsit un teren fertil de dezvoltare îl constituie aplicaţiile timp-real (TR). In acest sens au fost obţinute rezultate remarcabile în conducerea proceselor industriale, roboţilor şi manipulatorilor, în telecomunicaţii, aparatură de măsură, achiziţia datelor, în oferta bunurilor de larg consum, etc.

In paralel însă cu proliferarea sistemelor de calcul cu µP creşte rolul şi importanţa sistemelor de programe (software), remarcîndu-se o amplă preocupare pentru dezvoltarea acestui sector de activitate. Cu toate eforturile depuse însă în această direcţie se constată că, în ansamblu, costul programelor a scăzut de 3-4 ori în ultimii 20 de ani, în timp ce în aceeaşi perioadă, costul echipamentelor a scăzut în medie de 100 de ori, în principal prin scăderea vertiginoasă a costului componentelor electronice [Ba 83] .

In acest context general, prezenta lucrare îşi propune să abordeze domeniul sistemelor de operare timp-real destinate unor sisteme de calcul bazate pe microprocesoare.

1.2. Stadiul actual al dezvoltării sistemelor de operare (SØ) şi al sistemelor de operare timp-real (SØTR) pentru sisteme de calcul cu µP

O analiză succintă al stadiului actual al dezvoltării sistemelor de operare (SØ) pentru sisteme de calcul cu µP, pune în evidenţă faptul că firmele producătoare de software oferă o gamă largă de produse situate pe diferite niveluri de complexitate, începînd cu siste-

me simple de tip monitor şi terminînd cu sisteme sofisticate destinate unor sisteme multimicroprocesor (m/μP).

Dintre sistemele simple destinate /μP din familia 8080 (85) se remarcă sistemul IMSAI [IM] destinat testării şi implementării de programe de aplicaţii. El conţine un editor de teste, un asamblor, un monitor şi module simple pentru evidenţa fişierelor, toate ocupînd un volum de aproximativ 4 Kocteţi.

O etapă superioară în acest domeniu o reprezintă apariţia sistemului STARPLEX [Pe 81] , realizat de firma National Semiconductor pentru un microsistem bazat pe INTEL 8080, destinat asistării utilizatorului în testarea şi depanarea programelor. Este un sistem structurat pe niveluri, conţinînd facilităţi pentru gestionarea dispozitivelor periferice, încărcarea programelor, editarea de texte, editare de legături, inclusiv compilatoare pentru limbajele FORTRAN şi BASIC.
In acelaşi domeniu de complexitate se încadrează sistemele ISIS II [ISIS] , sistem puternic cu funcţii interactive,destinat prelucrării informaţiilor la nivelul fişierelor, sistemele CP/M-80 [Ki 79] şi MP/M-80 [Ki 82] care reprezintă produse dezvoltate de către diferite firme pentru /μP pe 8 biţi.Toate acestea sînt sisteme universale, cu grad sporit de generalitate destinate unor aplicaţii curente, nu însă unor aplicaţii timp-real. De regulă aceste sisteme nu conţin facilităţi de gestionare a timpului sau de sincronizare a activităţilor.

Aceste neajunsuri sînt rezolvate de către proiectanţii sistemului REX-80, destinat unor microsisteme construite pe baza /μP Z80 şi INTEL 8080 [Pe 81] . Acest sistem implementează multiprogramarea, utilizînd în acest scop un ceasornic timp-real şi un circuit pentru stabilirea priorităţilor întreruperilor. Este un sistem orientat pe gestionarea taskurilor,care utilizează structuri logice abstracte cum ar fi evenimentele şi canalele, facilitînd abordarea aspectelor legate de comunicarea între procese şi tratarea stimulilor externi. Dezavantajul său rezidă în faptul că este relativ voluminos deoarece îmbină aspecte caracteristice sistemelor universale cu aspecte specifice de timp real.

In literatura de specialitate apar referiri la unele sisteme destinate unor sisteme multiprocesor cum ar fi HYDRA [Wu 74], STARØS [JC 77] , AMØEBA [TM 81] , etc. Aceste referiri sînt însă foarte generale şi au un pronunţat caracter descriptiv.

In ceea ce priveşte realizările româneşti ale domeniului se remarcă sistemul de operare SFDX [** FE] implementat pe microcalculatoare din familia M 18, în diferite variante, similar ca performanţe cu sistemul ISIS II. In ceea ce priveşte SØTR, preocupările sînt mai restrînse; se remarcă totuşi sistemul MTOS [** 80], [** 82], realizat în cadrul I.C.I., sistem multitasking, multiutilizator, destinat unor aplicaţii TR, care permite execuţia concurentă a mai multor activităţi. Dimensiunile sale se situează între 6 şi 24 de Kocteţi în funcţie de configuraţia de generare.

Preocupări în acest domeniu au existat începînd din 1977 şi la Centrul de calcul al Institutului Politehnic, unde un colectiv de cercetători şi cadre didactice, din care face parte şi autorul au obţinut o serie de rezultate în concepţia şi implementarea de sisteme de operare TR pentru mini şi microsisteme de calcul.

Dacă în ceea ce priveşte descrierea unor sisteme de operare universale sau TR, există referiri bogate în literatura de specialitate, aceste referiri sînt mult mai restrînse atunci cînd este vorba de abordarea problemelor legate de proiectarea, realizarea sau implementarea unor astfel de sisteme. Sfera acestor referiri se restrînge şi mai mult cînd este vorba de sisteme dedicate unor aplicaţii TR.

Tratatele de bază ale proiectării şi implementării SØ [CD 73], [Ha 73], [Sh 74], [MD 74], [LD 81] nu ating decît în mod tangenţial aceste aspecte, şi aceasta în contextul unor sisteme mari şi mijlocii, referirile la microsisteme fiind practic inexistente, lucru explicabil şi prin perioada de timp în care au apărut.

Elementele de pornire ale unei abordări sistematice a domeniului timp-real, apar în lucrările lui Wirth [Wi 77a] şi Hoare [Ho 78] , fără ca acestea să abordeze în mod direct domeniul µP. Aceste lucrări definesc specificul unor aplicaţii TR şi

relevă importanţa factorului timp în acest context.

Se remarcă de asemenea şi unele încercări de conturare a unui software specific pentru astfel de aplicaţii. Aceste abordări au însă fie un grad prea mare de generalitate [Al 77] [BH 81], fie se limitează la domenii foarte restrînse [RG 79].

In ceea ce priveşte abordarea unor tehnici de proiectare şi realizare a unor SØTR, literatura de specialitate este şi mai restrînsă. Se remarcă totuşi, în acest sens unele lucrări care prezintă implementarea unor sisteme de operare pe sisteme mono/µP [Ta 80], [Ch 80] sau pe sisteme multi /µP [Mu 78b], [No 81a]. Aceste lucrări prezintă mai mult realizări din domeniu, fără a aborda problemele de esenţă ale studiului, proiectării şi implementării unor astfel de sisteme.

In acest context, prezenta lucrare îşi propune să-şi aducă într-o manieră originală contribuţia în abordarea problematicii timpului-real din punctul de vedere al proiectantului de sisteme de operare. Această abordare este restrînsă la sisteme mici mono şi multiprocesor bazate pe /µP, domeniu în care autorul a obţinut şi o serie de rezultate practice.

1.5. Descrierea generală a lucrării

In primul capitol al lucrării se realizează prezentarea cadrului general al lucrării cu referiri la realizările şi rezultatele obţinute pe plan mondial şi naţional în software-ul destinat timpului-real.

Capitolul 2 debutează cu precizarea noţiunii de timp-real şi legat de aceasta introduce conceptul de „ambianţă timp-real", concept definit în mod indirect, prin caracteristicile pe care trebuie să le îndeplinească un sistem de calcul care evoluează într-o astfel de ambianţă. Aceste caracteristici influenţează în mod direct arhitecturile hardware şi software ale sistemelor utilizabile în aplicaţii TR şi implicit activitatea de proiectare a acestora. In acest context, după definirea termenului de „arhitectură" într-o concepţie nouă, se studiază influenţa, cerinţelor ambianţei timp-real asupra arhitecturilor hardware, respectiv software a sistemelor. In cadrul acestui studiu se abordează

modurile de organizare ale sistemelor de calcul bazate pe µP realizîndu-se o clasificare a acestora. Această clasificare stă la baza precizării avantajelor şi limitărilor diferitelor moduri de organizare din perspectiva utilizării lor în domeniul TR. O atenţie deosebită se acordă structurilor multiprocesor în diferite moduri de organizare (sisteme de tip SIMD, MIMD), precum şi manierei de cuplare a µP într-o astfel de structură. Pornind de la concluziile rezultate din analiză, se propune o structură de sistem m µP, asimetrică, local distribuită, cu µP slab cuplate prin intermediul unei memorii comune, ca fiind cea mai potrivită pentru satisfacerea exigenţelor unei ambianţe TR. De asemenea sînt subliniate o serie de concluzii şi limitări introduse de arhitectura hardware, care trebuie să stea în atenţia proiectantului software de STR.

In acelaşi context sînt analizate în mod original şi structurile software. Pornind de la definiţia conceptului de SO universal, se subliniază că într-o ambianţă TR, funcţiile unui astfel de sistem se modifică,(în lucrare precizîndu-se în mod detaliat considerentele ce impun astfel de modificări) şi evoluează spre două entităţi de sine stătătoare - executivul - componentă cu caracter permanent, avînd drept sarcină principală optimizarea utilizării sistemului şi asigurarea performanţelor acestuia, şi „mediul de programare" componentă temporară, din perspectiva aplicaţiei, utilizată în etapa de creare,testare şi implementare.

In continuare se precizează funcţiile concrete care revin unui executiv timp-real şi unui „mediu de programare TR", elemente fundamentale ale proiectării unui SOTR.

Dacă marea majoritate a sistemelor de calcul sînt definite în momentul în care se precizează arhitecturile lor hard şi soft, pentru sistemele de timp-real, definirea completă se realizează numai în momentul în care se precizează şi programele aplicaţiei, respectiv procesele (activităţile, taskurile) care o implementează. Pornind de la această observaţie se defineşte într-o manieră nouă conceptul de proces, precizîndu-se stările prin care pot trece aceste procese din punct de vedere al sistemului de calcul şi interacţiunile care pot să apară între ele.

In acest context, se conturează o concepţie nouă,originală asupra unei aplicaţii timp-real. Astfel, aceasta este văzută ca o

structură de procese aplicative concurente care pentru a îndeplini cerinţele aplicaţiei, interacţionează şi îşi dispută resursele sistemului. Optimizarea acestei activităţi revine executivului TR care face parte integrantă din aplicaţie. „Mediul de pregramare TR" este o componentă aparte care serveşte la crearea şi testarea unor aplicaţii concrete.

Dacă funcţiile unui executiv rezultă direct din cerinţele ambianţei TR, maniera lor de proiectare, realizare şi implementare nu poate rezulta decît dintr-o analiză atentă a obiectivului optimizării, respectiv a sistemului de procese aplicative concurente. Un astfel de scop îşi propune capitolul 3 al lucrării. Partea iniţială a acestui capitol introduce aparatul matematic care va fi utilizat în lucrare. Cu ajutorul său se definesc mai multe noţiuni fundamentale dintre care se amintesc noţiunile de sistem de procese, de succesor, de predecesor, de secvenţă de execuţie. Se precizează de asemenea conceptele de combinaţie paralelă a sistemelor de procese, de intersecţie a acestora, de şir de procese, etc.

Cu ajutorul acestor noţiuni, un prim aspect abordat în analiza acestor sisteme îl constituie determinarea, respectiv proprietatea sistemelor de a furniza rezultate care să nu depindă de viteza sau de ordinea de execuţie a proceselor. În acest context, ţinînd cont de configuraţia resurselor sistemului şi de efectul pe care execuţia unui proces îl are asupra stării resurselor, se definesc noţiunile de sistem determinat şi procese neinterferente. În continuare se demonstrează un grup de teoreme care stabilesc condiţiile determinării unor sisteme de procese şi maniera în care se poate construi structura de sistem care permite gradul maxim de paralelism a execuţiei proceselor, fără alterarea proprietăţii de determinare.

Pornind de la aceste aspecte teoretice în continuare se propune un model experimental original pentru studiul determinării sistemelor de procese concurente. Modelul cuprinde o serie de algoritmi materializaţi în forma unor proceduri PASCAL care implementează metodologia studiului determinării abordînd trei aspecte:

1. Stabilirea grafului minim de precedenţă.
2. Studiul propriu-zis al determinării.
3. Determinarea grafului paralel maximal.

Modelul lucrează cu procese neinterpretate, acceptînd la intrare o descriere formalizată a proceselor sistemului şi a relaţiilor de precedenţă dintre ele în forma unui graf, stabileşte forma minimă a acestuia şi închiderea sa tranzitivă, elemente utilizate în studiul propriu-zis al determinării, iar în final furnizează la ieşire structura de graf paralel maximal direct implementabilă în aplicaţie.

Un al doilea aspect abordat în acest capitol se referă la studiul coordonării proceselor, activitate concepută ca o formă de interacţiune directă între procese. Formele concrete în care se manifestă coordonarea sînt excluziunea mutuală şi sincronizarea de condiţie. Pentru aceste forme se analizează două maniere de implementare: autocoordonarea şi coordonarea centralizată.

Analiza mecanismelor de sincronizare utilizate în autocoordonare, scoate în evidenţă o serie de dezavantaje şi ca atare inoportunitatea utilizării ei în sisteme timp-real. Dintre mecanismele destinate a implementa coordonarea centralizată se analizează două: monitorul monolitic şi semafoarele.

Conceptul de monitor monolitic reprezintă mai mult o tehnică de implementare a excluziunii mutuale, larg utilizată în sistemele cu µP. Avantajele pe care le oferă şi utilitatea sa au determinat formularea unei propuneri originale de extindere a acestui concept şi pentru sisteme m µP, slab cuplate prin intermediul unei memorii comune, în forma mecanismului PHS.

Cu toată simplitatea şi avantajele pe care le oferă, „tehnica monitorului monolitic" nu oferă suficiente facilităţi pentru a fi în măsură să asigure necesităţile de coordonare a unor procese într-un sistem TR. Ea rămîne însă ca o tehnică utilă de implementare a excluziunii mutuale în executivele TR, ca element structural al unor mecanisme de sincronizare mai complexe.

Mecanismul care satisface cerinţele de coordonare într-un executiv TR, îl reprezintă semafoarele. In lucrare se propune

semaforul generalizat extins materializat în operaţiile P şi V executate asupra unor structuri de date specifice.. Se prezintă maniera în care semafoarele se pot utiliza în implementarea simplă a excluziunii, a sincronizării de condiţie, a alocării resurselor. Pornind de la avantajele multiple ale acestui mecanism de coordonare, în lucrare se prezintă o extindere originală a conceptului de semafor pentru sisteme m/µP, în forma semafoarelor numite de autor speciale. Acest tip de semafoare prezintă o serie de particularităţi generate în principal de amplasarea lor în memoria comună a sistemului.

Activitatea complexă de sincronizare a interacţiunii proceselor într-o ambianţă TR, necesită alături de mecanismele de coordonare şi mecanisme de cooperare adecvate. In subcapitolul următor al lucrării se analizează cooperarea prin intermediul mesajelor, considerată drept interacţiune de nivel superior. Studiul teoretic al mecanismului de cooperare porneşte de la analiza relaţiei producător/consumator stabilind condiţiile în care această activitate se desfăşoară corect şi stabileşte o variantă teoretică de implementare bazată pe două semafoare şi un tampon de comunicare. Pe baza acestei argumentaţii teoretice se defineşte conceptul de mesaj şi mecanismul de cooperare prin mesaje bazat pe operatorii SEND şi RECEIVE, prezentîndu-se şi o serie de modalităţi de comunicare şi sincronizare realizabile prin intermediul mesajelor.

Pornind de la aceste aspecte se trec în revistă o serie de posibilităţi de implementare şi de utilizare corectă a mecanismului propus. Toate acestea sînt elemente originale care trebuie să stea în atenţia proiectantului de STR în două momente importante: în momentul realizării executivului (implementarea mecanismului) şi în momentul implementării aplicaţiei (utilizarea mecanismului).

In lucrare se prezintă extinderea originală a mecanismului de cooperare propus şi pentru sisteme m/µP în forma mesajelor externe, indicîndu-se metodologia de realizare a acestei extinderi.

Un alt aspect deosebit de important al analizei îl constituie activitatea de planificare în execuţie a proceselor. In începutul paragrafului consacrat acestor aspecte se precizează noţiunea de dispecer (planificator al execuţiei proceselor) şi funcţiile pe care acesta trebuie să le îndeplinească. Elementul fundamental al dispecerului îl reprezintă disciplina de planificare pe care acesta o implementează. In consecinţă se întreprinde o analiză sumară a disciplinelor de planificare cunoscute respectiv a celor bazate pe prioritate, pe termen, a disciplinelor neîntreruptibile şi a celor întreruptibile, scoţîndu-se în evidenţă pentru fiecare caracteristicile, avantajele şi dezavantajele.

SØTR are însă o serie de particularităţi care influienţează în mod direct disciplina de proiectare, particularităţi sesizate în mod original în § 3.5.2. De asemenea se precizează şi considerentele avute în vedere în abordarea planificării în execuţie a proceselor în sisteme cu µP slab cuplate, subliniindu-se faptul că într-un astfel de sistem, activitatea de planificare se restrînge la o sumă de planificări cvasiautonome caracteristice sistemelor mono µP componente, cu precizarea că procesele implicate pot fi afectate în plus de constrîngeri provenind de la procese exterioare sistemului.

Toate aceste observaţii sînt sintetizate într-un model destinat studiului planificării sistemelor închise de procese pe sisteme monoprocesor. Modelul presupune cunoscute structura de graf a sistemului proceselor aplicaţiei precum şi o serie de informaţii apriorice asupra proceselor implicate cum ar fi timpii de execuţie şi timpii de I/E. Astfel de informaţii sînt disponibile în cadrul sistemelor TR.

Modelul implementat în limbajul PASCAL furnizează la ieşire succesiunea proceselor şi timpii totali de planificare corespunzători diferitelor metode de planificare permiţînd studiul comparativ al acestora. Termenul de comparaţie îl reprezintă planificarea optimală, implementată în model în forma unei proceduri recursive. Procedura implementată deşi este rapidă, este suficient de laborioasă pentru a nu putea fi utilizată în planificarea dinamică a proceselor TR. Din acest motiv, în lucrare se prezintă un set de

discipline de planificare ale căror performanţe se apropie de performanţele planificării optime. Aceste discipline pot fi selectate de utilizator în funcţie de specificul aplicaţiei, de calitatea şi cantitatea de informaţii apriorice pe care le poate procura. Paragraful se încheie cu o serie de concluzii care conturează într-o manieră originală abordarea problemei planificării în execuţie a proceselor într-un STR

Un ultim aspect abordat în cadrul acestui capitol se referă la interblocarea proceselor. Fenomen cu urmări nefaste în special pentru STR, interblocarea proceselor este strîns legată de activitatea de alocare a resurselor. Din acest motiv, în lucrare se propune un model generalizat original, pentru studiul stării resurselor unui sistem de procese, implementat în forma unor proceduri PASCAL.

Sistemul de procese luat în considerare este o combinaţie paralelă a unor subsisteme de tip şir de procese.Această limitare nu este restrictivă, deoarece în ultimă instanţă execuţia unui sistem de procese concurente pe un sistem monoprocesor este de fapt un astfel de şir. Combinaţia paralelă a acestor şiruri de procese prefigurează sistemul multiprocesor sau execuţia unor sisteme de procese independente pe un sistem monoprocesor. Modelul permite atît studiul dinamicii necesarului şi utilizatului de resurse, a resurselor produse şi a celor consumate pentru un sistem tip şir de procese pornind de la necesarul de resurse pentru fiecare proces al şirului şi secvenţa de execuţie a acestora precum şi studiul dinamicii unei combinaţii paralele a unor astfel de sisteme El constă dintr-o serie de proceduri care permit simularea evoluţiei în timp a stării resurselor sistemelor considerate. Pornind de la un astfel de model, în continuare se defineşte din punct de vedere teoretic noţiunea de interblocare, indicîndu-se în acelaşi timp şi trei metode generale pentru ocolirea ei respectiv: prevenirea, detecţia şi evitarea, metode care sînt în continuare analizate în mod amănunţit în contextul STR.

Metoda cea mai indicată spre a fi folosită este prevenirea. In lucrare se prezintă mai multe posibilităţi care previn cauzele care provoacă interblocarea şi anume: elibera-

rea tuturor resurselor înainte de solicitarea unei noi resurse, prealocarea cantităţii maxime de resurse necesare unui şir, alocarea resurselor într-o ordine prestabilită, utilizarea sistemelor constrînse la o comunicare ierarhică. In funcţie de natura aplicaţiei pe care o implementează proiectantul de sisteme TR poate alege una sau alta dintre metodele propuse.

In continuare, în lucrare se prezintă un algoritm pentru detecţia interblocărilor. In legătură cu maniera de utilizare a acestui algoritm, într-un sistem TR, se avansează trei propuneri, subliniindu-se însă faptul că utilizarea sa măreşte simţitor regia sistemului de operare recomandîndu-se implementarea sa eventual cu opţiune a executivului sau a mediului de programare asociat.

In cadrul ultimului aspect luat în considerare în acest paragraf şi anume evitarea interblocărilor se prezintă un algoritm cu performanţe ridicate, care necesită drept informaţie apriorică doar cantitatea maximă de resurse necesară unui şir al sistemului. Algoritmul utilizează rutina de detecţie a interblocărilor definită anterior. Dezavantajul unei astfel de metode este că deşi asigură evitarea interblocărilor, măreşte regia sistemului, lucru care de regulă este incompatibil cu STR.

Pornind de la concepţia originală asupra STR conturată în capitolul 2 şi caracteristicile particulare ale sistemelor de procese TR prezentate în capitolul 3, capitolul următor al lucrării îşi propune să contureze şi să dezvolte o metodologie de proiectare, realizare şi implementare a unor astfel de sisteme.

Punctul de pornire al proiectării îl constituie stabilirea tipului de arhitectură de sistem de operare cel mai potrivit domeniului abordat. Din analiza unor caracteristici structurale ale sistemelor, care îşi găsesc corespondenţa perfectă în activitatea de programare, se ajunge la concluzia că cele mai potrivite tipuri de arhitecturi pentru STR sînt cele ierarhice structurate pe niveluri. In lucrare se prezintă structura unei astfel de arhitecturi şi metoda cea mai potrivită pentru realizarea ei. In continuare se trec în revistă o serie de factori care influenţează în mod direct activitatea de concepţie şi proiectare a STR. Este vorba în primul rînd despre necesitatea dezvoltării de software de aplicaţie TR care să asigure un raport convenabil între cost şi eficienţă, deziderat posibil de realizat cu ajutorul unui mediu de programare

corespunzător, utilizînd metoda adaptării generalului la cerinţele particularului. In al doilea rînd este vorba despre impactul conceptelor de obiect şi paradigmă asupra concepţiei, proiectării şi în mod deosebit a implementării SØTR.

In expunerea metodologiei de proiectare se abordează mai multe aspecte. In primul rînd se prezintă considerentele care stau la baza structurării funcţiilor executivului pe niveluri şi subniveluri partajate în executivul inferior (nucleul), executivul superior şi procesele aplicaţiei.

Pornind de la funcţiile pe care trebuie să le îndeplinească un executiv TR, pentru fiecare nivel şi subnivel în parte se face o trecere în revistă a celor mai potrivite modalităţi de implementare, conturîndu-se astfel o metodologie originală de proiectare şi realizare a unor astfel de sisteme. In cadrul acestei activităţi se fac referiri concrete la aspectele teoretice şi la metodele de rezolvare ale unor aspecte cheie ale proiectării, elemente studiate în capitolul anterior, cum ar fi: mecanisme de protecţie, de sincronizare, de cooperare, discipline specifice de planificare, tehnici de alocare a resurselor, de prevenire a interblocărilor, etc.

O atenţie deosebită se acordă în acest context, stabilirii şi implementării structurilor de date aferente executivului. In lucrare sînt prezentate obiectele stabilite a realiza acest deziderat. Pornind de la structura ierarhică a arhitecturii sistemului, aceeaşi disciplină a fost impusă şi acestui domeniu, obiectele fiind la rîndul lor cuprinse într-o structură ierarhică. Această abordare ierarhică, pe lîngă avantajele intrinseci pe care le oferă, se răsfrînge direct asupra activităţii de proiectare simplificînd şi sistematizînd această activitate.

Un al doilea aspect abordat, îl constituie proiectarea arhitecturii „mediului de programare". Se prezintă mai multe posibilităţi de implementare, propunîndu-se soluţia mediului de programare implementat pe un sistem de dezvoltare. Pentru această soluţie, se prezintă implicaţiile pe care le presupune implementarea ei într-o arhitectură TR ierarhică, precum şi maniera de realizare a funcţiilor pe care le presupune. In

ansamblul său „mediul de programare", în structura propusă, reprezintă o rezolvare originală a activităţii de dezvoltare de software de aplicaţie TR.

Un al treilea aspect care se dezvoltă se referă la proiectarea arhitecturii aplicaţiei. In lucrare sînt prezentate etapele în care se elaborează sistemul de procese care implementează aplicaţia, indicînd în mod concret metodele şi tehnicile cele mai potrivite pentru a fi utilizate în implementare, derivate pe de o parte din funcţiile executivului, iar pe de altă parte din studiul efectuat asupra acestor tipuri de sisteme prezentat în capitolul 2. Se prezintă de asemenea şi maniera în care se pot testa, verifica şi efectua măsurători asupra componentelor implementate, precum şi cîteva din activităţile pe care le presupune procesul de adaptare al aplicaţiei.

Un ultim aspect abordat se referă la problemele legate de extinderea arhitecturilor ierarhice la sisteme m/uP slab cuplate. Soluţia propusă, este aceea de sistem de operare distribuit prin replicare. Astfel fiecare sistem de calcul are propriul său executiv local, care este o componentă autonomă a sistemului distribuit. Această structură trebuie însă completată cu nucleul distribuit al sistemului m/uP şi cu componenta distribuită a schimbului de mesaje, componente care rezolvă problemele de interconectare ale /uP din punct de vedere software. In continuare se descrie metodologia de realizare a acestor componente, făcînd apel la generalizările şi extinderile propuse în capitolul 3, cu privire la mecanismele de protecţie, de sincronizare şi de cooperare pentru sisteme m/uP slab cuplate prin intermediul unei memorii comune. De asemenea se precizează şi cîteva maniere posibile de dezvoltare de software de aplicaţie pentru astfel de sisteme.

Pornind de la metodologia originală de proiectare a unor arhitecturi de SØTR prezentată în capitolul 4, următoarele două capitole îşi propun să prezinte cîteva exemple de aplicare a acestei metodologii în cadrul unor sisteme concrete implementate de autor. Această prezentare nu are un caracter exhaustiv ci mai mult orientativ, ea restrîngîndu-se la unele detalii mai semnificative de implementare. Astfel, în capitolul 5 al lucrării este abordată implementarea unor arhitecturi de SØTR pe sisteme de calcul mono /uP din familia I 8080/85. In prima parte a capitolului sînt

descrise trei astfel de sisteme de calcul bazate pe integratele 8080 respectiv 8085, care au constituit suportul implementării: două dintre ele au fost realizate în cadrul catedrelor Facultăţii de electrotehnică, iar cel de-al treilea, sistemul ECARØM-800, este produs de industria românească de tehnică de calcul. Pentru aceste sisteme sînt prezentate arhitecturile hardware la nivel de schemă bloc împreună cu facilităţile de TR. Pe aceste structuri hardware au fost implementate trei SØTR, denumite de autor SFIT85, SISTV respectiv SIECAR, cîte unul pentru fiecare sistem de calcul. Acestea nu diferă principial între ele, deosebirile concretizîndu-se în unele particularităţi introduse de structura hard. Un prim element abordat îl constituie prezentarea unor detalii de implementare a nucleului inferior. Punctul de pornire al implementării îl constituie stabilirea organizării structurilor de date ale nucleului: a tabelei de definiţie a procesului, a tranzacţiei procesorului şi a semaforului. Pasul următor îl reprezintă stabilirea diagramei de tranziţie a stărilor proceselor şi a influenţei pe care funcţiile sistemului o au asupra acesteia. În continuare, în cadrul capitolului se furnizează detalii de realizare a mecanismului de comutare, precizîndu-se o manieră originală de rezolvare a apelurilor sistem încuibate, precum şi precizări privind gestionarea timpului UC. Aici apar unele diferenţe între sisteme, deoarece SFIT85 este implementat pe o structură hardware care nu are ceas intern şi drept urmare divizarea timpului UC în acest sistem se realizează la iniţiativa programatorului printr-o funcţie specială.

În ceea ce priveşte dispecerul, în fiecare dintre sisteme au fost implementate două discipline de planificare: carusel simplu şi prioritate fixă. Se prezintă în continuare detalii de realizare a mecanismului de sincronizare respectiv a funcţiilor P şi V precum şi unele considerente legate de utilizarea conceptului de obiect. În finalul acestui subparagraf este abordată problema întreruperilor precizîndu-se în mod deosebit rolul IT de iniţializare, de timp şi de dispozitiv periferic. Se prezintă în acelaşi context, considerentele care au stat la baza implementării conceptului de monitor monolitic.

Prezentarea detaliilor de implementare a nucleului superior al unui SØTR debutează prin precizarea structurilor de date specifice: mesaj, zonă mesaje, ceas sistem şi tabelă ceas. In continuare sînt trecute în revistă elementele care materializează mecanismul de cooperare bazat pe mesaje. S-a acordat o atenţie deosebită problemelor timpului-real respectiv implementării ceasului de timp-sistem, cunoaşterii timpului curent precum şi implementării mecanismului de lansare întîrziată în execuţie a proceselor. Sînt descrise în acest sens, elementele esenţiale (rutine şi structuri de date) care concură la realizarea acestor obiective.

Dintre aspectele referitoare la executivul superior se prezintă numai detaliile de implementare a proceselor driver care tratează operaţiile I/E, detalii care au un caracter mai pronunţat de generalitate. Aceste procese se împart în procese care citesc informaţii şi procese care scriu informaţii. Pentru fiecare din aceste două categorii de procese se precizează elementele concrete care stau la baza realizării lor.

Un ultim aspect atins în cadrul acestui capitol se referă la implementarea unor „medii de programare". Varianta adoptată este cea corespunzătoare unui „mediu de programare" implementat pe un sistem de dezvoltare. Pentru cele trei sisteme prezentate, aceasta a fost conceput ca un program interactiv realizat în jurul unui interpretor de comenzi. Comenzile generale realizate se referă la crearea şi distrugerea proceselor, la interfaţa cu sistemul de dezvoltare propriu-zis (editor, asamblor, monitor), listarea stării proceselor, iniţializarea sistemului, relansarea execuţiei, contorizarea numărului de treceri prin anumite puncte. In sistemele care dispun de facilităţi legate de gestionarea timpului, au fost introduse funcţii de măsurare care permit înregistrarea timpului sistem la o trecere sau la o succesiune de N treceri (N precizat) prin anumite puncte stabilite ale codului executabil.

Capitolul 6 al lucrării prezintă succint, implementarea unei arhitecturi de SØTR pe un sistem de calcul m/µP. In partea de început a capitolului se realizează o descriere de principiu a arhitecturii hardware a sistemului m/µP realizat prin cuplarea la nivelul unei memorii comune şi a unui sistem de IT, a două microsisteme bazate pe µP 8080 respectiv 8085. Se insistă asupra elementelor specifice şi anume memoria comună caracterizată prin

regimurile de lucru normal şi special, ultimul servind la implementarea excluziunii mutuale a acceselor simultane şi sistemul de întreruperi interprocesoare. Pe această arhitectură hardware a fost proiectată şi realizată o arhitectură software de sistem de operare distribuit prin replicare, pornind de la metodologia precizată în capitolul 4. Acest sistem se prezintă ca o sumă de subsisteme de operare autonome, specifice fiecărui μP component în parte, ale căror procese pot însă comunica şi se pot sincroniza prin intermediul mesajelor externe şi al semafoarelor speciale. Sarcina rezolvării problemelor ridicate de acest tip de coordonare şi cooperare revine nucleului distribuit al sistemului m/μP şi componentei distribuite pentru cooperare. In continuare sînt prezentate cîteva din detaliile de implementare ale celor două componente.

In cazul nucleului distribuit sînt prezentate structurile de date specifice (semaforul special şi cutia poştală), detalii de realizare a mecanismului PHS şi a rutinelor care deservesc semafoarele speciale. Se acordă o atenţie deosebită rutinei V care datorită particularităţilor pe care le prezintă, necesită utilizarea sistemului de IT interprocesoare.

In ceea ce priveşte componenta distribuită pentru cooperare, sarcina acesteia este aceea de a realiza comunicarea prin mesaje externe, motiv pentru care se scot în evidenţă cîteva din particularităţile de implementare a mesajelor şi a funcţiilor specifice care le deservesc.

Capitolul se încheie prin prezentarea unor concluzii rezultate din activitatea practică, concluzii care conduc la o structurare ierarhică a mecanismelor de interacţiune, cu semnificaţii deosebite în sistematizarea şi ordonarea activităţii de implementare.

Cel de-al 7-lea capitol al lucrării prezintă o serie de concluzii rezultate din activitatea desfăşurată subliniind contribuţiile originale ale autorului în ceea ce priveşte problematica abordată. De asemenea sînt prezentate o serie de domenii în care rezultatele acestei cercetări pot fi aplicate, indicîndu-se unele direcţii şi posibilităţi de dezvoltare ulterioară.

2. Sisteme timp real

Scopul acestui capitol este de a fixa cadrul general al problematicii sistemelor timp-real precum şi de a preciza şi defini o serie de noţiuni care vor fi utilizate pe parcursul acestei lucrări.

2.1. Timp-real. Ambianţă timp-real

Dezvoltarea fără precedent a tehnologiilor de integrare a determinat proliferarea vertiginoasă a sistemelor de calcul, în mod deosebit a celor bazate pe microprocesoare, dezvoltare stimulată de scăderea continuă a preţului lor de cost. Aceasta a determinat extinderea ariei lor de utilizare prin abordarea unor noi domenii de aplicaţie cum ar fi: conducerea proceselor industriale, robotică, automatizări, telefonie, comunicaţii, achiziţia datelor, aeronautică, etc. In cadrul unor astfel de aplicaţii specializate, sistemele de calcul sînt nemijlocit conectate prin intermediul terminalelor, traductorilor şi a senzorilor la aplicaţia condusă. De la aceasta, în momente bine precizate de timp, sau în mod aleator, sistemul de calcul primeşte informaţii care trebuiesc prelucrate, iar rezultatele prelucrării furnizate de îndată ce devin disponibile, sau la cerere. Cel mai important parametru al unor astfel de sisteme este <u>timpul de răspuns</u> definit de regulă ca fiind intervalul de timp scurs între solicitarea unui serviciu al sistemului şi momentul onorării acestuia [Fr74] . Timpul de răspuns trebuie corelat cu cerinţele externe ale aplicaţiei, solicitările trebuind să fie rezolvate în intervale utile de timp, motiv pentru care aceste sisteme se numesc <u>timp-real</u> (TR).

In consecinţă, trăsăturile esenţiale ale sistemelor de prelucrare TR se referă la stabilirea unei comunicaţii directe între sistem şi aplicaţie şi la încadrarea timpului răspuns într-un interval prestabilit de timp. Acest interval variază în funcţie de specificul aplicaţiei, de caracteristicile sistemului de calcul, de funcţiile implementate şi acoperă intervalul de la cîteva fracţiuni de secundă la cîteva ore [DM77] .

Din punctul de vedere al prezentării rezultatelor prelucrate se pot discerne următoarele situaţii:

a) rezultatele prelucrate trebuiesc prezentate la momente bine precizate de timp (perioade de eşantionare, intervale de control) [Ca73] ;

b) rezultatul prelucrării trebuie prezentat în interiorul unui interval de timp (timp-real) [Cr80] .

Aplicaţiile timp-real, utilizate pe scară din ce în ce mai largă au determinat mutaţii profunde în arhitectura, performanţele şi tehnicile de programare ale sistemelor care le implementează. S-a conturat cu tot mai multă claritate termenul de ambianţă timp-real, definit în mod indirect prin caracteristicile pe care trebuie să le îndeplinească un sistem de calcul evoluînd într-o astfel de ambianţă.

Un sistem TR coordonează în general un anumit număr de activităţi simultane, afectate de constrîngeri de timp, asigurînd continuitatea, siguranţa şi performanţa funcţionării lor. Din punctul de vedere al unui astfel de sistem, ambianţa TR poate fi caracterizată astfel [Ha78] , [He83] :

1. Intr-o ambianţă TR, un sistem de calcul realizează funcţii avînd un caracter dedicat unei singure aplicaţii, sau unui grup restrîns de aplicaţii, opus caracterului general al unor sisteme universale.

2. Ambianţa TR presupune o largă varietate de echipamente periferice ca şi instrumente de măsură, motoare, valve, sisteme cu reacţie, filtre. Aceste dispozitive utilizează informaţii dintr-un spectru larg de forme analogice şi numerice şi o gamă largă de viteze.

3. In cadrul ambianţei TR, evenimentele se derulează cu mare viteză şi au un grad înalt de paralelism.

4. In ambianţa TR sistemul trebuie să răspundă unei largi varietăţi de semnale nedeterministe. Acestea sosesc în mod asincron (nedeterminat) şi trebuiesc rezolvate în anumite limite de timp, altfel informaţiile pe care le deţin se pot pierde sau îşi pot pierde semnificaţia [Wi77a] .

5. O ambianţă TR, datorită caracterului său dedicat, presupune un sistem de calcul cu o configuraţie fixă de procesoare şi echipamente periferice conţinînd un număr fix de procese (activităţi, taskuri).

6. Activitatea într-o ambianţă TR este continuă, procesele reluîndu-şi în mod ciclic activitatea în funcţie de solicitări, atîta timp cît sistemul de calcul funcţionează.

7. Multe din activităţile desfăşurate într-o ambianţă timp-real sînt foarte specializate şi spaţial distribuite.

In unele cazuri utilizarea unui singur procesor pentru comanda întregului sistem este improprie motiv pentru care se utilizează sisteme multiprocesor.

2.2. Moduri de organizare ale sistemelor de calcul bazate pe microprocesoare

In cadrul acestui paragraf, după definirea conceptului de arhitectură de sistem de calcul, vor fi abordate cîteva aspecte legate de modul de organizare al sistemelor de calcul bazate pe µP din punctul de vedere al arhitecturii lor hardware precum şi oportunităţen utilizării lor într-o ambianţă timp real. Se precizează că scopul acestei analize nu este acela de a prezenta în mod exhaustiv aspectele propuse ci de a scoate în evidenţă acele elemente esenţiale care prezintă importanţă din punctul de vedere al proiectantului software de sisteme TR.

2.2.1. Arhitectura sistemelor de calcul

Noţiunea de „arhitectură de sistem de calcul" are în interpretările actuale, un înţeles mai larg, motiv pentru care se vor face cîteva precizări în acest sens.

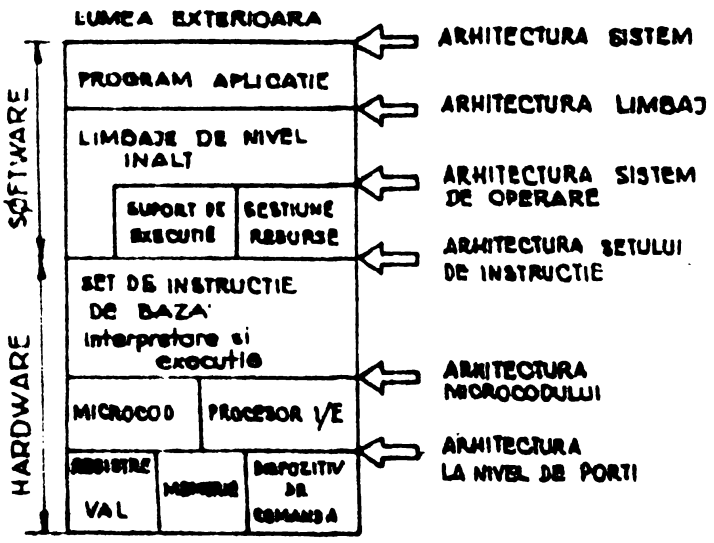


Fig.2.2.1. Arhitectura sistemelor de calcul.

In figura 2.2.1 se prezintă structura de principiu a unui sistem convenţional de calcul [***81]. Sistemul are o structură ierarhică care implementează la nivelurile de bază operaţiile simple (elementare), iar la nivelurile superioare operaţiile complexe. Fiecare nivel utilizează numai funcţiile nivelurilor aflate dedesubtul său (ierarhie de niveluri), astfel încît pe măsură ce creşte nivelul, creşte şi gradul de complexitate al funcţiilor implementate.

O arhitectură este o graniţă sau o interfaţă între două niveluri funcţionale. Ea poate fi definită ca fiind totalitatea

funcţiilor unui sistem de calcul aflate de o parte a unei interfeţe (graniţe) precizate, aşa cum sînt ele văzute de un utilizator situat de partea cealaltă a interfeţei.

Proiectanţii de sisteme, se situează de regulă pe un anumit nivel al interfeţei arhitecturale şi nu sînt preocupaţi de detaliile sistemului aflat dedesubt. Ei sînt interesaţi doar de funcţiile sistemului privite la nivelul interfeţei pe care se situează. Din acest punct de vedere există mai multe tipuri de arhitecturi:

1. Arhitectura sistemului - care reprezintă interfaţa sistem - lume exterioară.

2. Arhitectura limbajelor de programare - reprezintă interfaţa dintre programele de aplicaţii şi limbajele suport care de programare, presupunînd aplicaţii scrise în astfel de limbaje. Este cea mai utilizată arhitectură, accesibilă cercurilor largi de programatori.

3. Arhitectura sistemului de operare - reprezintă interfaţa dintre limbaj şi funcţiile suportului de execuţie şi ale gestionării resurselor care de obicei sînt realizate de către SØ.

4. Arhitectura setului de instrucţii convenţional reprezintă o interfaţă specială întrucît ea delimitează sistemul hardware de sistemul software. La acest nivel, instrucţiile elementare sînt recunoscute de către maşina hard, decodificate şi executate.

Celelalte două arhitecturi (arhitectura microcodului şi arhitectura la nivel de porţi) prezintă interes numai din punctul de vedere al proiectanţilor hardware de sisteme de calcul. Se precizează că în funcţie de sistemul particular de calcul studiat unele dintre aceste arhitecturi pot să lipsească. Spre exemplu în STR, arhitectura limbajelor de programare nivel înalt este slab reprezentată,de cele mai multe ori inexistentă. De regulă în astfel de sisteme, programele de aplicaţii se implementează în limbaje de asamblare utilizînd însă funcţiile sistemului de operare.

Analizînd sistemele de calcul din punctul de vedere al numărului de instrucţii şi al numărului de date prelucrate simultan se pot distinge mai multe tipuri de arhitecturi hardware de sisteme de calcul (fig.2.2.2.) [Mu78a] .

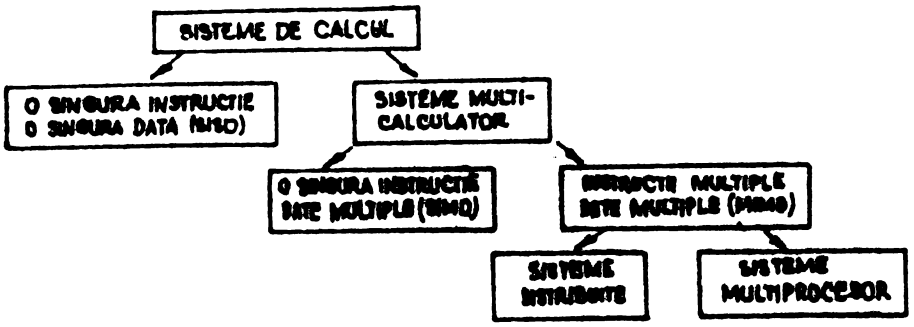


Fig.2.2.2. Clasificarea sistemelor de calcul.

2.2.2. Sisteme de tip SISD

Sistemele de tip SISD (o singură instrucţie, o singură dată) corespund sistemelor monoprocesor convenţionale, clasice de prelucrare. Astfel de sisteme sînt utilizate practic în toate domeniile, inclusiv în ambianţe TR. In cazul sistemelor bazate pe „ul", performanţele realizate nu sînt deosebite datorită vitezei lor relativ reduse de calcul.

Astfel de sisteme au la bază din punct de vedere conceptual calculatorul proiectat de Von Neumann, părintele sistemelor moderne de calcul. In forma sa cea mai simplă „calculatorul Von Neumann" este constituit din trei elemente: unitatea centrală de prelucrare (CPU), memoria şi un canal de legătură prin care se transmite la un moment dat un singur cuvînt între CPU şi memorie (instrucţie sau adresă). Sarcina programului care se execută poate fi rezumată la consultarea şi modificarea conţinutului memoriei, conform unui algoritm precizat, sarcină realizată în întregime prin circulaţia cîte unui singur cuvînt prin canal într-un sens sau altul. Din aceste motive canalul de legătură a fost denumit „strangularea Von Neumann" (bottleneck)[Ba78]. In mod paradoxal, cea mai mare parte a informaţiilor care circulă prin canal, nu sînt date utile propriu-zise ci nume de date sau operaţii şi date necesare calculului acestor nume. Din acest model simplificat al unui sistem clasic de calcul rezultă şi principalele sale limitări:

1. Nivelul relativ scăzut al performanţelor unei astfel de arhitecturi.

2. Caracterul strict secvenţial al programelor care se execută pe un astfel de sistem.

2.2.3. Sisteme multiprocesor

Sistemele multiprocesor s-au dezvoltat ca urmare a limitărilor introduse de sistemele monoprocesor, în primul rînd din considerente de sporire a vitezelor de calcul, de satisfacere a unor constrîngeri de timp impuse de ambianţa TR. Acest lucru se poate realiza prin prelucrarea paralelă a unor activităţi concurente, utilizînd mai multe procesoare, modalitate din ce în ce mai accesibilă pe măsură ce costul microprocesoarelor scade, iar puterea lor efectivă de calcul creşte [CP81]. Se pot distinge două categorii mari de sisteme respectiv de arhitecturi de sistemo multiprocesor: sisteme de tipul „o singură instrucţie - date multiple" (SIMD) şi sisteme de tipul „instrucţii multiple - date multiple" (MIMD)[Th79a,b].

A. Sisteme SIMD. In cadrul acestui tip de sistem un singur procesor de comandă aduce şi decodifică instrucţiile, dintre care unele sînt executate în acest procesor (salturi, salturi condiţionate) altele sînt transmise simultan spre execuţie altor procesoare interconectate. Se cunosc trei subclase de astfel de arhitecturi [We77a] :

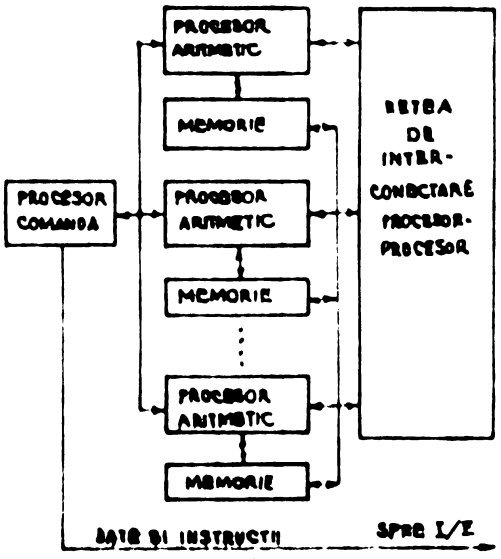


Fig.2.2.3. Schema bloc a unui sistem de calcul de tip SIMD.

- procesoare matriciale (Array procesors) care prelucrează simultan vectori de date;
- ansamble de calcul (Processing Ensembles) în care unitatea centrală este un ansamblu de sisteme şi elemente de prelucrare care comunică între ele transmiţînd mesaje;
- procesoare asociative (Asociative procesors) bazate pe conceptul de memorie asociativă, care operează şi au acces la date prin valoarea acestora şi nu prin adrese.

Dintre cele trei subclase de arhitecturi, procesoarele matriciale sînt singurele care îşi justifică preţul de cost şi aceasta doar în domenii de aplicaţii foarte specifice. Viteza

foarte mare de execuţie se obţine rulînd în mod paralel, pe mai multe procesoare un acelaşi cod (vector de instrucţii) asupra unor date diferite. Schema bloc a unui astfel de sistem apare în figura 2.2.3. Astfel de sisteme sînt utilizate şi în ambianţe TR cu totul particulare.

B. Sisteme MIMD

Arhitecturile setului de instrucţii pentru sistemele de tip MIMD se bazează pe paralelismul execuţiei unor procese independente care operează în mod concurent asupra unor seturi de date, îşi dispută resursele sistemului şi se sincronizează reciproc. Schema bloc a unui astfel de sistem apare în figura 2.2.4. Se cunosc două subclase de astfel de sisteme:

- Sisteme distribuite (Distributed systems) în care mai multe procesoare realizează funcţii specifice ca părţi ale unui singur sistem partiţionat. Procesoarele pot fi local distribuite (spre exemplu într-o fabrică, într-o încăpere, într-un agregat, în articulaţiile unui robot etc.) sau geografic distribuite în cadrul unor reţele de sisteme. Procesele (activităţile) şi funcţiile acestora trebuiesc cunoscute complet dinainte, astfel încît funcţiile sistemului să poată fi divizate între elementele de prelucrare individuale. Aceasta include segmentarea software-ului în părţi specializate pentru fiecare subsistem şi asigurarea unor variabile de control şi dispozitive periferice specifice pentru fiecare procesor. Comunicarea interprocesoare este de obicei restrînsă la transmiterea informaţiilor prin periferice sau linii de comunicaţie serie în cazul celor distribuite geografic sau prin memorii comune şi linii şi linii de comunicaţii în cazul sistemelor local distribuite.

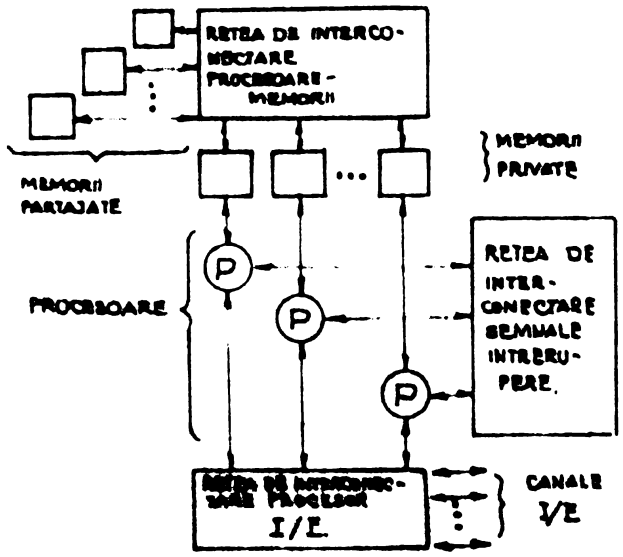


Fig.2.2.4. Schema bloc a unui sistem de tip MIMD.

Din cadrul acestor sisteme, ţinînd cont de caracteristicile unei ambianţe TR, cea mai potrivită arhitectură pentru astfel de aplicaţii este cea corespunzătoare unui sistem distribuit local.

-,Sisteme multiprocesor (Multiprocessor Systems) în care mai multe µP sînt conectate în cadrul unor reţele. Sînt guvernate de un sistem de operare unic care gestionează în mod dinamic procesele procesoarelor. Microprocesoarele pot fi identice şi capabile să execute orice proces (activitate) caz în care sistemul se numeşte simetric sau pot fi preasignate în a îndeplini anumite funcţii, caz în care sistemul se numeşte asimetric. Sistemele simetrice sînt utilizate în general în aplicaţii cu caracter universal, în care solicitările se modifică în mod continuu. Datorită echivalenţei µP-lor, sarcinile pot fi asignate oricăruia dintre ele, iar în caz de defecţiuni sistemul poate fi uşor reconfigurat. Prin contrast, sistemele asimetrice sînt configurate pentru un număr fix, predeterminat de procese, reasignarea acestora este mult mai dificilă, iar domeniul de aplicaţie mult specializat. Aceste restricţii însă aduc mari simplificări în SØ reducînd simţitor rolul acestuia. Pe măsură ce creşte gradul de asimetrie, tot mai multe părţi ale SØ devin tot mai specializate; la limită un astfel de sistem tinde spre unul distribuit local.

Din punctul de vedere al aspectelor luate în discuţie sistemele multiprocesor asimetrice sînt cele mai potrivite pentru a fi utilizate în ambianţe timp-real.

C. Cuplarea procesoarelor în cadrul sistemelor m/µP

Un criteriu important de clasificare şi realizare a sistemelor multiprocesor îl constituie maniera de cuplare a acestora. Din acest punct de vedere se disting sisteme cu procesoare strîns cuplate şi sisteme cu procesoare slab cuplate

Sistemele cu procesoare strîns cuplate operează sub jurisdicţia unei scheme stricte de comandă, implementată prin hard în ansamblul sistemului. Tendinţa de dezvoltare a acestui tip de sisteme este spre „supercalculatoare" a căror viabilitate este discutabilă.

Sistemele cu procesoare slab cuplate comunică în general prin intermediul unor memorii comune. Astfel de sisteme constau din mai multe module de calcul independente ale căror

procesoare sînt capabile să comunice între ele prin intermediul unor variabile partajate aflate în memoria comună. Problemele mai delicate sînt legate de arbitrajul cererilor simultane la memoria comună.

D. <u>Structură propusă</u>. Pornind de la analiza efectuată se poate concluziona că cele mai potrivite structuri de sisteme multiprocesor pentru aplicaţii timp-real se situează în domeniul sistemelor multiprocesor asimetrice, eventual distribuite local, avînd /uP-le slab cuplate prin intermediul unei memorii comune.

O astfel de structură de sistem, potrivită unor aplicaţii timp-real se prouno în figura 2.2.5.

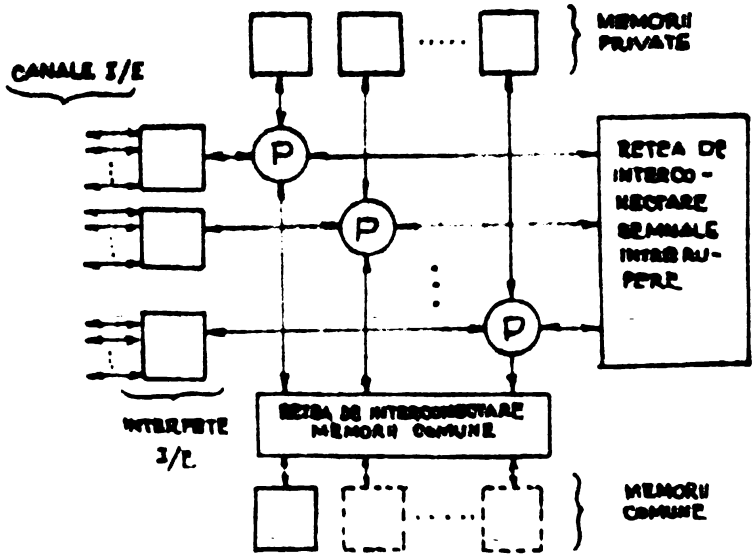


Fig.2.2.5. Structura unui sistem /uP destinat unor aplicaţii TR.

/uP componente sînt conectate prin intermediul unei reţele de semnale de întrerupere şi prin intermediul unei (unor) memorii comune. In rest fiecare /uP reprezintă o unitate de prelucrare autonomă dispunînd de memorie şi dispozitive periferice proprii [PT82a] , [PT86b] , [CG85] ,[CC84] .

2.2.4. <u>Concluzii</u>

Pornind de la cele prezentate în acest paragraf, din punctul de vedere al proiectantului software pot fi desprinse următoarele concluzii:

Referitor la cerinţele ambianţei timp-real:

1. Sistemele de calcul utilizate sînt de regulă specializate pe aplicaţii sau domenii de aplicaţii înrudite.

2. Sistemele au o funcţionare continuă ceea ce presupune o fiabilitate sporită şi măsuri suplimentare de rezolvare a situaţiilor deosebite.

3. Sistemele de calcul sînt astfel configurate încît să asigure timpi de răspuns corespunzători chiar la încărcări de vîrf. De regulă în aceste condiţii eficienţa utilizării echipamentului este mai puţin importantă decît viteza de reacţie la solicitările ambianţei sau siguranţa în funcţionare.

4. Sistemele de calcul TR dispun de regulă de facilităţi multiple de întrerupere.

5. Arhitecturile utilizate sînt cele corespunzătoare sistemelor monoprocesor şi sistemelor multiprocesor de tip asimetric slab cuplate. Acestea din urmă implementează „programarea concurentă" [KJ82] care presupune facilităţi suplimentare, relativ complexe din partea sistemului de operare.

Referitor la utilizarea µP-lor în realizarea sistemelor TR se pot sublinia următoarele:

6. Viteza redusă de calcul a µP-lor.

7. Dotarea relativ redusă a sistemelor cu dispozitive periferice, restrînsă la minimum necesar şi de regulă foarte specializată.

8. Memorie de dimensiuni reduse, strict adaptate necesităţilor aplicaţiei.

2.3. Sisteme de operare pentru aplicaţii timp-real

După ce a fost analizată arhitectura hardware a unui sistem de calcul TR, pe parcursul acestui paragraf se va aborda arhitectura software a unui astfel de sistem. Pentru aceasta se defineşte în prealabil conceptul de sistem de operare universal şi pornind de la acesta, conceptul de sistem de operare TR, precizîndu-se părţile sale componente şi funcţiile acestora.

2.3.1. Conceptul de sistem de operare

Se înţelege prin sistem de operare componenta software a sistemului de calcul care guvernează şi controlează resursele de echipament ca de exemplu unitatea centrală (procesor), memoria principală, memorii secundare, dispozitive periferice şi informaţii. Modulele sistemului de operare rezolvă conflictele de cereri de resurse, optimizează performanţele

şi simplifică utilizarea efectivă a sistemului. Un sistem de calcul este de fapt o interfaţă complexă între utilizator (program utilizator) şi hardware-ul calculatorului [MD74] .

Din cele prezentate, rezultă că un sistem de operare are sarcini deosebite în ceea ce priveşte gestionarea şi optimizarea utilizării resurselor sistemului. In acest sens el trebuie să realizeze următoarele funcţii:

a) Să păstreze evidenţa resurselor.

b) Să implementeze politicile alocării resurselor (cine primeşte resursă, ce primeşte, cînd primeşte, pentru cît timp şi în ce cantitate).

c) Să aloce efectiv resursele.

d) Să elibereze resursele la terminarea utilizării lor.

Resursele luate în considerare sînt:

- resursa unitate centrală (procesor), care este cea mai scumpă şi care trebuie utilizată cît mai eficient,

- resursa memorie formată din memoria principală rapidă (de regulă redusă ca dimensiuni) şi memoria masivă secundară mult mai lentă,

- resursa dispozitive periferice cuprinzînd echipamentele cu care unitatea centrală are schimburi de informaţii. Caracteristica esenţială a acestei resurse este viteza mult mai redusă a transferurilor în raport cu viteza unităţii centrale,

- resursa informaţii formată din fişiere, structuri de date, programe sistem, programe utilizator, procese (activităţi, taskuri) etc. [PC70] .

In general solicitările de resurse sînt păstrate în şiruri de aşteptare specifice, de unde sînt rezolvate în conformitate cu o anumită politică adoptată. In implementarea acestor politici se utilizează atît caracteristicile hardware ale sistemului de calcul cît şi particularităţile specifice ale resurselor tratate. Politicile au drept scop principal rezolvarea conflictelor de solicitare, simultan cu optimizarea utilizării resurselor. Beneficiarii direcţi ai acestor optimizări sînt programele de aplicaţii, care în acest mod îşi reduc timpul de trecere prin sistem. Implementarea politicilor este transparentă din punctul de vedere al utilizatorului, ea fiind însă direct resimţită în îmbunătăţirea performanţelor sistemului.

Pe lîngă optimizarea exploatării resurselor, un sistem de operare are sarcini deosebite în asistarea utilizatorului şi uşurarea activităţii acestuia. Dintre acestea se pot enumera: coordonarea transferurilor de informaţie între unitatea centrală şi dispozitivele periferice, coordonarea execuţiei lucrărilor, transmiterea unor mesaje de eroare cu semnificaţie deosebită pentru activitatea de depanare, realizarea interfeţei nivel logic/nivel fizic în cadrul operaţiilor periferice, facilităţi de dezvoltare şi punere la punct, măsurări de performanţe, etc.

In concluzie prin sistem de operare se înţelege acea parte a sistemului de calcul care asigură gestionarea optimală a resurselor sistemului precum şi o serie de funcţiuni pentru asistarea programelor, cu obiectivul de a asigura performanţe optime sistemului [Ba74] .

2.3.2. Sisteme de operare timp-real

Influenţa puternică pe care ambianţa TR o exercită asupra arhitecturii hardware a sistemelor de calcul TR se resimte şi în arhitectura software respectiv în sistemul de operare al acestui tip de sisteme. Factorii care materializează această influenţă se pot împărţi în două categorii:

A. Factori rezultaţi din utilizarea unor arhitecturi hardware specifice de sisteme de calcul. Natura influenţei acestor factori a fost precizată în § 2.3.1.4.

B. Factori rezultaţi din specificul ambianţei TR. Aceşti factori introduc o serie de caracteristici care particularizează specificul software al unui astfel de sistem. In continuare se amintesc cîteva dintre aceste caracteristici.

1. O primă caracteristică a arhitecturii software a unui sistem timp-real se referă la orientarea spre eveniment (event-driven) a acesteia. Conform ei, programele sînt selectate spre execuţie ca rezultat al recepţionării unor mesaje (semnale) provenind din ambianţa TR, care marchează realizarea unor evenimente. Această caracteristică este opusă orientării spre procese (process-driven), specifică sistemelor obişnuite de calcul, în care evenimentele care au loc în sistem sînt subordonate conţinutului şi structurii programului pe care îl execută [LD81] .

2. Un sistem TR este de fapt un coordonator, care solicită şi/sau acceptă mesaje provenind din ambianţă, le analizează şi lansează în execuţie programe pentru prelucrarea lor. Aceste programe numite procese (§2.4), la rîndul lor pot face referiri la date comune, pot coopera sau se pot coordona.

3. Din raţiuni de timp de răspuns, marea majoritate a proceselor se găsesc în memoria centrală a sistemului într-o formă executabilă. Din acest motiv la astfel de sisteme funcţia de gestiune a memoriei este practic inexistentă sau dacă există trebuie să fie simplă şi rapidă.

4. In sistemele TR ambianţa este aceea care exercită controlul asupra sistemului de calcul şi nu invers. De aceea SØTR trebuie să implementeze funcţii specifice care să permită acest lucru. Astfel procesele trebuie să aibă posibilitatea să elibereze procesorul în mod voluntar sau la sfîrşitul unui interval, precizat de timp, să li se asigure sau să îşi asigure intervale contigue de timp procesor necesare realizării comenzilor, să apeleze în mod facil rutinele sistemului, să manipuleze în mod intim dispozitivele periferice, etc.

5. Astfel de sisteme trebuie să acorde o atenţie deosebită tratării erorilor şi evenimentelor excepţionale, reconfigurării sistemului, redirectării operaţiilor periferice.

6. Caracterul dedicat al aplicaţiilor TR imprimă sistemelor o strînsă interconectare între soft şi hard, între aplicaţie şi sistem. In consecinţă, astfel de sisteme nu au caracterul de generalitate caracteristic sistemelor de operare universale.

7. Pornind de la definiţia sistemelor de operare universale precizată în subparagraful anterior, se poate observa că în cadrul sistemelor de operare TR cele două funcţii esenţiale (§ 2.3.1) îşi modifică în mod substanţial raportul. Astfel funcţia de asistare programatorului este practic inexistentă în exploatarea concretă a unei aplicaţii timp-real, ca atare sistemul se va ocupa exclusiv de gestionarea optimală a resurselor. Din acest motiv astfel de sisteme datorită caracterului lor mai restrîns se mai numesc şi „executive".

Funcţia de asistare a programatorului are însă un caracter preponderent în etapele de proiectare, realizare, implementare, testare şi punere la punct a unei aplicaţii timp-real şi înglobează o serie de facilităţi care în faza de exploatare nu mai

sînt necesare. Din acest motiv această funcţie este preluată de aşa numitul „mediu de programare" sau „sistem de dezvoltare". „Executivul" şi „mediul de programare" sînt două entităţi care nu este necesar să coexiste decît în etapa de testare a programelor. Rezultă deci că un sistem de operare timp-real are două componente:

- „executivul" care este implementat pe sistemul de calcul, conduce aplicaţia şi are un caracter permanent,
- „mediul de programare" utilizat pentru crearea şi testarea aplicaţiei TR. Ca atare, el poate fi implementat, pe un alt sistem de calcul dotat în acest sens, nu are un caracter permanent şi conlucrează cu executivul în cursul perioadei de testare.

Pornind de la obiectivele generale ale unui sistem de operare şi luînd în considerare influenţa ambianţei TR se pot preciza în continuare funcţiile unui „executiv timp-real" şi ale unui „mediu de programare timp-real".

2.3.3. Funcţiile unui Executiv timp-real

Un executiv timp-real coordonează în mod continuu activitatea unui sistem de calcul cu o configuraţie fixă de procesoare şi dispozitive periferice, conţinînd un număr fix de procese (activităţi) concurente. În acest sens el trebuie să îndeplinească în general următoarele funcţii:

(1) - implementarea unui mecanism de comutare, necesar partajării procesorului între diferite activităţi concurente.

(2) - gestionarea timpului unităţii centrale respectiv a diviziării acestuia între diferite activităţi.

(3) - implementarea politicii de planificare în execuţie a activităţilor.

(4) - implementarea unui mecanism de sincronizare.

(5) - trecerea activităţilor dintr-o stare în alta şi menţinerea evidenţei stării pentru fiecare activitate în parte.

(6) - tratarea flexibilă a IT interne (de tip nucleu).

(7) - iniţializarea sistemului.

(8) - gestionarea memoriei.

(9) - implementarea unui mecanism de cooperare şi comunicare interprocese.

(10) - păstrarea evidenței timpului real.

(11) - tratarea evenimentelor dependente de timp.

(12) - implementarea unui mecanism de alocare a resurselor.

(13) - implementarea unor funcții sistem suplimentare (auxiliare).

(14) - tratarea unor întreruperi externe (de tip executiv sau utilizator).

(15) - gestionarea dispozitivelor periferice și tratarea operațiilor I/E.

(16) - localizarea și rezolvarea unor tipuri de erori și avarii.

(17) - reconfigurarea sistemului.

(18) - redirectarea unor operații periferice.

Complexitatea unui astfel de executiv variază în funcție de configurația sistemului de calcul și de cerințele concrete ale aplicației. Pentru sisteme multiprocesor de tipul celui propus în § 2.2.3, mai apar în plus probleme legate de:

- gradul de distribuire și gradul de replicare al executivului sistemului. Prin <u>distribuire</u> se înțelege proprietatea de a implementa în mod singular anumite funcții ale executivului, apelul lor realizîndu-se prin intermediul mecanismelor de comunicare; prin <u>replicare</u> se înțelege multiplicarea în exemplare identice a unor servicii ale executivului și plasarea lor „în replică" pe mai multe procesoare ale sistemului [BL82] .

- gestionarea memoriei comune.

- implementarea mecanismelor de sincronizare, cooperare și comunicare între activități aparținînd unor procesoare diferite [CP81] .

2.3.4. <u>Funcțiile unui „mediu de programare timp-real"</u>

Un „mediu de programare TR" are un caracter interactiv și conține funcțiile necesare punerii la punct a unei aplicații concrete TR. In acest scop trebuie să implementeze următoarele categorii de funcții:

a) Funcția de dezvoltare a proceselor (activităților) aplicației care cuprinde:

(1) - Funcția de editare a unor texte sursă.

(2) - Funcția de asamblare a unor textesursă.

(3) - Funcții de depanare și testare în regim de monitor interactiv a programelor în cod obiect.

(4) - Facilităţi pentru asigurarea portabilităţii software-ului dezvoltat.

b) Funcţii de dezvoltarea aplicaţiei propriu-zise:

(5) - crearea şi distrugerea proceselor

(6) - iniţializarea şi invalidarea unor procese

(7) - listarea stării proceselor

(8) - iniţializarea şi punerea în execuţie a aplicaţiei

(9) - întreruperea şi reluarea execuţiei aplicaţiei.

c) Funcţii de testare şi măsurare:

(10) - facilităţi de contabilizare a numărului de treceri prin diferite puncte ale aplicaţiei,

(11) - facilităţi de determinare a timpului sistem,

(12) - facilităţi de măsurare a unor intervale de timp,

(13) - facilităţi de întîrziere a unor procese.

In mod evident, un mediu de programare TR este strîns legat de executivul pentru care dezvoltă aplicaţii.

2.4. Conceptul de proces. Interacţiuni între procese

Dacă marea majoritate a sistemelor de calcul sînt complet definite în momentul în care se precizează arhitecturile lor hardware şi software, în STR definirea completă se realizează numai în momentul în care se precizează şi programele aplicaţiei respectiv arhitectura sistemului de calcul(§ 2.2.1).

In cadrul acestei arhitecturi, procesele (activităţile taskurile) şi interacţiunile dintre ele joacă un rol fundamental motiv pentru care în acest paragraf se vor defini aceste concepte în sensul aspectelor abordate în cadrul lucrării.

2.4.1. Conceptul de proces

Intr-un program convenţional, procesul de prelucrare al informaţiei care trebuie să conducă la un rezultat dorit, apare în forma unei secvenţe de acţiuni elementare numite instrucţiuni. Aceste se execută pe un procesor, cîte una la un moment dat, fiecare instrucţiune începînd în momentul în care predecesoarea sa s-a terminat. O astfel de manieră de lucru se numeşte secvenţială.

In realitate execuţia unor programe aplicative nu e în mod necesar secvenţială. Ea poate consta din desfăşurarea simultană a mai multor activităţi relativ independente între ele. Fiecare din aceste activităţi, considerată aparte, este de natură secvenţială şi se numeşte proces (task, activitate) [KJ82], [JP81a] , [JP81b] . Noţiunea de proces, mult discutată, a făcut obiectul a numeroase cercetări în ceea ce priveşte definirea şi abordarea sa formală [WI77b] , [Ho78] , [MM79] .

Pe parcursul acestei lucrări, procesul se va defini ca fiind activitatea ce rezultă din execuţia unui program împreună cu datele sale pe un procesor secvenţial.

Din punct de vedere logic, fiecare proces are propriul său procesor şi propriul său program. In realitate după cum se va vedea în continuare, două sau mai multe procese pot partaja un acelaşi program (numit reentrant) sau un acelaşi procesor. Astfel un proces nu este echivalent cu un program şi nici cu un procesor ci cu pereche (procesor, program) în execuţie [Sh74].

Activitatea unui sistem de calcul constă în execuţia simultană a mai multor procese. Astfel de procese se numesc procese concurente. Concurenţa proceselor este caracterizată prin gradul de paralelism al execuţiei acestora. Dacă evoluţia proceselor concurente are loc pe un acelaşi procesor prin execuţia intreţesută a proceselor individuale avem de a face cu un paralelism logic, manieră de lucru cunoscută sub numele de multiprogramare. Dacă fiecare proces are însă propriul său procesor avem de a face cu un paralelism fizic, real. In acest caz, dacă procesoarele partajează o memorie comună maniera de lucru se numeşte multiprelucrare iar dacă procesoarele sînt conectate printr-o reţea de comunicaţii se numeşte prelucrare distribuită. In practică se admit şi metode hibride [AS83] , care îmbină în diferite moduri aceste maniere de lucru. Se mai precizează că execuţia proceselor concurente poate fi supusă la diferite restricţii (sincronizări) şi că procesele concurente pot realiza schimburi de informaţii (cooperări), cu alte cuvinte procesele pot interacţiona.

In practica programării concurente, vitezele relative de execuţie ale proceselor se consideră arbitrare şi necunoscute în sensul că nu se poate aprecia dinainte dacă unele dintre procese se termină înaintea altora. Se presupune totuşi că viteza de

execuţie a fiecărui proces este diferită de zero, ceea ce înseamnă că, dacă unele procese aşteaptă îndeplinirea unor acţiuni executate de alte procese, atunci aşteptarea nu se poate prelungi arbitrar. In particular, în cazul unor procese concurente, nu se acceptă ca anumite procese să blocheze nedefinit execuţia altor procese. De asemenea se precizează că efectul unui proces nu depinde de viteza sa de execuţie.

Programarea timp-real, reprezintă o etapă superioară în cadrul tipurilor de programare deoarece pe lîngă paralelismul execuţiei proceselor caracteristic programării concurente, ia în considerare şi viteza lor de execuţie [Wi77a] . De regulă procesele timp-real trebuie să se execute cu viteze prescrise, întrucît rezultatele furnizate de ele trebuie să se încadreze în limite precise de timp. Aceasta presupune o manieră diferită de tratare a lor din partea sistemului de operare aferent [Cr80] .

2.4.2. Stările proceselor

Referitor la orice proces aflat în evidenţa unui sistem de operare ne definesc următoarele trei stări fundamentale care se exclud reciproc, procesul găsindu-se în orice moment într-una din ele:

(1) Starea activ. Un proces se află în această stare dacă programul său este efectiv executat de către un procesor în momentul considerat. Numărul proceselor active într-un sistem, la un moment dat, este egal cu numărul procesoarelor sistemului.

(2) Starea aşteptare (blocat). Dacă un proces are nevoie de o resursă care este deja alocată, sau care încă nu a fost produsă, sau de informaţii care încă nu sînt disponibile, atunci el trece în starea de aşteptare a unui eveniment extern. Acest eveniment poate fi eliberarea (producerea) unei resurse sau recepţionarea unei informaţii din partea unui alt proces.

(3) Starea pregătit. In această stare se găsesc procesele care ar putea fi active dar nu pot fi lansate din lipsă de procesor.

Sistemul de operare dispune de structuri de date specifice care păstrează informaţii referitoare la starea tuturor proceselor pe care le are în evidenţă. Informaţiile acestea

se actualizează cu ocazia oricărei modificări de stare.

Se precizează că există următoarele tranziţii de stări ale proceselor în curs de execuţie (figura 2.4.1): din starea „activ" un proces trece în starea „aşteptare", dacă execuţia nu poate fi continuată din cauza absenţei unei resurse sau a unei informaţii devenită necesară. Ori care din aceste activităţi presupune un apel al funcţiilor sistemului de operare. Dacă execuţia unui proces se întrerupe, deoarece procesorul în cauză trebuie să execute altceva, atunci la iniţiativa acestei întreruperi procesul trece din starea „activ" în starea „pregătit". Din starea „aşteptare" un proces trece în starea „pregătit", dacă devine disponibilă resursa sau informaţia a cărei absenţă a cauzat aşteptarea. În sfîrşit, ori de cîte ori se eliberează un procesor, se trece un proces din starea „pregătit" (dacă există vreunul) în starea „activ" prin activitatea de planificare.

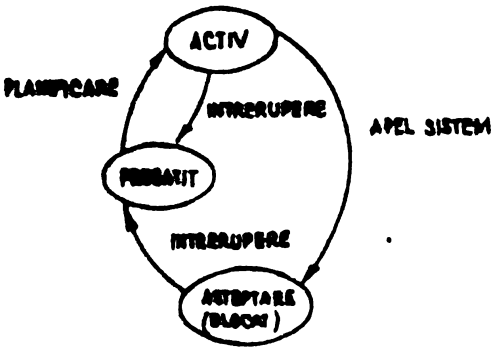


Fig.2.4.1. Tranziţiile de stări ale proceselor.

Se precizează că această structură de stări fundamentale precum şi tranziţiile dintre ele, stau la baza proiectării oricărui sistem de operare. Ele pot fi însă completate cu alte stări şi alte tranziţii în funcţie de particularităţile sistemului de operare care se doreşte a fi realizat.

2.4.3. Interacţiuni între procese

Conceptul de proces are numeroase aplicaţii în sistemele de operare. El a permis spre exemplu izolarea şi precizarea unor sarcini primitive ale SO, a simplificat studiul şi organizarea dinamicii unui SO şi a condus la dezvoltarea unor metodologii avansate de proiectare şi dezvoltare.

Studiul proceselor concurente şi al interacţiunii acestora este un subiect major al cercetărilor care se desfăşoară în domeniul SO.

Pentru a interacţiona, procesele concurente trebuie să comunice şi să se sincronizeze. Orice formă de interacţiune între două sau mai multe procese presupune deci două aspecte:

a) comunicare - prin intermediul căreia procesele pot face schimb de informaţii şi îşi pot influenţa în mod reciproc execuţia;
b) sincronizare - care permite proceselor ce se execută cu viteze

diferite să-şi alinieze de o asemenea manieră execuţiile încît comunicarea dintre ele să poată avea loc în condiţii de deplină siguranţă şi corectitudine.

Implementarea oricărei forme de interacţiune presupune realizarea unui mecanism de comunicare şi a unui mecanism de sincronizare specific.

In vederea sistematizării studiului teoretic şi practic al interacţiunilor interprocese, se propune gruparea lor în două categorii mari: cooperarea şi coordonarea (figura 2.4.2).

Coordonarea este acea formă de interacţiune interprocese în care comunicarea se realizează prin intermediul variabilelor partajate.

Cooperarea este acea formă de interacţiune în care comunicarea se realizează prin intermediul mesajelor.

In cadrul coordonării nu e necesară existenţa unui mecanism special de comunicare, aceasta realizîndu-se în mod direct prin intermediul variabilelor partajate. In schimb coordonarea presupune mecanisme adecvate de sincronizare, acestea avînd un rol preponderent în cadrul acestui tip de interacţiune, motiv pentru care coordonarea este denumită sincronizare.

Cooperarea în schimb presupune un mecanism special de comunicare, cu rol determinat în cadrul acestui tip de interacţiune, secondat de mecanisme specifice de sincronizare care sînt subordonate, uneori chiar cuprinse în mecanismul de comunicare. Din acest motiv cooperarea se confundă cu comunicarea.

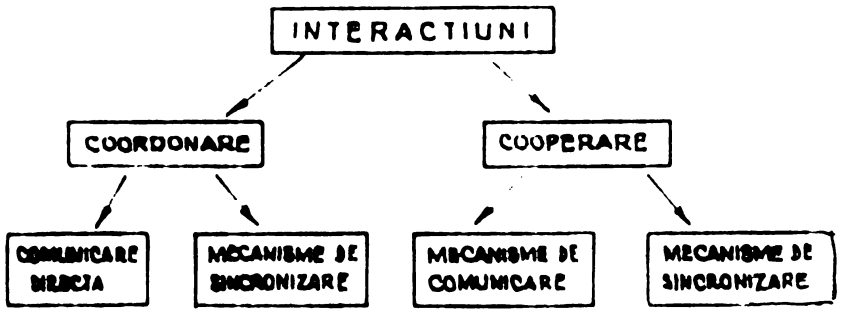


Fig. 2.4.2. Interacţiuni între procese.

In practica realizării SØ, cele două maniere de interacţiune nu se exclud reciproc, dimpotrivă ele sînt îmbinate în mod funcţional şi coexistă în scopul de a materializa funcţiile SØ. Intrepătrunderea lor este cu atît mai evidentă cu cît se constată că dacă manierele de comunicare se păstrează distincte, fiecare cu specificitatea sa, mecanismele de sincronizare sînt comune, proiectantul de sisteme utilizînd de regulă aceleaşi funcţii de sincronizare în ambele maniere de interacţiune.

Unele implementări merg şi mai departe ierarhizînd cele două moduri de interacţiune. In acest caz coordonarea este chiar mecanismul de sincronizare al cooperării (§ 6.4).

Desigur însă că există implementări care exclud complet una sau alta dintre cele două tipuri de interacţiune, funcţie de filozofia de proiectare preconizată.

3. Studiul caracteristicilor sistemelor de procese timp-real

Pornind de la ideea că elementul central al unui sistem de operare timp-real este procesul definit în capitolul anterior, în capitolul de faţă se prezintă analiza unor aspecte legate de determinarea sistemelor de procese, de coordonarea, cooperarea, planificarea şi interblocarea proceselor. Aceste aspecte sînt abordate din dorinţa de a determina acele elemente care influenţează în mod direct activitatea de proiectare şi implementare a unei aplicaţii timp-real în general şi a arhitecturii unui sistem de operare TR în special. Pentru început ne vor introduce cîteva notaţii de bază care permit abordarea matematică a problematicii propuse.

3.1. Sistem de procese

Din punct de vedere matematic, procesul reprezintă o unitate logică de calcul care va fi caracterizată exclusiv din punctul de vedere al comportării sale exterioare: date de intrare pe care le solicită (intrări), rezultate pe care le furnizează (ieşiri), acţiuni şi funcţii pe care le îndeplineşte şi timp de execuţie. Organizarea şi operaţiile interne ale unui proces nu interesează în acest context, procesele considerîndu-se neinterpretate. Drept urmare pe de o parte rezultatele analizei pot fi aplicate oricărui sistem de procese care respectă condiţiile iniţiale impuse, iar pe de altă parte caracteristica de neinterpretare, generalizează concluziile la nivelul oricărui sistem de procese, indiferent de funcţiile lor particulare.

Unui proces P i se asociază două evenimente: iniţierea şi terminarea sa. Se va preciza iniţierea procesului P prin $\overline{P}$ (start) şi terminarea prin $\underline{P}$ (stop). Dacă funcţia t(i) precizează timpul la care se întîmplă evenimentul i, vom presupune că intervalul de timp $t(\underline{P}) - t(\overline{P})$ este diferit de zero şi finit, atîta timp cît există resurse disponibile pentru P.

In sistemul fizic în care se execută un proces P, există un

set S al stărilor referitoare la configuraţia de resurse necesare execuţiei procesului (unitate centrală, memorie, dispozitive periferice, informaţie, etc.) care interesează la un anumit moment. Definirea şi interpretarea setului S va fi strict corelată cu scopul propus: studiul proceselor concurente respectiv al paralelismului execuţiei acestora. Ca stare li se vor asocia evenimentelor de iniţiere şi terminare a unui proces, o tranziţie de stare $s \rightarrow s'(s, s' \in S)$ unde s reprezintă starea din momentul iniţierii procesului iar s' starea din momentul terminării execuţiei procesului.

Fie $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ un set de procese şi fie $\leqslant$ o relaţie de ordonare parţială (precedenţă) în $\mathcal{P}$. Se va defini <u>sistemul de procese</u> ca fiind perechea $\mathcal{S} = (\mathcal{P}, \leqslant)$. Relaţia de ordonare parţială $P \leqslant P'$ precizează că procesul P se termină înainte cu P' să înceapă. In condiţiile în care relaţia de ordonare nu există în mulţimea $\mathcal{P}$ ($\leqslant = \emptyset$) avem de-a a face cu un sistem de procese independente $\mathcal{S} = (\mathcal{P}, \emptyset)$. Prin <u>procese independente</u> se înţeleg acele procese ale unui sistem $\mathcal{S}$ între care nu există relaţia $\leqslant$. Un sistem care conţine numai procese independente se va numi <u>sistem independent</u>.

Un sistem de procese $\mathcal{S}$ poate fi reprezentat prin intermediul unui <u>graf minim de precedenţă</u> care se construieşte astfel:

1. Fiecărui proces din $\mathcal{P}$ i se asociază un nod;
2. Intre două noduri P şi P' apare o săgeată direcţionată de la P la P' dacă şi numai dacă $P \leqslant P'$, şi nu există nici un P'' care să satisfacă relaţia $P \leqslant P'' \leqslant P'$ [r.3.1.1.].

Astfel graful minim de precedenţă poate fi definit ca cel mai mic grup de relaţii a căror închidere tranzitivă este $\leqslant$. (fig.3.1.1.(a)). Se precizează că <u>închiderea tranzitivă</u> presupune toate relaţiile de precedenţă admise între noduri fără a respecta condiţia [r.3.1.1.] (fig. 3.1.1.(b)).

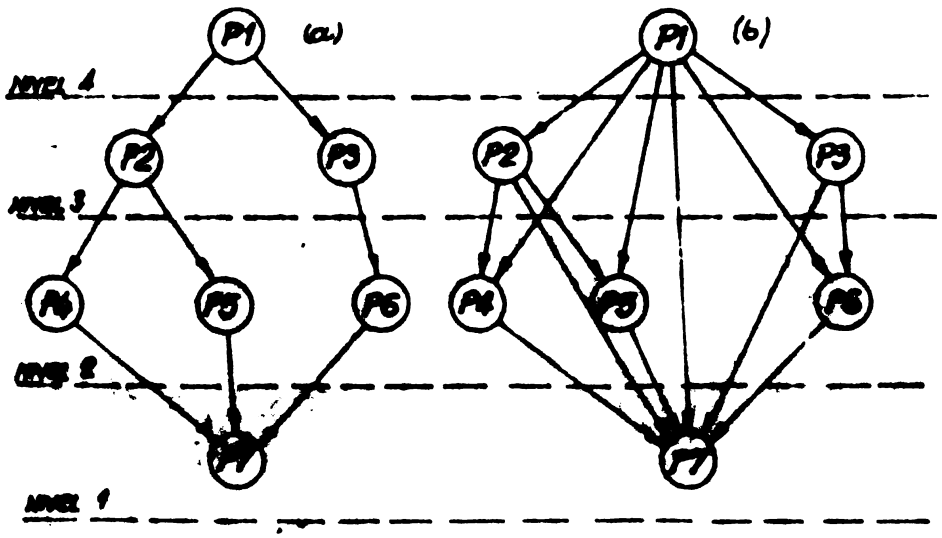


Fig.3.1.1. Graful de precedenţă al unui sistem de procese.

In continuare se vor considera sinonime noţiunile de sistem de procese, graf de precedenţă, sistem şi graf.

O parcurgere a nodurilor $n_1, n_2, \ldots, n$ ale unui graf G, reprezentată în forma $(n_1n_2)(n_2n_3)\ldots(n_{\ell-1}n_\ell)$ se va numi <u>drum</u>. Lungimea acestui drum este ℓ, unde ℓ este numărul de noduri parcurse (cuprinse în drum). p_j se numeşte <u>succesor</u> al lui p_i iar p_i <u>predecesor</u> al lui p_j, dacă într-un drum este îndeplinită condiţia $1 \leq i \leq j \leq \ell$. Dacă $j=i+1$ se utilizează noţiunile de <u>succesor imediat</u> respectiv <u>predecesor imediat</u>. Un proces fără predecesori se numeşte <u>iniţial</u>, iar un proces fără succesori se numeşte <u>terminal</u>. Dacă un proces P nu este nici succesorul nici predecesorul lui P', atunci P şi P' sînt independente. Un proces neterminal este la <u>nivelul ℓ</u>, dacă cel mai lung drum de la el spre procesul terminal este de lungime ℓ. Nivelul procesului terminal este 1. Definirea nivelelor într-o structură de graf, conduce la transformarea acesteia prin partiţionare într-o <u>structură ierarhică</u> pe niveluri conform următorului algoritm:

1° Procesul terminal reprezintă nivelul 1.

2° Nivelul i+1 constă din mulţimea proceselor ai căror succesori imediaţi aleătuiesc nivelul i.

3° Se repetă punctul 2° pînă la epuizarea proceselor structurii

Un exemplu de structură ierarhică apare în figura 3.1.1.(a). P_1 este proces iniţial, iar P_7 proces terminal; P_4, P_5 şi P_7 sînt succesorii lui P_2, iar P_1 şi P_2 sînt predecesorii lui P_4; P_4 şi P_5 sînt succesorii imediaţi ai lui P_2 şi predecesorii imediaţi ai lui P_7; P_4 şi P_5 sînt procese independente. Un exemplu de drum de lungime 4 îl constituie $(n_1n_2)(n_2n_3)(n_3n_4)$ unde $n_1 = P_1$, $n_2 = P_2$, $n_3 = P_4$ şi $n_4 = P_7$. Nivelul procesului P_4 este 2, iar al lui P_1 este 4. Structura ierarhică are 4 niveluri.

O <u>secvenţă de execuţie</u> a unui sistem $\mathcal{S} = (\mathcal{P}, \leq)$ format din n procese, este şirul $a = e_1e_2e_3\ldots e_{2n}$ de iniţieri şi terminări ale proceselor sistemului, care satisface constrîngerile de precedenţă ale lui $\mathcal{S}$. În mod concret:

1 Pentru fiecare $P \in \mathcal{P}$, simbolurile $\overline{P}$ şi $\underline{P}$ apar o singură dată în a

2 Dacă $e_i = \overline{P}$ şi $e_j = \underline{P}$, atunci $i < j$

3 Dacă $e_i = \underline{P}$ şi $e_j = \overline{P'}$, unde $P \leq P'$, atunci $i < j$.

Referitor la figura 3.1.1 se furnizează în continuare cîteva exemple de secvenţe de execuţie:

$\overline{P_1}\ \underline{P_1}\ \overline{P_2}\ \underline{P_2}\ \overline{P_3}\ \underline{P_3}\ \overline{P_4}\ \underline{P_4}\ \overline{P_5}\ \underline{P_5}\ \overline{P_6}\ \underline{P_6}\ \overline{P_7}\ \underline{P_7}$

$\overline{P_1}\ \underline{P_1}\ \overline{P_2}\ \overline{P_3}\ \underline{P_2}\ \overline{P_4}\ \overline{P_5}\ \underline{P_3}\ \underline{P_4}\ \overline{P_6}\ \underline{P_5}\ \underline{P_6}\ \overline{P_7}\ \underline{P_7}$

$\overline{P_1}\ \underline{P_1}\ \overline{P_2}\ \underline{P_2}\ \overline{P_3}\ \overline{P_4}\ \overline{P_5}\ \underline{P_5}\ \underline{P_4}\ \underline{P_3}\ \overline{P_6}\ \underline{P_6}\ \overline{P_7}\ \underline{P_7}$

O <u>secvenţă de execuţie parţială</u> este orice prefix al unei secvenţe de execuţie. In mod evident setul secvenţelor de execuţie corespunzătoare unui sistem $\mathcal{S}$ cuprinde secvenţele de evenimente care respectă constrîngerile sistemului. Un proces P se numeşte <u>activ</u> dacă după o secvenţă de execuţie parţială $a = e_1 e_2 \ldots e_K$, există un $i \leqslant k$ pentru care $e_i = P$ dar $e_j \neq P$ pentru orice $j \leqslant k$.

Dîndu-se un set S al stărilor în sensul celor anterior precizate se va nota cu $p = s_0 s_1 s_2 \ldots s_{2n} \in S$ secvenţa de stări corespunzătoare secvenţei de execuţie $a = e_1 e_2 \ldots e_{2n}$ unde s_0 este starea iniţială, iar $s_{i-1} \rightarrow s_i$ este tranziţia definită pentru evenimentul e_i.

In unele cazuri este convenabil ca un sistem de procese să aibă un singur proces iniţial şi un singur proces terminal, sistemul numindu-se în acest caz <u>închis</u>.

Un sistem închis se poate obţine dintr-unul deschis adăugînd formal la graful sistemului două procese vide P_0 şi P_{n+1}, unde P_0 este predecesorul tuturor proceselor iniţiale iar P_{n+1} succesorul tuturor proceselor terminale.

Presupunînd că $\mathcal{S}_1$ şi $\mathcal{S}_2$ sînt două sisteme închise de procese, graful sistemului $\mathcal{S}_1 . \mathcal{S}_2$, rezultat din concatenarea celor două sisteme se formează unind printr-o săgeată procesul final al lui $\mathcal{S}_1$ cu procesul iniţial al lui $\mathcal{S}_2$. O secvenţă de execuţie a lui $\mathcal{S}_1 . \mathcal{S}_2$ este orice şir $a = a_1 a_2$, unde a_1 este o secvenţă de execuţie a lui $\mathcal{S}_1$ iar a_2 o secvenţă a lui $\mathcal{S}_2$.

Un astfel de sistem închis se poate asocia unui eveniment a cărui realizare declanşează procesul iniţial şi care în continuare activează restul proceselor conform constrîngerilor prezente în graf.

In cadrul domeniului timp-real pot însă exista mai multe evenimente care pot declanşa mai multe procese iniţiale distincte corespunzătoare unor sisteme de procese. Fie $\mathcal{S}_1$ şi $\mathcal{S}_2$ două astfel de sisteme care nu au procese în comun. Noul sistem notat cu $\mathcal{S}_1 // \mathcal{S}_2$, este <u>combinaţia paralelă</u> a lui $\mathcal{S}_1$ şi $\mathcal{S}_2$, rezultat din simpla reuniune a acestora. Sistemul $\mathcal{S}_1 // \mathcal{S}_2$ conţine grafurile lui $\mathcal{S}_1$ şi $\mathcal{S}_2$ ca şi grafuri disjuncte, deoarece acestea nu au procese comune; fiecare proces din $\mathcal{S}_1$ este independent faţă de orice proces din $\mathcal{S}_2$.

Dacă sistemele $\mathcal{S}_1$ şi $\mathcal{S}_2$ au procese comune, se defineşte noţiunea de <u>intersecţie</u> a sistemelor $\mathcal{S}_1 \cap \mathcal{S}_2$, iar procesele comune se numesc <u>partajate</u> (fig.3.1.2) [AB78]. Se va arăta că un astfel de sistem se poate transforma într-unul echivalent care nu conţine însă procese partajate (§ 3.4.5).

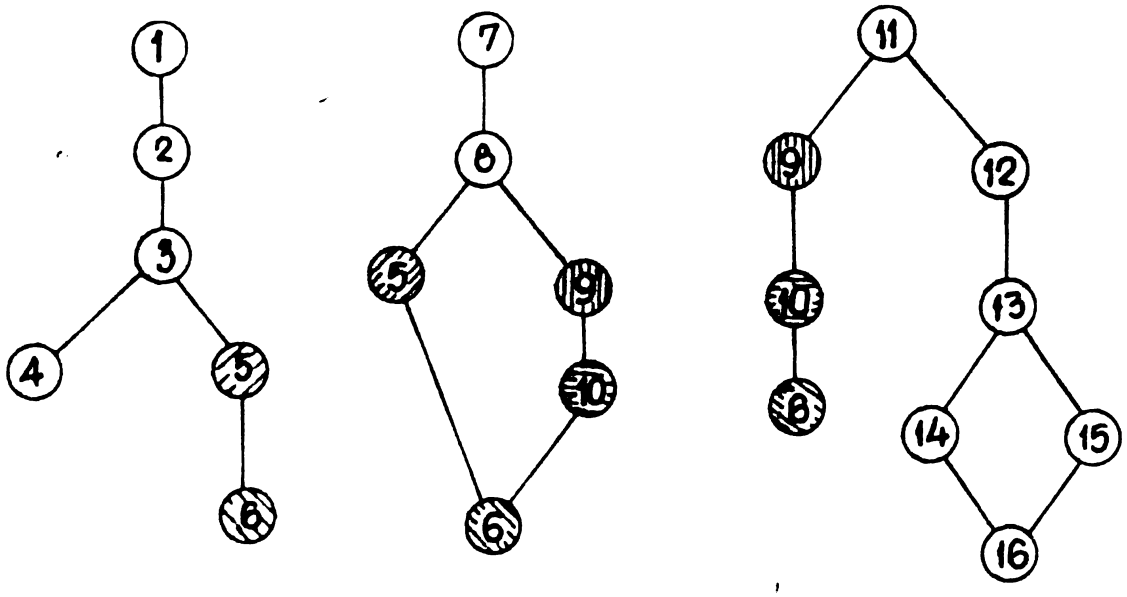


fig.3.1.2. Intersecţia sistemelor de procese.

Sistemele de operare timp-real gestionează adesea sisteme de procese care pot fi reiniţializate în mod arbitrar (la solicitarea unor evenimente externe spre exemplu).

Execuţia ciclică de n ori a unui sistem închis de procese $\mathcal{S}$, poate fi interpretată ca şi o concatenare de n sisteme identice pe care o vom nota $\mathcal{S}^n$. O secvenţă de execuţie a lui $\mathcal{S}^n$ are forma $\alpha = \alpha_1\alpha_2\ldots\alpha_n$ unde α_i este secvenţa de execuţie a iteraţiei i a lui $\mathcal{S}$. Vom numi un sistem <u>ciclic</u> sistemul care are forma $\mathcal{S}^n$ cu $n > 1$, nu sistemul care este o combinaţie paralelă a unor sisteme închise dintre care cel puţin unul este ciclic. O secvenţă de execuţie a unui sistem ciclic se va numi şi ea ciclică.

O importanţă aparte în studiul teoretic şi practic al sistemelor de procese îl constituie sistemul de tip „şir" de procese, care se defineşte astfel.

Fie $\mathcal{S} = (\mathcal{P}, \leqslant)$ un sistem având n procese. Sistemul $\mathcal{S}$ este un sistem de tip <u>"şir" de procese</u>, dacă pentru orice proces $P_i \in \mathcal{P}$ unde $1 \leq i < n$, avem satisfăcută relaţia $P_i \leqslant P_{i+1}$.

Din punctul de vedere al concurenţei, sistemele de procese se situează între două extreme. La o extremă se află sistemele independente care teoretic permit un grad maxim de paralelism (execuţia simultană paralelă a tuturor proceselor sistemului) iar la cealaltă extremă sistemele „şir" de procese care marchează practic absenţa concurenţei, procesele executîndu-se într-o ordine secvenţială impusă de relaţiile de constrîngere.

3.2. Sisteme determinate (Determinarea sistemelor)

Sistemele de operare timp-real sînt confruntate cu problema execuţiei paralele a unor procese care interacţionează în vederea implementării funcţiilor unei aplicaţii concrete. Este foarte important ca sistemul de procese în cauză, să furnizeze rezultate unice indiferent de viteza sau de ordinea de execuţie a proceselor. Un sistem care satisface această cerinţă se numeşte determinat. Sînt frecvente cazurile care pot conduce la nedeterminarea unui sistem format din procese independente: utilizarea în comun a unor resurse a unor locaţii comune de memorie, etc.).

Problema deosebit de importantă a determinării se rezolvă relativ simplu introducînd constrîngeri de precedenţă, între procese iniţial independente. Desigur o serie de constrîngeri rezultă din proiectarea logică a sistemului de procese. Nu la aceste constrîngeri se vor face referiri în continuare, ci la cele care se introduc suplimentar pentru determinarea sistemului.

Scopul acestui paragraf este de a stabili condiţiile necesare şi suficiente pentru determinarea unui sistem de procese şi de a propune o metodă pentru construcţia unui sistem determinat cu un grad maxim de paralelism.

Pentru abordarea problemei determinării se consideră că sistemul de procese se execută pe un sistem fizic reprezentat printr-un set ordonat $\mathcal{R} = (R_1, R_2, \ldots R_m)$ format din m resurse. Fiecare resursă R_i poate lua orice valoare în setul $\mathcal{V}$ de valori al sistemului. O resursă R_i poate reprezenta un periferic, o locaţie de memorie, un buffer, o zonă de informaţii, etc.

Starea sistemului va fi definită prin configuraţia valorilor particulare ale celor m resurse. Spre exemplu fie un sistem cu n procese şi fie $\alpha = e_1 e_2 \ldots e_{2n}$ o secvenţă de execuţie a sistemului iar $p = s_0 s_1 \ldots s_{2n}$ secvenţa de stări corespunzătoare. In aceste condiţii starea care urmează evenimentului $k (1 \leq k \leq 2n)$ va fi $s_k = [R_1(k), R_2(k), \ldots R_m(k)]$. Setul tuturor stărilor poate fi definit ca $S = \mathcal{V}^m$.

In continuare se va preciza efectul pe care execuţia unui proces îl are asupra stării resurselor. Fiecărui proces al sistemului $\mathcal{S}$ îi corespund două submulţimi din spaţiul stărilor. Mulţimea D_p (domeniu) cu d elemente şi mulţimea C_p (codomeniu) cu c elemente. La iniţierea procesului sînt consultate elementele lui D_p iar la terminarea sa sînt modificate valorile elementelor cuprinse în C_p. Astfel fiecare proces poate fi definit ca o funcţie $f_p : \mathcal{V}^d \rightarrow \mathcal{V}^c$.

Un sistem $\mathcal{S}(\mathcal{P},<)$ în care pentru fiecare proces P se cunosc doar D_P și C_P se numește neinterpretat. Interpretarea unui sistem $\mathcal{S}$ se referă la precizarea funcției f_P pentru fiecare proces P al sistemului.

In tranziția $s_K \rightarrow s_{K+1}$ nu se modifică nimic dacă e_{K+1} este un eveniment de inițiere; dacă e_{K+1} e o terminare de proces, atunci s_{K+1} diferă de s_K doar prin valorile R_i ale elementelor cuprinse în codomeniul C_P al lui P. Valorile elementelor afectate sînt determinate de funcția f_P anterior definită. Pentru a vizualiza evoluția stărilor se poate întocmi o matrice a istoriei sistemului pentru o anumită secvență de stări care conține m rînduri corespunzătoare celor m resurse și 2n+1 coloane corespunzătoare celor 2n+1 stări. In matrice apar valorile concrete ale resurselor.

Este însă mai convenabil și mai ușor de lucrat cu secvențele de valori modificate de sistem pentru fiecare resursă în parte. Se va defini secvența de valori $V(R_i,\alpha)$ pentru resursa R_i, ca fiind succesiunea de valori modificate de către terminarea proceselor P care apar în secvența (parțială) de execuție a și în a căror codomeniu apare R_i. Deci $V(R_i,a)=(v_1,v_2,\ldots v_s)$. Considerînd o secvență de execuție $a=e_1e_2\ldots e_{2n}$ și notînd cu ε șirul vid, se obține următoarea secvență de valori pentru resursa R_i: $V(R_i,\varepsilon) = R_i(0)$

$$V(R_i,e_1e_2\ldots e_K) = \begin{cases} V(R_i,e_1e_2\ldots e_{K-1}), R_i(k) \\ \qquad\qquad (\text{dacă } e_K = \underline{P} \text{ și } R_i \in C_P) \\ V(R_i,e_1,e_2,\ldots e_{K-1}) \quad \text{altfel} \end{cases}$$

Succesiunile de valori pentru resurse diferite pot avea lungimi diferite deoarece unele resurse pot apare în domeniile mai multor procese decît altele.

Avînd o secvență de valori $V(R_i,a)=(v_1,v_2,\ldots v_s)$, se va defini valoarea finală $F(R_i,a)=v_s$. In concordanță cu cele anterior expuse, se poate constata ușor că după execuția unei secvențe $a=e_1e_2\ldots e_K$, starea sistemului va fi $s_K=(F(R_1,a), F(R_2,a) \ldots F(R_m,a))$.

3.2.1. Definiții și teoreme

In acest paragraf se vor scoate în evidență cîteva definiții și teoreme referitoare la determinarea sistemelor de procese [CD73].

Definiția D.3.2.1. Un sistem $\mathcal{S}$ de procese este determinat dacă pentru orice stare inițială dată s_o, $V(R_i,a)= V(R_i,a')$ $(1 \leq i \leq m)$, pentru toate secvențele de execuție a, a' ale lui $\mathcal{S}$. Aceasta înseamnă că șirul valorilor unei resurse depinde numai de valoarea sa inițială în s_o.

Este evident că un sistem este nedeterminat dacă codomeniul unor procese intersectează domeniul sau codomeniul altor procese inde-

pendente. Pornind de la constatarea intuitivă că un sistem care conţine procese ale căror codomenii nu intersectează domeniile şi codomeniile proceselor independente, nu poate fi nedeterminat, se formulează următoarea definiţie.

Definiţia D.3.2.2. Procesele P şi P' sînt neinterferente dacă

1° P este succesor sau predecesor al lui P' sau

2° $C_P \cap C_{P'} = C_P \cap D_{P'} = D_P \cap C_{P'} = \emptyset$ [r.3.2.1].

Se spune că $\mathcal{P}=(P_1,P_2,\ldots,P_n)$ conţine procese mutual neinterferente dacă P_i şi P_j sînt neinterferente pentru toţi i şi j ($i \neq j$).

Consecinţa C.3.2.1. Fie $\mathcal{S}$ un sistem de n procese mutual neinterferente şi fie P un proces terminal al lui $\mathcal{S}$. Dacă $a = b_1 \overline{P}\ b_2 \underline{P}\ b_3$ este o secvenţă de execuţie a lui $\mathcal{S}$, atunci şi $a' = b_1 b_2 b_3\ \overline{P}\ \underline{P}$ este o secvenţă de execuţie a lui $\mathcal{S}$ pentru care $V(R_i,a) = V(R_i,a')$ pentru toţi $1 \leq i \leq m$.

Observaţie. Consecinţa demonstrează determinarea pentru tipuri de secvenţe particulare de execuţie a şi a' şi nu pentru toate secvenţele posibile.

Demonstraţie. Deoarece P nu are succesori în $\mathcal{S}$, a' este şi ea o secvenţă de execuţie validă, deoarece satisface condiţiile de precedenţă ale lui $\mathcal{S}$.

Procesul P modifică resursele R_i cuprinse în codomeniul său C_P. Deoarece procesele sînt mutual neinterferente avem $C_P \cap D_{P'} = \emptyset$, deci procesul P nu modifică domeniul nici unui alt proces P'. Rezultă că fiecare proces P' iniţiat în b_3 va avea acelaşi domeniu de valori în a' ca şi în a, deci $V(R_i,a) = V(R_i,a')$ pentru orice resursă care nu aparţine codomeniului C_P al lui P.

Din aceeaşi condiţie de neinterferenţă mutuală rezultă că $C_P \cap D_P = \emptyset$, deci că nici un proces nu modifică domeniul lui P. Înseamnă că pentru orice $R_j \in D_P$, $V(R_j,b_1) = V(R_j,b_1\ b_2\ b_3)$ deoarece nici un proces P' din $b_2 b_3$ nu modifică pe D_P. Deci $V(R_j,b_1) = V(R_j,b_1\ b_2\ b_3)$ pentru toţi $R_j \in D_P$, deci P modifică aceleaşi resurse R_i în C_P atît în a cît şi în a'. Dacă notăm cu v valoarea lui $R_i \in C_P$ modificată de P în a putem scrie:

$V(R_i,a) = V(R_i,b_1\ \overline{P}\ b_2\ \underline{P})$ - din ipoteză

$= (V(R_i,b_1\ \overline{P}\ b_2),v)$ - P modifică pe R_i

$= (V(R_i,b_1),v)$ - nici un P' din b_2 nu modifică C_P

$= (V(R_i,b_1\ b_2\ b_3),v)$ - nici un P' din $b_2 b_3$ nu modifică C_P

$= V(R_i,b_1\ b_2\ b_3\ \overline{P}\ \underline{P})$ - P modifică pe R_i

$= V(R_i,a')$ c.c.t.d.

Se va utiliza această consecinţă pentru a demonstra următoarea teoremă.

Teorema T.3.2.1. Un sistem care conţine procese mutual neinterferente este determinat.

Demonstraţia teoremei se va face prin inducţie. Faptul este evident pentru un sistem cu un singur proces. In continuare se va presupune că afirmaţia este valabilă pentru un sistem $\mathcal{S}$ $(\mathcal{P}, \leq)$ cu n-1 procese şi din aceasta se va deduce valabilitatea ei pentru un sistem cu n procese.

Fie a_1 şi a_2 două secvenţe de execuţie ale sistemului $\mathcal{S}$ cu n procese şi fie P un proces terminal al lui $\mathcal{S}$.

Conform consecinţei 3.2.1 se pot forma secvenţele a_1' şi a_2' astfel

$a_1' = a_1''\ \overline{P}\ \underline{P}$ în care $V(R_i, a_1) = V(R_i, a_1')1 \leq i \leq m$

$a_2' = a_2''\ \overline{P}\ \underline{P}$ în care $V(R_i, a_2) = V(R_i, a_2')1 \leq i \leq m$

Se observă însă că a_1'' şi a_2'' sînt secvenţe de execuţie ale unui sistem cu n-1 procese $\mathcal{S}' = (\mathcal{P} - \{P\}^2, \leq')$ obţinut din $\mathcal{S}$ prin eliminarea lui P şi a tuturor relaţiilor care se refereau la acest proces. Prin ipoteza demonstraţiei s-a presupus că un astfel de sistem este determinat deci $V(R_i, a_1'') = V(R_i, a_2'')$ pentru $1 \leq i \leq m$. Se analizează în continuare situaţia resurselor ce aparţin domeniului D_P şi codomeniului C_P pentru procesul P.

a) Valorile resurselor aparţinînd domeniului lui P sînt aceleaşi şi în a_1' şi în a_2' pentru că $F(R_j, a_1'') = F(R_j, a_2'')$ pentru $R_j \in D_P$. De aici rezultă că P introduce aceleaşi valori v în $R_i \in C_P$ atît pentru a_1' cît şi pentru a_2'.

b) Pentru resursele $R_i \notin C_P$:

$V(R_i, a_1) = V(R_i, a_1')$	consecinţa 3.2.1.
$= V(R_i, a_1'')$	pentru că $R_i \notin C_P$
$= V(R_i, a_2'')$	din ipoteză
$= V(R_i, a_2')$	pentru că $R_i \notin C_P$
$= V(R_i, a_2)$	consecinţa 3.2.1.

c) Pentru resursele $R_i \in C_P$:

$V(R_i, a_1) = V(R_i, a_1')$	consecinţa 3.2.1.
$= (V(R_i, a_1''), v)$	P introduce pe v în R_i
$= (V(R_i, a_2''), v)$	din ipoteză
$= V(R_i, a_2')$	P introduce v în R_i
$= V(R_i, a_2)$	consecinţa 3.2.1.

Teorema T.3.2.2. Fie $\mathcal{S}$ un sistem în care pentru fiecare proces P, se cunosc D_T şi $C_P \neq \emptyset$, dar nu se cunoaşte f_P. Sistemul $\mathcal{S}$ este de-

terminat pentru toate interpretările f_p ale proceselor sale numai dacă aceste procese sînt mutual neinterferente.

Demonstrație. Se presupune că procesele P și P' interferă; rezultă că ele sînt independente și deci există secvențele de execuție:

$$a = b_1 \overline{P}\,\underline{P}\,\overline{P'}\underline{P'}b_2$$
$$a' = b_1\,\overline{P'}\underline{P'}\overline{P}\,\underline{P}\,b_2$$

Se presupune că resursa $R_i \in (C_P \cap C_{P'})$. Se aleg în continuare două interpretări f_p și $f_{p'}$ astfel încît P să introducă valoarea u în R_i iar P' valoarea v unde $u \neq v$.

Atunci:

$$V(R_i, b_1\,\overline{P}\,\underline{P}\,\overline{P'}\,\underline{P'}) = (V(R_i, b_1),\ u,\ v)$$
$$V(R_i, b_1\,\overline{P'}\underline{P'}\overline{P}\,\underline{P}\ ) = (V(R_i, b_1),\ v,\ u)$$

ceea ce contravine determinanantei lui $\mathcal{S}$. Inseamnă că este absolut necesar ca $C_P \cap C_{P'} = \emptyset$.

Se presupune acum că $R_j \in (C_{P'} \cap D_P)$ și fie un $R_i \in C_P$ (deoarece $C_P \neq \emptyset$ din ipoteză). Se poate alege o funcție $f_{p'}$ astfel încît $F(R_j, b_1) \neq F(M_j, b_1\overline{P'}\underline{P'})$, deci P va consulta valori diferite în a și a'. Se alege în continuare f_P astfel încît P introduce u în a și v în a' unde $u \neq v$. In aceste condiții:

$$V(R_i, b_1\overline{P}\,\underline{P}\,\overline{P'}\underline{P'}) = V(R_i, b_1\overline{P}\,\underline{P}) \qquad (C_P \cap C_{P'} = \emptyset)$$
$$= (V(R_i, b_1),\ u)$$
$$V(R_i, b_1\overline{P'}\underline{P'}\overline{P}\,\underline{P}) = V(R_i, b_1\overline{P'}\underline{P'}),v)$$
$$= (V(R_i, b_1),v) \qquad (C_P \cap C_{P'} = \emptyset)$$

Deoarece $u \neq v$ înseamnă că $\mathcal{S}$ nu este determinat. Rezultă că $C_{P'} \cap D_P = \emptyset$ trebuie să fie adevărată. Se observă că dacă $C_P = \emptyset$, nu se ajunge la aceeași contradicție. Prin simetrie se poate demonstra că $C_P \cap D_{P'} = \emptyset$ trebuie de asemenea să fie adevărată pentru ca sistemul $\mathcal{S}$ să fie determinat.

Ca o aplicație a teoremelor T.3.2.1 și T.3.2.2 se enunță o teoremă referitoare la paralelismul maxim al unui sistem de procese în care constrîngerile de precedență sînt introduse numai din necesități de determinare. In prealabil însă se vor face cîteva precizări.

Din definiția D.3.2.1 rezultă că într-un sistem determinat de procese unei stări inițiale date îi corespunde o singură secvență de valori pentru fiecare resursă R_i. Două sisteme avînd același set de procese se numesc echivalente dacă sînt determinate și dacă pentru o stare inițială dată produc aceeași secvență de valori pentru fiecare resursă a sistemului.

Un sistem de procese $\mathcal{S}$ și graful său G este paralel maximal,

dacă $\mathcal{S}$ este determinat şi înlăturarea unui arc (P,P') din G face ca P şi P' să devină interferente. Astfel (P,P') este un arc al grafului paralel maximal dacă

$$(C_P \cap C_{P'}) \cup (C_P \cap D_{P'}) \cup (C_{P'} \cap D_P) \neq \emptyset \qquad [r.3.2.2.]$$

Teorema T.3.2.3. Pornind de la un sistem determinat de procese $\mathcal{S}(\mathcal{P}, \lessdot)$ se construieşte un nou sistem $\mathcal{S}'(\mathcal{P}, \lessdot')$ unde $\lessdot'$ este închiderea tranzitivă a relaţiei

$$X = \left\{ (P,P') \in \lessdot \mid (C_P \cap C_{P'}) \cup (C_P \cap D_{P'}) \cup (C_{P'} \cap D_P) \neq \emptyset \right\}.$$

Sistemul $\mathcal{S}'$ este sistemul paralel maximal unic echivalent cu $\mathcal{S}$.

Demonstraţie. Fie G' graful minim de precedenţă al lui $\mathcal{S}'$ (corespunzător mulţimii X şi deci lui $\lessdot'$). In general $G' \subset X \subset \lessdot'$. Se presupune că se îndepărtează un arc (P,P') din G'. Deoarece nu mai există un alt drum de la P la P' (din definiţia grafului minim de precedenţă [r.3.1.1]) şi deoarece P şi P' satisfac [r.3.2.2] iar $X \supset G'$, rezultă că P şi P' devin interferente. Deci graful G' este paralel maximal.

In continuare se va demonstra determinarea sistemului $\mathcal{S}'$. Deoarece $\mathcal{S}$ este determinat, sînt valabile relaţiile $P \lessdot P'$ sau $P' \lessdot P$ pentru fiecare pereche de procese care satisface [r.3.2.2]. Din definiţia lui X rezultă că avem de asemenea $P \lessdot' P'$ sau $P' \lessdot' P$. Deoarece toate procesele din $\mathcal{S}'$ sînt mutual neinterferente conform teoremei T.3.1.1, $\mathcal{S}'$ este determinat.

Fiecare relaţie de precedenţă din $\mathcal{S}'$ apare şi în $\mathcal{S}$, deci fiecare secvenţă de execuţie din $\mathcal{S}$ este generată de $\mathcal{S}'$. Fie a' o secvenţă de execuţie a lui $\mathcal{S}'$, care nu e generată de $\mathcal{S}$ şi fie a o secvenţă de execuţie comună pentru $\mathcal{S}$ şi $\mathcal{S}'$. Determinarea lui $\mathcal{S}'$ implică $V(R_i, a') = V(R_i, a)$, $1 \leq i \leq m$ pentru o stare iniţială dată. Deoarece are aceleaşi secvenţe de valori ale resurselor ca şi $\mathcal{S}$ rezultă că cele două sisteme sînt echivalente. Mai rămîne de rezolvat problema unicităţii. Se observă că $P \lessdot' P'$ dacă şi numai dacă există un lanţ

$P = P_{i_1} \lessdot P_{i_2} \lessdot \ldots \lessdot P_{i_k} = P' (k > 1)$ în care fiecare pereche de procese $P_{i_j}, P_{i_{j+1}}$ satisface [r.3.2.2]. Deoarece P_i aparţin grafului G a lui $\mathcal{S}$ rezultă că lanţul prezentat este unic şi întrucît această afirmaţie trebuie să fie adevărată pentru orice sistem paralel maximal de procese echivalent cu $\mathcal{S}$, din unicitatea ei rezultă unicitatea lui $\mathcal{S}'$.

3.2.2. Model pentru studiul determinării sistemelor de procese concurente

In acest subparagraf se vor interpreta rezultatele teoretice prezentate înainte şi se va furniza o metodologie de studiu a determinării sistemelor de procese concurente bazată pe un număr de algoritmi

concepuţi de autor.

Metodologia presupune trei aspecte intercorelate:

A. Determinarea grafului minim de precedenţă.

B. Studiul propriu-zis al determinării sistemului.

C. Determinarea grafului paralel maximal.

A. In vederea determinării grafului minim de precedenţă, din condiţiile concrete ale aplicaţiei timp-real se stabilesc procesele, funcţiile acestora şi relaţiile de constrîngere care rezultă direct din condiţiile de proiectare (spre exemplu succesiunea în timp a execuţiei unor procese). In această etapă sistemului $\mathcal{S}$ i se asociază graful G' care materializează procesele şi relaţiile de constrîngere iniţiale dîntre ele, conţinînd cîte un arc pentru fiecare relaţie definită în sistem. Procesele se numerotează în mod arbitrar.

Graful G' al sistemului se prezintă în forma unui set de perechi de valori (P_i, P_j), prezenţa fiecărei perechi precizînd relaţia $P_i < P_j$ între procesele P_i şi $P_j \in \mathcal{S}$. Fiecare pereche de valori materializează un arc al grafului G' în următoarea interpretare: P_i este procesul sursă iar P_j procesul destinaţie al arcului asociat relaţiei (fig.3.2.1.(a)).

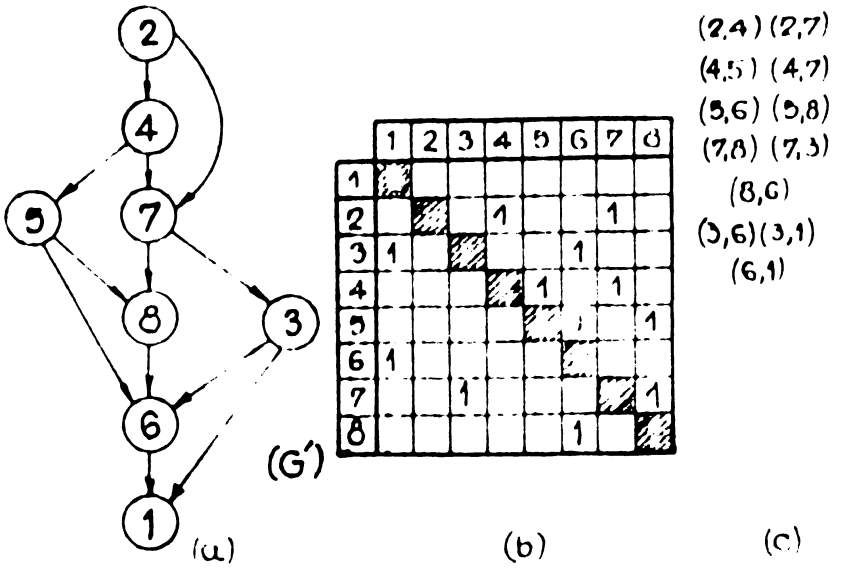


Fig.3.2.1. Sistemul de procese şi graful G' rezultat din condiţii de proiectare.

O primă problemă care poate apare este aceea a sistemelor conţinînd procese partajate. Astfel de sisteme pot fi transformate în sisteme echivalente care nu conţin procese partajate după cum se va arăta în §.3.4.5, astfel încît, în studiul determinării se pot aborda sistemele normale închise de procese.

In continuare se pune problema determinării închiderii tranzitive a relaţiei $\lessdot$ în sistemul $\mathcal{S}$, element deosebit de util în numerotarea proceselor şi în determinarea grafului paralel maximal.

A.3.2.1. <u>Algoritm pentru determinarea închiderii tranzitive a unei relaţii definite pe mulţimea proceselor unui sistem $\mathcal{S}$</u>. Acest algoritm primeşte ca date de intrare un graf de precedenţa pentru un sistem $\mathcal{S}$ $(\mathcal{P}, \lessdot)$ şi furnizează la ieşire închiderea tranzitivă a relaţiei $\lessdot$. In acest scop se identifică toate relaţiile de precedenţă care există între perechile de procese ale sistemului.

Algoritmul presupune următorii paşi:

1. Se alcătuieşte tabloul GRAF care cuprinde toate arcele grafului sistemului $\mathcal{S}$. Spre exemplu pentru graful din fig.3.2.1(a) se obţine următorul tablou: GRAF =(2,4)(2,7)(4,5)(4,7)(5,6)(5,8)(7,8)(7,3)(8,6)(3,6)(3,1)(6,1) (fig.3.2.1(b)(c)).

2. Se aplică procedura GRAFTRANZ (fig.3.2.3). In cadrul acestei proceduri se introduc pe rînd elementele I ale tabloului GRAF în tabelul TRANZ cu indicele M2. Pentru fiecare element introdus se parcurg elementele tabloului TRANZ (indice J), verificînd dacă procesul destinaţie al elementului recent introdus (TRANZ(.M2.)) se regăseşte printre procesele sursă ale celorlalte elemente. In caz afirmativ se adaugă şirului TRANZ (dacă nu există) elementul format din procesul sursă al elementului M2 şi procesul destinaţie al elementului J (procedura CĂUTA 1), deoarece între aceste procese există o relaţie de precedenţă. Inserţia se face cu ajutorul indicelui M1 care tot timpul marchează sfîrşitul tabloului TRANZ. Structura datelor aferente algoritmului apare în figura 3.2.2.

```
CONST   M=50;LUNG=(:10:);
        NRPROCESE=8; NRRESURSE=5;
TYPE  RELATIE=RECORD
                SURSA:INTEGER;
                DEST:INTEGER
              END;
      G=ARRAY(.1..M.) OF RELATIE;
      ELEMENT=SET OF 1..NRRESURSE;
      T1=ARRAY(.1..NRPROCESE.) OF ELEMENT;
VAR   DERIM,GRAF1,GRAF,TRANZ,X,GRAFPAR:G;
      CP,DP:T1;
      COMP:RELATIE;
      M1,M2,M3,N1,N2:INTEGER;
      T:BOOLEAN;
```

Fig.3.2.2. Structura datelor aferente algoritmilor.

Pentru exemplul precizat, în urma execuţiei procedurii GRAFTRANZ se obţine următorul set de relaţii (închiderea tranzitivă): TRANZ =(2,4)(2,5)(2,7)(2,8)(2,3)(2,6)(2,1)(4,5)(4,7)(4,8)(4,6)(4,1)(4,3)(5,8)(5,6)(5,(7,8)(7,3)(7,6)(7,1)(8,6)(8,1)(3,6)(3,1)(6,1)(fig.3.2.4(a),(b).

Pornind de la închiderea tranzitivă a relaţiei $\lessdot$ în sistemul $\mathcal{S}$ obţinută prin aplicarea algoritmului A.3.2.1, se renumerotează procesele în ordinea descrescătoare a numărului de relaţii în care sînt implicate,

realizîndu-se o sortare topologică a acestora. Ca urmare a acestei sortări se obţine următoarea proprietate importantă:dacă P_i şi P_j sînt două procese renumerotate pe baza sortării topologice şi dacă

```
PROCEDURE CAUTA1(COMP:RELATIE;VAR TRANZ:G; VAR T:BOOLEAN);
VAR GASIT:BOOLEAN; K:INTEGER;
BEGIN
     GASIT:=FALSE; K:=1; T:=FALSE;
          REPEAT
               IF COMP=TRANZ(.K.)THEN GASIT:=TRUE;
               K:=K+1
          UNTIL (K>M1) OR(GASIT);
          IF (K>M1)AND (NOT GASIT) THEN
               BEGIN
                  M1:=M1+1;TRANZ(.M1.):=COMP;T:=TRUE
               END;
END;
PROCEDURE GRAFTRANZ(GRAF:G;VAR TRANZ:G);
VAR I,J : INTEGER;
BEGIN
     M1:=1; T:=FALSE;TRANZ(.1.):=GRAF(.1.);
     FOR I:=2 TO N1 DO
     BEGIN
          COMP:=GRAF(.I.); CAUTA1(COMP,TRANZ,T);
          IF T THEN
              BEGIN
                  M2:=M1;
                  WHILE (M2<=M1) DO
                  BEGIN
                       J:=1;
                       WHILE (J<M2) DO
                       BEGIN
                            IF TRANZ(.M2.).DEST=TRANZ(.J.).SURSA THEN
                            BEGIN
                                COMP.SURSA:=TRANZ(.M2.).SURSA;
                                COMP.DEST:=TRANZ(.J.).DEST;
                                CAUTA1(COMP,TRANZ,T);
                            END;
                            J:=J+1
                       END;
                       M2:=M2+1
                  END;
              END;
     END;
END;
```

Fig.3.2.3. Procedura GRAFTRANZ pentru determinarea închiderii tranzitive a relaţiei $\prec$.

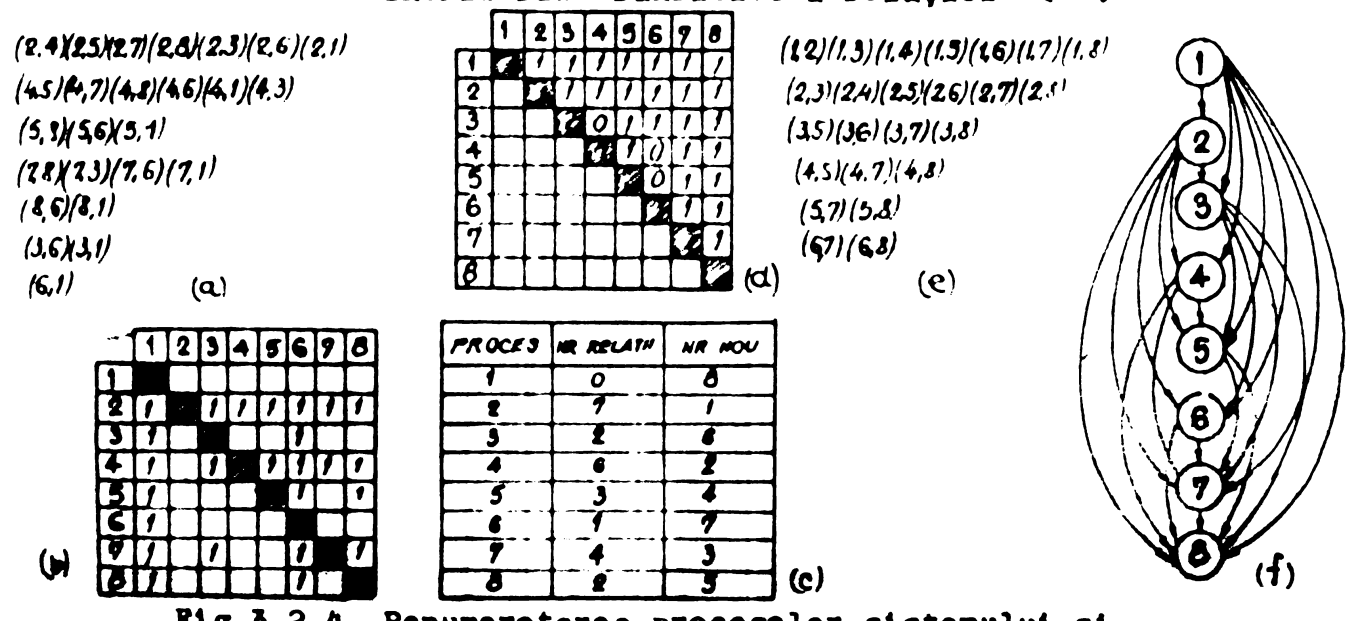


Fig.3.2.4. Renumerotarea proceselor sistemului şi reprezentarea închiderii tranzitive.

perechea $(P_i,P_j) \in G$, aceasta presupune implicit că $i < j$[Kn 74] .

Această proprietate simplifică mult căutările în tabelele de relaţii. Pentru exemplul considerat renumerotarea proceselor sistemului şi reprezentarea închiderii tranzitive a relaţiei $\lessdot$ în sistemul $\mathcal{S}$ apare în fig.3.2.4(c)(d)(e)(f).

Ca urmare a renumerotării proceselor, graful G' al sistemului d vine cel din figura 3.2.6(a). De obicei acest graf are o structură mai complicată întrucît conţine o serie de arcuri redundante. Se precizeaz că un arc (P,P') este redundant, dacă mai există un drum de la P la P' care conţine mai mult de un arc. Pentru a găsi forma implementabilă ce mai simplă a unui sistem de procese a fost conceput următorul algoritm

A.3.2.2. Algoritm pentru determinarea grafului minim de precede ţă al unui sistem de procese.

Acest algoritm presupune cunoscut un set al relaţiilor $\lessdot$ dintr procesele unui sistem $\mathcal{S}(\mathcal{P}, \lessdot)$ (eventual închiderea tranzitivă a acestei relaţii) şi determină graful de precedentă minim al sistemului eliminînd relaţiile redundante [r.3.1.1]. El constă din următorii pa

1. Se alcătuieşte tabelul GPRIM al sistemului $\mathcal{S}$ care cuprinde relaţiile sistemului ordonate în ordinea descrescătoare a procesului sursă. Pentru graful din figura 3.2.6 (a) se obţine următorul tablou: GPRIM = (7,8)(6,7)(6,8)(5,7)(5,8)(4,5)(4,7)(4,8)(3,5)(3,6)(1,2).

2. Se aplică procedura MINGRAF (fig.3.2.5) care pornind de la tabloul GPRIM identifică şi elimină arcurile redundante construind un nou tabel GRAF1 care conţine numai arcurile strict necesare reprezentă- rii sistemului. In acest scop se analizează pentru fiecare element

```
PROCEDURE CAUTA2(VAR GRAF1:G);
VAR GASIT :BOOLEAN;I,J:INTEGER;
BEGIN
     I:=0;GASIT:=FALSE;
          REPEAT I:=I+1;
               IF COMP.GRAF(.I.) THEN GASIT:=TRUE;
          UNTIL(I=N2)OR(GASIT);
               IF GASIT THEN
               BEGIN FOR I:=I TO N2-1 DO
                    BEGIN
                         GRAF(.I.):=GRAF1(.I+1.);
                    END;
                    N2:=N2-1
               END;
END;
PROCEDURE MINGRAF(GPRIM:G;VAR GRAF1:G);
VAR I,J:INTEGER;
BEGIN
     FOR I:=1 TO M1 DO
     GRAF1(.I.):=GPRIM(.I.);N2:=M1;
     FOR I:=2 TO M1 DO
          BEGIN J:=1;
               WHILE (J=M1) DO
                    BEGIN
                         COMP.SURSA:=GPRIM(.I.).SURSA;
                         IF GPRIM(.I.).DEST=GPRIM(.J.).SURSA THEN
                              BEGIN
                                   COMP.DEST:=GPRIM(.J.).DEST;
                                   CAUTA2(GRAF1)
                              END;
                         J:=J+1
                    END;
          END;
END;
```

Fig.3.2.5.Procedura pentru aflarea grafului minim de precedenţă

(arc) I al tabloului GPRIM, dacă procesul său destinaţie se regăseşte printre procesele sursă ale celorlalte elemente J. Dacă acest lucru e adevărat se elimină din tabel (dacă există) relaţia formată din procesul sursă al elementului I şi procesul destinaţie al elementului J, deoarece între cele două procese există un alt drum format din mai mulţi paşi, deci e un arc redundant. Deoarece eliminarea se face prin mutarea elementelor (procedura CAUTA2-fig.3.2.5) în prealabil tabloul GPRIM se mută în tabloul GRAF1 asupra căruia se execută modificările. Structura datelor algoritmului apare în fig.3.2.2. Aplicînd iterativ algoritmul descris va rezulta graful minim de precedenţă al sistemului $\mathcal{S}$ (fig.3.2.6 (b),(c),(d)).

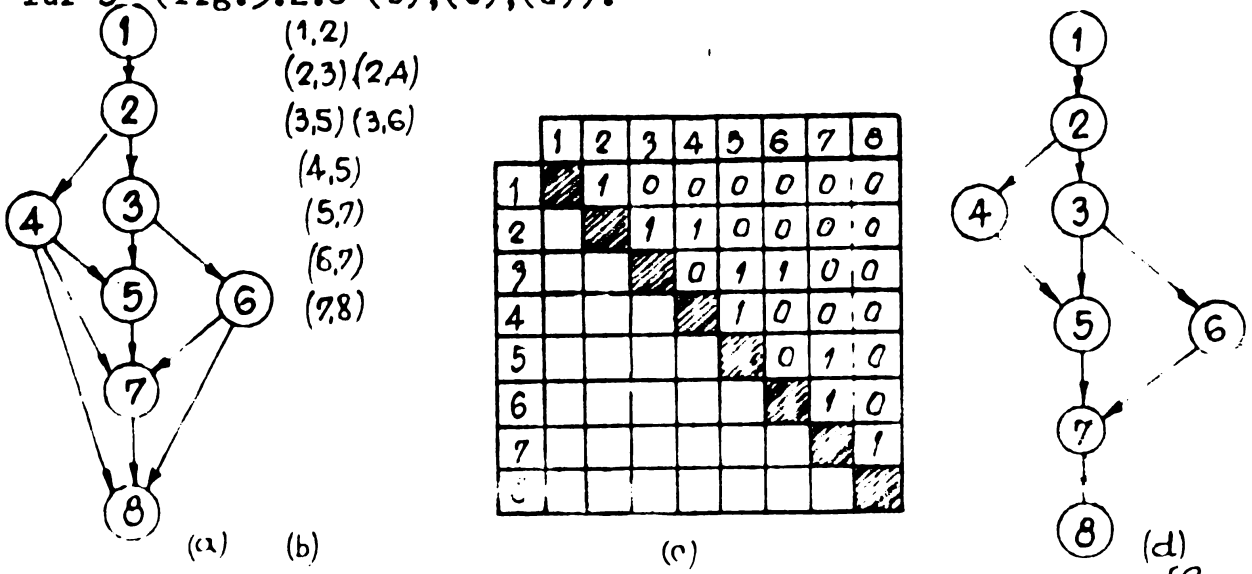


Fig.3.2.6. Graful minim de precedenţă al sistemului $\mathcal{S}$.

Această formă a grafului direct implementabilă este utilizabilă şi în studiul determinării sistemului.

In concluzie în vederea determinării grafului minim de precedenţă se procedează astfel:

M.3.2.1.
1. Se realizează graful G' al sistemului.
2. Se determină închiderea tranzitivă a relaţiei de precedenţă definite în sistem aplicînd A 3.2.1.
3. Se renumerotează procesele conform ordonării lor topologice.
4. Se determină graful minim de precedenţă aplicînd A 3.2.2.

B. In studiul determinării se porneşte de la teorema T3.2.1, care precizează că un sistem care conţine procese mutual neinterferente este un sistem determinat (D.3.2.1). Reluînd definiţia proceselor mutual neinterferente (D.3.2.2) se poate observa că studiul determinării se restrînge la submulţimea proceselor independente ale

sistemului, întrucît procesele între care există relaţia $\lessdot$ sînt prin definiţie neinterferente. Mulţimea perechilor proceselor independente se găseşte simplu din matricea tranziţiilor corespunzătoare închiderii tranzitive, şi anume sînt procesele care au 0 în această matrice (fig.3.2.4(d)). In exemplul prezentat, procese independente sînt perechile (3,4)(4,6)(5,6).

Teorema T.3.2.2 precizează că într-un sistem determinat procesele sistemului pot fi neinterpretate, cu condiţia să le cunoaştem domeniul. Rezultă că nu este necesar să se cunoască funcţiile proceselor (f_P) ci doar domeniul şi codomeniul acestora relativ la mulţimea resurselor considerate. Domeniul şi codomeniul se pot determina din condiţiile de proiectare.

Rezultă că pînă la demonstrarea determinării unui sistem de procese concurente mai rămîne de verificat dacă pentru perechile (P,P') de procese independente ale sistemului se verifică cerinţa 2° din D.3.2.2: $C_P \cap C_{P'} = C_P \cap D_{P'} = D_P \cap C_{P'} = \emptyset$, lucru uşor de făcut. Dacă pentru toate perechile de procese independente această relaţie se verifică, rezultă că procesele independente sînt mutual neinterferente şi deci sistemul este determinat (conform T.3.2.1).

In consecinţă se propune următoarea manieră de lucru:

M3.2.2.
1. Pornind de la graful minim al sistemului determinat prin M3.2.1, se aplică A.3.2.1 pentru a se determina închiderea tranzitivă.
2. Din închiderea tranzitivă (reprezentată matricial) se stabilesc perechile de procese independente.
3. Pentru aceste procese se precizează din condiţiile de proiectare domeniile şi codomeniile relativ la mulţimea resurselor luate în considerare.
4. Se verifică pentru perechile de procese independente îndeplinirea relaţiei [r.3.2.1] ;
   $C_P \cap C_{P'} = C_P \cap D_{P'} = D_P \cap C_{P'} = \emptyset$.

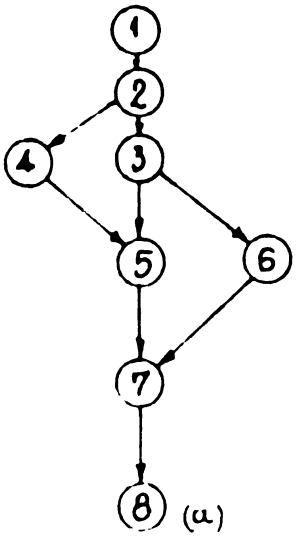
Dacă relaţia se verifică pentru toate perechile de procese, rezultă că sistemul este determinat.

Dacă se constată faptul că un sistem este nedeterminat, între perechile de procese independente care nu îndeplinesc [r.3.2.1] se introduce cîte o relaţie de precedenţă, aceasta conducînd la determinarea sistemului.

C. O problemă de maximă importanţă pentru sistemele de procese concurente timp-real, este cea referitoare la determinarea grafului paralel maximal al unui sistem de procese în care constrîngerile de precedenţă sînt stabilite din considerente de determinare. Se propune

în acest sens următorul algoritm:

A.3.2.3. <u>Algoritm pentru determinarea grafului paralel maximal al unui sistem $\mathcal{S}$ determinat de procese</u>. Se presupune că sistemul determinat se prezintă în forma unui graf G. Se presupune de asemenea că procesele sistemului partajează un set de m resurse cunoscute şi că se cunosc domeniile şi codomeniile proceselor relativ la aceste resurse. Un domeniu (codomeniu) se prezintă în forma unui element a cărui structură precizează resursele aferente procesului căruia îi este asociat. Spre exemplu fie sistemul $\mathcal{S}$, care are graful G şi ale cărui procese partajează resursele $\mathcal{R} = \{R_1, R_2, R_3, R_4, R_5\}$ ca în figura 3.2.7.



(a)

	IN DOMENIUL PROCESELOR	IN CODOMENIUL PROCESELOR
R1	1,2,7,8	4
R2	1,7	5
R3	3,4,8	1
R4	3,4,5,7	2,7
R5	6	3,6,8

(b)

Fig.3.2.7. Graful G al sistemului şi partajarea resurselor.

Algoritmul pentru aflarea grafului paralel maximal presupune următorii paşi:

1. Se aplică algoritmul A.3.2.1, pentru determinarea închiderii tranzitive a relaţiei de precedenţă $<$ pentru sistemul $\mathcal{S} = (\mathcal{P}, <)$ de n procese. Ca rezultat al aplicării algoritmului vom obţine tabloul TRANZ (fig.3.2.4(e)

2. Din condiţii de proiectare se stabileşte repartizarea resurselor. Considerînd că avem 5 resurse, repartizarea acestora se prezintă ca în figura 3.2.7(b). Pornind de la repartizarea resurselor se alcătuiesc tabelele CP şi DP care au respectiv cîte un element de forma CP(I) = (R1,R2,R3,R4,R5) pentru fiecare proces al sistemului. R_j ia valoarea j sau 0 după cum resursa se găseşte sau nu în codomeniul (domeniul) procesului I. Pentru situaţia anterior prezentată tabelele CP şi DP apar în figura 3.2.8.

3. Se aplică procedura MAXPAR pentru determinarea grafului maximal paralel (determinarea setului X de relaţii $<$) fig.3.2.9. Structura datelor utilizate în cadrul procedurii apare în fig.3.2.2. In cadrul acestui algoritm se urmăreşte detectarea acelor perechi de procese care satisfac [r.3.2.2] . In acest scop, în procedura MAXPAR pentru fiecare proces I al sistemului se fac următoarele:

	I	CP(I)				
P1	1	0	0	3	0	0
P2	2	0	0	0	4	0
P3	3	0	0	0	0	5
P4	4	1	0	0	0	0
P5	5	0	2	0	0	0
P6	6	0	0	0	0	5
P7	7	0	0	0	4	0
P8	8	0	0	0	0	5

R1 R2 R3 R4 R5

CP=(0,0,3,0,0) (0,0,0,4,0)(0,0,0,0,5)(1,0,0,0,0)
(0,2,0,0,0)(0,0,0,0,5)(0,0,0,4,0)(0,0,0,0,5)

	I	DP(I)				
P1	1	1	2	0	0	0
P2	2	1	0	0	0	0
P3	3	0	0	3	4	0
P4	4	0	0	3	4	0
P5	5	0	0	0	4	0
P6	6	0	0	0	0	5
P7	7	1	2	0	4	0
P8	8	1	0	3	0	0

R1 R2 R3 R4 R5

DP=(1,2,0,0,0)(1,0,0,0,0)(0,0,3,4,0)(0,0,3,4,0)
(0,0,0,4,0)(0,0,0,0,5)(1,2,0,4,0)(1,0,3,0,0)

Fig.3.2.8. Alcătuirea tabelelor CP şi DP.

a) Se verifică realizarea relaţiei $C_p \cap C_{p'}$. Pentru aceasta se caută resursele ce aparţin codomeniului C_p al procesului I şi se verifică dacă aceste resurse apar în codomeniile $C_{p'}$ ale proceselor J următoare lui I, deoarece conform sortării topologice procesul I poate fi în relaţia $\lessdot$ numai cu procese cu număr mai mare ca el. Se alcătuieşe perechi având pe I ca proces sursă şi pe J ca proces destinaţie şi se verifică dacă relaţia (I,J) aparţine tabloului TRANZ (procedura CAUTA3). In caz afirmativ relaţia(I,J) este o relaţie $\lessdot'$, ea aparţine deci lui X şi se trece în tabloul X, care este tabloul ce conţine relaţiile de tip $\lessdot'$. In caz negativ relaţia se neglijează deoarece $I \not\lessdot J$.

b) Se verifică într-o manieră asemănătoare realizarea relaţiei $C_p \cap D_{p'}$. Pentru aceasta se caută resursele ce aparţin codomeniului C_p al procesului I şi se verifică dacă ele nu apar printre resursele domeniilor $D_{p'}$ ale proceselor J următoare lui I. In continuare se trece la completarea tabloului X ca şi în cazul a.

c) Se verifică realizarea relaţiei $D_p \cap C_{p'}$. In acest scop se caută resursele ce aparţin domeniului D_p al procesului I printre resursele codomeniilor $C_{p'}$ ale celorlalte procese. Se trece şi aici la completarea tabelului X dacă este cazul.

Procedura MAXPAR furnizează la ieşire tabloul X care conţine toate relaţiile $\lessdot'$ care satisfac [r.3.2.1] (fig.3.2.10(a),(b),(c)).

4. Se aplică tabelului X,algoritmul A.3.2.2 pentru determinarea grafului minim de precedenţă pentru un sistem de procese. In cadrul algoritmului se elimină relaţiile redundante şi ceea ce rămîne

este tabloul **MAXPAR** care materializează graful sistemului paralel maximal echivalent cu $\mathcal{S}$, graf care apare în figura 3.2.10(d),(e),(f).

```
PROCEDURE CAUTA3(TRANZ:G;VAR   X    :G);
VAR GASIT:BOOLEAN;I,K:INTEGER;
BEGIN
     K:=0;GASIT:=FALSE;
     REPEAT K:=K+1;
          IF COMP=TRANZ(.K.) THEN GASIT:=TRUE;
     UNTIL(K=M1)OR GASIT;
          IF GASIT THEN
               BEGIN
               GASIT:=FALSE;
                    FOR I:=1 TO M3 DO
                         BEGIN
                              IF COMP=X(.I.) THEN GASIT:=TRUE
                         END;
                    IF NOT GASIT THEN
                         BEGIN
                              M3:=M3+1; X(.M3.):=COMP;
                         END
               END;
END;
PROCEDURE MAXPAR (CP,DP:T1; TRANZ:G; VAR X:G);
VAR I,I1,J,K:INTEGER  ;
BEGIN
     M3:=0;
     FOR I:=1 TO NRPROCESE DO
     BEGIN
          FOR J:=1 TO NRRESURSE DO
          BEGIN
               IF J IN CP(.I.) THEN
               BEGIN I1:=I+1;
                    FOR K:=I1 TO NRPROCESE DO
                    BEGIN
                    IF J IN CP(.K.) THEN
                         BEGIN
                              COMP.SURSA:=I;COMP.DEST:=K;
                              CAUTA3(TRANZ,X)
                         END;
                    END;
               END;
          END;
          FOR J:=1 TO NRRESURSE DO
          BEGIN
               IF J IN DP(.I.) THEN
               BEGIN
                    I1:=I+1;
                    FOR K:=I1 TO NRPROCESE DO
                    BEGIN
                         IF J IN DP(.K.) THEN
                         BEGIN
                              COMP.SURSA:=I;COMP.DEST:=K;
                              CAUTA3(TRANZ,X)
                         END;
                    END;
               END;
          END;
          FOR J:=1 TO NRRESURSE DO
          BEGIN
               IF J IN DP(.I.) THEN
               BEGIN
                    I1:=I+1;
                    FOR K:=I1 TO NRPROCESE DO
                    BEGIN
                         IF J IN CP(.K.) THEN
                         BEGIN
                              COMP.SURSA:=I;COMP.DEST:=K;
                              CAUTA3(TRANZ,X)
                         END;
                    END;
               END;
          END;
     END;
END;
```

Fig.3.2.9. Procedura MAXPAR.

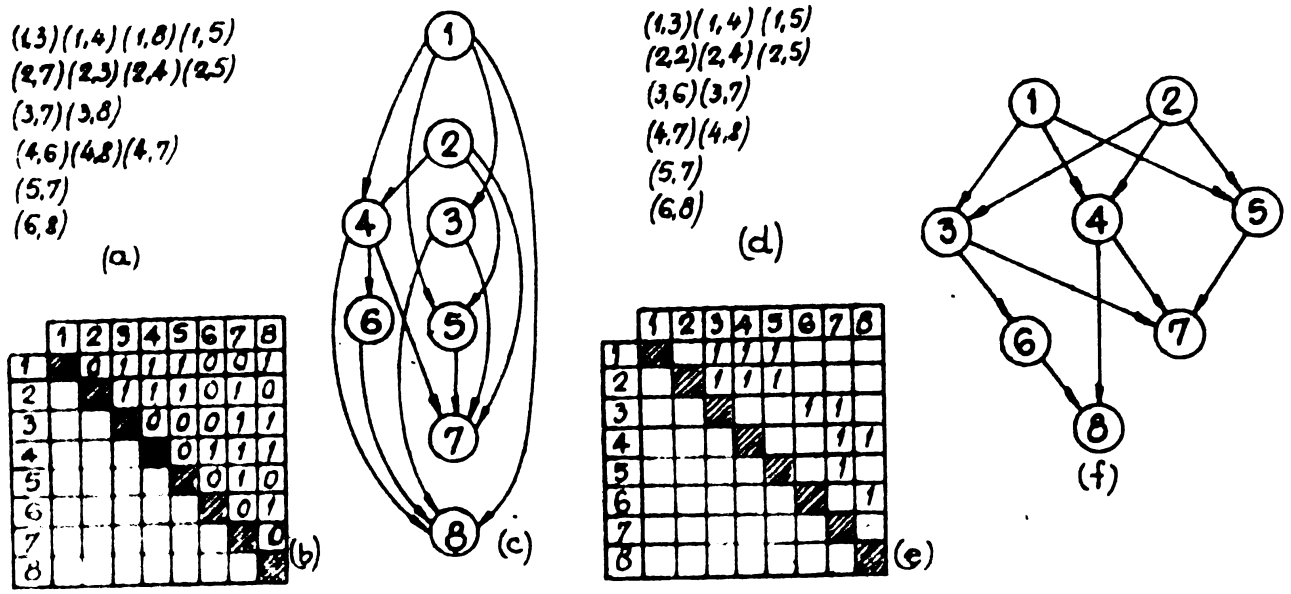


Fig.3.2.10. Graful maximal paralel şi forma sa minimă.

In concluzie pentru determinarea grafului maximal paralel se procedează astfel.

M.3.2.3. 1. Se determină graful minim de precedenţă conform lui M3.2.1.

2. Se aplică algoritmul A3.2.3 care determină graful maximal paralel.

Sistemul obţinut în această manieră este un sistem direct implementabil.

3.3. Coordonarea proceselor

După cum s-a precizat în § 2.4.3 coordonarea sau interacţiunea de nivel inferior este acea formă de interacţiune interprocese în care comunicarea se realizează în mod direct prin intermediul unor variabile partajate.

Comunicarea directă se referă la faptul că procesele comunică în mod direct modificînd în mod asincron conţinutul unor resurse, zone sau date comune (globale), numite variabile partajate. Problema care se pune în aceste condiţii este de a proteja variabilele partajate de accesul simultan al mai multor procese, cu alte cuvinte de a coordona accesul proceselor la variabilă.

Pentru a realiza acest lucru sînt necesare două tipuri de sincronizări: excluziunea mutuală şi sincronizarea de condiţie.

A. Excluziunea mutuală presupune o astfel de coordonare a acceselor la o variabilă partajată, încît ea să nu fie niciodată utilizată de mai mult decît un singur proces la un moment dat. In acest sens excluziunea mutuală trebuie să asigure tratarea unei secvenţe de instrucţii (accesul la variabila partajată) ca o operaţie indivizibilă. O astfel de secvenţă de instrucţii (acces) care trebuie să apară în execuţie ca o operaţie indivizibilă se numeşte regiune critică. In

acest context, termenul de excluziune mutuală este legat de execuţia exclusiv mutuală a secţiunilor critice care se referă la o aceeaşi variabilă partajată. Trebuie precizat că efectele execuţiei paralele a proceselor (multiprogramare sau multiprelucrare) sînt direct observabile în cazul în care procesele în cauză modifică o variabilă partajată. In astfel de cazuri, în absenţa coordonării, un proces poate utiliza prin intermediul acestei variabile rezultatele intermediare produse de execuţia incompletă a unui alt proces. Dacă două sau mai multe procese nu au variabile comune, atunci execuţia lor nu este necesar să fie mutual exclusivă. Un exemplu de excluziune mutuală îl constituie o bancă de date referitoare la starea meteorologică momentană. La aceste date au acces mai multe procese concurente care le pot actualiza (institutele meteorologice) sau le pot consulta (beneficiari). Fiecare proces de actualizare se exclude mutual cu orice alt proces, fie de actualizare fie de consultare. Pe de altă parte, procesele de consultare nu se exclud între ele. In termeni precizaţi în § 3.1, excluziunea mutuală poate fi definită astfel.

D 3.3.1. Fie $a = e_1 e_2 \ldots e_K$ o secvenţă de execuţie parţială a sistemului de procese $\mathcal{S} = (\mathcal{P}, <)$. Se spune că procesul P1 este activ după a dacă $e_i = \overline{P1}$ pentru un i oarecare, $(i \leq k$ şi $e_j \neq \underline{P1})$ pentru orice j, $i < j \leq k$. Procesele P1 şi P2 se exclud mutual în a dacă şi numai dacă cel mult unul din procesele P1 şi P2 este activ după fiecare prefix al lui a.

Luînd în considerare caracteristicile proceselor timp-real în cele ce urmează ne vor avea în vedere două procese ale căror programe conţin cîte o secţiune critică prin care procesele au acces la o variabilă partajată. Aceste procese sînt considerate ca fiind cicline (fig.3.3.1). Sînt valabile următoarele ipoteze:

1) Accesele la variabila partajată sînt realizate ca şi operaţii indivizibile. Referirile simultane sînt secvenţializate într-o ordine arbitrară de către mecanismul de sincronizare.

2) Secţiunile critice nu pot avea asociate priorităţi.

3) Vitezele relative ale proceselor sînt necunoscute.

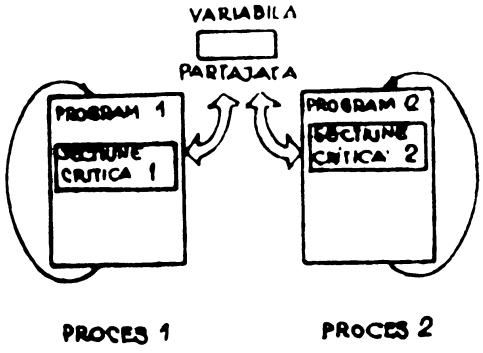


Fig.3.3.1. Procese cicline cu secţiuni critice.

4) Un program poate fi oprit înafara regiunii sale critice.

Pentru a implementa excluziunea mutuală, mecanismele de sincronizare trebuie să îndeplinească următoarele cerinţe:

1) Un proces al cărui program se execută înafara regiunii sale critice, nu poate fi blocat de un alt proces care intră în regiunea sa critică.

2) Mecanismul de sincronizare trebuie să evite formarea unei bucle de tipul „după dumneavoastră" la intrarea unor procese în regiunea critică, care amînă la nesfîrşit decizia de acces (blocare nedefinită).

3) Mecanismul de sincronizare trebuie să fie „principial"[A883]. Aceasta presupune că nici un proces nu poate fi întîrziat pentru totdeauna datorită unei condiţii care se repetă infinit de des, prin selecţia preferenţială a regiunii critice a unui alt proces care o tratează.

B. Sincronizarea de condiţie. Coordonarea proceselor concurente este necesară şi în cazul în care variabila partajată la care procesele au acces se referă la o stare improprie execuţiei unei operaţii particulare. Orice proces care încearcă să execute o astfel de operaţie este întîrziat, pînă cînd starea variabilei se va schimba, ca rezultat al acţiunii altor procese care execută operaţii asupra ei. Deoarece în acest caz variabila partajată este o condiţie, acest tip de sincronizare se va numi sincronizare de condiţie.

Un exemplu de sincronizare de condiţie apare cînd, un proces „producător" încearcă să execute o operaţie de depozitare într-un buffer (variabilă partajată) care este plin. Acest proces trebuie întîrziat pînă cînd în buffer va fi creat spaţiu disponibil de către alte procese „consumatoare".

Un alt exemplu de sincronizare de condiţie îl constituie întîrzierea unui proces de tratare a unui eveniment asincron (întrerupere, semnal de timp, etc.) pînă la apariţia efectivă a evenimentului.

Excluziunea mutuală şi sincronizarea de condiţie pot fi implementate în două moduri:

a) de către procesele concurente care utilizează variabile globale pentru coordonare. In acest caz fiecare proces implicat conţine mecanisme proprii de coordonare (autocoordonare)

b) de către sistemul de operare aferent aplicaţiei, caz în care coordonarea este realizată în mod centralizat pe baza unor convenţii prestabilite de semnalizare între procesele concurente şi sistemul de operare (coordonare centralizată).

In continuarea acestui paragraf vor fi analizate mai multe maniere de implementare a coordonării respectiv mai multe mecanisme de sincronizare. Se precizează că această analiză este strict limitată

la mecanismele fundamentale de nivel coborît care prezintă interes din punctul de vedere al proiectantului de SØ.

3.3.1. Mecanisme de sincronizare utilizate în autocoordonare

O manieră de implementare a sincronizării în autocoordonare, constă în poziţionarea şi testul unei variabile partajate. Această metodă este rezonabilă pentru implementarea sincronizării de condiţie dar nu şi pentru excluziunea mutuală. Pentru a semnala o condiţie un proces poziţionează o variabilă partajată, un alt proces care aşteaptă realizarea acestei condiţii testează această variabilă pînă cînd găseşte valoarea dorită. Această tehnică de întîrziere prin testare repetată se numeşte „aşteptare activă" (busy-waiting) [Pa82]. Ea poate fi utilizată şi în implementarea excluziunii mutuale dar nu în mod direct ci în forma unor protocoale speciale care să asigure realizarea cerinţelor impuse acesteia. Aceste protocoale sînt utilizate în cadrul fiecărui proces, la intrarea şi ieşirea din regiune critică. Există mai multe metode propuse în acest scop [CD73], [Sh74], [We77b], [An79a], [Ho83], [AS83]. Dintre acestea se va prezenta soluţia propusă de matematicianul olandez Dekker. In acest scop se defineşte o variabilă partajată RIND care va fi iniţializată pe 1 sau 2 (RINDUL-MEU sau RINDUL-LUI) şi două variabile SOLICIT(1) şi SOLICIT (2) care se iniţializează pe false. Fiecare din procesele 1 şi 2 au cîte două variabile locale „EU" şi „CELALALT" care sînt iniţializate pe 1 şi 2, pentru procesul 1 şi pe 2 şi 1, pentru procesul 2. Algoritmul este următorul:

A 3.3.1. Protocolul de intrare în regiunea critică:

```
BEGIN
     SØLICIT(EU) : = true;
     RIND : = RINDUL-LUI
                  REPEAT
                  UNTIL NOT SOLICIT(CELALALT)ØR(RIND=RINDUL-
END;                                                    MEU)
```

Protocolul de ieşire din regiunea critică:

```
BEGIN
     SØLICIT (EU): = false;
END;
```

Se demonstrează că acest algoritm satisface cele trei cerinţe (§ 3.3) impuse unui mecanism de sincronizare [Ho83], [AS83].

Protocoalele de intrare şi ieşire pot fi simplificate dacă procesorul (procesoarele) dispun de o instrucţie de tipul „testează şi poziţionează" care implementează prin cablaj două referiri la

memorie ca o operaţie indivizibilă [Ju77]. Spre exemplu dacă se doreşt testarea şi modificarea variabilei partajate booleene FANIØN, este necesar să i se asocieze o altă variabilă FØST-FANIØN şi să se aplice o astfel de instrucţie. Efectul va fi următorul:

- instrucţia TESTEAZA SI MODIFICA (FANIØN, FØST-FANIØN):

secvenţa indivizibilă { (1) FØST-FANIØN:=FANIØN; { valoarea veche a fanionului s atribuie variabilei FØST-FANI
(2) FANIØN:=true; { se modifică şi adevărat valoarea lui FANIØN }.

Utilizînd o variabilă partajată booleană ØCUPAT şi o variabilă locală fiecărui proces (FØST-ØCUPAT), excluziunea mutuală se poate implementa astfel:

A.3.3.2. Protocolul de intrare în secţiunea critică.

```
BEGIN
    REPEAT
       TESTEAZA SI MØDIFICA (ØCUPAT, FØST-ØCUPAT);
    UNTIL NØT FØST-ØCUPAT;
END;
```

Protocolul de ieşire din secţiunea critică.

```
ØCUPAT = false ;
```

Sînt valabile următoarele observaţii:

1) Ambii algoritmi pot fi generalizaţi pentru mai multe procese; în acest caz A.3.3.1 se complică şi mai mult, în timp ce A.3.3.2 rămîne nemodificat.

2) A.3.3.2 nu este implementabil pe sisteme cu /u procesoare, deoarece pînă în prezent, acestea nu implementează instrucţii de tipu „testează şi modifică".

3) Metodele de implementare ale mecanismelor de sincronizare bazat pe tehnica „aşteptării active" au două mari dezavantaje

a) sînt dificil de proiectat, de înţeles şi de utilizat

b) risipesc timpul de execuţie al procesorului.

Datorită acestor motive, autocoordonarea nu este indicată ca metodă de implementare a coordonării în sistemele de procese timp-rea

3.3.2. Mecanisme de sincronizare utilizate în coordonarea centralizată. Conceptul de "monitor monolitic"

Mecanismele de sincronizare ce vor fi prezentate în continuare fac parte din categoria celor utilizate în coordonarea centralizată. Esenţa acestor tehnici de sincronizare constă din faptul că procesele nu se sincronizează în mod direct; toate coordonările sînt rezolvate în mod centralizat de către SØ.

O metodă simplă şi eficientă de rezolvare a asincronismelor, implementabilă prin proiectare în SØ, se bazează pe conceptul de monitor monolitic [Ho83].

Preluînd sarcinile coordonării centralizate, un SØ este confruntat cu numeroase situaţii critice pe care trebuie să le rezolve cu prioritate, în regim de operaţii indivizibile, uneori în intervale precizate de timp şi fără să fie întrerupt (spre exemplu actualizarea bazei proprii de date sau tratarea unei întreruperi). Pentru a rezolva astfel de situaţii procesorul utilizează tehnica monitorului monolitic. Această tehnică se bazează pe dezactivarea/activarea IT.

M.3.3.1. Fiecare apel sistem (provenind de la procese) sau întrerupere (provenind de la orologiu, dispozitive periferice sau exterior) dezactivează întreruperile. Rutinele SØ se execută într-un regim special cu întreruperile dezactivate ferite de posibilitatea de a fi întrerupte. Acest regim este părăsit de îndată ce se revine într-un proces utilizator prin funcţia de planificare.

Metoda monitorului monolitic este avantajoasă datorită simplităţii şi posibilităţii de a trata eficient evenimente asincrone. Cu ajutorul ei se poate implementa direct excluziunea mutuală şi indirect sincronizarea de condiţie. De regulă însă, monitorul monolitic nu este utilizat în mod direct, ci ca şi mecanism fundamental de sincronizare în implementarea unor primitive de sincronizare mai complexe (bazate pe semafoare sau mesaje).

Monitorul monolitic are însă două dezavantaje:

1) Execuţia oricărei rutine a SØ presupune dezactivarea întreruperilor tuturor dispozitivelor. Aceasta înseamnă că dispozitivele nu pot primi în această perioadă noi comenzi, lucru care în unele situaţii nu poate fi acceptat. Pentru unele dispozitive, în special cele care lucrează în timp-real, dacă răspunsurile sînt prea lente se pot pierde informaţii. Din acest motiv, monitorul monolitic trebuie utilizat cu discernămînt, doar în acele părţi ale SØ care reclamă în mod absolut necesar acest lucru. De obicei aceste părţi sînt concentrate în nucleul (Kernel-ul) sistemului (§ 4.3.1).

2) Un al doilea dezavantaj rezidă în faptul că monitorul monolitic trebuie să conţină o bază de date completă, privitoare la toate resursele sistemului. Dimensiunile şi complexitatea acesteia pot deveni prohibitive.

Cu toate neajunsurile, metoda monitorului monolitic, utilizată în anumite contexte structurale, rămîne o metodă de bază în rezolvarea aspectelor legate de excluziunea mutuală cu atît mai mult cu cît majoritatea µP-lor implementează instrucţii specializate de dezactivare a IT.

3.3.3. Extinderea conceptului de monitor monolitic. Mecanismul PHS

Conceptul de monitor monolitic, rezolvă în mod corespunzător problema sincronizării proceselor prin intermediul SØ pentru sisteme de calcul monoprocesor. El devine însă ineficient în cadrul sistemelor multiprocesor. In continuare se va prezenta o metodă originală de extindere a acestui concept pentru sisteme multiprocesor slab cuplate prin intermediul unei memorii comune, sisteme care se încadrează în arhitectura propusă în § 2.2.3. In memoria comună se păstrează variabilele partajate utilizate pentru comunicarea între procese aparţinînd unor procesoare diferite. Procesele au acces normal direct la memoria comună în regim de actualizare şi consultare. Pentru a rezolva însă problema accesului exclusiv este necesară implementarea unui mecanism specializat (protecţie hard-soft) [Cr82],[CP82],[CG83]. In condiţiile în care apar mai multe cereri de acces simultan la memoria comună, de la procese aparţinînd unor procesoare diferite, excluziunea mutuală a acestora se rezolvă printr-o schemă hardware de arbitrare a cererilor, care dă cîştig de cauză la un moment dat, unui singur procesor.

Această schemă rezolvă în mod elegant problemele accesului la memoria comună în regim de consultare nu însă şi cele ale regimului de actualizare, deoarece un acces cîştigat, poate fi pierdut înainte ca procesorul respectiv să-şi fi terminat modificările.

M 3.3.2. Pentru a preveni acest fapt s-a introdus un regim special de acces la memoria comună. Trecerea în regimul special de acces se face prin program la iniţiativa unui procesor. Regimul special nu conferă procesorului nici un spor de prioritate înainte de cîştigarea accesului. După cîştigarea acestuia însă, nici un alt procesor nu mai primeşte dreptul de a citi sau de a scrie în memoria comună, pînă ce procesorul care o utilizează exclusiv, renunţă la regimul special. Pentru a rezolva însă în mod complet problema propusă, acest regim trebuie combinat cu un mecanism de protecţie software. In acest scop, fiecărei zone partajate i se asociază un index de zonă boolean care poate lua valorile „liber" sau „ocupat". Mecanismul complet PHS constă din două funcţii de sincronizare bazate pe următorul algoritm:

A 3.3.3.
```
Funcţia ACCES;
BEGIN
    REPEAT
       REPEAT
       UNTIL INDEX-ZØNA = LIBER;
      SØLICIT REGIM SPECIAL;
      IF INDEX-ZØNA = ØCUPAT THEN RENUNT REGIM SPECIAL;
    UNTIL INDEX-ZØNA = LIBER;
 INDEX-ZØNA: = ØCUPAT;
 RENUNT REGIM SPECIAL
END;
```

```
Funcţia ELIB;
BEGIN
  SØLICIT REGIM SPECIAL;
  INDEX-ZØNA: = LIBER;
  RENUNT REGIM SPECIAL
END;
```

Se observă că mecanismul PHS este o extindere generalizată a conceptului de monitor monolitic. Mecanismul are în principiu aceleaşi avantaje şi aceleaşi dezavantaje ca şi monitorul monolitic. In plus s-a încercat însă, ca prin proiectare, să blocheze cît mai puţin memoria comună, fiind preferat un regim de testare repetată din partea procesoarelor solicitante. Acest mecanism va fi utilizat în implementarea SØTR ca un submecanism al monitorului monolitic pentru acele secţiuni critice ale SØ care au nevoie de acces la variabile partajate din memoria comună a sistemului multi µP. Se impune însă o utilizare atentă şi numai în condiţii de strictă necesitate.

Cu toată simplitatea şi avantajele pe care le presupune, tehnica „monitorului monolitic" nu oferă suficiente facilităţi pentru a fi în măsură să asigure necesităţile de coordonare a unor procese în cadrul unui sistem TR. Ea rămîne însă valabilă ca o tehnică utilizată în executivele timp-real pentru implementarea simplă a excluziunii mutuale, ca element fundamental al unor mecanisme de sincronizare mai complexe.

3.3.4. Semafoare

Unul dintre mecanismele fundamentale de sincronizare a proceselor concurente care satisface cerinţele unui executiv TR îl constituie semaforul preconizat de către Dijkstra [Di68a],[Di68b] şi definit astfel:

D.3.3.2. Un semafor este o variabilă partajată s, avînd o valoare întreaga nenegativă, asupra căreia se definesc două operaţii: P şi V
P(s) întîrzie procesul care o execută într-o listă de aşteptare pînă cînd s >0, iar apoi execută s: = s-1.
V(s) execută s: = s + 1 şi eliberează din lista de aşteptare [illegible].

In lucrarea de faţă se va considera în general aceeaşi definiţie pentru semafor, cu precizarea că variabila partajată poate să aibă orice valoare, inclusiv valori negative (semafor generalizat extins)[Cr80]. In aceste condiţii, precizînd că PROCES execută operaţiile P(s) şi V(s) asupra semaforului s vom avea:

```
P(s) :  s := s-1 ;
        IF  s<0  THEN BLØCHEAZA (PRØCES)
                 ELSE CØNTINUA  (PRØCES);

V(s) :  s := s + 1
        IF s<= 0 THEN ELIBEREAZA (PROCES-DIN-LISTA);
         CØNTINUA (PROCES);
```

Semaforul poate fi iniţializat cu orice valoare întreagă. Operaţiile P şi V sînt indivizibile. Cînd $s \leq 0$ se spune că semaforul lucrează în regim de blocare, iar valoarea s precizează numărul de procese aflate în lista de aşteptare a semaforului (blocate în semafor). Acest număr este util în rezolvarea blocărilor infinite care ar putea fi generate de evenimente excepţionale (spre exemplu abortarea unui proces).

Proprietatea de „principialitate" a mecanismului, adică aceea ca un proces să nu rămînă infinit blocat într-un semafor se asigură teoretic dacă numărul de operaţii V executate asupra semaforului este suficient. In practică acest aspect este rezolvat dacă se asigură o egalitate strictă între numărul de operaţii P şi numărul de operaţii V care se execută asupra unui semafor aflat în regim de blocare.

Semafoarele sînt unelte foarte generale, larg utilizate în rezolvarea problemelor de sincronizare [Ha74],[PC80],[JP81a],[JP81b],[KJ82]. In acest sens în continuare se furnizează cîteva metode de utilizare a lor.

M.3.3.3. Implementarea excluziunii mutuale cu ajutorul semafoarelor se realizează asociind variabilei partajate un semafor S cu valoarea iniţială 1 şi realizînd la intrarea în fiecare secţiune critică care se referă la acea variabilă, o operaţie P asupra lui S iar la ieşire o operaţie V(S). (Fig.3.3.2).

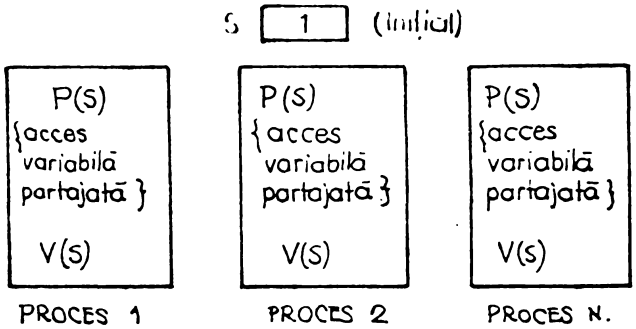


Fig.3.3.2. Implementarea excluziunii mutuale.

M.3.3.4. Implementarea sincronizării de condiţie se realizează ataşînd la variabila partajată care reprezintă condiţia, un semafor care să realizeze sincronizarea.

Valoarea iniţială a semaforului este 0. Procesul care se sincronizează face un P pe acest semafor, procesul care actualizează condiţia face un V. Acest mecanism se numeşte sincronizare simplă (asimetrică) şi poate fi utilizat în tratarea întreruperilor (fig.3.3.3a) (vezi § 4.3.2). Se mai admite şi un alt gen de sincronizare şi anume sincronizarea simetrică (fig.3.3.3(b)).

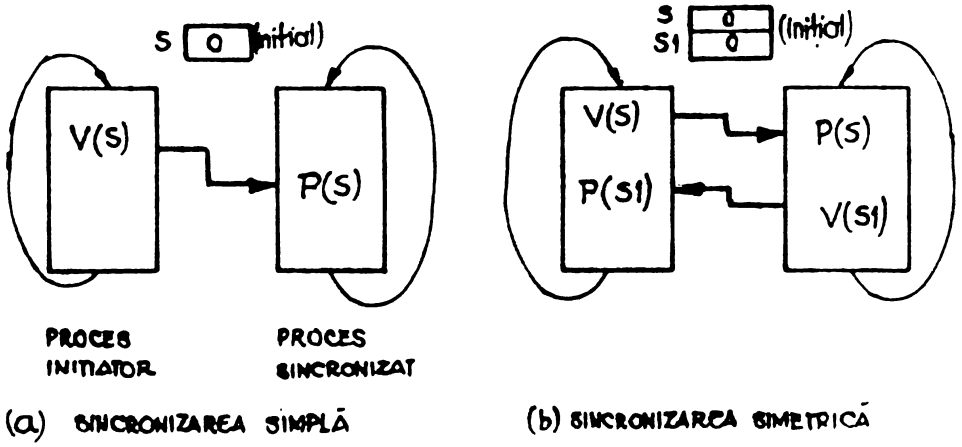


Fig.3.3.3. Implementarea sincronizării de condiţie.

Observaţie. Deoarece ne aflăm în domeniul aplicaţiilor TR care în general presupun procese ciclice, activate de anumite semnale, interblocarea (§ 3.6.) nu poate să apară, deoarece procesele practic nu se termină. Condiţia necesară este ca să existe un proces iniţiator (care să facă prima operaţie V).

M 3.3.5. Semafoarele generalizate extinse pot fi utilizate cu eficienţă pentru sincronizarea de condiţie în cazul alocării resurselor. Astfel, fiecărui tip de resursă i se asociază cîte un semafor care are ca valoare iniţială, numărul de unităţi disponibile ale resursei. Operaţiile de alocare se traduc prin operaţii P iar cele de eliberare prin operaţii V asupra semaforului specific.

Această metodă asigură atît evidenţa utilizării resurselor cît şi mecanismul de sincronizare aferent. Este recomandabilă atît prin eficienţă cît şi prin simplitate.

Problema esenţială care se pune la implementarea semafoarelor este legată de realizarea indivizibilităţii operaţiilor P şi V. În acest scop se poate folosi mecanismul bazat pe „aşteptarea activă" (A.3.3.1 sau A.3.3.2) sau monitorul monolitic (M3.3.1). Metoda presupusă a fi utilizată în această lucrare se bazează pe implementarea operaţiilor P şi V ca şi rutine ale nucleului SØ care lucrează în regim de monitor monolitic (vezi § 5.2).

3.3.5. Semafoare speciale

Conceptul de semafor generalizat poate fi extins şi pentru sisteme multiprocesor [Cr82],[CP82],[PP83a],[CG83]. Un astfel de semafor care poate sincroniza activitatea unor procese aparţinînd unor procesoare diferite se va numi semafor special. Contextul în care se defineşte acest semafor este cel al sistemelor multiprocesor slab cuplate prin memorie comună precizat în § 2.2.3. Se subliniază faptul că fiecare procesor dispune înafara memoriei comune, de o memorie proprie în care are acces exclusiv.

Semafoarele speciale coincid din punct de vedere funcţional cu semafoarele generalizate extinse. Din punctul de vedere al definirii şi implementării apar însă unele particularităţi generate în principal de faptul că nici un procesor nu are acces la memoria privată a unui alt procesor. Astfel:

a) Semafoarele speciale se păstrează în memoria comună a sistemului.

b) Lista de aşteptare a proceselor blocate într-un astfel de semafor se păstrează de asemenea în memoria comună.

b) Operaţiile P şi V efectuate asupra unui astfel de semafor se implementează în mod diferit faţă de cele uzuale, necesitînd prezenţa unui sistem de întreruperi interprocesoare. Cu toate acestea în cadrul sistemului, rutinele P şi V vor fi concepute unitar, astfel încît utilizatorul să nu sesizeze nici o diferenţă între un semafor obişnuit şi unul special.

d) Implementarea semafoarelor speciale necesită prezenţa unui mecanism special destinat rezolvării accesului exclusiv la memoria comună; este vorba de mecanismul PHS prezentat.

In rest datorită identităţii lor funcţionale, toate cele precizate în legătură cu posibilităţile de utilizare a semafoarelor normale sînt valabile şi pentru semafoarele speciale.

3.4. Cooperarea proceselor

Cooperarea sau interacţiunea de nivel superior este acea formă de interacţiune în care comunicarea se realizează prin intermediul transmiterii mesajelor [An79b], [Ba81]. Mesajul îşi are originea în conceptul de semafor a cărui mecanism de sincronizare a fost completat cu un mecanism de transmitere a datelor. Astfel mesajul este o structură mai complexă care prin operaţii specifice permite atît transmiterea unor informaţii între procese concurente cît şi sincronizarea execuţiei acestora. Procesele îşi transmit şi recepţionează mesaje în loc să citească şi să scrie variabile partajate.

Cooperarea este strîns legată de relaţia producător/consumator [Di68b], [MD74], [Ha73], [JP82]. Din punctul de vedere al numărului de procese implicate în această relaţie se disting patru tipuri de cooperări:

1) un producător/un consumator
2) un producător/mai mulţi consumatori
3) mai mulţi producători/un consumator
4) mai mulţi producători/mai mulţi consumatori.

In cadrul acestor tipuri un rol determinant îl are relaţia un producător/un consumator care stă la baza mecanismului de comunicare prin mesaje. Din ea vor fi derivate ulterior şi celelalte tipuri de cooperări

3.4.1. Relaţia producător/consumator

In analiza relaţiei producător/consumator se utilizează un model simplu compus din două sisteme ciclice închise [CD73]. Sistemul $\mathcal{S}_1$ produce resurse şi le depune într-un buffer de dimensiune N iar sistemul $\mathcal{S}_2$ extrage resurse din buffer şi le consumă. Modelul nu pierde nimic din generalitate dacă se consideră că sistemele $\mathcal{S}_i$ sînt formate din cîte un singur proces. Dacă cele două sisteme sînt executate concurent ele se află într-o relaţie producător/consumator. Funcţionarea corectă a acestei relaţii presupune următoarele condiţii

C.3.4.1. $\mathcal{S}_1$ trebuie blocat cînd bufferul e plin şi pus să aştepte apariţia unor locuri libere (prin consum).

C.3.4.2. $\mathcal{S}_2$ trebuie blocat cînd bufferul e gol şi pus în aşteptarea apariţiei unor resurse (prin producţie).

In mod concret presupunînd că la fiecare reluare $\mathcal{S}_1$, produce o resursă, iar $\mathcal{S}_2$ consumă una, numărul de reluări ale lui $\mathcal{S}_2$ nu trebuie să depăşească pe cel al reluărilor lui $\mathcal{S}_1$ iar $\mathcal{S}_1$ nu poate fi cu mai mult de N reluări în faţa lui $\mathcal{S}_2$. Se consideră evident faptul că o anumită resursă produsă de către un producător poate fi consumată de către un singur consumator. Cu alte cuvinte o resursă produsă poate fi consumată o singură dată.

Pentru început se va trata problema (k)-sincronizării.

D3.4.1. Fie $\mathcal{S} = \mathcal{S}_1^m // \mathcal{S}_2^n (m \geq 1, n \geq 1)$ şi fie a o secvenţă de execuţie a lui $\mathcal{S}$. In orice prefix b a lui a, fie $N_1(b)$ numărul de reluări ale lui $\mathcal{S}_1$. Pentru un $k > 0$ dat, se spune că $\mathcal{S}_1$ este (k)-sincronizat cu $\mathcal{S}_2$ în a dacă $N_1(b) \leq N_2(b) + k$ pentru toate prefixele b ale lui a.

Referitor la această definiţie se pot face următoarele observaţii: a) nu există nici o secvenţă a care să aibă această proprietate dacă $m > n + k$.

b) relaţia definită nu este simetrică.

c) $\mathcal{S}_1$ (k)-sincronizat cu $\mathcal{S}_2$ presupune că pentru toate prefixele b ale lui a, numărul de resurse produse de $\mathcal{S}_1$ nu depăşeşte niciodată cu mai mult de k-unităţi numărul de resurse consumate de $\mathcal{S}_2$.

In implementarea (k)-sincronizări se pot utiliza semafoare. Fie $\mathcal{S}_1$ şi $\mathcal{S}_2$ două sisteme închise şi s un semafor. Definim $\mathcal{S}_1' = [P(s), \mathcal{S}_1]$ prin prefixarea lui $\mathcal{S}_1$ cu o operaţie P asupra lui s şi $\mathcal{S}_2' = [\mathcal{S}_2, V(s)]$ sufixînd pe $\mathcal{S}_2$ cu o operaţie V(s). In aceste condiţii se poate enunţa următoarea propoziţie.

P3.4.1. Fie $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}'_1$ și $\mathcal{S}'_2$ definiți ca mai sus și fie I valoarea inițială a semaforului s. Se definește $\mathcal{S} = (\mathcal{S}'_1)^m // (\mathcal{S}'_2)^n$ ($m \geqslant 1$, $n \geqslant 1$) și fie $\mathcal{A}$ setul de secvențe de execuție valide pentru $\mathcal{S}$. In aceste condiții, în fiecare $a \in \mathcal{A}$, $\mathcal{S}'_1$ este (k)-sincronizat cu $\mathcal{S}'_2$. $\mathcal{A}$ este nevid dacă și numai dacă $m \leqslant n + k$.

Revenind la problema cooperării prin relația producător/consumator și considerînd un buffer de dimensiune N se poate constata că pentru o funcționare corectă este necesar ca:

1. $\mathcal{S}_1$ să fie (N)-sincronizat cu $\mathcal{S}_2$ (pentru ca $\mathcal{S}_1$ să nu mai producă resurse dacă bufferul este plin) (din C3.4.1).

2. $\mathcal{S}_2$ să fie (0)-sincronizat cu $\mathcal{S}_1$ (pentru ca $\mathcal{S}_2$ să nu încerce să consume resurse dintr-un buffer gol (din C.3.4.2).

(k)-sincronizarea rezolvă condiția C.3.4.1, dacă $m \leqslant n+k$ și dacă $\mathcal{S}'_1$ începe să producă înainte ca $\mathcal{S}'_2$ să încerce să consume.

Dacă însă $\mathcal{S}'_2$ își începe execuția înaintea lui $\mathcal{S}'_1$ făcînd un V asupra lui s, mecanismul nu mai funcționează (se depășește dimensiunea maximă a bufferului k=N+1). Rezultă că, pentru a implementa corect coordonarea mai este nevoie de încă o sincronizare.

P.3.4.2. Fie $\mathcal{S} = (\mathcal{S}_1)^m // (\mathcal{S}_2)^n$ ($m \geqslant 1$, $n \geqslant 1$) și fie a o secvență de execuție pentru $\mathcal{S}$. Se spune că $\mathcal{S}_1$ și $\mathcal{S}_2$ sînt (k, ℓ)-sincronizate în a dacă $\mathcal{S}_1$ este (k)-sincronizat cu $\mathcal{S}_2$ și $\mathcal{S}_2$ este (ℓ)-sincronizat cu $\mathcal{S}_1$.

Pentru ca astfel de secvențe de execuție să existe, este necesar ca ℓ, k, m și n să satisfacă anumite condiții.

P.3.4.3. Fie $\mathcal{S}_1$ și $\mathcal{S}_2$ și $\mathcal{S}$ cei definiți mai sus. Fie $\mathcal{A}$ setul de secvențe de execuție în care $\mathcal{S}_1$ este (k)-sincronizat cu $\mathcal{S}_2$ și $\mathcal{S}_2$ este (ℓ)-sincronizat cu $\mathcal{S}_1$. $\mathcal{A}$ este nevid dacă și numai dacă $k+\ell \geqslant 1$ și $-\ell \leqslant m-n \leqslant k$ (r.3.4.1).

Demonstrație

$$N_1(b) \leqslant N_2(b)+k \quad \{ \text{(k)-sincronizare} \} \qquad \text{(r.3.4.2)}$$
$$N_2(b) \leqslant N_1(b)+\ell \quad \{ \text{(ℓ)-sincronizare} \} \qquad \text{(r.3.4.3)}$$

Prin adunare rezultă $k+\ell \geqslant 0$. Dar $k+\ell = 0$ presupune $k = -\ell$, sau înlocuind în (r.3.4.3)

$$N_1(b) \geqslant N_2(b) + k.$$

Comparînd această relație cu [r.3.4.2] rezultă că pentru toate prefixele b ale lui a avem $N_1(b) = N_2(b)+k$ ceea ce în mod evident nu poate fi adevărat. Rezultă deci că $k+\ell \neq 0$ deci $K+\ell > 1$.

Din relațiile

$$m \leqslant n+k \quad \{ \text{(k)-sincronizare} \}$$
$$n \leqslant m+\ell \quad \{ \text{(ℓ)-sincronizare} \}$$

rezultă imediat $-\ell \leqslant m-n \leqslant k$ (r.3.4.4).

In implementarea (k,ℓ)-sincronizării se pot utiliza semafoarele. Aplicînd în mod simetric P.3.4.1 se obține:

P.3.4.4. Fie $\mathcal{S}'_1$ și $\mathcal{S}'_2$ dați ca și în P.3.4.1 și fie semaforul g. Se definește $\mathcal{S}''_1 = [\mathcal{S}'_1, V(g)]$ sufixînd pe $\mathcal{S}'_1$ cu o operație V asupra lui g și $\mathcal{S}''_2 = [P(g), \mathcal{S}'_2]$, prefixînd pe $\mathcal{S}'_2$ cu o operație P(g). Fie $\mathcal{A}$ un set de secvențe de execuție pentru $\mathcal{S} = (\mathcal{S}''_1)^m // (\mathcal{S}''_2)^n$ și fie k și ℓ valorile inițiale ale semafoarelor s respectiv g. In aceste condiții pentru toți $a \in \mathcal{A}$, $(\mathcal{S}''_1)$ și $(\mathcal{S}''_2)$ sînt (k,ℓ)-sincronizați.

Analizînd rezultatele obținute și rememorînd cerințele relației producător/consumator se poate enunța următoarea propoziție:

P.3.4.5. Fie un buffer $\mathcal{B}$ de dimensiune N și fie $\mathcal{S} = \mathcal{S}_1 // \mathcal{S}_2$. Condiția necesară și suficientă ca $\mathcal{S}_1$ să coopereze corect cu $\mathcal{S}_2$ prin $\mathcal{B}$ în cadrul relației producător/consumator (C.3.4.1, C.3.4.2) este ca $\mathcal{S}_1$ să fie (N,O) sincronizat cu $\mathcal{S}_2$.

Implementarea acestei relații de cooperare se face utilizînd P.3.4.4 în care se ia inițial s=N și g=O.

3.4.2. Mecanism de cooperare prin mesaje pentru sisteme de procese concurente

Pe baza celor expuse în § 3.4.1 se propune în continuare definirea teoretică a unui mecanism de cooperare pentru sisteme de procese concurente bazat pe transmiterea mesajelor. Pentru aceasta se va defini în primul rînd noțiunea de mesaj, iar apoi vor fi precizați operatorii care realizează efectiv cooperarea proceselor: SEND (expediere mesaj) și RECEIVE (REC) (primire mesaj).

D.3.4.2. Mesajul se definește ca fiind resursa consumabilă care intervine în relația producător/consumator. Din punct de vedere structural mesajul este conceput ca și o structură de date purtătoare de informații semnificative pentru procese (de regulă șiruri de caractere).

In contextul acestui capitol, mesajele se identifică cu resursele care se depun și se consumă din buffer.

Pentru a defini operatorii SEND și REC se procedează astfel. Se restrîng $\mathcal{S}''_1$ și $\mathcal{S}''_2$ la acele porțiuni din $\mathcal{S}_1$ și $\mathcal{S}_2$ care efectiv produc (expediază), respectiv consumă (recepționează) mesaje.

D.3.4.3. Fie $\mathcal{S}''_1$ și $\mathcal{S}''_2$ două porțiuni de proces precizate ca mai sus. In aceste condiții se definește operatorul SEND ca fiind identic cu $\mathcal{S}''_1$ iar operatorul RECEIVE (REC) ca fiind identic cu $\mathcal{S}''_2$.

In cele ce urmează se va demonstra că mecanismul de cooperare propus implementează în mod corect relația producător/consumator.

Se presupune că există un proces ciclic producător care conţine un SEND şi un proces ciclic consumator care conţine un operator REC şi că cele două procese cooperează prin intermediul unui buffer de dimensiune N. Din D.3.4.3 rezultă că SEND şi REC sînt direct derivaţi din $\mathcal{P}''_1$ respectiv $\mathcal{P}''_2$. Atunci din P.3.4.4. rezultă că procesul producător (cel care conţine SEND-ul) este (N,0) sincronizat cu procesul consumator (cel care conţine REC-ul). In aceste condiţii din P.3.4.5. rezultă că cele două procese cooperează corect în cadrul relaţiei producător/consumator (c.c.t.c).

3.4.3. Modalităţi de implementare a comunicării prin intermediul transmiterii mesajelor

Avînd la dispoziţie un mecanism principial de comunicare prin mesaje bazat pe relaţia producător/consumator, se propune în continuare studiul modalităţilor de implementare a comunicării şi sincronizării proceselor, utilizînd un astfel de mecanism. In ceea ce priveşte comunicarea, pornind de la observaţia că orice comunicare interprocese presupune un proces emiţător şi un proces receptor, în funcţie de modul în care acestea sînt precizate, se pot implementa mai multe modalităţi de comunicare.

A. Comunicare prin numire directă, în care fiecare proces comunicant îl numeşte pe celălalt. Această comunicare este denumită de tip canal (fig.3.4.1a). Astfel SEND-ul precizează numele receptorului, iar mesajul poate fi recepţionat numai de către procesul nominalizat; REC-ul precizează numele emiţătorului, iar mesajul va fi recepţionat numai dacă provine de la acesta. Această manieră de comunicare este restrictivă, dar este uşor de implementat şi de utilizat [Ch80]. Este ideală pentru cooperarea de tip un producător/un consumator, dar nu şi pentru restul tipurilor de cooperare.

B. Comunicare prin "căsuţe poştale". O căsuţă poştală poate apare ca şi destinaţia unui proces producător (SEND) şi ca şi sursa unui proces consumator (REC). Astfel un mesaj trimis unei căsuţe poştale poate fi recepţionat de orice proces care execută un RECEIVE asupra căsuţei poştale (fig.3.4.1.b). Acest mecanism este potrivit pentru toate tipurile de cooperări, dar este dificil de implementat [AS83].

C. Comunicarea prin "porturi", este un caz particular al comunicării prin căsuţe poştale în care fiecărui proces receptor i se asociază o căsuţă poştală de unde el se aprovizionează cu mesaje pe care le consumă. Astfel numele căsuţei poştale devine similar cu al procesului receptor şi el va fi precizat numai în SEND. RECEIVE-ul nu mai precizează sursa, deoarece procesul citeşte din propria sa „căsuţă poştală" care se numeşte "port" (fig.3.4.1.c).

„Porturile" sînt simplu de implementat deoarece toate RECEIVE-urile care se adresează unui „port" se fac din același proces. Ele pot fi utilizate în implementarea practică a tuturor tipurilor de cooperare.

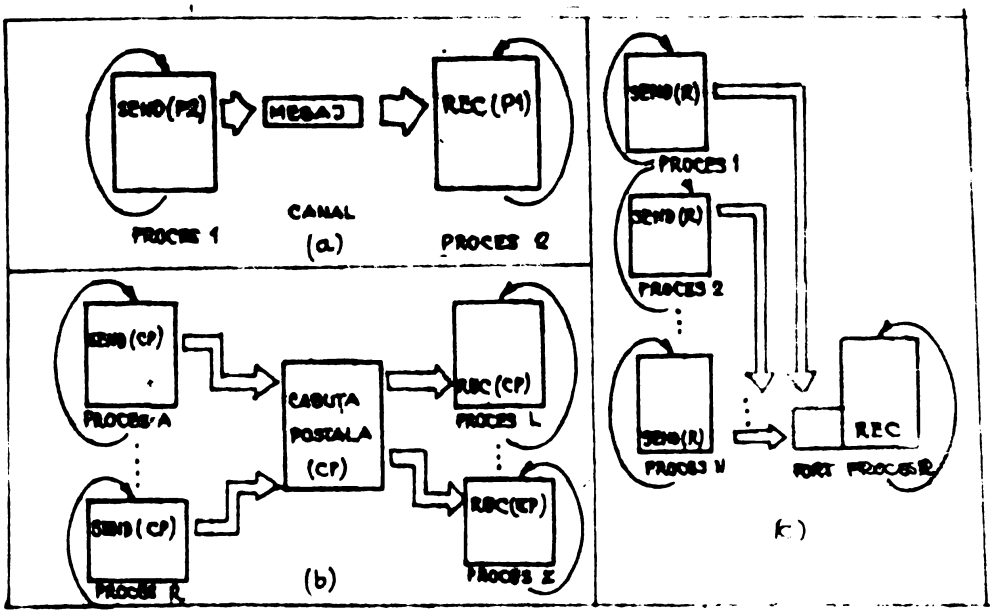


Fig.3.4.1. Modalități de implementare a comunicării prin transmiterea mesajelor.

3.4.4. Comunicări sincronizate. Modalități de implementare a sincronizării prin intermediul transmiterii mesajelor.

În ceea ce privește studiul modalităților de implementare a sincronizării, se pornește de la observația importantă că un mecanism de cooperare prin mesaje poate realiza pe lîngă funcția de comunicare și pe aceea de sincronizare a proceselor implicate în comunicare, adică poate realiza comunicări sincronizate. În funcție de maniera de implementare a sincronizării distingem mai multe posibilități de comunicări sincronizate. Acestea sînt strîns legate de proprietatea de blocare a operatorilor SEND și REC. Se spune despre un operator că are proprietatea de blocare dacă execuția sa permite întîrzierea procesului care l-a apelat. Se vor analiza din această perspectivă operatorii SEND și REC.

Pentru operatorul SEND, punctul de pornire al analizei îl constituie dimensiunea maximă a bufferului de transmitere a mesajelor, implicat în relația producător/consumator.

Din acest punct de vedere se disting trei cazuri și implicit trei maniere de sincronizare prin transmiterea mesajelor:

a) Dimensiunea bufferului este egală cu N. Acest caz a fost discutat pe larg în § 3.4.1 și s-a văzut că pentru o cooperare corectă procesele producător/consumator trebuie să fie (N,0) sincronizate. În acest caz proprietatea de blocare devine activă pentru SEND cînd numărul de resurse produse ajunge egal cu dimensiunea bufferu-

lui şi dispare cînd acest lucru nu mai este valabil. Se va denumi această manieră de transmitere a mesajelor comunicare N sincronizată, iar operatorul se va numi SEND cu blocare temporară.

b) Dimensiunea bufferului este infinită. In acest caz nici un proces emiţător nu va fi întîrziat din cauza unui SEND. Se va denumi această manieră de transmitere a mesajelor comunicare asincronă iar operatorul corespunzător SEND fără blocare. Ea permite producătorului să o ia ori cît de mult înaintea consumatorului în ceea ce priveşte producţia mesajelor.

c) Dimensiunea bufferului este zero. In acest caz procesul producător va fi întîrziat de către SEND pînă cînd se va executa REC-ul corespunzător de către procesul consumator. Cînd se realizează acest lucru se transmite mesajul şi ambele procese continuă. Această manieră de comunicare se numeşte sincronă. Se observă că în acest caz SEND-ul are proprietatea de blocare, de aceea el este denumit SEND cu blocare (blocking-send). SEND-ul cu blocare stă la baza unui mecanism avansat de interacţiune implementat ca şi construcţie de limbaj de nivel superior în limbajul ADA sub numele de „rendezvous".[Ic79a],[Ic79b],[KJ82]

In ceea ce priveşte RECEIVE-ul forma cea mai utilizată este aceea care se bucură de proprietatea de blocare. REC cu blocare implementează în mod implicit sincronizarea deoarece un proces consumator este întîrziat dacă bufferul este gol, pînă la sosirea primului mesaj. Cu toate acestea în unele implementări se utilizează operatorul REC fără blocare care are avantajul că permite unui proces să primească mai multe mesaje şi apoi să selecteze unul pentru a-l prelucra.

O ultimă formă posibilă de REC este cea cu acceptarea doar a acelor mesaje care determină ca o condiţie precizată în REC să fie adevărată. Procesului consumator i se permite astfel să obţină informaţii despre mesaj înainte de a-l primi efectiv. Această formă de recepţie se numeşte RECEIVE condiţional şi prin intermediul ei se pot rezolva practic toate tipurile de sincronizări. Implementarea ei este însă relativ complicată.

3.4.5. Posibilităţi de implementare şi utilizare a mecanismului de cooperare propus.

In § 3.4.2 a fost definit d.p.d.v. teoretic un mecanism bazat pe transmiterea mesajelor care implementează în mod corect relaţia producător/consumator. In spiritul celor discutate în § 3.4.3 şi § 3.4.4 se propune particularizarea acestui mecanism prin precizarea unor modalităţi concrete de implementare a manierelor de comunicare şi de sincronizare cele mai potrivite pentru domeniul sistemelor de procese concurente timp-real. De asemenea se propune o trecere în revistă a posibilită

lor de utilizare ale mecanismului de cooperare astfel completat.

Tinînd cont de caracteristicile sistemelor de operare timp-real se abordează următoarele aspecte:

A). Stabilirea dimensiunii maxime N a bufferului de comunicare se realizează pornind de la P3.4.3. Asimilînd pe N cu k şi luînd $\ell = 0$ (dimensiunea minimă a bufferului) din r.3.4.4. rezultă:

$$N \geq m-n \qquad [r.3.4.5]$$

Tinînd cont că într-un SØTR, poate fi determinat în mod experimental numărul de SEND-uri, respectiv numărul de REC pe care le execută un proces într-un anumit interval semnificativ de timp (vezi § 4.5), eventual ca o medie a unui număr corespunzător de determinări repetate, se propune următoarea metodă.

M.3.4.1. Pentru fiecare din procesele i implicate într-o relaţie producător/consumator se determină:

$$m_i = \frac{\sum_{j=1}^{N_{1i}} m_{ij}}{N_{1i}} \quad \text{şi} \quad n_i = \frac{\sum_{j=1}^{N_{2i}} n_{ij}}{N_{2i}}$$

unde $m_{ij} = \frac{\text{Număr SEND-uri}}{\text{Interval de timp}(T)}$ pentru procesul i în determinarea j iar,

$n_{ij} = \frac{\text{Număr REC-uri}}{\text{Interval de timp}(T)}$ pentru procesul i în determinarea j.

Se presupune că se fac N_{1i} determinări pentru m_i şi N_{2i} determinări pentru n_i. Considerînd numărul proceselor implicate egal cu P se determină m şi n după relaţiile:

$$m = \max (m_i) \quad i = 1,P$$
$$n = \min (n_i) \quad i = 1,P$$

Conform relaţiei r.3.4.5 se alege o astfel de valoare pentru N încît $N \geq m-n$.

Stabilirea lui N este deosebit de semnificativă dacă sistemul lucrează într-un regim staţionar, fără fluctuaţii prea mari în funcţionare. In aceste condiţii stabilirea unei valori corespunzătoare pentru N asigură un grad maxim de paralelism lucru esenţial în condiţii de TR. Sînt însă de interes şi valori înafara domeniului stabilit de P.3.4.4, caz în care intră în acţiune mecanismul de întîrziere. Deoarece în acest caz procesele se aşteaptă unele pe altele, gradul de paralelism este atenuat în mod corespunzător.

B). In ceea ce priveşte implementarea manierei de comunicare se propune comunicarea prin "porturi". Aceasta presupune asocierea cîte unui buffer de comunicare, în calitate de port, fiecărui proces

receptor şi indicarea în SEND a destinaţiei mesajului (precizarea procesului destinatar, respectiv a port-ului acestuia). Operatorul REC nu are nevoie de informaţii suplimentare întrucît consumarea mesajelor se realizează doar din bufferul (port-ul) asociat procesului care l-a iniţiat. In buferul de comunicare mesajele se depun în ordinea sosirii lor şi sînt consumate în aceeaşi ordine (disciplina FIFØ). Această manieră de comunicare are avantajul că este relativ uşor implementabilă şi permite realizarea simplă a tipurilor de comunicare care se apreciază ca fiind acoperitoare pentru domeniul sistemelor dedicate timp-real.

M.3.4.2. Cooperarea de tip mai mulţi producători/un consumator presupune ca fiecare proces producător să expedieze cîte un mesaj avînd ca destinaţie procesul consumator. Aceasta se poate realiza în versiune normală (ciclică) (fig.3.4.1.c), în versiune „SAU" (fig.3.4.2.a) sau în versiune „SI" (fig.3.4.2.b),

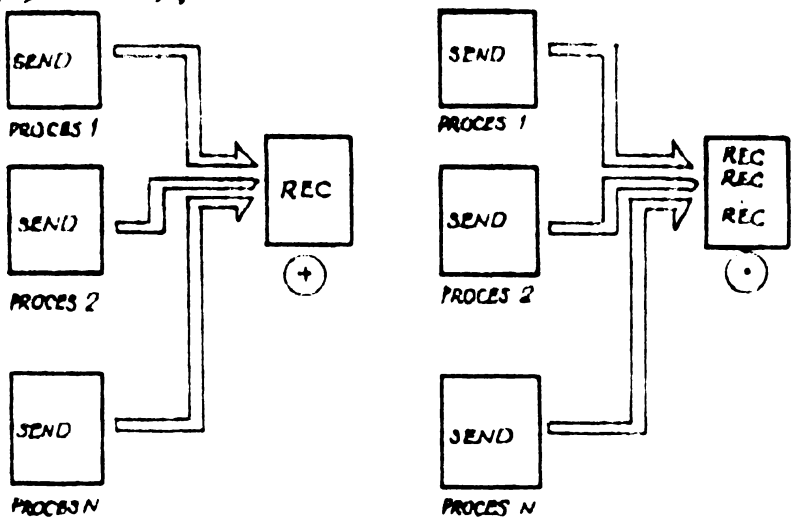


Fig.3.4.2. Versiuni de cooperării de tip mai mulţi producători/un consumator.

M.3.4.3. Cooperarea de tip un producător/mai mulţi consumatori presupune ca procesul producător să expedieze cîte un mesaj pentru fiecare proces consumator utilizînd în acest o suită de operaţii SEND (fig.3.4.3)

C) Referitor la implementarea sincronizării, din considerente de proiectare şi de implementare a SØTR se propune realizarea SEND-ului cu blocare temporară, deci comunicarea N sincronizată. In unele cazuri, dacă este necesar se poate implementa şi comunicarea asincronă. Aceasta însă presupune

a) fie o determinare precisă a dimensiunii maxime a bufferelor astfel încît să nu apară blocări din acest motiv

b) fie implementarea unui regim de alocare dinamică pentru spaţiul de memorie destinat bufferelor şi renunţarea la limitarea lor superioară.

In ceea ce priveşte operatorul REC, se propune implementare cu blocare ca fiind cea mai potrivită pentru domeniul luat în considerare.

Mecanismul de coordonare astfel particularizat permite în plus faţă de cele precizate implementarea comunicării sincrone şi a variantelor acesteia (comunicarea de tip corutină şi comunicarea de tip subrutină) [Ho78],[KJ82] prin utilizarea a două perechi de operatori SEND, REC după cum rezultă din fig.3.4.4.

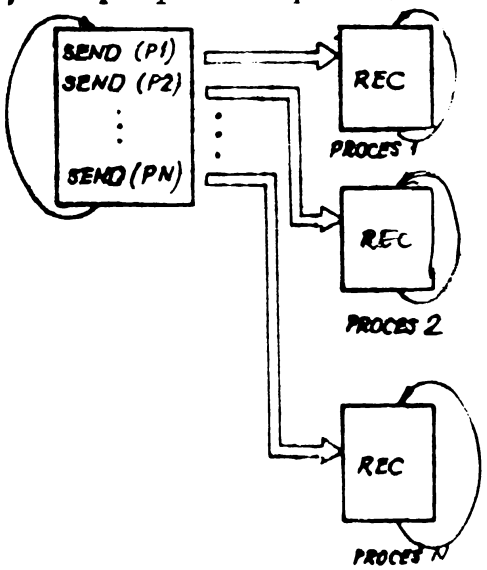


Fig.3.4.3. Coordonare de tip un producător/mai mulţi consumatori.

D) Semnificaţia informaţiei mesajului poate fi luată în considerare la realizarea comunicării sau a sincronizării în două moduri:

a) prin convenţii stabilite în faza de proiectare a mecanismului şi care vor fi cuprinse în structura sistemului dobîndind un caracter permanent

b) prin convenţii stabilite în etapa de implementare a aplicaţiei (scrierea programelor), bazate pe facilităţile oferite de mecanism şi care au un caracter flexibil.

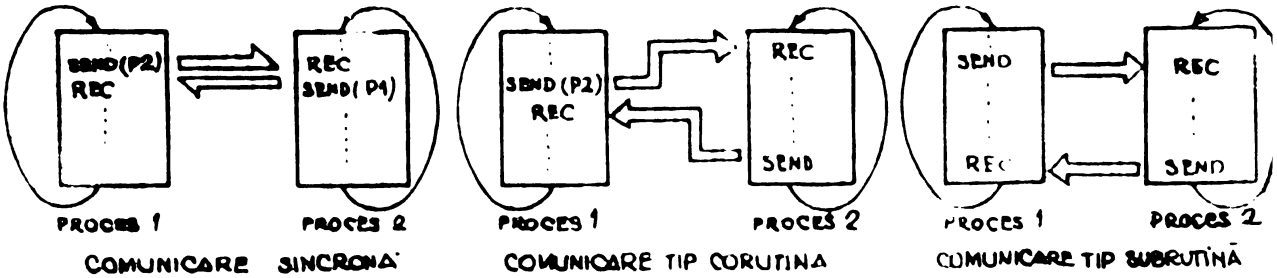


Fig.3.4.4. Comunicarea sincronă.

Din prima categorie se aminteşte:

M.3.4.4. Implementarea unui tip special de mesaj numit mesaj prioritar. Aceasta se poate realiza prin intermediul unei informaţ specifice care se inserează în conţinutul mesajului. După cum s-a precizat în § 3.4.5, fiecare proces consumator are asociat în calitate de „port" un buffer de comunicare în care mesajele se depun în ordinea sosirii. Pentru mesajul prioritar, această ordine nu este respectată; un astfel de mesaj se depune în capul bufferul

astfel încît el devine primul mesaj care va fi consumat. Sarcina testării informaţiei care precizează mesajul prioritar revine sistemului de operare, respectiv operatorului SEND, care trebuie să fie înzestrat cu facilităţile necesare rezolvării înlănţuirii amintite. Implementarea acestui tip de mesaj are o importanţă deosebită pentru STR în care pot să intervină evenimente excepţionale (alarme, defecţiuni, depăşiri de domeniu, etc.) care necesită o tratare urgentă.

Din categoria (b)de convenţii se aminteşte:

M.3.4.5. Precizarea prin convenţie a unor condiţii în textul mesajului, care testate fiind prin program, influenţează maniera de execuţie a procesului consumator. Se prezintă în continuare două posibilităţi în acest sens:

M.3.4.5.a. Stabilirea identităţii procesului emiţător, prin înserarea numelui acestuia în conţinutul mesajului, pe baza unei convenţii de proiectare sau de utilizare a SOTR. Aceasta permite spre exemplu selectarea mesajelor de către un proces receptor prin neglijarea celor care nu îi sînt destinate. De asemenea în acelaşi mod se poate implementa comunicarea prin numire directă. Dezavantajul metodei constă în pierderea mesajelor greşit destinate.

M.3.4.5.b. Realizarea mesajului cu răspuns, prin inserţia unei informaţii specifice în conţinutul mesajului. Prezenţa acestei informaţii crează prin convenţie următoarele obligaţii:

1° obligaţia ca procesul consumator să transmită un mesaj de răspuns procesului producător (emiţător)

2° obligaţia procesului producător de a se bloca într-un REC pînă la primirea mesajului de răspuns. Sarcina testării informaţiei specifice revine proceselor cooperante care trebuie să respecte obligaţiile stabilite prin convenţie.

E. Implementarea relaţiei de precedenţă. După cum s-a văzut în § 3.1 şi § 3.2 un rol important în realizarea sistemelor de procese timp-real revine relaţiei de precedenţă (de ordonare parţială) (§ 3.1) Mecanismul de comunicare propus permite implementarea simplă a acestei relaţii în forma unei perechi de operatori SEND-REC după cum urmează.

M.3.4.6. Pentru fiecare săgeată a grafului de precedenţă G al sistemului de procese, se înserează o pereche de operatori SEND-REC astfel: SEND-ul sufixează procesul de origine al săgeţii iar REC-ul prefixează procesul destinaţie (fig.3.4.5).

Probleme de implementare ridică sistemele care conţin procese partajate (§ 3.1). In vederea transformării acestor tipuri de sisteme în sisteme normale se poate utiliza metoda multiplicării proceselor .

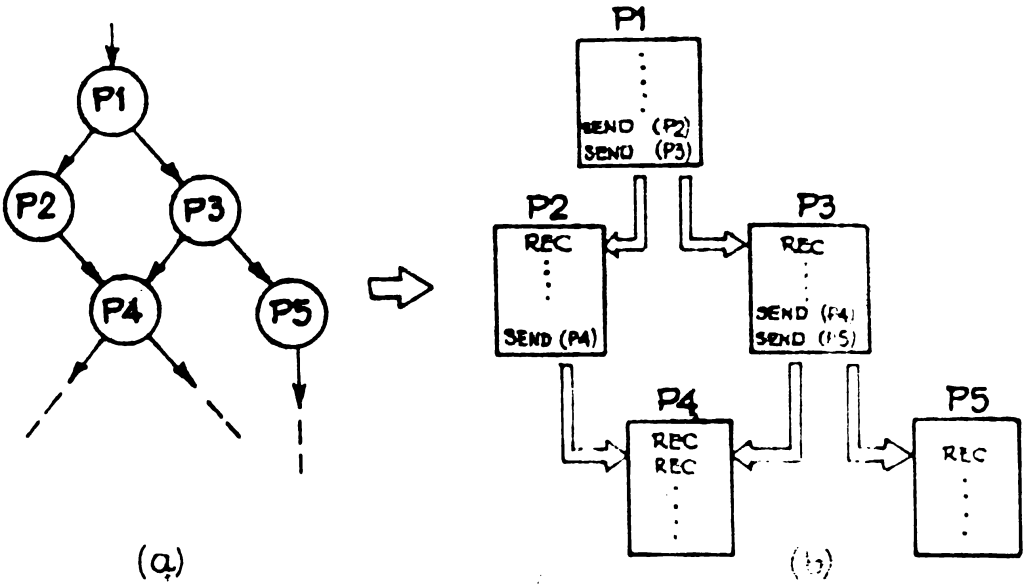


Fig.3.4.5. Implementarea relaţiei de precedenţă.

M.3.4.7. Metoda multiplicării presupune multiplicarea proceselor partajate astfel încît să fie rezolvate toate solicitările provenind din sistem. Implementarea multiplicării se poate realiza în două maniere.

a) multiplicarea propriu-zisă a procesului prin inserarea sub nume diferite a mai multor copii ale aceluiaşi proces (fig.3.4.6 Această tehnică este inerentă sistemelor multiprocesor. Se poate aplica şi sistemelor monoprocesor dacă condiţiile de viteză prescrise se impun cu necesitate acest lucru.

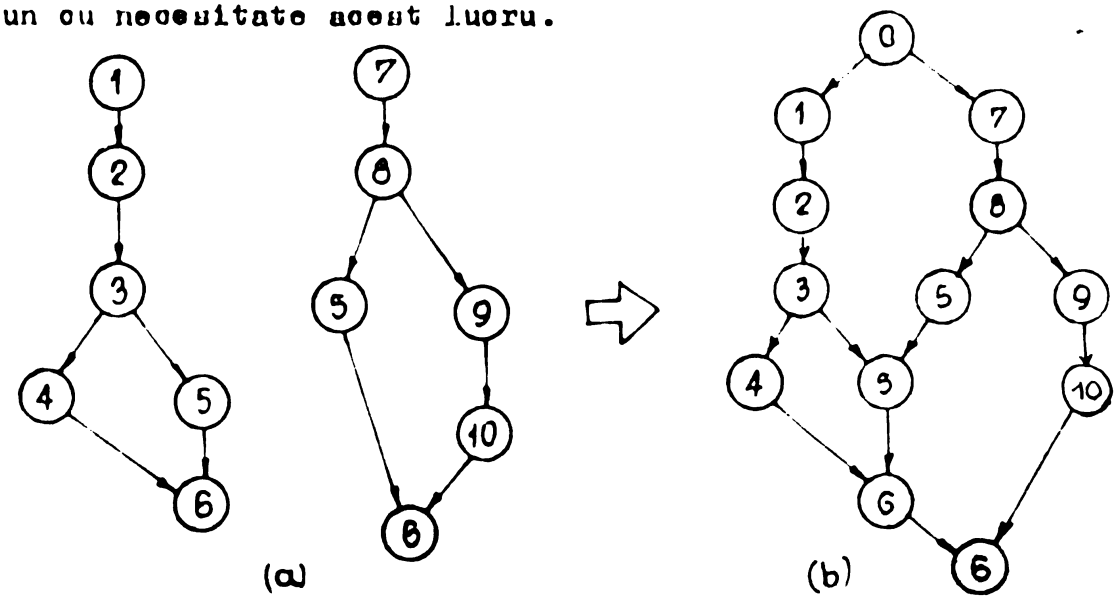


Fig.3.4.6. Multiplicarea proceselor prin replicare.

b) execuţia secvenţială a procesului partajat prin reluarea sa pentru fiecare solicitare în parte (fig.3.4.7). Aceasta presupune utilizarea informaţiei conţinută în mesaj în vederea influenţării manierei de execuţie a procesului.

Este posibil ca în funcţie de împrejurări să se aplice ambele tehnici şi chiar combinaţii ale acestora pentru acelaşi proces.

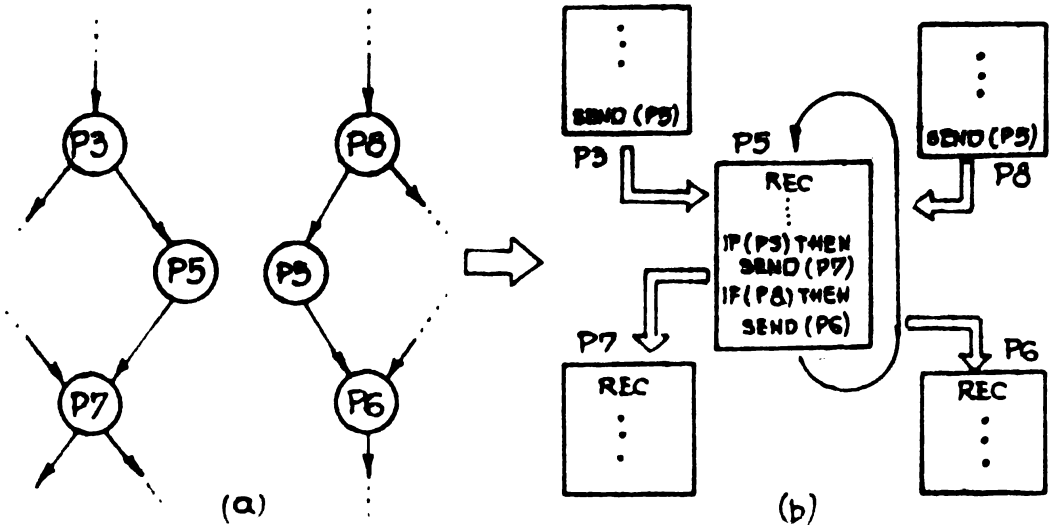


Fig.3.4.7. Multiplicarea proceselor prin reluare.

Cu aceste precizări se apreciază că mecanismul de cooperare propus reprezintă un instrument suficient de puternic pentru a rezolva problemele interacţiunilor din cadrul unor sisteme de procese concurente.

3.4.6. Extinderea mecanismului de cooperare propus pentru sisteme de calcul multiprocesor. Mesaje externe.

Mecanismul de cooperare propus poate fi extins astfel încît să permită coordonarea unor procese aparţinînd unor procesoare diferite prin intermediul unor mesaje numite externe. Această extindere se face pentru sisteme de calcul multiprocesor slab cuplate prin memorie comună (§ 2.2.3).

Extinderea se realizează în următoarea manieră.

M.3.4.8. Fiecărui /uP al sistemului i se asociază un port special situat în memoria comună. In acest port, se depun mesajele externe în ordinea sosirii lor, cu excepţia celor prioritare care se depun în faţă. In populaţia de procese a fiecărui /uP există un proces special aparţinînd SØ care are sarcina de a extrage mesajele din acest port specific al /uP în cauză şi de a|le expedia proceselor cărora le sînt adresate. Sincronizarea acestui proces ciclic se realizează prin intermediul unui semafor special aflat în memoria comună pe care el execută operaţii P iar cei care îi transmit mesaje operaţii V.

Acest mecanism este complet transparent din punctul de vedere al proceselor producător/consumator; toate elementele precizate, necesare în transmiterea mesajelor fiind îndeplinite de către SØ. Astfel toate cele precizate cu privire la posibilităţile de cooperare ale

mecanismului rămîn valabile şi pentru procese aparţinînd unor proce soare diferite.

Implementarea mesajelor externe presupune utilizarea semafoarelor speciale (§ 3.3.5) şi a mecanismului PHS (§ 3.3.3). In cadrul unui /uP, transmiterea efectivă a mesajelor între procesul sistem care distribuie mesajele şi procesele cărora le sînt destinate se realizează prin operatori SEND normali.

3.5. Planificarea proceselor

3.5.1. Planificarea în execuţie a proceselor independente. Discipline de planificare.

Planificarea în execuţie a proceselor se concretizează în activitatea de alocare a resursei unitate centrală (în cazul sistemelor monoprocesor) sau a unităţilor centrale (în cazul sistemelor multiprocesor). La un moment dat, un singur proces poate fi executat pe o unitate centrală şi se spune că aceasta este alocată procesului respectiv. Dacă mai multe procese independente solicită serviciile procesorului, este necesară planificarea execuţiei lor conform unei discipline de planificare, activitate care constă de fapt în administrarea unei (unor) cozi de aşteptare. Sarcina aceasta revine unui modul specializat al sistemului de operare numit dispecer. Acesta acţionează într-unul din următoarele momente:

1° Cînd un proces aflat în execuţie, semnalizează faptul că din anumite motive (interne sau externe) trebuie să-şi întrerupă activitatea.

2° La apariţia unui eveniment (terminare operaţie I/E, semnal extern, întrerupere) aşteptat de un proces care se află în starea blocat pe acel eveniment.

3° La epuizarea unei cuante de timp [MD74], [LD81].

Sarcinile dispecerului sînt:

1° Păstrarea evidenţei stărilor proceselor.

2° Implementarea unei politici de alocare a procesorului şi deci implicit a unei discipline de planificare în execuţie a proceselor.

3° Alocarea efectivă a procesorului, proceselor.

4° Dealocarea procesorului în momentele în care procesul curent nu-şi mai poate continua execuţia. Aceasta presupune reactualizarea stării procesului şi salvarea stării acestuia pentru o reluare ulterioară.

Dintre aceste elemente, un rol aparte revine disciplinelor de planificare în execuţie a proceselor despre care există o bogată literatură de specialitate [CD73],[Ha73],[Sh74],[TC78],[Su78],[RS79], [St80],[Do81],[CA82].

Politica de alocare a procesorului se referă la acele perioade din existenţa procesului, în care acesta dispune de toate resursele necesare începerii execuţiei sale şi devine pretendent la ocuparea UC (starea pregătit § 2.4.2). In acest moment, procesul este introdus în coada (cozile) de aşteptare,pentru planificarea ocupării procesorului. Administrarea cozii este făcută pe baza unor <u>informaţii apriorice</u> asupra procesului (timp de execuţie, prioritate) şi/sau pe baza unor informaţii istorice evaluate de sistem (măsurări, numărări de treceri etc.). Utilizarea acestor informaţii, combinată cu anumite opţiuni (aplicarea preemţiunii procesorului) sau restricţii de ocupare,permite obţinerea unei largi diversităţi de politici de alocare a procesorului şi implicit a unor diverse discipline de planificare în execuţie a procesoarelor independente [Pa82].

Se prezintă în continuare cîteva dintre disciplinele fundamentale de planificare urmate de o analiză critică a lor. In acest scop relativ la un proces, se defineşte noţiunea de <u>timp de trecere</u> (timp total) TT ca fiind intervalul de timp absolut scurs între momentul lansării în execuţie şi momentul terminării execuţiei procesului şi noţiunea de <u>timp de execuţie</u> TE ca fiind intervalul de timp necesar execuţiei procesului, dacă acesta ar ocupa singur unitatea centrală. Referitor la timpul de execuţie TE prezintă importanţă <u>timpul curent</u> TC - timpul de procesor de care a beneficiat efectiv un proces pînă la un moment dat şi <u>timpul restant</u> TR_e - intervalul de timp procesor de care mai are nevoie procesul pînă la terminarea sa. De asemenea prezintă interes şi numărul de procese aflate simultan în evidenţa distribuitorului la un moment dat notat cu NP (număr procese pregătite). De obicei valorile medii ale lui TT şi NP caracterizează performanţele unei discipline de planificare.

1° <u>Discipline bazate pe prioritate.</u> Prioritatea este un număr ataşat fiecărui proces, care prin valoarea sa reglementează ordinea de lansare în execuţie a mai multor procese aflate simultan în starea pregătit. Planificarea proceselor se poate realiza fie într-o singură coadă de aşteptare, din care se selectează pentru execuţie procesul cel mai prioritar, fie în mai multe cozi de aşteptare, caz în care lucrările dintr-o anumită coadă sînt luate în considerare numai în momentul în care toate cozile asociate priorităţilor mai mari sînt goale. Există

discipline de planificare cu prioritate fixă (prioritatea atribuită unui proces nu se poate modifica) sau variabilă (în funcţie de condiţiile concrete de planificare). Disciplinele bazate pe prioritate pot fi întreruptibile sau nu.

2° Discipline bazate pe termen („deadline"). Se aplică în acele cazuri în care terminarea execuţiei unor procese trebuie să aibă loc cu necesitate înainte de scurgerea unui interval prestabilit de timp numit de obicei dată limitată [Pa82], spre exemplu în aplicaţiile timp-real. Pentru a se implementa o astfel de disciplină sînt necesare informaţii apriorice asupra timpului de execuţie al procesului şi informaţii despre încărcarea sistemului. Cunoscînd astfel de informaţii se poate spre exemplu asocia procesului o prioritate variabilă care să crească pe măsură ce se apropie termenul. Se poate acorda de asemenea procesului în cauză o prioritate ridicată care să-i permită în orice condiţii încadrarea în timpul prestabilit.

NR.	DISCIPLINA	INTRERUPTIBILA	UTILIZEAZA INFORMATII APRIORICE
1	PRIORITATE	DA (NU)	NU (DA)
2	TERMEN	DA (NU)	DA
3	FIFO	NU	NU
4	LIFO	NU	NU
5	SPT	NU	DA
6	RR	DA	NU
7	PCQ	DA	NU (DA)
8	SET	DA	NU
9	SRPT	DA	DA

Fig.3.5.1. Discipline de planificare.

3° Discipline neîntreruptibile. În cadrul acestui tip de politici de alocare nu există posibilitatea întreruperii procesului aflat în execuţie în vederea rechiziţionării (preemţiunii) procesorului. Din momentul în care alocarea devine efectivă, unitatea centrală nu este eliberată decît după terminarea execuţiei procesului care a dobîndit-o. Există mai multe astfel de discipline.

a) Disciplina FIFØ (primul venit-primul servit) în care fiecare proces devenit executabil este introdus la sfîrşitul cozii de aşteptare, urmînd a fi planificat în execuţie, după ce toţi predecesorii săi, în timp, şi-au terminat execuţia.

b) Disciplina LIFØ (ultimul venit-primul servit), în care se administrează o coadă de tip stivă: ultimul proces introdus în coadă este cel care se va lansa primul în execuţie. Disciplina poate conduce uneori la blocarea nedefinită în această coadă a unor procese.

c) Disciplina SPT (shortest-processing time). Este o disciplin care favorizează lucrările scurte întrucît întotdeauna se alocă proce sorul procesului cu cel mai scurt timp de execuţie. Este o disciplină cu bune performanţe care necesită informaţii apriorice.

4° Discipline întreruptibile. In cadrul acestui tip de politi ci de alocare există posibilitatea întreruperii procesului în curs de execuţie în vederea locării procesorului unui alt proces. In acest ca un proces nu va fi executat într-o singură trecere prin UC ci în mai multe. Suspendarea execuţiei procesului poate avea loc la epuizarea unui interval precizat de timp procesor (numit cuantă), la apariţia unui eveniment prioritar (întrerupere) sau din motive legate de execu ţia procesului (absenţa unor resurse, sincronizări, etc.). Există mai multe astfel de discipline.

a) Disciplina carusel simplu RR (round-robin) în varianta sa numită de bază este acea disciplină în care procesorul se alocă pe durata unei cuante de timp (time-slice sau quantum) procesului urmă tor aflat în coada de aşteptare care este considerată circulară. Se garantează pentru această metodă că orice proces este servit într-un interval finit de timp, care însă nu depinde de timpii săi estimaţi de execuţie. Cuanta ia valori de ordinul milisecundelor şi este fixată funcţie de tipul şi obiectivele SŞ. Există şi variante mai elaborate ale planificării RR în care spre exemplu se modifică inter valul de timp procesor alocat unui proces, în funcţie de prioritatea sa, astfel încît un proces prioritar primeşte mai multe cuante decît un proces mai puţin prioritar.

b) Disciplina carusel-multiplu FCQ (Feedback cycle queues) [LD81] reprezintă o extensie a metodei precedente, prin definirea mai multor valori crescătoare pentru cuanta de timp. Pentru fiecare cuantă de timp se administrează o coadă circulară. Un proces care solicită procesor este introdus iniţial în coada corespunzătoare cuantei celei mai reduse şi pe măsură ce este replanificat, este trecut pe rînd în cozile corespunzătoare unor cuante crescînde. Procesele aparţinînd unei cozi nu sînt planificate în execuţie pînă cînd toate cozile corespunzătoare unor cuante mai mici decît cea corespunzătoare cozii curente sînt vide. Procesele foarte lungi, ajunse în coada corespunzătoare cele mai mari cuante, dacă nu s-au terminat revin în această coadă pînă la terminare.

e) Disciplina SET (shortest elapsed time) este disciplina conform căreia procesorul se alocă procesului cu timpul curent (TC) minim. Dacă mai multe procese au acelaşi timp curent minim, se

rulează doar aceste procese în RR. Disciplina favorizează lucrări scurte.

d) Disciplina SRPT (shortest-remaining processing time) planifică spre execuţie procesul cu timpul restant TR_eminim, scop în care are nevoie de informaţii apriorice (valoarea timpului de execuţie TE). Un proces rulează practic pînă la terminare. La apariţia unei IT care marchează apariţia unui eveniment aşteptat şi deci apariţia unui nou proces pregătit, procesul curent nu este selectat în continuare numai dacă timpul total de execuţie al noului proces este mai mic ca TR_eal procesului în curs.

Un tabel al disciplinelor prezentate apare în fig.3.5.1. Analiza comparativă a acestor discipline conduce la următoarele concluzii [CD73] :

1° Disciplina SRPT este cea mai pretenţioasă (necesită informaţie apriorică şi este întreruptibilă) în schimb oferă cele mai bune performanţe. S-a demonstrat că atît TT mediu cît şi NP mediu sînt minime pentru această disciplină. Totuşi ea are un dezavantaj şi anume: pot apare întreruperi foarte dese, dacă apar procese din ce în ce mai scurte. Din acest motiv se utilizează variante ale disciplinei SRPT în care:

1° se limitează numărul de IT admise în unitatea de timp,

2° se maximalizează durata pentru care sînt luate în considerare întreruperile.

2° Dintre disciplinele neîntreruptibile, SPT oferă cele mai bune performanţe. Dezavantajul ei constă în faptul că necesită informaţie apriorică (TE), lucru care pentru SPTR dedicate nu reprezintă însă un impediment.

Referitor la disciplinele care nu necesită informaţii apriorice se pot trage următoarele concluzii.

3° Dacă timpii de execuţie TE ai proceselor independente sînt apropiaţi ca şi valoare în acest caz disciplina FIFO este cea mai avantajoasă. RR şi SET nu se comportă bine deoarece toate procesele se execută cvasisimultan, toate se încep, toate sînt reţinute şi toate se termină aproximativ odată. Datorită acestui fapt rezultă un TT mediu mai mare ca la FIFO.

4° Dacă însă dispersia timpilor de execuţie ai proceselor este mare, atunci FIFO se comportă prost producînd atît TT cît şi NP relativ mari. In aceste condiţii RR oferă performanţe mai bune.

3.5.2. Particularităţi ale planificării proceselor în SØTR

Disciplinele de planificare prezentate s-au referit în special la procesele independente. Intrucît SØTR nu le sînt caracteristice decît într-o măsură mai restrînsă procesele independente, şi utilizarea acestor discipline ca atare este mai restrînsă. Pentru a aborda planificarea proceselor în acest context, se vor preciza cîteva particularităţi ale SØTR.

1. In sistemele timp-real dedicate unor aplicaţii există o populaţie fixă de procese, care rămîne nemodificată în timp. In sistem nu se creează şi nu se distrug procese, ele doar se reiau ciclic.

2. Procesele unei aplicaţii TR se află într-una din stările activ, pregătit sau blocat după cum s-a precizat în § 2.4.2.

3. Intre procese există relaţii de constrîngere rezultate din condiţii de proiectare, de determinare (§ 3.2), de coordonare (§ 3.3), de cooperare (§ 3.4) sau de interblocare (§ 3.6).

Datorită acestor constrîngeri, un sistem de procese pentru o aplicaţie TR, se prezintă în general ca avînd o structură de graf închis (§ 3.1).

4. Intr-un sistem TR dedicat, procesele ce intervin fiind cunoscute, se pot determina cu precizie timpii de execuţie (TE) ai proceselor, astfel încît activitatea de planificare poate dispune de informaţii apriorice.

5. Datorită structurii fixe a proceselor, în astfel de sisteme, în timp, se pot culege şi informaţii istorice (frecvenţe de apel, număr de treceri, grad de utilizare) care pot să servească la creşterea eficienţei planificării.

Problema care se pune poate fi formulată astfel: avînd o structură fixă de procese, să se realizeze o astfel de planificare încît respectînd constrîngerile lor interne de precedenţă să rezulte un timp total de planificare minim. In acest scop se pot utiliza informaţii apriorice şi informaţii istorice.

Pentru sistemele monoprocesor planificarea minimă constă din execuţia secvenţială a proceselor (într-o anumită ordine) astfel încît timpul minim de execuţie pentru o trecere prin structura de graf este egal cu suma timpilor de execuţie ai proceselor. Ceea ce se poate realiza eventual printr-o planificare corespunzătoare se referă la obţinerea unor timpi de trecere mai reduşi pentru anumite procese. Totuşi şi pe sisteme monoprocesor există rezerve pentru reducerea acestui timp minim de planificare prin suprapunerea activităţii în unitatea centrală cu operaţiile periferice (I/E), element ce va fi analizat în paragraful următor.

Pentru sistemele multiprocesor există o bogată metodologie de planificare a execuţiei proceselor care se bazează pe migraţia proceselor [CD73]. Pentru structura sistemului multiprocesor preconizat în lucrare pentru aplicaţii timp-real (§ 2.2.3) aceşti algoritmi nu prezintă un interes deosebit din următoarele motive:

1° Aceste sisteme au o structură fixă, procesele fiind prerepartizate pe procesoare. In această prerepartizare statică ar putea fi luaţi eventual în considerare algoritmii de planificare pentru sisteme multiprocesor însă, de regulă, ea este legată mai mult de funcţiile pe care trebuie să le îndeplinească μP respectiv, de locul fizic pe care acesta îl ocupă, de poziţia sa ierarhică în arhitectura sistemului, etc.

2° Intr-un astfel de sistem nu există posibilitatea migrării proceselor, decît eventual în condiţii excepţionale.

3° Influenţele reciproce interprocese se reduc la activităţile de coordonare şi cooperare care se realizează prin memoria comună şi IT.

In concluzie într-un astfel de sistem multiprocesor, activitatea de planificare se restrînge la planificarea pe mai multe sisteme monoprocesor, în sensul definit anterior cu precizarea că această planificare poate fi afectată în plus de constrîngeri exterioare sistemului.

Un ultim aspect care trebuie precizat se referă la întreruperi. Restrîngînd planificarea la sisteme monoprocesor, principial IT nu pot influenţa timpul de planificare, decît în mod negativ, mărind acest timp prin timpul necesar rezolvării lor. Datorită însă faptului că se lucrează într-o ambianţă TR din care IT nu pot fi excluse (externe, operaţii I/O, semnale interprocesoare, etc.) ele trebuie avute în vedere şi acceptate. In concluzie disciplinele de planificare implementate trebuie să fie <u>întreruptibile</u>.

3.5.3. <u>Model pentru studiul planificării sistemelor închise de procese. Concluzii</u>.

După cum s-a precizat o sursă de reducere a timpului de planificare în sistemele monoprocesor o constituie maximalizarea suprapunerii timpului de execuţie al procesorului cu cel al execuţiei operaţiilor I/E [RF76],[TC78],[Ro79].

Spre exemplu pentru structura de graf prezentată în figura 3.5.2(a), ale cărei caracteristici apar în aceeaşi figură (b) execuţia serială normală conduce la o planificare de lungime 242, în timp de planificarea optimă cu suprapunere conduce la 165 unităţi de timp, deci cu 32 % mai redusă.

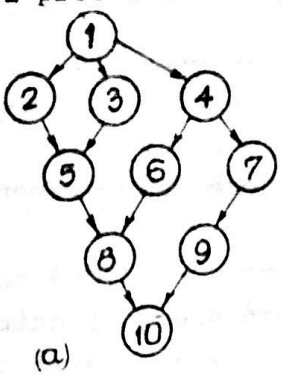
In continuare se va prezenta un model realizat de autor pentru studiul planificării sistemelor închise de procese pe sisteme monoprocesor. Fiecare proces se presupune segmentat în două părţi: partea de instrucţii executabilă pe microprocesor şi partea de I/E fiecare necesitînd pentru execuţie un anumit interval de timp. Pentru a reprezenta grafic diferitele posibilităţi de planificare se vor utiliza diagrame Gantt în care deasupra axei timpului se reprezintă timpul procesor iar dedesubt timpul I/E. Porţiunile haşurate marchează perioade de inactivitate [CD73] (fig.3.5.2(c)).

Se presupun cunoscute informaţii apriorice respectiv timpii procesor şi timpii I/E ai proceselor.

In planificare se iau în considerare următoarele observaţii:

1° Un proces nu se poate planifica pentru execuţie pînă ce toţi predecesorii săi s-au terminat (o 3.5.1).

2° Segmentul I/E al unui proces se poate executa numai după ce segmentul procesor corespunzător aceluiaşi proces s-a încheiat. (o 3.5.



(a)

PROCES	TIMP PROCESOR	TIMP I/E	NR. SUCC	SUCC 1	SUCC 2	SUCC 3
1	3	7	3	2	3	4
2	19	13	1	5	-	-
3	8	6	1	5	-	-
4	14	9	2	6	7	-
5	10	14	1	8	-	-
6	11	14	1	8	-	-
7	10	16	1	9	-	-
8	16	9	1	10	-	-
9	10	17	1	10	-	-
10	17	19	0	-	-	-

(b)

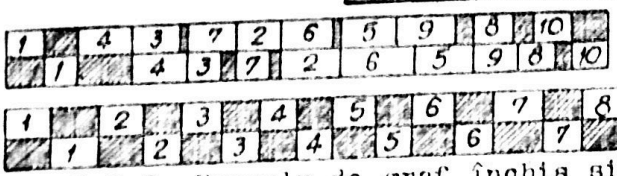


Fig.3.5.2. Exemplu de graf închis şi planificările optimală şi serială aferente.

Datele care se furnizează la intrarea modelului sînt cele din fig.3.5.2.(b) şi ele constau din structura grafului proceselor şi timpii fiecărui proces.

Modelul implementat în limbajul PASCAL, furnizează la ieşire timpii corespunzători diferitelor metode de planificare permiţînd studiul comparativ al acestora. Metodele se implementează ca şi proceduri ale modelului. Baza de date utilizată apare în fig.3.5.3. In continuare se vor aprofunda cîteva aspecte specifice.

A. Determinarea timpului este realizată de o rutină specializată a modelului numită TIMP (fig.3.5.3). Timpul sistemului se caracterizează prin două valori:

```
CONST  NL=^(:10:)^; MAXSUCC=3;MAXPROC=32;
TYPE NOD=RECORD
           TIMP_PROC   :INTEGER;
           TIMP_IO     :INTEGER;
           NR_PRED     :INTEGER;
           NR_PRED_REZ:INTEGER;
           NR_SUCC     :INTEGER;
           INC_PROC    :INTEGER;
           SF_PROC     :INTEGER;
           INC_IO      :INTEGER;
           SF_IO       :INTEGER;
           TTERM_PRED  :INTEGER;
           SUC : ARRAY(.1..MAXSUCC.) OF INTEGER
         END;
     LISTA=ARRAY(.0..MAXPROC.) OF INTEGER;
VAR GRAF:ARRAY(.1..MAXPROC.) OF NOD;
    SECV,SECV_MIN:ARRAY(.1..MAXPROC.) OF INTEGER;
    KSECV,NSECV,NSECV_MIN,NRPROC:INTEGER;
    INCEPUT,SFIRSIT,J:INTEGER;
    TPROC_CRT,TPROC_MIN,TIO_CRT,TIO_MIN:INTEGER;
    NIV:ARRAY(.1..MAXPROC.) OF LISTA;
    LST,LST1:LISTA;
PROCEDURE TIMP( NC:INTEGER);
VAR T,T1,NS,I:INTEGER;
BEGIN
  T:=MAX(GRAF(.NC.).TTERM_PRED,TPROC_CRT);
  GRAF(.NC.).INC_PROC:=T;
  TPROC_CRT:=T+GRAF(.NC.).TIMP_PROC;
  GRAF(.NC.).SF_PROC:=TPROC_CRT;
  T1:=MAX(TPROC_CRT,TIO_CRT);
  GRAF(.NC.).INC_IO:=T1;
  TIO_CRT:=T1+GRAF(.NC.).TIMP_IO;
  GRAF(.NC.).SF_IO:=TIO_CRT;
    FOR I:=1 TO GRAF(.NC.).NR_SUCC DO
      BEGIN
        NS:=GRAF(.NC.).SUC(.I.);
        GRAF(.NS.).TTERM_PRED:=TIO_CRT
      END;
END;
```

Fig.3.5.3. Baza de date şi procedura de determinare a timpilor.

- TPRØC - CRT - timpul procesor corespunzător ultimului proces planificat

- TIØ - CRT - timpul I/E corespunzător aceluiaşi proces. Această valoare reprezintă de fapt valoarea timpului planificării la momentul considerat şi reprezintă timpul de trecere TT al ultimului proces planificat.

Pentru procesul care se planifică, se iau în considerare TIMP PRØC (timpul procesor), TIMP-IØ (timpul I/E) şi TTERM-PRED (timpul de terminare al ultimului procesor). Se precizează că:

TIMP-PRØC + TIMP-IØ = TE (timpul de execuţie).

Presupunînd că procesul NC a fost planificat în execuţie, procedura TIMP realizează următoarele funcţii:

1° Determină noua valoare a lui TPRØC-CRT conform relaţiei:

$$\text{TPRØC-CRT} := \max\{\text{TPRØC-CRT}, \text{TTERM-PRED}\} + \text{TIMP-PRØC}$$

2° Determină noua valoare a lui TIØ-CRT pe baza relaţiei:

$$\text{TIØ-CRT} := \max\{\text{TPRØC-CRT}, \text{TIØ-CRT}\} + \text{TIMP-IØ}$$

3° Actualizează cîmpurile TTERM-PRED ale succesorilor procesului planificat (NC) cu valoarea TIØ-CRT corespunzătoare terminării sale.

Procedura TIMP se apelează ori de cîte ori un proces este selectat pentru execuţie.

B. Planificarea optimă. Pentru a determina planificarea optimă (cu timp de planificare minim) a fost realizată o rutină care generează în mod recursiv toate secvenţele de planificare valide posibile pentru un sistem specificat de procese ţinînd cont de cele precizate anterior (o 3.5.1, o 3.5.2). Pentru a realiza acest lucru, rutina se bazează pe conceptul de nivel definit în § 3.1; pentru fiecare nivel se păstrează o listă a proceselor aparţinătoare (NIV). Procedura primeşte ca date de intrare numărul procesului şi nivelul pe care acesta se găseşte; în funcţie de structura grafului se parcurg toate posibilităţile de planificare completîndu-se în mod dinamic lista nivelurilor. Pentru fiecare planificare se determină timpul total şi în final se reţine planificarea minimă. Implementarea acestei proceduri apare în fig.3.5.4.

```
PROCEDURE GEN(KLISTA, KNIV: INTEGER);
VAR NC,KN,KNIV1,LGLC,I,NS,K,J:INTEGER;
BEGIN
  KSECV:=KSECV+1;NC:=NIV(.KNIV,KLISTA.); LGLC:=NIV(.KNIV,0.);
  TIMP(NC);SECV(.KSECV.):=NC;KN:=0;KNIV1:=KNIV+1;
  FOR I:=1 TO LGLC DO
    IF NIV(.KNIV,I.) <> NC THEN
      BEGIN
        KN:=KN+1;NIV(.KNIV1,KN.):=NIV(.KNIV,I.);
        NIV(.KNIV1,0.):=KN
      END;
  IF GRAF(.NC.).NR_SUCC <> 0 THEN
    BEGIN
      FOR I:=1 TO GRAF(.NC.).NR_SUCC DO
        BEGIN
          NS:=GRAF(.NC.).SUC(.I.);
          GRAF(.NS.).NR_PRED_REZ:=GRAF(.NS.).NR_PRED_REZ+1;
          IF GRAF(.NS.).NR_PRED_REZ = GRAF(.NS.).NR_PRED THEN
            BEGIN
              KN:=KN+1;NIV(.KNIV1,KN.):=NS;NIV(.KNIV1,0.):=KN
            END
        END;
    END;
  IF KN <> 0 THEN
    BEGIN
      FOR K:=1 TO KN DO
        GEN(K,KNIV1)
  END ELSE
      BEGIN
      IF TIO_CRT<TIO_MIN THEN
        BEGIN
          NSECV_MIN:=NSECV+1;
          FOR I:=1 TO NRPROC DO
            SECV_MIN(.I.):=SECV(.I.);TIO_MIN:=TIO_CRT;
            TPROC_MIN:=TPROC_CRT
        END;
      END;
  IF GRAF(.NC.).NR_SUCC <> 0 THEN
      BEGIN
        FOR I:=1 TO GRAF(.NC.).NR_SUCC DO
          BEGIN
            NS:=GRAF(.NC.).SUC(.I.);
            GRAF(.NS.).NR_PRED_REZ:=GRAF(.NS.).NR_PRED_REZ - 1
          END;
      END;
      KSECV:=KSECV-1;
      IF KSECV>0 THEN
        BEGIN
          NC:=SECV(.KSECV.);TIO_CRT:=GRAF(.NC.).SF_IO;
          TPROC_CRT:=GRAF(.NC.).SF_PROC
        END;
END;
```

Fig.3.5.4. Procedura pentru determinarea planificării optime.

Procedura deşi este rapidă, este suficient de laborioasă pentru a nu putea fi utilizată în planificarea dinamică ("on line"). Ea este concepută pentru a fi utilizată „off line" avînd drept scop identificarea planificării minime ca şi criteriu de comparaţie pentru celelalte metode de planificare. Rezultatele furnizate de ea prezint însă interes în concepţia şi proiectarea algoritmului de planificare pentru o aplicaţie reală după cum se va vedea ulterior.

In continuare se prezintă cîteva discipline de planificare mai puţin laborioase, care pot fi utilizate în regim dinamic şi ale căror performanţe vor fi raportate la planificarea optimă.

C. Planificarea P1. Pornind de la condiţiile date ale modelului se observă că pentru a obţine o planificare cît mai redusă este necesar:

1° Să se selecteze cu prioritate în execuţie, procesele care au cel mai scurt timp procesor, astfel încît succesorii acestora să devină planificabili cît mai curînd.

2° Ca un proces pregătit, cu timpul I/E minim să fie planificat cît mai tîrziu (relativ la celelalte procese pregătite pentru execuţie) întrucît timpul său I/E va influenţa într-o masură redusă timpul total de execuţie al acestui set de procese.

Luînd în considerare aceste observaţii se propune următorul algoritm:

A.3.5.1. 1° Se trece procesul iniţial în LST1 (lista de lucru)

2° Se planifică primul proces din LST1 spre execuţie (se trece în lista SECV-care conţine secvenţa proceselor planificate)

3° Pentru procesul selectat, toţi succesorii planificabili (care au toţi predecesorii terminaţi) se trec în lista de lucru LST.

4° Se reorganizează lista LST în LST1 conform subalgoritmului:

S.A.3.5.1. (ORDØNARE 1)

a) Se alege procesul cu cel mai scurt timp TIMP-PRØC şi se trece în capul listei LST1.

b) Se alege procesul cu cel mai scurt timp TIMP-IØ şi se pune în coada listei LST1.

c) Se repetă punctele a) şi b) pînă la epuizarea proceselor listei LST. Procesele selectate se pun în urma, respectiv înaintea celor anterioare.

5° Se reia algoritmul de la 2° pînă la epuizarea tuturor proceselor.

D. Planificarea P2. In metoda prezentată după ce este selectat procesorul cu cel mai scurt timp procesor, procesul cu cel mai scurt timp e plasat la sfîrşit. Se poate însă întîmpla ca acest proces să aibă un timp procesor chiar mai scurt decît timpul său IE, caz în care este dificil de apreciat dacă e mai bine ca acest proces să fie plasat la începutul sau la sfîrşitul listei. Pornind de la aceste observaţii se propune următorul algoritm:

A.3.5.2. Acest algoritm este asemănător cu A.3.5.1 cu excepţia pasului 4° care se modifică astfel:

SA 3.5.2. (ORDØNARE 2)

a) Se alege procesul cu cel mai scurt timp. Dacă timpul cel mai scurt este un timp procesor, procesul corespunzător se trece în capul listei (LST1). Dacă timpul cel mai scurt este un timp IE procesul se trece în coada listei.

b) Se reia punctul a) pînă la epuizarea proceselor din LST. Procesele selectate se pun în urma respectiv înaintea celor selectate anterior.

Se propun în continuare alţi algoritmi de planificare, care însă spre deosebire de P1 şi P2 care nu iau în considerare nivelurile ierarhice reluînd ordonarea după fiecare proces, intervin doar la trecerea de pe un nivel ierarhic pe altul. Datorită acestui fapt, aceşti algoritmi sînt mai economici şi după cum se va vedea în continuare şi mai eficienţi.

E. Planificarea P3 constă în aplicarea planificării P1 pe nivele ierarhice conform următorului algoritm:

A.3.5.3.

1° Se pune procesul iniţial în LST1.

2° Se selectează primul proces din LST1, iar succesorii săi planificabili se trec în LST care cuprinde procesele nivelului următor.

3° Se ordonează procesele nivelului următor din lista LST conform subalgoritmului SA 3.5.1 şi se obţine LST1.

4° Se selectează spre execuţie procesele ordonate ale nivelului următor (LST1), care devine astfel nivelul curent. Pe măsură ce sînt selectate procesele, succesorii lor planificabili se trec în LST, pregătindu-se nivelul următor.

5° Se reia pasul 3° pînă cînd LST devine vidă (nu mai există succesori).

F. Planificarea P4 constă în aplicarea planificării P2 pe nivele ierarhice. Aceasta se poate realiza cu ajutorul algoritmului următor:

A.3.5.4. Acest algoritm coincide cu A.3.5.3 cu excepţia pasului 3° în care subalgoritmul de ordonare va fi SA 3.5.2.

G. Planificarea P5. Metodele prezentate, presupun algoritmi de sortare pentru a realiza ordonarea. In aplicaţiile timp-real, regia acestei ordonări poate fi însă prohibitivă. Din acest motiv, o metodă simplă de planificare, care înlătură acest dezavantaj, consă în selectarea proceselor din nivelul curent de la stînga la dreapta (în ordinea în care sînt introduse în LST). După ce a fost selectat ultimul proces din dreapta nivelului curent se trece la procesul din stînga nivelului următor, pînă la epuizarea tuturor proceselor.

Un exemplu de implementare al unei discipline de planificare (P3) apare în fig.3.5.5 (procedura METODA-A).

Modelul a fost utilizat în studiul unor tipuri de structuri caracteristice de procese cu timpi proces şi IE aleatori. Pentru un grup de încercări, tabelul comparativ al performanţelor metodelor de planificare analizate apare în fig.3.5.6. Se precizează că abaterile sînt calculate în procente relative la planificarea optimă. Din rezultatele obţinute experimental se pot deduce următoarele:

1° Planificarea optimă este superioară celei aleatoare cu aproximativ 29 %.

2° Planificarea restrînsă la un singur nivel ierarhic este superioară celei extinse pe mai multe niveluri (abateri medii de 1-3 % faţă de 13-16 %).

3° Metoda P5 obţine rezultate comparabile cu metodele P3 şi P4, fără însă a implica nici o sortare. Dispersia abaterii faţă de planificarea optimă este însă mai mare, deoarece depinde de numerotarea proceselor care se face aleator; ordinea de planificare este ordinea în care procesele apar de la stînga la dreapta în nivelul ierarhic.

H. Concluzii. Pornind de la cele prezentate, se pot formula următoarele concluzii.

1° In SOTR se impune, acolo unde este posibil, suprapunerea activităţii procesorului cu activitatea dispozitivelor periferice.

2° Se pot utiliza mai multe metode de planificare. Astfel:

M.3.5.1. Dacă se dispune de informaţie apriorică se recomandă disciplinele de planificare dinamică P3 şi P4, a căror adîncime de planificare se extinde la un nivel ierarhic, care obţin performanţe apropiate de planificarea optimă.

M.3.5.2. Dacă informaţia apriorică se obţine dificil, se recomandă disciplina P5 care obţine rezultate bune, afectate însă de o dispersie mai mare.

```
PROCEDURE TREC_SUC(VAR J:INTE&ER;SECV:LISTA;VAR LST:LISTA);
VAR I,K,NS,NC:INTEGER;
BEGIN
  FOR I:=INCEPUT TO SFIRSIT DO
    BEGIN
      NC:=SECV(.I.);
      FOR K:=1 TO GRAF(.NC.).NR_SUCC DO
        BEGIN
          NS:=GRAF(.NC.).SUC(.K.);
          GRAF(.NS.).NR_PRED_REZ:=GRAF(.NS.).NR_PRED_REZ+1;
          IF &RAF(.NS.).NR_-RED_REZ=&RAF(.NS.).NR_-RED THEN
            BEGIN
              J:=J+1;LST(.J.):=NS
            END;
        END;
    END;
END;
PROCEDURE ORDONARE1(J:INTEGER;VAR LST,LST1:LISTA);
VAR M,N,L,NC1,NSEL,TMINP,TMINIO:INTEGER;
    KK:INTEGER;
BEGIN
  M:=1;N:=J;TMINP:=30000;TMINIO:=30000;
  REPEAT
    NSEL:=0;TMINP:=30000;TMINIO:=30000;
    FOR L:=1 TO J DO
      BEGIN
        NC1:=LST(.L.);
        IF NC1<>0 THEN
          IF GRAF(.NC1.).TIMP_PROC<TMINP THEN
            BEGIN
              TMINP:=GRAF(.NC1.).TIMP_PROC;
              NSEL:=NC1;KK:=L
            END;
      END;
    IF NSEL<>0 THEN
    BEGIN
      LST1(.M.):=NSEL;LST(.KK.):=0;M:=M+1
    END;
    NSEL:=0;
    FOR L:=1 TO J DO
      BEGIN
        NC1:=LST(.L.);
        IF NC1<>0 THEN
          IF GRAF(.NC1.).TIMP_IO<TMINIO THEN
            BEGIN
              TMINIO:=GRAF(.NC1.).TIMP_IO;
              NSEL:=NC1;KK:=L
            END;
      END;
    IF NSEL <> 0 THEN
    BEGIN
      LST1(.N.):=NSEL;LST(.KK.):=0;N:=N-1
    END;
  UNTIL((M-N)=1)
END;
PROCEDURE METODA_A;
VAR I:INTEGER;
BEGIN
  WHILE SFIRSIT<=NRPROC DO
    BEGIN
      J:=0;TREC_SUC(J,SECV,LST);
      ORDONARE1(J,LST,LST1);
      INCEPUT:=SFIRSIT+1;
      FOR K:=1 TO J DO
        BEGIN
          SFIRSIT:=SFIRSIT+1 ;
          SECV(.SFIRSIT.):=LST1(.K.);
          TIMP(SECV(.SFIRSIT.))
        END;
    END;
END;
```

Fig.3.5.5. Exemplu de implementare al unei discipline de planificare.

M.3.5.3. Cu bune rezultate se pot utiliza disciplinele de planificare bazate pe prioritate statică. Utilizînd informații apriorice, cu

ajutorul modelului prezentat se determină planificarea optimă. Priorităţile statice se stabilesc conform ordinii proceselor în această planificare. Aceste priorităţi pot fi ulterior corectate pe baza informaţiilor istorice culese.

METODA	INCERCARI (%)						ABATERE MEDIE (%)
	1	2	3	4	5	6	
SERIALA	32	24	29	29	32	32	29,6
P1	16	1,4	1,6	5,4	21	24	13,9
P2	18	0	15	14	26	27	16,6
P3	0,6	1,4	1,7	0	0,6	0,6	0,8
P4	1,2	3,6	3,4	5,4	4,8	1,2	3,2
P5	6	0	0,5	1	1,8	1,2	1,7

Fig.3.5.6. Tabel comparativ al performanţelor metodelor de planificare.

In funcţie de particularităţile concrete ale unei aplicaţii TR, se pot utiliza şi alte discipline de planificare derivate direct din disciplinele fundamentale prezentate sau realizînd combinaţii ale acestora.

In implementare şi în aprecierea performanţelor, trebuie ţinut cont de faptul că disciplinele de planificare studiate sînt caracterizate de două componente:

1° Componenta statică (deterministă) în cadrul căreia planificarea poate fi prevăzută pe baza informaţiei apriorice şi a structurii grafului de precedenţă asociat sistemului de procese.

2° Componenta dinamică (asincronă) guvernată de interfaţa cu mediul exterior care este în general greu de anticipat. Elementele acestei componente pot fi cunoscute în timp pe baza informaţiilor istorice culese pe parcursul funcţionării sistemului, sau din funcţionarea unor sisteme sau aplicaţii similare. Ţinînd cont de aceste observaţii se recomandă următoarea metodologie.

M.3.5.4. 1° Se stabileşte şi se implementează una din disciplinele de planificare anterior precizate.

2° Se introduc în sistem elemente care înregistrează informaţii istorice (număr de apeluri, intervale de apariţie a întreruperilor externe, număr de treceri prin anumite puncte, etc.).

3° Pe baza informaţiilor istorice se pot aduce corecturi metodei iniţiale de planificare, urmărind optimizarea unuia sau mai multor parametri (criterii) ai sistemului.

3.6. Interblocarea proceselor

In cadrul sistemelor timp-real o importanţă aparte trebuie acordată interblocării proceselor. Un sistem constînd din mai multe procese concurente este interblocat, dacă evoluţia proceselor sale este blocată un timp nedefinit deoarece o parte din aceste procese

păstrează resurse aşteptate de celelalte [CD73], [MD74], [De81], [Pa82].

Considerînd numărul de instrucţii executate ca măsură a evoluţiei unui proces, pentru două procese concurente P1 şi P2 se poate reprezenta grafic curba de evoluţie relativă. Această curbă are un aspect discontinuu sau continuu după cum cele 2 procese se execută întreţesut (prin multiprogramare) pe o aceeaşi unitate centrală sau pe două unităţi centrale distincte (fig.3.6.1 a şi b).

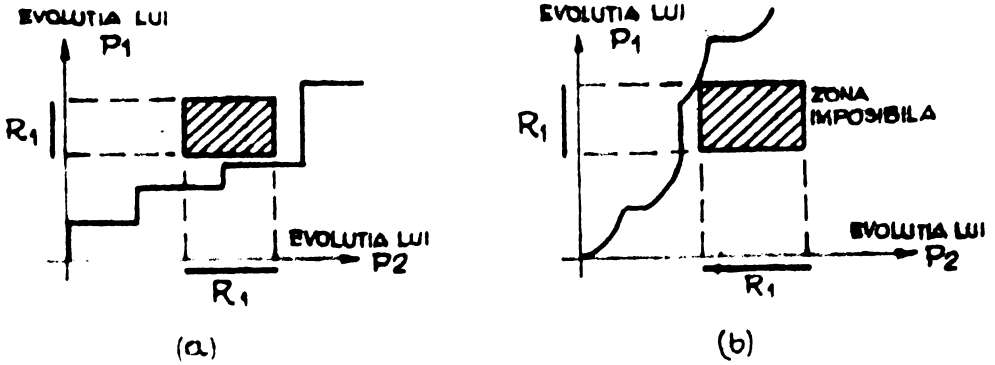


Fig.3.6.1. Curba de evoluţie relativă a două procese.

Dacă cele două procese au nevoie de o aceeaşi resursă R_1, intersecţia zonelor în care P_1 respectiv P_2 solicită această resursă generează o „zonă imposibilă" care nu poate fi traversată de către curba de evoluţie. Pornind de la această curbă în figura 3.6.2(a) este reprezentată o situaţie de interblocaj, în care procesele P_1 şi P_2 solicită pe rînd resursele R_1 şi R_2 respectiv R_2 şi R_1. Dacă curba de evoluţie intră în zona N (zona nesigură) interblocajul este iminent din cauza ireversibilităţii evoluţiei (P_1 deţine pe R_1 şi necesită pe R_2 iar P_2 deţine pe R_2 şi necesită pe R_1). Se observă însă că dacă resursele sînt solicitate în aceeaşi ordine de către ambele procese, interblocajul nu poate să apară.

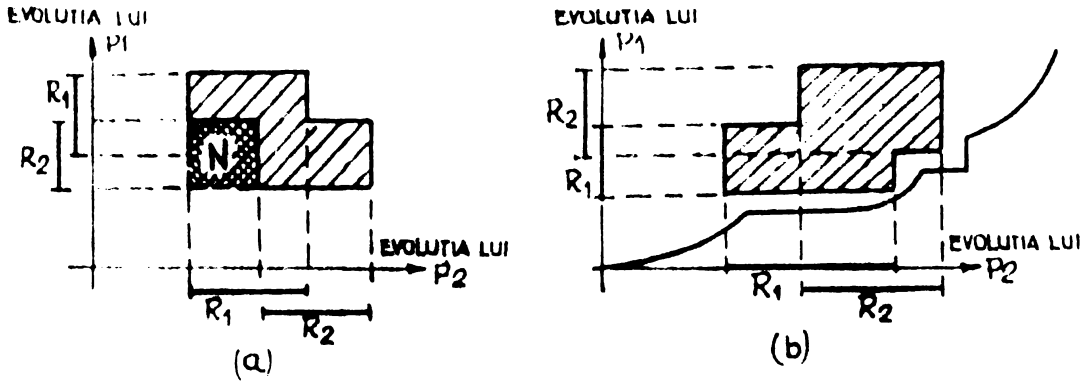


Fig.3.6.2. Reprezentarea interblocajului.

Pentru ca regiunile nesigure să existe sînt necesare trei condiţii: (1) exoluziunea mutuală (fiecare proces deţine controlul exclusiv asupra resurselor pe care le utilizează), (2) resurse nepreemtibile (un proces nu eliberează resursele pe care le deţine decît după ter

minarea utilizării lor) şi (3) aşteptarea ciclică de resurse (fiecare proces păstrează resurse în timp ce aşteaptă să i se elibereze celelalte resurse de care are nevoie pentru a continua.

3.6.1. Model generalizat pentru studiul stării resurselor unui sistem de procese.

In vederea abordării matematice a problemei interblocării se va elabora un model generalizat al stării resurselor pentru un sistem de procese. Sistemul luat în considerare este o combinaţie paralelă a unor subsisteme de tip şir (§ 3.1) formate din şiruri liniare de procese. Din punctul de vedere al resurselor se consideră că în cadrul unui şir de procese, starea resurselor este constantă pe parcursul execuţiei unui proces, dar se modifică de la un proces la altul.

Limitarea legată de considerarea unui subsistem ca fiind format dintr-un şir de procese nu este restrictivă, întrucît în ultimă instanţă execuţia unui sistem de procese concurente pe un sistem monoprocesor revine la un astfel de model. Combinaţia paralelă de şiruri de procese prefigurează sistemul multiprocesor sau execuţia unor sisteme de procese independente pe un acelaşi sistem monoprocesor.

In studiul problemei interblocărilor, sistemul fizic va fi asimilat cu un set de m tipuri de resurse R_1, R_2,...,R_m fiecare avînd z_i unităţi disponibile. Vectorul $z = (z_1, z_2, \ldots z_j, z_{j+1}, \ldots z_m)$ reprezintă capacitatea sistemului.

Se vor considera două categorii de resurse [Bh74] :

1. Resurse fixe care sînt un set finit de unităţi identice cu următoarele proprietăţi:

a) Numărul de unităţi este constant;

b) Orice unitate este fie „disponibilă" fie „alocată" unui proces;

c) O resursă fixă poate fi retrocedată de către un proces numai după ce a fost alocată acestuia.

Exemple de resurse fixe sînt componentele hardware (memoria, memoriile auxiliare, procesoarele, perifericele) şi componentele software (fişiere de date, procese, tabele, semafoare).

2. Resurse consumabile sînt un set de resurse cu următoarele proprietăţi:

a) Numărul de unităţi de resurse este variabil deoarece aceste resurse pot fi „consumate" sau „produse" de către procese;

b) Un proces „producător" creşte numărul de resurse retrocedînd una sau mai multe resurse pe care le creează.

c) Un proces „consumator" descreşte numărul de resurse primind iar apoi consumînd una sau mai multe resurse. Ca atare resursele primite, nu mai sînt retrocedate.

Exemple de resurse consumabile sînt semnalele de sincronizare, mesajele, întreruperile I/E sau de timp, tranzacţiile proceselor etc.

In cadrul vectorului z cele două categorii de resurse se consideră grupate: cele fixe în prima parte (indicii 1 la j) cele consumabile în cea de-a doua parte a vectorului (indicii j+1 la m).

Se consideră în continuare un şir de procese $\mathcal{S}_i = P_i(1)P_i(2)..P_i(n_i)$ unde $1 \leq n_i$. Pentru fiecare proces se introduce noţiunea de <u>necesar de resurse</u> $n_i(\ell) = (n_{i1}(\ell), n_{i2}(\ell),...,n_{im}(\ell)$ unde $n_{ij}(\ell)$ reprezintă necesarul de resursă fixă sau consumabilă R_j pentru procesul $P_i(\ell)$ $(1 \leq \ell \leq n_i)$ aparţinînd şirului i. Valorile $n_{ij}(\ell)$ corespunzătoare resurselor solicitate sînt prin convenţie pozitive.

Deoarece într-un proces o anumită resursă consumabilă este fie solicitată fie produsă (cazurile în care acelaşi proces produce şi consumă o resursă nu interesează din punctul de vedere al şirului de procese i considerat), există posibilitatea ca în vectorul necesar să fie reprezentate şi resursele consumabile produse de către proces. Pentru a deosebi aceste resurse de cele solicitate, prin convenţie, componentele $n_{ij}(\ell)$ corespunzătoare lor se înregistrează cu semn negativ. Astfel componentele vectorului $\tilde{n}_i(\ell)$ pot lua următoarele valori:

a) valori (+) care precizează cantitatea de resursă fixă sau consumabilă necesară procesului $P_i(\ell)$ (resursă solicitată).

b) valori (-) care precizează cantitatea de resursă consumabilă produsă de către proces (resursă produsă).

c) valori nule care precizează faptul că procesul nu utilizează resursa respectivă.

Pentru resursele fixe se consideră că toate resursele necesare unui proces se alocă înainte de lansarea sa în execuţie şi că la terminarea sa, acesta retrocedează toate resursele pe care le-a primit. Fiind însă vorba de un şir de procese, procesúl care se termină nu va elibera decît acele resurse de care nu are nevoie procesul următor, iar acesta nu va primi decît acele resurse pe care le solicită în plus faţă de procesul terminat. Această categorie de resurse nu modifică capacitatea sistemului.

Pentru resursele consumabile se consideră de asemenea că unui proces i se alocă la început toate resursele necesare execuţiei sale, dar că aceste resurse sînt consumate (deci nu mai sînt restituite). De asemenea procesul poate produce el însuşi resurse, ca atare capacitatea sistemului se modifică în funcţie de resursele produse şi cele consumate.

Pentru şirul luat în discuţie se presupune o secvenţă unică de execuţie a = $\bar{P}_i(1)\underline{P_i}(1)\ldots\underline{P_i}(n_i) = e_1\ e_2\ \ldots\ e_{2n_i}$, avînd secvenţa de stări asociată $p_i = s_o\ s_1\ s_2\ \ldots\ s_{2n_i}$.

Pentru a caracteriza o stare s_K a proceselor şirului se vor defini următorii vectori: $\widetilde{U}_i(k)=(U_{i1}(k), U_{i2}(k),\ldots,U_{im}(k))$ şi $\widetilde{N}_i(k)=(N_{i1}(k),N_{i2}(k),\ldots,N_{im}(k))$ unde $U_{ij}(k)$ reprezintă cantitatea din resursa R_j utilizată de şirul i la momentul k, iar $N_{ij}(k)$ - reprezintă cantitatea din resursa R_j, necesară şirului i la momentul k

De asemenea se mai definesc vectorii $\widetilde{PR}_i(k)=(0,0,0,\ldots 0, PR_{ij+1}(k)\ldots PR_{im}(k))$ şi $\widetilde{C\O N}_i(k)=(0,0,\ \ldots\ 0,\ C\O N_{ij+1}(k)\ldots C\O N_{im}(k))$ unde $PR_{i\ell}(k)$ şi $C\O N_{i\ell}(k)$ reprezintă cantitatea din resursa R_ℓ produsă respectiv consumată de către şirul i la momentul k. Este evident că aceşti vectori au valori numai pentru categoria de resurse consumabile z_ℓ ($j < \ell \leq m$).

Pentru a se putea preciza dinamica acestor vectori se mai introduc noţiunile de <u>necesar relativ</u> $\widetilde{q}_i(k)=(q_{i1}(k),q_{i2}(k),\ldots, q_{im}(k))$ şi <u>retrocedare relativă</u> $\widetilde{r}_i(k)=(r_{i1}(k),r_{i2}(k),\ldots r_{im}(k))$ unde $q_{ij}(k)$ reprezintă cantitatea de resursă R_j necesară la momentul k (în plus faţă de ceea ce deţine şirul) pentru a satisface solicitarea procesului următor, iar $r_{ij}(k)$ reprezintă cantitatea din resursa R_j care poate fi eliberată la terminarea unui proces, astfel încît solicitarea procesului următor să fie acoperită.

Dacă $q_i(k)$ este nul se va nota aceasta prin $\widetilde{q}_i(k) = \widetilde{0}$ unde $\widetilde{0}$ reprezintă un m-uplu de zerouri.

Se reaminteşte faptul că pe întreg parcursul acestui paragraf toţi vectorii care se referă la resurse au o structură partajată, respectiv din cele m elemente, cele cuprinse între 1 şi j se referă la resursele fixe, iar cele cuprinse între j+1 şi m la cele consumabile. Modelul permite şi luarea în considerare a extremelor (numai resurse fixe sau numai resurse consumabile) prin poziţionarea corespunzătoare a lui j.

Vectorii $\widetilde{q}_i(\ell)$ (necesarul de resursă al procesului care urmează să fie executat) şi $\widetilde{r}_i(\ell-1)$ (retrocedarea procesului care tocmai s-a terminat) pot fi obţinuţi din vectorii $\widetilde{n}_i(\ell)$ (necesarul procesului care urmează) şi $n_i(\ell-1)$ necesarul procesului terminat) conform următorului algoritm.

<u>A.3.6.1</u>

A. Pentru resurse fixe:

1° Se scad vectorii $\widetilde{n}_i(\ell)$ şi $\widetilde{n}_i(\ell-1)$ componentă cu componentă.

2° Componentele care rezultă pozitive se trec în $\tilde{q}_i(\ell)$ ele reprezentînd necesarul relativ al noului proces, iar cele care rezultă cu minus se trec cu semn pozitiv pe locurile corespunzătoare în $\tilde{r}_i(\ell-1)$ ele reprezentînd retrocedarea relativă a vechiului proces.

3° Restul pozițiilor din $\tilde{r}_i(\ell-1)$ și $\tilde{q}_i(\ell)$ se completează cu zerouri.

B. Pentru resurse consumabile:

1° $\tilde{q}_i(\ell)$ are drept componente, valorile care apar cu semnul (+) în $\tilde{n}_i(\ell)$, acestea reprezentînd resurse solicitate de procesul curent.

2° $\tilde{r}_i(\ell-1)$ are drept componente, valorile care apar cu semnul (+) în $\tilde{n}_i(\ell-1)$, acestea reprezentînd resurse solicitate de procesul anterior. Aceste resurse au fost consumate, ele însă apar în vectorul retrocedare pentru a putea fi diminuată în mód corespunzător, cantitatea totală de resurse aflată la dispoziția șirului (vezi D.3.6.1).

3° Restul pozițiilor din $\tilde{q}_i(\ell)$ și $\tilde{r}_i(\ell-1)$ se completează cu zerouri. Se precizează că indicele i ia valori în domeniul $(1,n_i)$.

Forma Pascal a acestui algoritm apare în figura 3.6.3, împreună cu baza de date care va fi utilizată pentru toate procedurile care modelează interblocarea (procedura RRELATIVE).

Procedura RRELATIVE primește ca intrări doi vectori de tip resursă NMIC1 și NMIC reprezentînd pe $\tilde{n}_i(\ell)$ respectiv $\tilde{n}_i(\ell-1)$ și furnizează pe QMIC($\tilde{q}_i(\ell)$) și RMIC ($\tilde{r}_i(\ell-1)$). (fig.3.6.3) .

Pentru determinarea vectorilor $\widetilde{PR}_i(\ell)$ și $\widetilde{CØN}_i(\ell)$ se pornește de la $\tilde{n}_i(\ell)$ conform următorului algoritm:

Algoritm.

A) Pentru componentele corespunzătoare resurselor consumabile.

1° $\widetilde{PR}_i(\ell)$ are drept componente acele valori din $\tilde{n}_i(\ell)$ care au semn negativ. Valorile se iau în modul.

2° $\widetilde{CØN}_i(\ell)$ are drept componente valorile ce apar cu semnul (+) în $\tilde{n}_i(\ell)$.

3° Restul componentelor consumabile iau valoarea zero.

B) Pentru resurse fixe:

1° Toate componentele corespunzătoare resurselor fixe iau valoarea zero. (Procedura PRØCØN figura 3.6.4).

In aceste condiții pentru șirul i, dinamica vectorilor $\tilde{U}$, $\tilde{N}$, $\widetilde{PRØ}$, $\widetilde{CØN}$ și $\tilde{z}$, poate fi descrisă conform următorului model, considerîn ca valori inițiale la momentul 0 $\tilde{N}_i(0)=\tilde{q}_i(1)$ și $\tilde{U}_i(0)=\tilde{0}$:

```
TYPE  RESURSA=ARRAY(.1..NRRESURSE.) OF INTEGER ;
      EVOLUTIE1=ARRAY(.1..NRPROC.) OF RESURSA ;
      SISTEM =ARRAY(.1..NRLANTURI.) OF RESURSA ;
      STARE1 =(INCEPUT,SFIRSIT) ;
      EVENIMENT =RECORD
                   STARE : STARE1 ;
                   PROCES: INTEGER;
                   LANT  : INTEGER
                 END;
      SECVEXEC  =ARRAY (.0.. NREVENIM.) OF EVENIMENT ;
      RESDEN =ARRAY(.1..NRLANTURI.) OF  EVOLUTIE1 ;
      DOMENIU=ARRAY(.1..NRLANTURI.) OF INTEGER;
VAR   NMIC:EVOLUTIE1;
      OMIC,RMIC,NMARE,UMARE,ZMIC,PRO,CON:RESURSA;
      K,L,U,IB,NLR:INTEGER;
      ZERO:RESURSA;
      UBARE,NBARE,NPRIM,MAXR,PRODG,CONG:SISTEM;
      NMICG:RESDEN;
      SECV: SECVEXEC;
      MI:DOMENIU;
PROCEDURE RRELATIVE(NMIC1,NMIC:RESURSA;
                    VAR OMIC,RMIC:RESURSA);
   VAR DIF,I:INTEGER ;
   BEGIN
     FOR I:=1 TO NRRESURSE1 DO
       BEGIN
         DIF:= NMIC1(.I.)-NMIC(.I.);
         IF  DIF>0  THEN
           BEGIN
             OMIC(.I.):=DIF;RMIC(.I.):=0
           END;
         IF (DIF<0) THEN
           BEGIN
             RMIC(.I.):=-DIF;OMIC(.I.):=0
           END;
         IF (DIF=0) THEN
          BEGIN
             RMIC(.I.):=0;OMIC(.I.):= 0
          END;
       END;
     FOR I:=NRRESURSE1+1 TO NRRESURSE DO
       BEGIN
         IF NMIC1(.I.)>0 THEN OMIC(.I.):= NMIC1(.I.);
         IF NMIC(.I.)>0  THEN RMIC(.I.):= NMIC(.I.);
       END;
   END;
```

Fig.3.6.3. Baza de date şi procedura de determinare a vectorilor necesari şi retrocedare relativă.

D.3.6.1

a) Starea k ($0 \leq k \leq 2n_1$) în care se ajunge după iniţierea unui proces $P_1(\ell')$ ($1 \leq \ell' \leq n_1$) este:

$$\tilde{N}_1(k) = \tilde{0};\ \tilde{U}_1(k) = \tilde{U}_1(k-1) + \tilde{N}_1(k-1)$$

b) Starea de k în care se ajunge după terminarea unui proces $P_1(\ell)$ este:

$$\tilde{N}_1(k) = \tilde{q}_1(\ell+1);\ \tilde{U}_1(k) = \tilde{U}_1(k-1) - r_1(\ell);$$

$$\tilde{z}(k) = \tilde{z}(k) - C\tilde{\emptyset}N_1(\ell) + \tilde{PR}_1(\ell);\ \text{unde } \tilde{q}_1(\ell+1),$$

$\tilde{r}_1(\ell)$, $\tilde{PR}_1(\ell)$ şi $C\tilde{\emptyset}N_1(\ell)$ se determină prin algoritmii A.3.6.1. şi A.3.6.2.

Pornind de la această definiţie, pentru un sistem de tip şi de procese putem determina dinamica vectorilor $\tilde{N}$, $\tilde{U}$, $C\tilde{\emptyset}N$, $\tilde{PR}$ şi $\tilde{z}$, dacă cunoaştem necesarul de resurse pentru fiecare proces al lanţului şi secvenţa de execuţie (procedura PASLANT fig.3.6.4).

```
PROCEDURE PROCON(NMIC:RESURSA;VAR PRO,CON:RESURSA);
 VAR I:INTEGER;
 BEGIN
   PRO:=ZERO; CON:=ZERO;
   FOR I:=NRRESURSE1+1 TO NRRESURSE DO
     IF NMIC(.I.)<0 THEN PRO(.I.):=-NMIC(.I.)
                    ELSE CON(.I.):=NMIC(.I.);
 END;
PROCEDURE CAUTA(SECV:SECVEXEC;M:INTEGER;
                VAR S:INTEGER;VAR  GASIT:BOOLEAN);
  VAR Q:INTEGER;
  BEGIN
    S:=K;GASIT:=FALSE;
    IF S<>NREVENIM THEN
      REPEAT
        S:=S+1;Q:=SECV(.S.).LANT;
        IF M=Q THEN GASIT:=TRUE ;
      UNTIL (GASIT) OR (S=NREVENIM)
  END;
PROCEDURE PASLANT(SECV:SECVEXEC;NMIC:EVOLUTIE1;
                  VAR NMARE,UMARE,PRO,CON:RESURSA);
    VAR L,P,S,R:INTEGER;
        GASIT:BOOLEAN;
   BEGIN
     P:=SECV(.K.).PROCES;
     PROCON(NMIC(.P.),PRO,CON);
     IF SECV(.K.).STARE=INCEPUT THEN
       BEGIN
         SUMA(UMARE,UMARE,NMARE);
         NMARE:=ZERO
       END;
     IF SECV(.K.).STARE=SFIRSIT THEN
       BEGIN
         L:=SECV(.K.).LANT;P:=SECV(.K.).PROCES;
         CAUTA(SECV,L,S,GASIT);
           IF GASIT THEN
             BEGIN
               R:=SECV(.S.).PROCES;
               RRELATIVE(NMIC(.R.),NMIC(.P.),QMIC,RMIC)
             END
                   ELSE
             BEGIN
               RRELATIVE(ZERO,NMIC(.P.),QMIC,RMIC)
             END;
         NMARE:=QMIC;DIFERENTA(UMARE,UMARE,RMIC);
         SUMA(ZMIC,ZMIC,PRO); DIFERENTA(ZMIC,ZMIC,CON);
       END;
   END;
PROCEDURE INITBARE(SECV:SECVEXEC;NMICO:RESOEN;
                   VAR NBARE,UBARE,PRODO,CONO:SISTEM);
  VAR M,P,S:INTEGER;
      GASIT:BOOLEAN;
    BEGIN
      FOR M:=1 TO NRLANTURI DO
        BEGIN
          CAUTA(SECV,M,S,GASIT);
          P:=SECV(.S.).PROCES;
WRITE ('INITBARE',M,P,S,K,EOL);
          RRELATIVE(NMICO(.M,P.),ZERO,QMIC,RMIC);
          NBARE(.M.):=QMIC;UBARE(.M.):=ZERO;
          PRODO(.M.):=ZERO;CONO(.M.):=ZERO;
WRITE('INITBARE');
SCR(NBARE(.M.),4);SCR(UBARE(.M.),4);
WRITE(EOL);
        END;
    END;
PROCEDURE PASSIST(SECV:SECVEXEC;NMICO:RESOEN;
                  VAR UBARE,NBARE,PRODO,CONO:SISTEM);
  VAR I:INTEGER ;
  BEGIN
    IF K=0 THEN  INITBARE(SECV,NMICO,NBARE,UBARE,PRODO,CONO)
           ELSE
             BEGIN
               I:=SECV(.K.).LANT;
               PASLANT(SECV,NMICO(.I.),NBARE(.I.),UBARE(.I.),
                       PRODO(.I.),CONO(.I.));
WRITE('PASIST');
           FOR I:=1 TO NRLANTURI DO
           BEGIN
            SCR(NBARE(.I.),4);SCR(UBARE(.I.),4);SCR(QMIC,4);SCR(RMIC,4)
           END;
    WRITE(EOL);
             END;
  END ;
```

Fig.3.6.4.Proceduri pentru modelarea interblocărilor

Acest model poate fi generalizat pentru precizarea stărilor unui sistem $\mathcal{S} = \mathcal{S}_1 // \mathcal{S}_2 // \dots // \mathcal{S}_n$ care este o combinație paralelă a n sisteme de tip șir. Fiecare stare s_K a sistemului este descrisă prin următoarele matrici:

$$\|U(k)\| = \begin{Vmatrix} U_1(k) \\ U_2(k) \\ \vdots \\ U_n(k) \end{Vmatrix}; \|N(k)\| = \begin{Vmatrix} N_1(k) \\ N_2(k) \\ \vdots \\ N_n(k) \end{Vmatrix}; \|PR(k)\| = \begin{Vmatrix} PR_1(k) \\ PR_2(k) \\ \vdots \\ PR_n(k) \end{Vmatrix}; \|CON(k)\| = \begin{Vmatrix} CØN_1(k) \\ CØN_2(k) \\ \vdots \\ CØN_n(k) \end{Vmatrix}$$

Dacă $a=e_1e_2\dots e_k\dots$ este o secvență de execuție pentru sistemul $\mathcal{S}$ iar NMICG necesarul de resurse al tuturor proceselor șirurilor i ($1 \leq i \leq n$), atunci aplicînd procedura anterioară pentru șirul activ al sistemului, putem determina în fiecare moment dinamica matricilor $\|U\|$, $\|N\|$, $\|PR\|$, $\|CØN\|$ și dinamica vectorului $\hat{z}$. Un șir $\mathcal{S}_i$ al sistemului este activ cînd $\tilde{U}_i(k) + \tilde{N}_i(k) \neq 0$.

Acest lucru este realizat de procedura PASIST (fig.3.6.4). Aceasta primește la intrare secvența de execuție a sistemului (SECV) și necesarul de resurse pentru toate șirurile de procese (NMICG) unde NMICG(.I.) este necesarul pentru șirul I. Procedura actualizează matricile UBARA, NBARA, CØNG și PRØDG precum și vectorul ZMIC. In toate matricile modificările se execută doar în rîndul corespunzător șirului activ. Un exemplu de aplicare al procedurii pentru un set de resurse fixe apare în figura 3.6.5.

NRPROC = 3
NRRESURSE = 4
NREVENM = 10
NRLANTURI = 2
NRRESCONS = 0

		R1	R2	R3	R4
LANT 1	P1	0	1	2	0
	P2	1	1	0	1
	P3	1	1	2	0
LANT 2	P1	2	0	2	2
	P2	3	0	3	0
	0	0	0	0	0
	Z	3	3	3	3

SECV.

	STARE	PROCES	SIR
1	I	1	1
2	S	1	1
3	I	1	2
4	S	1	2
5	I	2	2
6	S	2	2
7	I	2	1
8	S	2	1
9	I	3	1
10	S	3	1

	$\tilde{N}_1$	$\tilde{N}_2$	$\tilde{U}_1$	$\tilde{U}_2$
0	0120	2022	0000	0000
$\tilde{P}_{11}$	0000	2022	0120	0000
$\tilde{P}_{11}$	1001	2022	0100	0000
$\tilde{P}_{12}$	1001	0000	0100	2022
$\tilde{P}_{12}$	1001	1010	0100	2020
$\tilde{P}_{22}$	1001	0000	0100	3030
$\tilde{P}_{22}$	1001	0000	0100	0000
$\tilde{P}_{21}$	0000	0000	1101	0000
$\tilde{P}_{21}$	0020	0000	1100	0000
$\tilde{P}_{31}$	0000	0000	1120	0000
$\tilde{P}_{31}$	0000	0000	0000	0000

Fig.3.6.5. Exemplu de aplicare al modelului generalizat.

Modelul generalizat pentru studiul stării resurselor este valabil atît pentru resurse fixe cît și pentru resurse consumabile sau combinații ale acestora.

O tranziție corespunzătoare unei inițieri de proces este permisă numai dacă necesarul solicitat din fiecare resursă R_j nu depășește

disponibilul din acel moment. Se va defini deci şi un vector „disponibil" $\tilde{d}(k)=(d_1(k),\ldots,d_m(k))$ unde

$$\tilde{d}(k) = \tilde{z}(k) - \sum_{i=1}^{n} \tilde{U}_i(k) \qquad \text{(r.3.6.1)}$$

Deci în şirul $\mathcal{S}_i$, o tranziţie de stare corespunzătoare evenimentului $e_{K+1}=F_i(\ell)$ este permisă dacă $\tilde{N}_i(k) \leq \tilde{d}(k)$. Se spune despre o secvenţă de execuţie că este <u>validă</u> dacă toate tranziţiile sale de iniţiere sînt permise.

3.6.2. <u>Definirea noţiunii de interblocare</u>

Pornind de la modelul generalizat prezentat se poate defini noţiunea de interblocare.

<u>D.3.6.2</u>. Fie $\mathcal{S} = \mathcal{S}_1 // \mathcal{S}_2 // \ldots // \mathcal{S}_n$ un sistem de procese unde fiecare $\mathcal{S}_i (1 \leq i \leq n)$ este un şir. Fie $a=e_1e_2\ldots e_K$ o secvenţă de execuţie parţială a lui $\mathcal{S}$ şi fie $p=s_0s_1\ldots s_K$ secvenţa parţială de stări corespunzătoare. Presupunînd că există un set $MI \neq \tilde{O}$ de indici ai lanţurilor sistemului astfel încît pentru fiecare $i \in MI$ să avem

$$\tilde{N}_i(k) \not\leq \tilde{d}(k) + \sum_{j \notin MI} \tilde{U}_j(k)$$

atunci $\mathcal{S}$ este blocat. De asemenea sînt interblocate şirurile $\mathcal{S}_i$ cu $i \in MI$.

inter

Definiţia precizează că o interblocare există atunci cînd un set de şiruri active, aparţinînd unui sistem, nu îşi pot începe execuţia procesului următor din lipsă de resurse. Chiar dacă toate şirurile neinterblocate eliberează toate resursele pe care le deţin, totuşi nu vor fi suficiente resurse pentru ca măcar unul dintre şirurile interblocate să-şi continue execuţia. Deoarece s-a presupus că nici un şir nu poate solicita mai multe resurse decît poate oferi sistemul, rezultă că pentru ca să apară interblocarea trebuie să existe cel puţin două şiruri.

Există mai multe moduri de a rezolva interblocările.

a) <u>Prevenirea</u> prin care se încearcă ocolirea factorilor care generează interblocări prin restricţii introduse la proiectarea sistemului sau prin restricţii de utilizare a resurselor.

b) <u>Detecţia</u> presupune un algoritm de constatare a unei interblocări şi luarea de măsuri pentru rezolvarea situaţiei.

c) <u>Evitarea</u>. Sistemul de operare, pe baza unor informaţii cunoscute aprioric, controlează de o asemenea manieră evoluţia proceselor încît evită regiunile nesigure. Se vor analiza pe rînd aceste metode în contextul sistemelor timp-real.

3.6.3. <u>Prevenirea interblocărilor</u>

Una dintre metodele cele mai indicate de rezolvare a interblocărilor în cadrul STR este metoda prevenirii. Prevenirea se realizează evi-

tînd pe cît este posibil condiţiile care pot genera interblocări: sincronizarea proceselor, nonpreemţiunea şi aşteptarea circulară.

1) Sincronizarea proceselor şi legat de aceasta, excluziunea mutuală, nu pot fi evitate întrucît, sincronizarea este mecanismul esenţial de coordonare a activităţii proceselor unui STR (§ 3.3). In condiţiile în care însă, se proiectează şi se implementează un mecanism de sincronizare „principial", iar utilizarea acestuia se face cu atenţie, probabilitatea de apariţie a interblocărilor din acest motiv este redusă.

2) Nonpreemţiunea. Analizînd definiţia D.3.6.1, se poate observa că dacă fiecare şir (proces), în momentul k în care solicită resurse, nu deţine nici o resursă ($\widetilde{N}_i(k) \neq \widetilde{0}$ şi $\widetilde{U}_i(k) = \widetilde{0}$), atunci membrul drept al inegalităţii se reduce la $\widetilde{z}$ (capacitatea sistemului). Reamintind că una din ipotezele admise se referă la faptul că nici un şir nu poate solicita resurse peste capacitatea sistemului rezultă că în acest caz interblocările devin imposibile

Pornind de la această observaţie se conturează două metode de prevenire a interblocărilor.

M.3.6.1. Inaintea fiecărei solicitări de resurse, şirul trebuie să elibereze toate resursele pe care le deţine (chiar dacă mai are nevoie de ele) şi apoi să le solicite din nou.

Această metodă nu este însă aplicabilă tuturor tipurilor de resurse din cauza caracteristicilor fizice ale acestora (spre exemplu resursele nepreemtibile).

M.3.6.2. Fiecărui şir al sistemului i se prealocă cantitatea maximă de resurse de care are nevoie.

Acest lucru este însă posibil numai în condiţiile existenţei unei cantităţi suficiente de resurse, lucru mai rar întîlnit la sistemele dedicate şi timp-real. Dezavantajul acestei metode constă în faptul că unele resurse pot rămîne multă vreme neutilizate.

3) Aşteptarea circulară. Evitarea aşteptării circulare a fost uşurată de introducerea unor metode grafice pentru reprezentarea interblocărilor [Sh74], [Ho83]. Spre exemplu, una dintre ele presupune reprezentarea resurselor unui sistem la momentul k, în forma unui graf direcţionat G_K, conţinînd un set de m noduri corespunzătoare tipurilor de resurse $R_1, R_2, \ldots, R_m$, un set de noduri corespunzătoare utilizatorilor şi un set de săgeţi care se definesc astfel: graful G_K conţine o săgeată $(R_j, \mathcal{P}_i)$ dacă lanţul $\mathcal{P}_i$

deţine resursa $R_j(U_{ij}(k) \neq 0)$ şi o săgeată $(\mathcal{S}_i, R)$ dacă solicită resursa $R_\ell(N_{i\ell}(k) \neq 0)$. Cu alte cuvinte după evenimentul k, lanţul $\mathcal{S}_i$ deţine resursa R_j şi solicită resursa R_ℓ (fig.3.6.6(a)).

Se consideră situaţia următoare: se presupune că există mai multe lanţuri $\mathcal{S}_i$ fiecare deţinînd şi solicitînd resurse. Astfel lanţul $\mathcal{S}_{i1}$ deţine resursa R_{j1} şi solicită resursa R_{j2}. Aceasta este deţinută de lanţul $\mathcal{S}_{i2}$ care solicită R_{j3} aparţinînd lanţului $\mathcal{S}_{i3}$. Interblocarea apare în momentul în care bucla se închide: $\mathcal{S}_{i3}$ solicită R_{j1} (fig.3.6.6(b)).

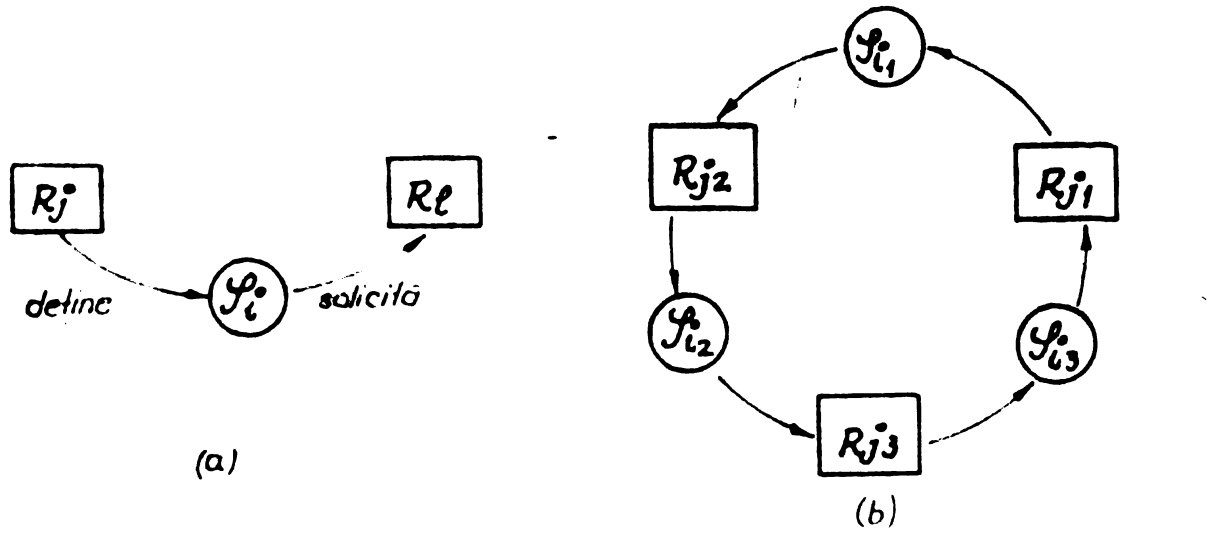


Fig.3.6.6. Reprezentarea grafică a resurselor.

Intuitiv, se poate enunţa următoarea propoziţie.

P.3.6.1. Dacă o stare s_K conţine o interblocare, graful G_K asociat stării respective conţine o buclă.

Această condiţie referitoare la G_K este una necesară dar nu şi suficientă (spre exemplu în cazul în care există mai multe resurse de un anumit tip). Condiţia devine suficientă în momentul în care, din fiecare tip de resursă implicată în buclă mai există o singură unitate.

Pentru evita formarea buclelor se pot aplica mai multe metode. Una dintre ele se referă la alocarea ordonată a resurselor. Aceasta presupune ordonarea mulţimii $\mathcal{R} = (R_1, R_2, \ldots, R_m)$ a tipurilor de resurse prin aplicarea unei relaţii $\lhd$ de ordonare liniară definită astfel: $R_1 \lhd R_2$ presupune că într-un sistem care solicită resurse, resursa R_2 se alocă numai după ce s-a alocat resursa R_1.

Se presupune că mulţimea R a fost ordonată prin renumerotarea tipurilor de resurse şi că avem $R_1 \lhd R_2 \lhd \ldots \lhd R_m$.

Se poate demonstra următoarea teoremă:

T.3.6.1. Într-un set de şiruri constrîns la o alocare ordonată a resurselor nu poate apare interblocarea.

Demonstrație. Se presupune că în graful asociat stării resurselor unui sistem, la momentul k există o buclă (interblocare) care conține nodurile R_{i1}, R_{i2},...,R_{in}. Deoarece șirul resurselor e supus la o alocare ordonată, parcurgînd buola se poate scrie:
$R_{i1} \lessdot R_{i2} \lessdot \ldots \lessdot R_{in} \lessdot R_{i1}$. Rezultă că:

a) $R_{i1} \lessdot R_{in}$ (din tranzitivitate)

b) $R_{in} \lessdot R_{i1}$ (din închiderea buclei)

Existența ambelor relații într-un sistem cu mulțimea resurselor ordonată este imposibilă, deci premisa de la care s-a pornit (că sistemul conține o interblocare) este falsă.

Pornind de la T.3.6.1 se poate concepe următoarea metodă de alocare.

M.3.6.3. Resursele solicitate de un sistem se alocă numai într-o ordine prestabilită.

Aplicarea acestei metode permite evitarea sigură a interblocărilor. Metoda este recomandabilă în special pentru sistemele timp-real, în care informația predictivă asupra sistemului permite o astfel de alocare a resurselor.

După cum s-a arătat în § 3.1, un rol central în cadrul sistemelor de procese îl joacă relația de precedență. În § 3.4 s-a demonstrat că o posibilitate de implementare a relației de precedență o reprezintă mecanismul SEND-REC de comunicare prin mesaje. Mesajele fiind de fapt resurse consumabile, utilizarea lor poate conduce la situația formării unor bucle în sensul celor precizate de T.3.6.1, Prin urmare cooperarea proceselor poate conduce la interblocare. Deoarece alocarea ordonată, nu este potrivită resurselor consumabile, pentru a preveni formarea buclelor în această situație se poate recurge la metoda comunicării ierarhice.

Pentru a implementa această metodă se pornește de la observația că procesele unui sistem pot fi organizate într-o structură de niveluri ierarhice conform celor precizate în § 3.1.

Într-o astfel de structură comunicările au loc într-un singur sens de sus în jos comenzile transmițîndu-se doar de pe niveluri superioare spre cele inferioare, iar răspunsurile în sens invers (fig.3.6.7).

Pentru o astfel de manieră de comunicare se poate demonstra următoarea teoremă.

T.3.6.2. Într-un sistem de procese constrîns la o comunicare ierarhică, nu poate apare interblocarea.

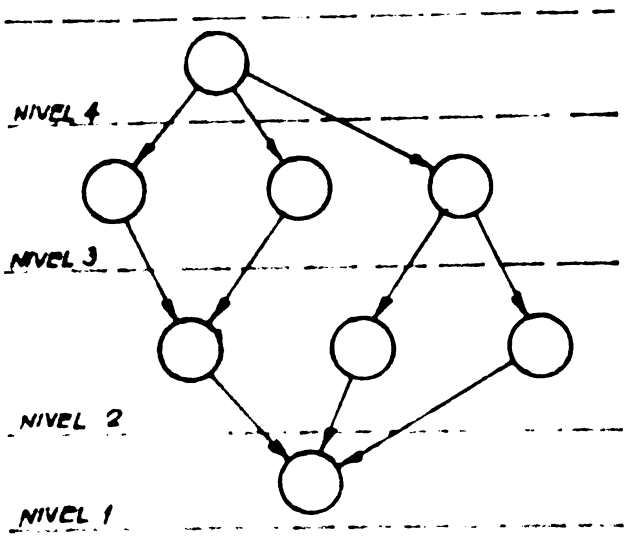


Fig.3.6.7. Structură (comunicare) ierarhică.

Asimilînd relaţia $\lessdot$ definită anterior cu o relaţie de comunicare între două procese situate unul pe un nivel superior altul pe un nivel inferior (implementată printr-un SEND-REC), teorema se demonstrează exact ca şi predecesoarea ei, arătînd că formarea unei bucle nu este posibilă.

Comunicarea ierarhică stă la baza structurilor ierarhice în care conform teoremei T.3.6.2 nu pot apare interblocări datorate comunicării prin mesaje. Din acest motiv, acest tip de structuri au o importanţă cu totul particulară pentru SØTR (§ 4.2). Este uşor de observat că o astfel de structură ierarhică se suprapune perfect peste structura unui graf al unui sistem închis de procese.

In funcţie de situaţiile concrete ale proiectării sistemelor, se pot folosi după necesităţi şi combinaţii ale metodelor de prevenire prezentate. Se apreciază prevenirea ca fiind metoda cea mai indicată spre a fi utilizată în rezolvarea problemei interblocării SØTR.

3.6.4. Detecţia interblocărilor

Problema detecţiei interblocărilor prezintă un mare interes pentru specialişti în acest domeniu existînd o bogată literatură. [Mu68], [CD73], [MD74], [Ho83] etc.

In cadrul celor prezentate în acest paragraf, pentru sistemele tip şiruri de procese, se poate concepe un algoritm care să detecteze apariţia unei interblocări şi şirurile implicate în ea. Un astfel de algoritm utilizează procedura de parcurgere a stărilor resurselor unui sistem (PASSIST) şi definiţia interblocării (D.3.6.2). Astfel, algoritmul preconizat simulează execuţia şirurilor de procese pornind de la starea curentă, încercînd să găsească un set de şiruri care din lipsă de resurse disponibile se interblochează.

Algoritmul utilizează drept intrări matricile care caracterizează starea sistemului $\|U\|$, $\|N\|$, $\|PR\|$, $\|CØN\|$ şi $\tilde{z}$ furnizînd la ieşire un set de indici MI de şiruri interblocate şi numărul acestora IB. Dacă setul e vid (IB=0), înseamnă că starea e sigură şi că nu poate conduce la interblocări.

A.3.6.3.

1° Se iniţializează domeniul MI al indicilor şirurilor de procese pe valoarea 1 şi IB cu valoarea numărului şirurilor. Se calculează $\tilde{d}$ conform lui (r 3.6.1) şi se atribuie valoarea calculată variabilei locale VAL.

2° Se caută în mulţimea indicilor şirurilor, un astfel de şir i încît $\tilde{N}_i \leq \tilde{VAL}$. Dacă nu se găseşte nici unul algoritmul se termină.

3° Se elimină indicele i din domeniul MI, se decrementează IB şi se adaugă la $\tilde{VAL}$ mulţimea resurselor U_i utilizate de şirul i : $\tilde{VAL} = \tilde{VAL} + \tilde{U}_i$. Se mai adaugă lui $\tilde{VAL}$ mulţimea resurselor produse de şirul I ($\tilde{PR}_i$) şi se scade mulţimea elementelor consumate de acesta ($\tilde{CON}_i$). Se revine la pasul 2°. In figura 3.6.8 se observă procedura DETECTIE care implementează algoritmul şi un exemplu concret de utilizare al acestuia.

Algoritmul se comportă diferit faţă de resursele fixe şi faţă de resursele consumabile. Astfel dacă pentru cele fixe, valoarea lui VAL creşte în mod continuu pe măsură ce se elimină şiruri din set (deci ordinea de căutare nu are nici o semnificaţie) pentru resursele consumabile există posibilitatea ca VAL să se diminueze (prin consumarea de resurse), caz în care ordinea de căutare este importantă. In astfel de cazuri pentru mărirea acurateţei avem nevoie de informaţii predictive legate de ordinea de execuţie a proceselor în diferitele şiruri. Acest lucru este realizabil în sistemele de operare timp-real dedicate.

Fiind dat un astfel de algoritm se propun trei metode de utilizare.

M.3.6.4. Implementarea algoritmului de detecţie în sistemul de operare şi utilizarea sa ori de cîte ori se produc modificări în utilizarea resurselor. Rezolvarea unei interblocări detectate se poate face în mod brutal prin îndepărtarea şirurilor blocate sau dacă acest lucru nu este posibil prin întîrzierea acestora. Metoda poate fi îmbunătăţită prin căutarea unui set minimal de şiruri a căror îndepărtare rezolvă interblocarea sau aplicînd un criteriu de cost minim.

Dezavantajul acestei metode constă în faptul că el măreşte mult regia sistemului de operare, lucru care în general nu se acceptă la sistemele timp-real.

M.3.6.5. Implementarea algoritmului de detecţie ca o opţiune a mediului de "programare" (§ 2.3.2). In aceste condiţii algorit-

mul poate fi utilizat doar în cazurile în care, simulînd execuţia sistemului, se observă apariţia interblocărilor. Rezolvarea acestora se face prin introducerea de condiţii de constrîngere suplimentare care să evite situaţiile care au fost detectate.

```
PROCEDURE DETECTIE (NBARE,UBARE:SISTEM;ZMIC:RESURSA;
                    VAR MI:DOMENIU; VAR IB:INTEGER);
  VAR I:INTEGER;VAL:RESURSA;
      ADEV:BOOLEAN ;
  BEGIN
   VAL:=ZMIC;
   FOR  I:=1 TO NRLANTURI DO
    BEGIN
      MI(.I.):=1;
      DIFERENTA(VAL,VAL,UBARE(.I.));
    END ;
   I:=0; IB:=NRLANTURI;
   REPEAT
     I:=I+1;
     IF MI(.I.)<>0 THEN
       BEGIN
         COMP(NBARE(.I.),VAL,ADEV);
         IF ADEV THEN
           BEGIN
             IB:=IB-1;SUMA(VAL,VAL,UBARE(.I.));
             SUMA(VAL,VAL,PROD0(.I.));DIFERENTA(VAL,VAL,COND(.I.));
             MI(.I.):=0;I:=0
           END;
       END;
   UNTIL (IB=0) OR((I=NRLANTURI)AND(NOT ADEV))
  END;
```

NR PROC - 3
NR. RESURSE - 4
NR. EVENIM - 10
NR. LANTURI - 2

		R1	R2	R3	R4
L_1	P1	0	1	2	0
	P2	1	1	0	1
	P3	1	1	2	0
L_2	P1	2	0	2	2
	P2	3	0	3	0
	P3	0	0	0	0

SECV.	STARE	PROC	LANT
0	I	1	1
1	S	1	1
2	I	1	2
3	S	1	2
4	I	2	1
5	S	2	1
6	I	3	1
7	S	3	1
8	I	2	2
9	S	2	2

STARE

	$\tilde{N}_1$	$\tilde{N}_2$	$\tilde{U}_1$	$\tilde{U}_2$	MI L_1	MI L_2
0	0120	2022	0000	0000	0	0
1	0000	2022	0120	0000	0	0
2	1001	2022	0100	0000	0	0
3	1001	0000	0100	2022	0	0
4	1001	1010	0100	2020	0	0
5	0000	1010	1101	2020	0	0
6	0020	1010	1100	2020	1	1

Fig.3.6.8. Implementarea algoritmului de detecţie.

M.3.6.6. Implementarea algoritmului de detecţie ca o opţiune a sistemului de operare (executivului). In aceste condiţii algoritmul va fi solicitat de către programator cînd apare suspiciunea unei interblocări (spre exemplu unul sau mai multe procese sînt blocate o lungă perioadă de timp).

3.6.5. Evitarea interblocărilor

Algoritmi care realizează evitarea interblocărilor sînt complecşi laborioşi şi necesită informaţii apriorice cu privire la utilizarea resurselor [Mu68], [CD73], [MD74] .

Dintre algoritmii existenţi se prezintă unul cu performanţe bune, care însă nu necesită ca informaţie apriorică decît limita superioară a necesarului de resurse pentru fiecare şir L_i al sistemului [CD73]. Acest lucru este relativ uşor de determinat practic, pentru orice configuraţie de procese aparţinînd unui sistem de cal TR.

In modelul care va fi analizat în continuare se va considera că singura informaţie ataşată unui şir L_i este vectorul $\tilde{M}_i = (M_{i1}, M_{i2}, \ldots, M_{im})$ unde M_{ij} reprezintă cantitatea maximă de resursă j de care are nevoie şirul i pe tot parcursul existenţei sale.

$$M_{ij} = \max_{1 \leq \ell \leq p_i} \left\{ q_{ij}(\ell) + \sum_{h=1}^{\ell-1} (q_{ij}(h) - r_{ij}(h)) \right\}$$

O a doua ipoteză simplificatoare se referă la faptul că în acest model, structura şirurilor (secvenţa de execuţie a proceselo nu este cunoscută decît după execuţia lor efectivă.

In fundamentarea raţionamentului se porneşte de la următoarea observaţie. Fiind dată o stare s_K a sistemului $\mathcal{S}$ şi o cerere $\tilde{N}(k)$, se analizează dacă aceasta poate fi onorată în condiţii de siguranţă. Dacă cererea a fost onorată starea S_{K+1} va diferi de starea S_K prin aceea că $\tilde{N}_i(k+1) = \tilde{0}$ (ea nu mai solicită nimic decar ce solicitarea i-a fost satisfăcută) şi $\tilde{U}_i(k+1) = \tilde{U}_i(k) + \tilde{N}_i(k)$ (a crescut cantitatea de resurse utilizate).

Problema care se pune este următoarea: în ce condiţii, plecînd de la starea s_K care este sigură, se poate ajunge în starea s_{K+1}, evitînd interblocarea.

Intrucît nu cunoaştem decît maximul de resurse necesare pentru un şir L_i al sistemului, pentru a fi siguri în evitarea interblocării se propune cazul cel mai defavorabil pentru necesarul de resurse şi anume: fiecare şir activ i al sistemului solicită maximul de resurse şi nu eliberează nici una din resursele pe care le are alocate, deci solicitarea sa va fi $\tilde{M}_i - \tilde{U}_i(k+1)$.

In aceste condiţii problema interblocării se poate reformula astfel. Se consideră o secvenţă de procese P_j corespunzătoare şirurilor L_i active în starea s_{K+1}. Fiecare proces P_j modelează tot ceea ce a mai rămas din şirul L_i pînă la terminarea acestuia. In general în acest proces sînt incluse mai multe procese dar datorită presupunerii că deja s-a eliberat cantitatea maximă de resurse necesară şirului, nici unul din aceste procese nu va mai solicita şi nu va mai elibera nici o resursă pînă la epuizarea şirului.

Pornind de la vectorul de resurse disponibile $\tilde{d}(k+1)$ trebuie găsită o secvenţă de execuţie a proceselor independente $P'_1, P'_2, \ldots, P'_a$ pentru care vectorii solicitare şi retrocedare relativă sînt furnizaţi de următoarele relaţii:

$$\tilde{q}'_i = \tilde{M}_i - \tilde{U}_i(k+1)$$
$$\tilde{r}'_i = \tilde{M}_i \qquad i \in \text{Domeniului indicilor şirurilor active.}$$

In aceste condiţii se poate formula următoarea propoziţie.

P.3.6.2. Fiind dată o stare s_{K+1} se defineşte vectorul necesar astfel:

1) dacă $\tilde{U}_i(k+1)+\tilde{N}_i(k+1) > \tilde{O}$ atunci $\tilde{N}_i(k+1)=\tilde{M}_i - \tilde{U}_i(k+1)$ pentru $1 \leq i \leq n$.

2) în restul cazurilor $\tilde{N}_i(k+1) = \tilde{O}$.

Cu ajutorul vectorilor $\tilde{N}_i$ se alcătuieşte matricea de stare a necesarului sistemului $\|N'(k+1)\|$ care are pe rînduri vectorii $\tilde{N}_i(k+1)$ $(1 \leq i \leq n)$. Starea s_{K+1} este sigură, (nu conţine o interblocare) dacă starea $(\|U(k+1)\|, \|N'(k+1)\|)$ nu conţine o interblocare.

Rezultă din această propoziţie că ori de cîte ori se solicită resurse noi în sistem, verificarea siguranţei stării următoare se poate realiza astfel.

A.3.6.4. Pornind de la starea sistemului $\|U\|$, $\|N\|$ şi cunoscînd necesarul maxim de resurse pentru fiecare şir $\|MAXR\|$, se determină matricea $\|N'\|$ conform propoziţiei P.3.6.2. Stării $\|U\|$, $\|N'\|$, $\|CON\|$ $\|PR\|$ şi $\tilde{z}$ i se aplică algoritmul de detectare a interblocării (A.3.6.3). Dacă nu se detectează o interblocare (IB=0) starea este sigură.

Forma PASCAL a acestui algoritm apare în figura 3.6.9.

```
PROCEDURE UTILMAX(UBARE,NBARE:SISTEM;DMIC:RESURSA;
                  MAXR:SISTEM;VAR MI:DOMENIU;VAR NPRIM:SISTEM);
  VAR               I:INTEGER;TEMP:RESURSA ;
  BEGIN
    FOR I:=1 TO NRLANTURI DO
      BEGIN
        SUMA(TEMP,UBARE(.I.),NBARE(.I.));
        IF (TEMP<>ZERO) THEN
          DIFERENTA(NPRIM(.I.),MAXR(.I.),UBARE(.I.))ELSE
          NPRIM(.I.):=ZERO
      END;
    DETECTIB(UBARE,NPRIM,DMIC,MI,NLB)
  END;
```

Fig.3.6.9. Algoritm pentru evitarea interblocărilor.

In cazul detectării interblocărilor se iau măsuri similare celor expuse în paragraful precedent (abortare sau amînare).

Avantajul algoritmului constă în faptul că nu mai avem nevoie de necesarul de resurse al fiecărui proces în parte şi nici de secvenţa de execuţie a proceselor ci doar de necesarul maxim de resurse al lanţului

Dezavantajul provine din faptul că întrucît ipotezele formulate presupun cazul cel mai nefavorabil, soluţia este acoperitoare, dar condiţiile în care ea se realizează nu sînt întotdeauna necesare. Cu alte cuvinte, din cauza solicitării maxime presupuse a se exercita tot timpul, se va detecta o interblocare chiar în situaţiile care în realitate nu o produc. Pentru sistemele timp-real, este un lucru acceptabil tocmai din condiţiile de siguranţă care li se impun.

4. Arhitectura software a sistemelor de operare timp-real

4.1. Conceptul de structură în programare

In cea mai generală accepţiune a sa, un sistem este format dintr-o mulţime de obiecte şi o mulţime de relaţii între aceste obiecte. Proprietăţile sistemului derivă din atributele obiectelor şi din natura relaţiilor dintre ele. Un element de bază al unui sistem îl constituie structura sa. Descrierea obiectelor sistemului şi a relaţiilor dintre ele este de fapt descrierea structurii sistemului.

Noţiunea de structură intersectează sfera de interes a celor mai diverse domenii ale ştiinţei, structuralismul pătrunzînd în biologie, psihologie, sociologie şi din ce în ce mai profund în ştiinţa calculatoarelor în general şi în programare în special. Deoarece structurile implicate în aceste domenii se constituie în sisteme cu grade diferite de complexitate, pentru analiza sistemelor complexe a fost introdus termenul de "holon" [Ko74], [Ko76] provenind din sinteza lui „holos" (întreg) şi "on" sugerînd partea (proton, neutron, etc.). Se demonstrează că sisteme complexe aparţinînd unor domenii foarte diferite (biologie, psihologie, automatică, structuri sociale, sisteme de calcul) pot fi reprezentate ca şi ierarhii de holoni. Referitor la aceste ierarhii se pot face următoarele observaţii cu caracter de generalitate.

A. Se constată spre exemplu că în sfera producţiei de bunuri materiale, la proiectarea unui produs nou, complex (holon), nu se porneşte niciodată de la zero; proiectarea şi realizarea noului se bazează pe existenţa unor subansamble bine puse la punct, dezvoltate de-a lungul unor perioade mai lungi de timp şi care încorporează

o serie de concepte teoretice şi practice validate prin experienţă, cărora eventual li se aduc mici modificări în vederea integrării lor în noul ansamblu. In mod similar se constată că şi brevetele naturii sînt asemănătoare, majoritatea fiinţelor vii avînd aceeaşi compoziţie structurală foarmată din oase, muşchi, nervi, vase de sînge, etc, catalogate din acest motiv drept organe analoage. Odată ce natura a adaptat un patent pentru un proces sau o componentă vitală (holon),ea nu renunţă uşor la el; îi face modificările minime astfel încît în diferite circumstanţe să aibă funcţii diferite. Spre exemplu picioarele aripile zburătoarelor, înotătoarele peştilor, braţele sînt atît de diferite încît s-ar putea crede că au structuri diferite. In realitate ele nu sînt decît adaptări strategice ale unor structuri existente - respectiv membrul anterior al strămoşului comun al reptilelor. Deşi au funcţii total diferite (alergare, căţărare, înot, zbor), aranjamentul oaselor rămîne acelaşi. Se constată că produsul sau organul a devenit un „holon evolutiv stabil".

B. In lumea care ne înconjoară se observă că sistemele complexe care trec prin forme intermediare stabile evoluează mult mai rapid decît acele sisteme care trec direct de la forma amorfă la structura de sistem complex. In primul caz evoluţia spre forme complexe este ierarhică. In natură se observă clar predominanţa ierarhiilor în cadrul sistemelor complexe, deoarece dintre toate formele complexe posibile, ierarhiile sînt acelea care au capacitatea potenţială să evolueze.

C. O a treia observaţie se referă la proprietăţile dinamice şi maniera în care o astfel de ierarhie poate fi pusă în funcţiune. Se constată că aceasta se realizează pe baza unui mod clasic de acţiune bazat pe principiul comutării, unde un semnal codificat implicit, care poate fi relativ simplu, pune în mişcare un mecanism complex pregătit pentru acţiune. Astfel în comportamentul instinctual există reflexe simple ce sînt declanşate de stimuli simpli (spre exemplu un zgomot puternic). In activităţi complexe care necesită dobîndirea unei îndemînări se întîmpină acelaşi proces, de această dată însă dobîndit în timp, drept urmare a simţirii gradate a detaliilor unor comenzi implicite primite din vîrful ierarhiei. Semnalul primit de la nivelurile superioare nu trebuie să specifice ceea ce holonul trebuie să execute; în schimb el acţionează ca şi o comandă simplă codificată care determină intrarea în acţiune a holonului. Odată intrat în acţiune, holonul va transforma comanda implicită într-una explicită activînd subunităţile aflate în subordine strategică, urmărind prin mesaje şi răspunsuri (reacţii) modul în care comenzile lansate sînt îndeplinite.

In continuare se va analiza modul în care conceptele prezentate se găsesc în activitatea de programare şi în arhitectura produselor software. Structuralismul a pătruns în programare odată cu introducerea conceptului de programare structurată şi definirea unităţilor structurale de bază - modulele - numite procese, activităţi sau taskuri. Acestea sînt secvenţe de program avînd un caracter unitar şi stabil care îndeplinesc un număr redus de acţiuni bine precizate (§ 2.4.1). Un program constă dintr-o ierarhie de module (procese). Un modul al ierarhiei pe de o parte reprezintă un „întreg" (privit ca subsistem) iar pe de altă parte el poate fi o „parte" constitutivă a unui întreg de nivel superior (sistem). Un astfel de modul (proces) poate fi asimilat cu un holon. De asemenea nu întîmplător se vorbeşte despre sistem de calcul, sistem de operare, sistem de procese etc.

Observaţiile anterior formulate, prin particularizare, rămîn valabile şi pentru domeniul programării şi pot fi luate în considerare de proiectantul software.

A. Astfel în realizarea unui produs software complex, ne pornepte de la un fond deja existent de programe, module şi algoritmi care sînt adaptaţi şi asamblaţi conform necesităţilor programului. De regulă se utilizează algoritmi consacraţi cu performanţe ridicate metode verificate şi optimizate. Această tendinţă, din ce în ce mai vizibilă în programare îmbracă forme diferite începînd cu primitivele consacrate utilizate în proiectarea şi realizarea sistemelor de operare, continuînd cu macroinstrucţiile şi terminînd cu bibliotecile şi cu pachetele de programe specializate.

B. In ceea ce priveşte evoluţia sistemelor software se constată că ierarhizarea a pătruns mai încet datorită absenţei, la început, a unei metodologii şi a unei discipline de proiectare. A fost perioada în care complexul domina ierarhia, perioadă în care structurile amorfe se constituiau direct în sisteme complexe. Pe măsură însă ce s-a acumulat experienţă de programare, s-au cristalizat unele metode şi principii şi s-au făcut descoperiri teoretice importante, sistemele ierarhice au devenit din ce în ce mai răspîndite. La ora actuală se utilizează din ce în ce mai mult această metodă în realizarea unor noi sisteme complexe.

C. In sfîrşit, în ceea ce priveşte modul de acţionare a unor astfel de programe ierarhice se constată o analogie perfectă bazată pe acelaşi mecanism al comutării la iniţiativa unui semnal simplu (apel de rutină, mesaj sau semnal de întrerupere).

Pentru proiectantul software, sarcina creatoare constă în a defini şi încadra diversele activităţi într-un sistem ierarhic complex şi de a grupa procesele în unităţi operaţionale (holoni) într-o astfel de manieră încît subsistemele realizate să fie slab cuplate, permiţînd o manieră simplă dar eficientă de comandă a ierarhiei.

4.2. Arhitectura unui sistem de operare timp-real

4.2.1. Arhitectura ierarhică a unui sistem TR

Termenul de structură se reflectă direct în arhitectura unui sistem de calcul [Cr84b] .

Luînd în considerare aspectele prezentate pînă acum în această lucrare şi corelîndu-le cu cele precizate în subcapitolul anterior din punct de vedere structural, cele mai potrivite arhitecturi pentru sistemele timp-real sînt arhitecturile ierarhice structurate pe niveluri.

Într-o astfel de arhitectură ierarhică nivelurile inferioare implementează funcţiile de bază (elementare), iar pe măsură ce nivelul creşte, creşte şi complexitatea funcţiilor implementate. Regula de bază a unei astfel de arhitecturi se referă la faptul că funcţiile (obiectele) aflate pe niveluri superioare pot apela (utiliza) doar funcţiile de pe nivelurile inferioare lor; cu alte cuvinte orice nivel se bazează numai pe nivelurile aflate sub el şi nu pe cele aflate deasupra[CR81a].

In acest fel comenzile circulă în sistem numai de sus în jos prin mecanismul apelurilor proceselor sau rutinelor. Se precizează însă că există şi informaţii care circulă de jos în sus respectiv (semnale, răspunsuri, reacţii) care au menirea de a activa procesele ierarhiei drept răspuns al unor comenzi anterioare, sau de a furniza informaţii referitoare la maniera de execuţie a comenzilor. Aceste informaţii se pot opri la diferite niveluri ale ierarhiei.

O astfel de structură ierarhică apare în figura 4.2.1. Suportul arhitecturii software propuse îl constituie maşina hardware care constituie baza ierarhiei. In continuare apare executivul sistemului iar pe nivelul superior, procesele aplicaţiei (aplicaţia propriu-zisă). Se observă că de fapt există o structură formată din trei arhitecturi hardware, executiv (sistem de operare) şi sistem propriu-zis. La rîndul lor aceste arhitecturi au o structură ierarhică lucru care va fi prezentat în paragrafele următoare. Interfaţa cu ambianţa TR se realizează la nivelul arhitecturii hardware. Semnalele provenite din ambianţă se propagă de jos în sus, activînd prin mecanismul de tratare al întreruperilor procese specializate aflate pe diferite niveluri sau funcţii ale executivului. In continuare acestea pot genera comenzi care se materializează în apeluri de procese sau funcţii aflate pe nivelurile inferioare.

Adoptarea unei astfel de structuri are mai multe avantaje:

1° Reprezentabilitate şi claritate.

2° Modularitate.

3° Sens unic al circulaţiei comenzilor, - de sus în jos - şi sens unic al circulaţiei răspunsurilor (reacţiilor)- de jos în sus -.

4° Ortogonalitate înţelegînd prin aceasta proprietatea sistemului de a accepta modificări ale unor părţi ale structurii fără repercursiuni asupra celorlalte.

5. Implementare simplă.

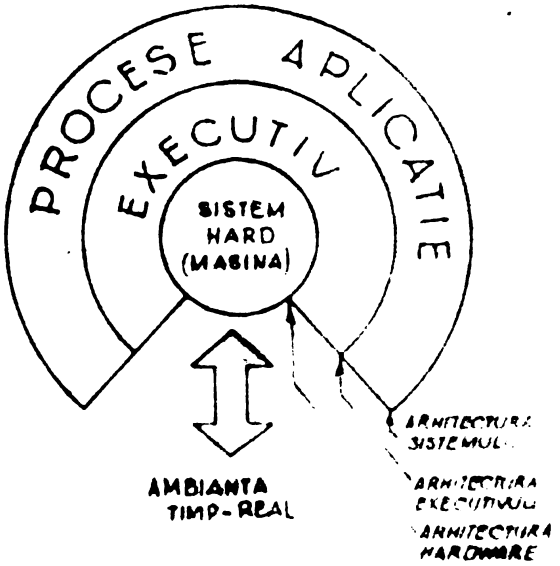


Fig.4.2.1. Arhitectura unui sistem timp-real.

4.2.2. Metoda ierarhică de construcţie

In proiectarea şi realizarea unei arhitecturi ierarhice de sistem se poate utiliza metoda construcţiei ierarhice [CRU1a] .

M.4.2.1. Metoda constă în proiectarea, realizarea şi testarea succesivă a nivelurilor sistemului, începînd de la bază şi înaintînd spre straturile superioare. Nivelurile sînt astfel concepute încît fiecare implementează un grup unitar de funcţii sau mecanisme care depind numai de nivelurile aflate dedesupt şi în nici un caz de cele aflate deasupra.

Sistemul se realizează pornind de la bază spre vîrf (tehnica bottom-up), proiectînd implementînd şi testînd fiecare nivel în parte. In această manieră, construcţia sistemului nu este dificilă şi practic, activitatea de punere la punct se realizează în paralel cu implementarea sa.

Pentru fiecare nivel în parte, metoda propusă de realizare este metoda programării structurate completată cu conceptul de orientare spre obiect (§ 4.2.4).

4.2.3. Factori care influenţează activitatea de concepţie şi proiectare a STR

A. Un prim factor îl constituie necesitatea dezvoltării de software TR specific diferitelor tipuri de aplicaţii, element care conferă noi dimensiuni activităţii de proiectare a STR. Astfel pe de o parte trebuiesc proiectate şi realizate arhitecturi de sisteme

care să satisfacă în mai mare măsură cerinţele unei ambianţe (TR), cerinţă care oferă un caracter particular (dedicat) proiectării. Pe de altă parte în proiectarea acestor arhitecturi trebuie urmărit un cadru stabil, conceput de o asemenea manieră încît programele realizate să fie reutilizabile în diferite aplicaţii, chiar pentru diferite arhitecturi hardware, cerinţă care oferă un caracter general proiectării [RG79] . Cele două cerinţe contradictorii sînt reprezentate grafic în figura 4.2.2. Astfel punctul 1 indică un software cu un grad maxim de reutilizare (independent de hard şi de aplicaţie) dar puţin eficient. Punctele 2 şi 4 reprezintă puncte de maximă eficienţă din punct de vedere al hardului respectiv al aplicaţiei, dar cu un grad redus de reutilizare. Punctul 3 reprezintă un software eficient. Suprafaţa 1234 reprezintă nivelul posibil de eficienţă pentru diferitele niveluri de dependenţă faţă de maşină sau aplicaţie.

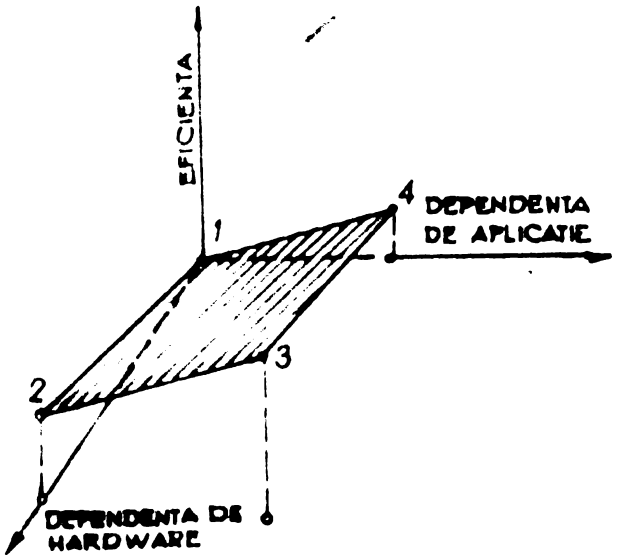


Fig.4.2.2. Reprezentarea grafică a raportului eficienţă/dependenţă.

M.4.2.2. Metoda preconizată a rezolva aceste probleme este metoda adaptării. Conform acestei metode software-ul timp-real se proiectează şi realizează pentru punctul 1 având un caracter pronunţat de generalitate, urmînd ca prin adaptări succesive să fie potrivit (acordat) într-o manieră controlată atît pe structura hardware cît şi pe aplicaţie, tinzîndu-se apre punctul 3.

Sarcina dezvoltării aplicaţiilor timp-real, respectiv a implementării, testării şi adaptării lor, revine „mediului de programare" componentă a SØTR care implementează funcţia de asistare a programatorului şi care pentru a materializa aceste deziderate dispune de funcţii specifice. Impactul pe care arhitectura „mediului de programare" îl exercită asupra arhitecturii generale a sistemului este scos în evidenţă în paragraful destinat proiectării acestei componente (§ 4.4).

B. Un alt factor important care influenţează activitatea de proiectare şi mai ales maniera de implementare a STR îl reprezintă conceptul de obiect.

In activitatea de concepţie şi proiectare a SØ, metoda programării structurate a fost dezvoltată cu scopul de a permite dezvoltarea de software eficient, uşor de întreţinut, cu un preţ de cost mai scăzut şi cu o productivitate mai ridicată. Pentru a îndeplini aceste scopuri

programarea structurată utilizează tehnici de descopunere şi de proiectare din aproape în aproape (stepwise refinement) [VB78], vizînd în primul rînd circulaţia controlată a fluxului informaţiei în sisteme şi programe individuale. In acest scop fiecare proces, (funcţie) se descompune în module bine definite precizîndu-se riguros interfeţele dintre ele: parametri, structuri de date, organizarea tabelelor, maniere de reprezentare a datelor, etc.

Cu toate că d.p.d.v. structural această metodă oferă avantaje majore, pot apare probleme deosebite dacă sînt necesare modificări ale datelor sau ale structurilor de date în cadrul modulelor. Astfel de modificări au implicaţii asupra tuturor modulelor care le utilizează.

Orientarea spre obiect pune la dispoziţie o metodă care extinde conceptul de structură din programarea convenţională şi asupra datelor şi reprezentării acestora.

Aceasta presupune pe de o parte, proiectarea structurată a bazei de date iar pe de altă parte o restructurare a modulelor ierarhiei astfel încît să se reducă la minimum numărul de module care au acces la o anumită structură de date (obiect). In plus aceste module se asociază cu datele (obiectele) pe care le deservesc.

In noua structură, modulele curente vor apela astfel de proceduri care manipulează obiecte pentru a modifica sau utiliza date în loc să realizeze accesul direct la ele. Astfel, deşi datele asociate unui obiect pot fi utilizate de mai multe module, numai procedurile direct asociate obiectului pot manipula structura de date a acestui.

Drept urmare, modificarea structurii sau a reprezentării datelor unui obiect, necesită modificarea doar a procedurilor direct asociate, în locul modificării tuturor modulelor care utilizează aceste date. In acest sens multe din sistemele moderne de operare sînt concepute ca o colecţie de tipuri de obiecte. Pentru fiecare tip de obiect sistemul dispune de un set de apeluri prin care tipul poate fi utilizat şi manipulat [JP79], [PN79], [LR80], [KC81], [PK81], [CL82], [BL82], [CC83].

Pornind de la aceste consideraţii se propune completarea metodei construcţiei ierarhice cu conceptul de orientare spre obiect, concretizat în structurarea datelor în obiecte ierarhice clar definite şi restrîngerea numărului de module care realizează accesul la ele. In acest fel se accentuează caracteristica de ortogonalitate a arhitecturii sistemului.

C. Tendinţele actuale care se manifestă în concepţia şi proiectarea SØ tind să desemneze structuri sau concepte fundamentale într-un sistem de operare, care marchează în mod pregnant sistemul şi pe care le denumesc paradigme. Paradigmele ocupă un rol central în activitatea de proiectare şi mai ales în cea de utilizare a sistemului imprimînd un caracter pronunţat de specificitate. Varietatea paradigmelor utilizate este foarte mare: fişierele în UNIX [RT74], capabilităţi în Hydra [Wu74], obiecte în STAR/ØS [JC77], [SF77], [SB77], servicii în AMØEBA [TM81]. Spre exemplu în UNIX toate comunicările sînt gîndite ca şi citiri şi scrieri în fişiere, în sistemele tradiţionale bazate pe capabilităţi, noţiunile de bază se referă la realizarea de operaţii asupra obiectelor, în AMØEBA la transmiterea şi primirea de mesaje de la porţi protejate, etc. In acest context, pornind de la observaţia că în SØTR un rol fundamental îl are circulaţia informaţiei şi legat de aceasta activităţile de coordonare şi cooperare a proceselor care în cadrul metodologiei propuse se realizează prin mesaje, se propune pentru categoria de sisteme definite în această lucrare paradigma mesaj.

4.3. Arhitectura unui „executiv" pentru un sistem TR

4.3.1. Structura ierarhică a arhitecturii executivului

In proiectarea arhitecturii executivului unui SØTR, pentru un sistem monoprocesor, punctul de pornire îl constituie funcţiile pe care acesta trebuie să le îndeplinească, funcţii precizate în § 2.3.3. [Cr80], [CP81], [Cr81a], [Cr84a].

Din punctul de vedere al modului lor de execuţie, funcţiile executivului pot fi împărţite în trei categorii:

a) funcţii care se execută integral cu întreruperile mascate, în regim de „monitor monolitic" (§ 3.3.2). In această categorie se încadrează funcţiile numerotate între (1)-(8)(§ 2.3.3).

b) funcţii care se execută parţial în regim de „monitor monolitic" funcţiile numerotate (9) - (12).

c) funcţii care nu se execută cu IT mascate - cele cu numere între (13) - (18).

Această partajare oferă argumente pentru gruparea funcţiilor executivului pe niveluri. Astfel funcţiile care se execută în regim de întreruperi mascate se pot dirija spre nucleul executivului (executiv inferior) grupate pe două subniveluri: cel inferior cu regim de monitor monolitic total (a) şi cel superior cu regim de monitor monolitic parţial (b). Acest lucru este necesar şi pentru a înlătura dezavantajele monitorului monolitic, prezentate în § 3.3.2,

acţionînd în sensul reducerii dimensiunilor sale. Restul funcţiilor executivului (c) se pot dirija spre nivelul superior al executivului şi ele pot fi implementate în forma unor procese sistem întreruptibile, asincrone care utilizează funcţiile nucleului. Toate acestea conduc la o arhitectură de executiv ierarhică structurată pe niveluri (fig.4.3.1).

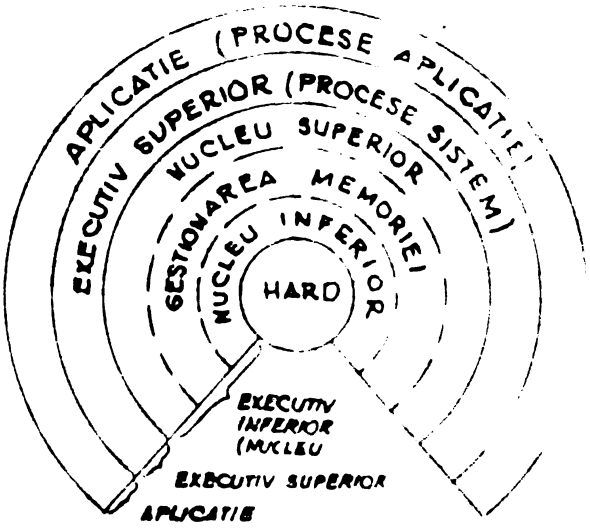


Fig.4.3.1. Structura ierarhică a executivului unui sistem TR.

Conceptul de nucleu (Kernel) este un concept modern mult utilizat în literatura de specialitate [Ha70], [SG77], [RR81], [Ho83], care datorită avantajelor oferite a fost adoptat şi în cadrul lucrării de faţă.

4.3.2. Nucleul inferior al executivului

A. Funcţiile nucleului inferior. Nucleul inferior al executivului cuprinde funcţiile care se execută integral în regim de monitor monolitic (M.3.3.1). Aceste funcţii se implementează în forma unor module şi rutine de executiv specifice care utilizează structuri de date specifice. Ele sînt activate de apeluri de rutine provenite din sistem sau de semnale şi IT provenite din sistem sau din ambianţa TR.

In continuare se vor trece în revistă cîteva maniere de concepţie şi posibilităţi de implementare a funcţiilor nucleului inferior.

(1). Mecanismul de comutare se poate implementa în forma a două rutine care se apelează ori de cîte ori se intră sau se iese din nucleu. Aceste rutine acţionează asupra stivei procesului care se execută curent în UC, salvînd starea procesului (adresa de revenire, conţinutul registrelor generale şi pointerul curent la stivă în stiva proprie. Mecanismul trebuie să permită şi rezolvarea unor apeluri încuibate (funcţii sistem care apelează alte funcţii sistem şi trebuie să fie complet transparent din punctul de vedere al procesului utilizator (la revenire în proces să redea conţinutul iniţial nealterat al stivei). In această manieră sistemul lucrează pentru fiecare proces în stiva proprie a procesului. Dotarea fiecărui proces cu stiva sa proprie, strict dimensionată, deşi are

dezavantajul unui consum mai ridicat de memorie, asigură o mare flexibilitate şi fiabilitate implementării şi simplifică mult mecanismul de comutare.

Dintre structurile de date ale nucleului, mecanismul are acces direct la tabelele de definiţie ale proceselor.

(2). Gestionarea timpului unităţii centrale se realizează prin divizarea acestuia între procesele pregătite pentru execuţie (multiprogramare). Divizarea timpului UC se poate implementa în două moduri:

a) la iniţiativa sistemului de întreruperi,IT de timp, de I/E, de sincronizare, etc.)

b) la iniţiativa proceselor (sincronizări de condiţie,funcţii speciale de comutare, etc.) [Tab0] .

In primul caz implementarea multiprogramării se asigură în mod automat, sarcina divizării timpului revenind exclusiv sistemului, în cazul al doilea această sarcină revine parţial sau total (în dependenţă de implementare) proceselor utilizator. Această ultimă manieră are avantajul simplităţii implementării dar este relativ dificil de utilizat şi de pus la punct. In toate cazurile, gestiunea timpului UC se încheie cu apelul rutinei care implementează politica de planificare în execuţie a proceselor. Se utilizează mecanismul de comutare şi modulele care păstrează evidenţa stării proceselor.

(3). Politica de planificare în execuţie a proceselor este implementată de un modul specializat al nucleului numit dispecer ale cărui atribuţii şi manieră de acţiune au fost precizate în § 3.5.1. Metodele de planificare propuse a fi utilizate pentru sistemele TR au fost prezentate detaliat în § 3.5.3. Se reamintesc cîteva din concluziile formulate:

a) Pentru sporirea eficienţei utilizării UC, este necesară suprapunerea activităţii UC cu activitatea periferice lor prin tratarea acestora la nivel de întreruperi.

b) Se pot utiliza metode de planificare dinamică atît în caz că se dispune de informaţie apriorică (M.3.5.1) cît şi în lipsa acesteia (M.3.5.2).

c) Se pot utiliza metode bazate pe prioritatea statică în cadrul cărora priorităţile proceselor se stabilesc cu ajutorul modelului pentru studiul planificării sistemelor de procese închise (M.3.5.3). Informaţiile apriorice se determină cu ajutorul funcţiilor de măsurare implementate în mediul de programare.

d) In cadrul metodei adaptării, un rol esenţial revine metodelor de planificare bazate pe informaţii istorice (M.3.5.4).

Sarcina esenţială, de a selecta următorul proces apt pentru execuţie şi de a-l lansa efectiv, dispecerul şi-o îndeplineşte apelînd modulele care modifică starea proceselor şi rutina de refacere a stării program. El intervine direct în tranzacţia procesorului pentru a înscrie procesul selectat.

Referitor la aceste aspecte, în sistemele de operare moderne se manifestă din ce în ce mai pregnant tendinţa ca acestea să implementeze mecanisme, iar utilizatorii să implementeze politici (strategii) pe baza mecanismelor puse la dispoziţie de sistem [Wu74], [CC83]. Această tendinţă extrem de utilă pentru sistemele cu caracter universal, nu poate fi definitorie pentru SDTR în care gradele de libertate admise sînt mai restrînse şi impun implementarea unor politici chiar în sistem din raţiuni de eficienţă şi performanţă. Chiar şi în aceste cazuri, există însă posibilitatea de a opta pentru diferite variante, după cum s-a precizat mai înainte.

(4). Mecanismul de sincronizare care se presupune, este cel bazat pe semafoare generalizate extinse (§ 3.3.4). Acest mecanism poate fi implementat în forma a două rutine indivizibile P şi V care acţionează asupra unei structuri de date specifice: obiectul semafor. Rutinele P şi V apelează rutinele mecanismului de comutare, modulele de modificare a stării proceselor şi modulul dispecer. Acest mecanism poate fi utilizat direct în implementarea excluziunii mutuale (M.3.3.4). De asemenea el stă la baza implementării altor funcţii şi mecanisme ale sistemului după cum se va vedea în continuare.

(5). Trecerea activităţilor dintr-o stare în alta şi menţinerea evidenţei fiecăreia se poate realiza prin module specifice ale nucleului care au acces la tabelele de definiţie ale proceselor. În concepţia şi implementarea acestor module se utilizează conceptul de obiect.

(6). În ceea ce priveşte tratarea flexibilă a întreruperilor interne şi externe se reaminteşte că întreruperile sînt activităţi ale căror rol este de a cupla rapid microprocesorul la un eveniment extern care îi solicită atenţia decuplîndu-l de la activitatea pe care o desfăşura în acel moment. Modul de tratare al IT depinde în primul rînd de maniera hardware în care acestea sînt implementate. În general, întreruperile sînt sesizate de către nucleul inferior al executivului prin lansarea asincronă în execuţie a unor rutine specializate dedicate unor grupuri de IT sau unor IT singulare. Din punctul de vedere al tratării software se disting două tipuri de întreruperi:

a) IT interne (de tip nucleu) care influenţează în mod direct activitatea nucleului (ceasul de timp real, întreruperi specifice) şi care sînt tratate de către acesta prin intermediul rutinelor specializate. Se precizează însă că aceste IT trebuie să fie cît mai reduse ca număr iar tratarea lor trebuie să fie simplă şi eficientă pentru ca, pe de o parte să nu mărească prea mult regia sistemului, iar pe de altă parte ţinînd cont că ele sînt tratate în regim de monitor monolitic să nu reducă timpul de răspuns (viteza de reacţie) a sistemului. Aceasta poate avea drept consecinţă directă pierderea unor întreruperi care soseso prea des, lucru inadmisibil într-o ambianţă TR.

b) IT externe (de tip executiv sau utilizator) care sînt sesizate de către nucleu dar care sînt tratate de către executivul superior sau direct de către utilizator, prin intermediul unor procese speciale (sistem sau utilizator) cu priorităţi adecvate. Pentru tratarea acestor IT se poate utiliza o metodă derivată din implementarea sincronizării de condiţie (M.3.3.4). Astfel, fiecărei IT (sau grup de IT) care soseşte pe o linie,i se asociază un semafor cu valoarea iniţială 0. Adresa fiecărui semafor este cunoscută de către procesul sistem (utilizator) care tratează IT respectivă şi care execută iniţial o operaţie P asupra semaforului blocîndu-se în el. Rutinele nucleului care sesizează IT,execută o operaţie V pe semaforul specific deblocînd procesul care devine astfel apt spre a fi selectat şi lansat în execuţie de către dispecer.

Acest lucru se întîmplă cu atît mai curînd cu cît prioritatea procesului este mai mare.

(7). Funcţia de iniţializare a sistemului se poate implementa în forma unei rutine care tratează întreruperea generată de punerea în funcţiune a sistemului. Apariţia acestei întreruperi produce în mod automat saltul la o rutină de iniţializare aflată în nucleul inferior care:

- iniţializează unele zone de date
- iniţializează unele contoare şi semafoare ale sistemului
- iniţializează ceasul sistem
- iniţializează sistemul de întreruperi
- pe baza unei tabele repertoriu a proceselor, iniţializează tabele de definiţie ale proceselor
- trece în starea „pregătit" toate procesele sistemului
- dă comanda dispecerului, care va selecta şi va lansa pe rînd în execuţie procesele, în ordinea priorităţilor (sau conform politicii implementate).

Toate aceste activităţi sînt necesare deoarece executivul împreună cu procesele sistem şi procesele aplicaţie se găsesc în memoria EPRØM a sistemului, în timp ce structurile de date aferente lor, în memoria RAM care este volatilă. Drept urmare, prima activitate care trebuie executată la conectarea sistemului este tocmai iniţializarea bazei de date a executivului, sarcină complexă care revine acestei rutine. Iniţializarea zonelor proprii de date ale proceselor se realizează prin rutine specializate ale acestora, a căror execuţie precede codul propriu-zis al procesului (§ 5.2).

(8). Gestionarea memoriei apare ca un sub nivel intermediar între subnivelurile executivului deoarece utilizează unele funcţii ale nucleului şi este apelată de funcţiile nucleului superior. Din cele ce au fost precizate în capitolul 2 (§ 2.2.4) a rezultat însă cu claritate că STR dispun de memorii de dimensiuni reduse strict adaptate aplicaţiei, motiv pentru care funcţia de gestionare a memoriei este foarte restrînsă. Marea majoritate a executivelor TR nu implementează funcţii de gestiune a memoriei, aceasta realizîndu-se prin alocare statică optimizată (zone de memorie strict dimensionate funcţie de caracteristicile aplicaţiei).

B. Structuri de date ale nucleului inferior. Structurile de date ale nucleului inferior se constituie în jurul a trei obiecte: tabelă de definiţie proces, tranzacţie procesor şi semafor.

(1) Tabela de definiţie a procesului este o structură de obiect specifică fiecărui proces care conţine: starea procesului, prioritatea, semafoarele asociate portului de comunicare, cîmp pentru înlănţuirea procesului în coada de aşteptare a unui semafor, pointerul stivei proprii, pointerul zonei de mesaje (port-ul), adresa zonei de cod şi alte informaţii dependente de implementare. Mulţimea tabelelor proceselor sistemului formează tabela de procese, tabelă la care au acces doar modulele nucleului care asigură trecerea activităţilor dintr-o stare în alta.

(2) Tranzacţia procesorului este structura care păstrează evidenţa procesului activ în procesor. Se actualizează de către dispecer.

(3) Semaforul este o structură care conţine 2 cîmpuri: semaforul propriu-zis şi pointerul la primul proces care aşteaptă. La aceste cîmpuri au acces numai rutinele P şi V.

4.3.3. Nucleul superior al executivului

A. Funcţiile nucleului superior. După cum s-a precizat în § 4.3.1, nucleul superior include funcţiile care se execută parţial în regim de monitor monolitic şi care utilizează funcţiile nucleului inferior. Dintre acestea se amintesc:

(9). Mecanismul de cooperare şi comunicare interprocese. Mecanismul propus a fi implementat este cel definit în § 3.4.2, care se bazează pe conceptul de mesaj (D.3.4.2). El poate fi implementat în forma a două rutine executiv care realizează funcţiile operatorilor SEND respectiv REC (D.3.4.3), rutine care acţionează asupra unei structuri de date specifice: bufferul de comunicare care păstrează mesajele. Politica de gestionare a bufferului este FIFO (primul venit, primul servit). Posibilităţile de sincronizare şi comunicare ale mecanismului au fost precizate (§ 3.4.5).

Din analiza efectuată în acelaşi paragraf, au rezultat o serie de elemente care trebuiesc luate în considerare în etapa de proiectare, în vederea particularizării implementării mecanismului. Dintre acestea se precizează:

(a) Posibilitatea stabilirii dimensiunii maxime N a bufferului de comunicare (M.3.4.1). Pentru a realiza însă acest lucru este necesar un mediu de programare cu funcţii de măsură adecvate. In ceea ce priveşte alocarea zonelor de memorie pentru buffere, în absenţa funcţiei de gestionare a memoriei care ar permite gestionarea dinamică a zonelor bufferelor, se realizează prin rezervarea statică a unei zone pentru transmiterea mesajelor în cadrul fiecărui proces.

(b) Utilizarea tehnicii „port"-urilor în implementarea mecanismului de comunicare şi cooperare.

(c) Implementarea SEND-ului cu blocare temporară şi a REC-ului cu blocare.

(d) Implementarea mesajului de tip prioritar (M.3.4.4). Mecanismul de comunicare se bazează pe funcţiile nucleului inferior respectiv pe mecanismele de comutare şi sincronizare ale acestuia. Semafoarele asociate „port"-ului se păstrează în tabela de definiţie a procesului, împreună cu adresa zonei de mesaje (buffer de comunicare). Ele au rolul de a asigura pe de o parte (N,0)-sincronizarea (§ 3.4.2), iar pe de altă parte de a rezolva accesul în regim de excluziune mutuală la buffer, deoarece SEND-ul şi REC-ul sînt funcţii întreruptibile.

(10). Păstrarea evidenţei timpului real. Legat de timpul real în sistemele TR se pun următoarele probleme:

(a) implementarea ceasului de timp intern care guvernează activitatea sistemului.

(b) implementarea unor ceasuri suplimentare, cu diferite baze de timp necesare sincronizării unor evenimente.

(c) implementarea unei funcţii care să permită utilizatorului să cunoască timpul curent.

(a). Implementarea ceasului de timp intern al sistemului este legată de prezenţa unui orologiu hardware (de regulă programabil) (I 8253) care la anumite intervale de timp generează întreruperi. Aceste întreruperi sînt tratate de către o rutină specializată a executivului care actualizează baza de date referitoare la timpul sistem. Această rutină se găseşte în nucleul inferior deoarece se execută cu IT mascate.

(b). Ceasurile suplimentare pot fi implementate şi ele în două moduri:

(1) prin circuite hardware (spre exemplu I 8253 are trei ceasuri interne în aceeaşi capsulă). Dacă este necesar se poate multiplica numărul de capsule, după cum s-a procedat la sistemul FELIX-800 care conţine 5 circuite I8253.

(2) prin software. In acest caz, prin program se generează diferite baze de timp pornind de la cuanta de bază a ceasului intern, utilizînd contoare software.

(c). Implementarea unei funcţii de cunoaştere a timpului sistem se poate realiza în forma unei rutine întreruptibile care consultă ceasul sistem şi furnizează timpul citit.

(11). Tratarea evenimentelor dependente de timp, este legată de implementarea unui mecanism de lansare întîrziată în execuţie a unor procese, durata întîrzierii fiind prescrisă. Acest mecanism se poate implementa în forma unei rutine care asigură întîrzierea procesului (apelant) un interval precizat de timp. Mecanismul de întîrziere se bazează pe semafoare.

Rutina acţionează asupra unei tabele care conţine intrări pentru procesele care se pot întîrzia, fiecare intrare cuprinzînd durata de întîrziere şi semaforul de întîrziere predestinat procesului respectiv, cu valoarea iniţială nulă. In momentul apelului rutinei, în intrarea corespunzătoare procesului apelant, se iniţializează durata cu valoarea prescrisă a întîrzierii, iar pe semaforul corespunzător se execută o operaţie P care produce blocarea procesului. La scurgerea unei cuante prestabilite de timp se baleiază această tabelă, se decrementează duratele pentru intrările active,

iar acolo unde durata a devenit nulă, se execută o operaţie V asupra semaforului respectiv,deblocînd procesul.

Dimensiunea tabelei poate fi precis stabilită, deoarece aplicaţia fiind cunoscută în întregime se poate cunoaşte exact numărul proceselor care se întîrzie.

(12). Implementarea unui mecanism de alocare a resurselor. După cum s-a precizat în § 3.6.1, într-un sistem de calcul TR se consideră două categorii de resurse:

(a) resurse fixe

(b) resurse consumabile.

(a) Dintre resursele fixe, fac obiectul alocării doar acelea care nu au fost prerepartizate proceselor. Ele sînt grupate într-un "pool" de resurse de unde sînt alocate. Mecanismul de alocare pentru resursele fixe se bazează pe utilizarea semafoarelor generalizate extinse şi a fost prezentat în capitolul 3 (M.3.3.5). Mecanismul se poate implementa în forma a două rutine: una care alocă resursa, alta care o eliberează. Fiecărui tip de resursă i se asociază un semafor specific a cărui valoarea iniţială este egală cu numărul de resurse disponibile. Solicitarea face o operaţie P asupra acestui semafor, eliberarea o operaţie V.

Metoda aceasta asigură atît evidenţa utilizării resurselor cît şi mecanismul de sincronizare aferent întrucît la epuizarea unei resurse de un anumit tip, provoacă blocarea procesului solicitant pînă la proxima eliberare a unei resurse de acel tip. Mecanismul este simplu şi eficient. Semafoarele asociate resurselor aparţin bazei de date a nucleului. Mecanismul utilizează funcţiile P şi V ale nucleului inferior.

(b) Deoarece în cadrul STR gama de resurse consumabile se poate restrînge la mesaje şi întreruperi de diferite tipuri, este uşor de observat că problema alocării acestora este rezolvată implicit prin implementarea mecanismului de cooperare şi comunicare interprocese prin mesaje respectiv prin mecanismul de tratare al IT. Baza de date utilizată în acest scop este cea specifică mecanismelor amintite.

Legat de alocarea resurselor trebuie reconsiderată problema interblocărilor. In acest sens este necesară reluarea concluziilor rezultate din analiza aprofundată a interblocărilor efectuată în subcapitolul 3.6.

Se porneşte de la teorema T.3.6.2 care precizează că într-un sistem de procese constrîns la o comunicare ierarhică nu poate apare interblocarea. Aceasta este situaţia ideală în care problema interblocărilor este rezolvată automat, dar de multe ori procesele aplicaţiei

nu pot fi încadrate în totalitate într-o astfel de ierarhie.

Ca atare, în aceste condiţii se pot aborda trei căi de rezolvare.

(a) In primul rînd se poate face apel la una din metodele de evitare a interblocărilor: prin eliberarea tuturor resurselor (M.3.6.1) prin alocarea cantităţii maxime de resurse (M.3.6.2) sau prin alocarea într-o ordine prestabilită (M.3.6.3). In oricare din aceste cazuri, rutinele de alocare şi de eliberare devin mai complicate, ele trebuind să implementeze metoda aleasă. In plus trebuiesc definite şi structuri de date suplimentare necesare metodei implementate.

(b) In al doilea rînd se recomandă (M.3.6.5), metodă care permite detecţia „off-line" a interblocărilor, prin implementarea mecanismului de detecţie (§ 3.6.4) ca o opţiune a „mediului de programare". La detecţia unor situaţii de interblocare se adoptă măsuri adecvate prin introducerea unor relaţii de precedenţă suplimentare între procesele implicate în interblocare.

(c) In situaţia în care se ajunge la concluzia că este absolut necesară implementarea unui mecanism de detecţie a interblocărilor în executivul sistemului (M.3.6.4) sau ca opţiune a acestuia (M.3.6.6), se recomandă ca el să fie plasat într-un proces sistem al executivului superior (§ 4.3.5). In acest caz structura datelor executivului trebuie completată cu structuri de date specifice acestei activităţi. In aceleaşi coordonate poate fi rezolvată şi problema evitării interblocărilor conform celor precizate în § 3.6.5.

B. Structuri de date ale nucleului superior. Structurile de date ale nucleului superior se constituie în jurul a patru obiecte: mesaj, zonă mesaj, ceas sistem şi tabela ceas.

(1) Mesajul este un obiect, care d.p.d.v. sistem cuprinde două tipuri de informaţii:

(a) informaţii sistem (lungime mesaj, proces destinaţie, tip de mesaj)

(b) text propriu-zis.

(2) Zona mesaj (portul) este specifică fiecărui proces în parte şi în ea se păstrează mesajele adresate procesului. Adresa acestei zone împreună cu semafoarele asociate ei se păstrează în tabela de definiţie a procesului. In această zonă au acces numai rutinele SEND şi REC.

(3) Ceasul sistem este o structură de date care păstrează ceasul sistemului. Aceasta poate fi modificat doar de rutina specia-

lizată a executivului.

(4) Tabela ceas conţine intrări pentru toate procesele care se pot întîrzia în sistem. Structura unei intrări, a fost precizată anterior. La această tabelă au acces funcţia de întîrziere şi rutinele de tratare a ceasurilor suplimentare.

4.3.4. Structurarea ierarhică a obiectelor nucleului

In proiectarea structurilor de date ale nucleului se utilizează orientarea spre obiect, în spiritul celor precizate în § 4.2.4.

Este interesant de remarcat faptul că obiectele nucleului pot fi definite de o asemenea manieră încît să alcătuiască o structură de date ierarhică,(fig.4.3.2), care respectă principiul stabilit al ierarhiei. Astfel o structură (obiect) situată pe un nivel superior, se bazează numai pe structurile de date aflate pe nivelurile inferioare ei. Această abordare ierarhică, pe lîngă avantajele intrinseci pe care le conferă (§ 4.2.1), se răsfrînge direct asupra activităţii de proiectare simplificînd şi sistematizînd această activitate.

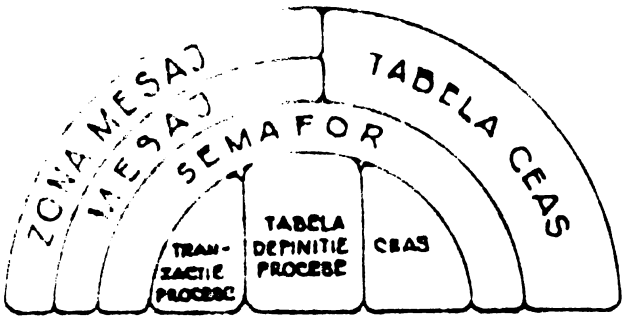


Fig.4.3.2.Structura ierarhică a obiectelor bazei de date a nucleului executivului.

4.3.5. Executivul superior. Procese sistem

A. Funcţiile executivului superior. Executivul superior cuprinde funcţiile SØ care sînt intreruptibile (se execută cu IT nemascate). Pentru simplificarea activităţii de proiectare şi implementare,acestea se pot realiza în forma unor procese sistem asincrone activate prin mecanismul întreruperilor (semafoarelor) sau prin mecanismul mesajelor (13). Funcţiile sistem suplimentare sînt activităţi ale sistemului care nu au caracter de generalitate şi care de regulă depind de specificul aplicaţiei concrete, de tipul de sistem sau de maniera de structurare şi de implementare. Ele se grupează în procese cu prioritate medie, de obicei mai ridicată decît a proceselor aplicaţiei. Spre exemplu procesul „poştaş" în sistemele multiprocesor (§ 4.6), procesele destinate gestionării unor resurse speciale, implementării unor funcţii de detecţie a interblocărilor, de măsurare, implementării unor politici specifice, etc. De regulă aceste procese sînt activate prin mecanismul semafoarelor.

(14). Tratarea întreruperilor externe. După cum s-a precizat în § 4.3.2, cu excepţia IT de tip nucleu care sînt tratate în totalitate în executivul inferior, restul IT sînt rezolvate de nivelurile superioare ale arhitecturii software. Este vorba pe de o parte de IT care vizează activitatea executivului şi pe de altă parte de IT care vizează direct aplicaţia. Tratarea acestor IT se realizează de către procese specializate (sistem sau utilizator) conform metodologiei prezentate în § 4.3.2, bazate pe mecanismul semafoarelor. Caracteristica comună a acestor procese este aceea că ele se blochează pe un semafor specific, din care sînt deblocate la o IT. Ele se pot găsi în executivul superior pentru IT sistem diversele niveluri ale ierarhiei aplicaţiei pentru IT extern

(15). Gestionarea dispozitivelor periferice şi tratarea ţiilor I/E se realizează prin intermediul unor procese sistem specializate, cîte unul pentru fiecare dispozitiv periferic, numite drivere. Aceste procese acţionează la iniţiativa proceselor aplicaţie prin mecanismul mesajelor. Fiecare driver are un caracter ciclic tratează mesajele caracteristice dispozitivului periferic pe care-l deserveşte. Structura unui astfel de proces apare în figura 4.3.3. Procesul începe cu un REC prin intermediul căruia primeşte comenzile în forma unor SND-uri, şi în care se blochează în absenţa acestora. In continuare pe lungimea specificată în comandă, transferă informaţii la nivel de octet sau de bit, funcţie de tipul dispozitivului periferic şi de circuitele de interfaţă ale acestuia. După fiecare lansare de operaţie periferică, procesul se blochează în semaforul specific aşteptînd IT de terminare a operaţiei lansate, materializată într-o operaţie V asupra aceluiaşi semafor. Bucla se reia pînă la epuizarea lungimii de transfer solicitate. In final driverul transmite procesului solicitant un SEND cuprinzînd informaţia citită sau confirmarea terminării operaţiei I/E solicitate, (în cazul scrierii), dacă se precizează în mod expres acest lucru (vezi mesajul cu răspuns (M.3.4.5.b)).

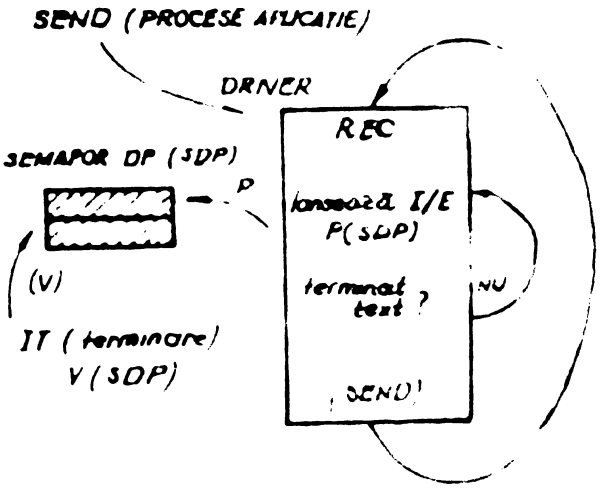


Fig.4.3.3. Structura de principiu a unui driver.

Această manieră de lucru asigură suprapunerea activităţii UC cu cea a DP-urilor întrucît în timpul în care driverul este blocat, în UC se execută alte procese aflate în starea pregătit. Desigur există posibilitatea tratării operaţiilor periferice şi în absenţa întreruperilor, caz în care testarea terminării operaţiei lansate se face de către driver prin „testare activă". In acest caz nu mai este posibilă suprapunerea activităţii UC cu cea a perifericuluí deoarece odată lansat, driverul îşi continuă execuţia pînă la terminarea completă a textului pe care îl citeşte (scrie). Nu este o metodă indicată, deoarece iroseşte timpul UC prin aşteptare activă.

(16),(17),(18). In ceea ce priveşte localizarea şi rezolvarea unor tipuri de erori şi avarii aceasta este strîns legată de reconfigurarea sistemului şi redirectarea unor operaţii periferice [CP78]. Această activitate se poate realiza în mai multe feluri:

(a) prin procesele care tratează IT şi care pot conţine secvenţe de detectare şi rezolvare a unor tipuri de erori. Astfel driverele pot conţine secvenţe de analiză a răspunsului DP-ului şi în caz de erori reparabile pot încerca rezolvarea lor prin reluarea operaţiei periferic

(b) prin procese specializate pe IT de avarii cu sarcini precise de semnalizare şi eventual de reconfigurare a sistemului sau redirectare a operaţiilor periferice. Aceste activităţi au o importanţă deosebită în sistemele care lucrează în ambianţe care necesită un grad sporit de fiabilitate, în care din condiţii de siguranţă, unele dintre echipamente se păstrează în mai multe exemplare sau dacă acest lucru nu e posibil funcţiile unor echipamente defecte sînt preluate de către altele în stare de funcţionare.

Funcţiile de redirectare şi reconfigurare au un grad ridicat de specificitate depinzînd în mod direct pe de o parte de structura hardware a sistemului, de natura echipamentelor sale periferice, iar pe de altă parte de caracteristicile concrete ale aplicaţiei.

B. <u>Structuri de date ale executivului superior.</u> Executivul superior nu implementează structuri de date proprii, utilizînd structurile definite în nucleu. Aceasta însă nu exclude posibilitatea ca procesele sistem să-şi definească structuri de date locale în funcţie de necesităţi.

4.4. <u>Arhitectura „mediului de programare"</u>

După cum s-a mai precizat sarcina esenţială a „mediului de programare" se referă la asistarea programatorului în dezvoltarea de aplicaţii TR.

Din concluziile prezentate în § 2.2.4 rezultă însă că sistemele cu microprocesor care implementează aplicaţii concrete au o configura-

ţie restrînsă din raţiuni de eficienţă, deci ele nu pot implementa funcţiile unui „mediu de programare". In consecinţă sarcina dezvoltării de aplicaţii trebuie rezolvată astfel. In timp s-au conturat în acest sens două metode: simularea şi sistemele de dezvoltare.

4.4.1. „Mediu de programare" simulat

Una din tehnicile larg răspîndite încă de la apariţia microprocesoarelor constă în traducerea şi verificarea programelor pentru microsisteme cu ajutorul unor calculatoare de capacitate mai mare [Pe81]. In acest caz, calculatorul mare, numit „cross-computer", implementează două programe:

a) un program asamblor care acceptă programe în limbajul de asamblare al microprocesorului traducîndu-le în codul maşină al acestui

b) un program simulator care execută programul obiect furnizat de asamblor, interpretînd fiecare instrucţie. In ambele faze, programatorul are acces la diverse informaţii despre program care îl ajută la depistarea şi corectarea unor erori (listing sursă, mesaje de eroare, vidaje de memorie parţiale, conţinut registre, execuţie pas cu pas, etc). Programul simulator poate fi complicat şi mai mult prin completarea sa cu funcţii de măsurare, trasare, evidenţa timpului, etc. necesare dezvoltării de aplicaţii timp-real.

Cu toate acestea facilităţi, unele greşeli de program rămîn nedescoperite, în special cele legate de conexiunea cu partea de echipament, respectiv cele legate de timpul-real, motiv pentru care această metodă este puţin utilizată la ora actuală.

Pentru a facilita testarea ansamblului echipamente-programe, s-au dezvoltat noi tehnici. Acestea se bazează pe emularea (simularea prin hardware) a microprocesorului, cu ajutorul unui sistem relativ complex cuprinzînd un minicalculator sau un alt microprocesor. Emulatorul execută programele realizate pentru aplicaţia respectivă, fiind prevăzut cu facilităţi complexe de depanare a programelor. Utilizarea acestei tehnici este avantajoasă pentru aplicaţii care se dezvoltă în serie medie şi mare, deoarece presupune o investiţie iniţială mult mai mare decît metoda simulării prin program.

Metoda implementării „mediului de programare" prin simulare prezintă avantajul implementării facile (chiar prin generare automată de cross-software) însă preţul de cost al programelor realizate prin simulare este mai ridicat decît al celor realizate prin alte metode, iar unele aspecte legate de întreruperi, operaţii I/E, timp-real nu pot fi puse în evidenţă decît în momentul utilizării

programelor [Pe81] .

4.4.2. „Mediu de programare" implementat pe un sistem de dezvoltare

Dezavantajele metodelor de simulare, au condus la ideea implementării „mediului de programare" pe un sistem de calcul bazat pe un microprocesor identic sau aparţinînd familiei de µP pentru care se dezvoltă aplicaţia. Un astfel de sistem se numeşte sistem de dezvoltare.

Un sistem de dezvoltare conţine pe lîngă µP propriu-zis o memorie în general mai mare decît a sistemelor de aplicaţie, precum şi o gamă mai largă de periferice (display, lector - perforator de bandă, imprimantă, casete magnetice, floppy-discuri, etc.). Pe lîngă configuraţia de echipamente, sistemul de dezvoltare este prevăzut şi cu un sistem de operare propice realizării, testării şi depanării programelor.

In coordonatele metodologiei de proiectare prefigurate în acest capitol, pentru un sistem de dezvoltare aplicaţii TR se propune o structură software originală de arhitectură ierarhică structurată pe niveluri. Această arhitectură rezultă din completarea arhitecturii propuse pentru un sistem TR (§ 4.2.1) cu o structură suplimentară, numită „mediu de programare" ale cărui funcţii au fost precizate (§ 2.3.4). „Mediul de programare" utilizează funcţiile executivului şi are acces direct la maşina hardware pentru a implementa funcţiile monitorului interactiv (fig.4.4.1).

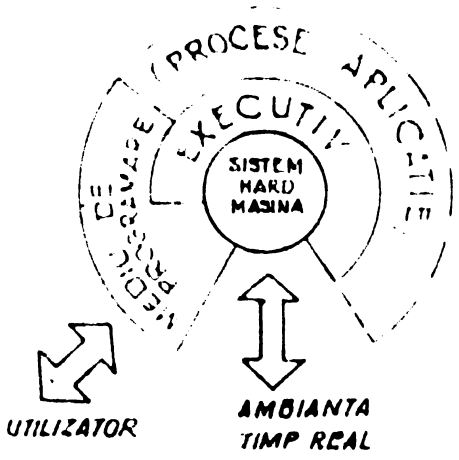


Fig.4.4.1. Arhitectura unui sistem de dezvoltare TR.

Această componentă este însă slab cuplată cu restul arhitecturii, sarcinile sale referindu-se exclusiv la dezvoltarea de procese aplicaţie, la dezvoltarea de aplicaţii propriu-zise şi la teste şi măsurători de timp-real.

Conform celor prezentate în § 4.1 şi în § 4.2.3, din punctul de vedere al aplicaţiei, executivul reprezintă partea general reutilizabilă, valabilă pentru o categorie mai largă de aplicaţii, iar procesele aplicaţiei reprezintă partea variabilă particulară a unei aplicaţii concrete acţionînd într-o ambianţă concretă. Rolul mediului de programare, în plus faţă de cele anterior precizate, este acela de a genera partea variabilă şi de a armoniza interfaţa dintre cele două părţi conform metodei adaptării (M.4.2.2). Pentru a implementa

deziderateleacestei metode „mediul de programare" poate utiliza trei categorii de informaţii:

a) informaţii rezultate din funcţiile de testare şi de măsurare proprii

b) informaţii rezultate din exploatarea practică a unor sisteme echivalente

c) informaţii furnizate de studiul pe modele „off-line" a arhitecturii sistemului de procese concurente care materializează aplicaţia. De regulă, aceste modele presupun procese neinterpretate, motiv pentru care ele se pot implementa în limbaje superioare şi pe alte categorii de sisteme de calcul. Spre exemplu, modelele preconizate în această lucrare şi anume: model pentru studiul determinării sistemelor (§ 3.2.2), model pentru studiul planificării sistemelor închise de procese (§ 3.5.3) şi model pentru studiul stării resurselor unui sistem de procese (§ 3.6.1), au fost implementate în limbajul PASCAL şi rulate pe calculatoarele FELIX C 256 (512) şi M18.

4.4.3. Implementarea arhitecturii unui „mediu de programare" pe un sistem de dezvoltare

Un „mediu de programare" se poate implementa în forma unui program interactiv construit în jurul unui interpretator de comenzi. Acesta primeşte comenzile de la utilizator, lansează rutinele care le execută şi revine aşteptînd comanda următoare. După cum s-a mai precizat, „mediul de programare" este slab cuplat cu restul arhitecturii fiind invizibil pentru aceasta. În schimb el are acces la componentele arhitecturii, putînd genera unele componente ale acesteia (procese aplicaţie).

În ceea ce priveşte implementarea funcţiilor de dezvoltare a proceselor (§ 4.3.4), asamblarea (1) şi editarea (2) sînt clasice şi nu ridică probleme deosebite sub aspectul proiectării. Monitorul interactiv (3) trebuie să implementeze minimum funcţii de vizualizare a conţinutului unor zone de memorie, inserţia şi substituţia conţinutului unor locaţii, vizualizarea şi poziţionarea conţinutului registrelor, comenzi de lansare în execuţie a codului obiect, implementarea unor puncte de întrerupere (break-pointuri).

O menţiune specială trebuie făcută în legătură cu asigurarea portabilităţii (4) software-ului generat [Crăiţ] . În acest sens se conturează două metode.

(a) Utilizarea benzii perforate drept suport al programului generat în vederea implantării software-ului pe sistemul destinaţie.

(b) Dotarea sistemului de dezvoltare cu facilităţi pentru arderea de memorii EPROM în care se înscrie direct software-ul generat.

Funcţiile de dezvoltare a proceselor nu apelează în mod direct funcţiile executivului. Acestea sînt însă apelate după necesităţi de către procesele aplicaţie care se generează.

In ceeea ce priveşte funcţiile de dezvoltare a aplicaţiei (5)-(9) ele servesc la crearea, punerea în funcţiune şi urmărirea stării aplicaţiei. In acest scop, ele au acces direct la baza de date a executivului, respectiv la tabelele de definiţie ale proceselor pe care le crează şi le exploatează. Pentru aceasta, mediul de programare foloseşte o tabelă repertoriu a proceselor pe care le are în evidenţă. Funcţia de iniţializare a sistemului apelează direct rutina specializată a executivului.

Funcţiile de testare şi măsurare (10)-(13), apelează în mod direct funcţiile executivului pentru determinarea timpului sistem, măsurarea unor intervale de timp sau întîrzierea unor procese. Ca manieră de lucru, intervenţia funcţiilor de măsurare în codul obiect se poate realiza la nivelul unor punote de măsurare prevăzute de către utilizator. Iniţial aceste punote sînt inerte (NØP-uri) ele putînd fi activate prin comenzi lansate din „mediul de programare". Dialogul dintre codul obiect şi rutinele de măsurare se realizează prin intermediul unor întreruperi generate prin program (RST-uri). Rutinele de măsurare propriu-zise se pot implementa în forma unor rutine care tratează aceste întreruperi.

Se pot imagina şi versiuni mai sofisticate, care să nu necesite prevederea în codul obiect a punctelor de măsurare, lucru care însă complică mult rutinele de tratare.

4.5. Arhitectura aplicaţiei. Procese aplicaţie

In cadrul STR, arhitectura aplicaţiei este strîns legată şi depinde în mod intim de arhitectura executivului utilizînd funcţiile acestuia (fig.4.3.1).

Din motivele prezentate pe parcursul acestui capitol, şi pentru sistemul de procese care implementează aplicaţia se recomandă o arhitectură ierarhică structurată pe niveluri. Acest lucru este bazat pe observaţia desprinsă din capitolul 3 dedicat studiului acestor sisteme, conform căreia forma lor cea mai generală este cea corespunzătoare unui graf închis. Ori din punct de vedere structural, structura de graf este identică cu o arhitectură ierarhică structurată pe niveluri.

Concepţia, proiectarea, realizarea şi testarea unei astfel de arhitecturi se realizează în mai multe etape.

In prima etapă, sarcina cea mai dificilă care revine proiectantului este aceea ca pornind de la condiţiile concrete ale aplicaţiei, să delimiteze procesele, să precizeze clar funcţiile acestora să stabilească interacţiunile dintre ele şi pe această bază să implementeze procesele într-o arhitectură ierarhică (structură de graf).

Se precizează că procesele sistem nu trebuiesc introduse structura de graf, ele fiind considerate procese colaterale care execută în regim de subrutină sistem pentru procesele care le apelează.

Odată sistemul stabilit, se trece la etapa a doua în care verifică dacă acesta conţine procese partajate, caz în care conform metodei multiplicării (M.3.4.7) va fi transformat într-un sistem normal. In continuare în cadrul acestei etape se determină forma minimă a grafului sistemului (M.3.2.1), se studiază, dacă este cazul, determinarea sistemului (M.3.2.2) şi se determină graful paralel maximal (M.3.2.3). Structura astfel obţinută reprezintă graful (arhitecturu) implementabil.

Pentru a realiza aceste obiective se utilizează modelul pentru studiul determinării sistemelor de procese definit în § 3.2.2.

. Se precizează că etapa II-a nu e necesar să fie parcursă dacă structura de graf rezultată în urma primei etape este simplă şi nu justifică acest lucru. Se mai precizează faptul că în ambele etape, procesele sînt neinterpretate (§ 3.1 , pentru acest moment funcţiile lor concrete neprezentînd importanţa.

In continuare se poate trece la activitatea de realizare, implementare şi testare individuală a proceselor arhitecturii. In această activitate se va ţine cont de implementarea relaţiilor de precedenţă interprocese în cadrul structurii de graf (M.3.4.6) ,de implementarea cooperărilor de tip producător/consumator în variantele mai mulţi producători/un consumator (M.3.4.2) şi un producător/mai mulţi consumatori (M.3.4.3). In implementare este necesar să se aibă în vedere utilizarea unor informaţii conţinute în mesajele care circulă între procese stabilind convenţii precise în acest sens, care trebuiesc riguros respectate. Este vorba spre exemplu de stabilirea identităţii procesului emiţător (M.3.4.5a) sau de implementarea mesajelor cu răspuns (M.3.4.5b). In funcţie necesităţi utilizatorul poate stabili şi alte convenţii.

Procesele aplicaţiei pot utiliza funcţiile şi mecanismele executivului fără restricţii. In funcţie de necesităţi, fiecare proces îşi poate defini propria sa bază de date. Variabilele partajate pot fi protejate prin semafoare specifice sau pot fi închise în procese manipulator care rezolvă problema accesului exclusiv la ele (procese monitor). Mecanismul de sincronizare recomandat este cel bazat pe mesaje. Mecanismul bazat pe semafoare se va utiliza doar pentru procesele sistem sau pentru procesele aplicaţiei care tratează întreruperi, utilizarea lui incorectă conducînd rapid la interblocări.

O atenţie deosebită trebuie acordată proceselor care tratează întreruperi externe, procese care din punctul de vedere al sistemului sînt procese care iniţiază ramuri ale ierarhiei. Activitatea de generare propriu-zisă presupune scrierea textului sursă a fiecărui proces în parte, asamblarea, testarea şi corectarea lui. In paralel se va realiza şi gestionarea statică a memoriei prin precizarea adresei zonei de cod şi a adresei şi dimensiunilor zonei de date proprii a fiecărui proces în parte. In cadrul gestionării se va ţine cont şi de zona de date a executivului.

In etapa următoare, cu ajutorul mediului de programare, se pot efectua măsurători de timp asupra proceselor individuale în vederea stabilirii informaţiilor apriorice necesare modelului de studiu al planificării sistemelor închise de procese. Efectuarea acestei activităţi, precum şi studiul cu ajutorul modelului precizat a arhitecturii aplicaţiei depinde în mod direct de strategia preconizată de utilizator în abordarea politicii de alocare a UC proceselor. De asemenea la acest moment există suficiente informaţii referitoare la structura grafului sistemului, a resurselor proceselor şi a disciplinei de planificare pentru a se aborda dacă se consideră necesar acest lucru, modelul „off-line" de studiu a interblocărilor.

Ultima etapă este destinată activităţii de creare a aplicaţiei, punerea în funcţiune, testare, măsurarea şi adaptare. Această activitate laborioasă şi complexă se realizează de asemenea prin mijlocirea mediului de programare.

După crearea propriu-zisă a arhitecturii aplicaţiei (executiv + procese aplicaţie) se poate trece la simularea unor evenimente sau semnale produse de ambianţă şi la studiul comportării sistemului drept răspuns la aceste evenimente. Prin funcţiile mediului de programare se pot spre exemplu realiza:

- determinarea timpului de răspuns la un anumit eveniment,
- determinarea unor timpi de execuţie a unor procese, a unor ramuri ale structurii, a unor rutine,
- contabilizarea timpilor şi a numărului de treceri prin dife-

rite puncte ale arhitecturii,

- constatarea unor interblocări,
- urmărirea activităţii dispecerului, etc.

Pe baza acestor constatări în cadrul activităţii de adaptare se pot lua o serie de măsuri şi opera o serie de modificări menite a ameliora performanţele sistemului. Dintre acestea se amintesc

- modificarea priorităţii unor procese, care nu asigură timpi de răspuns corespunzători şi chiar a politicii de alocare a UC-ului
- reajustarea unor secvenţe,
- modificarea unor politici, a unor strategii de acţiune,
- reconsiderarea unor sincronizări, etc.

Această activitate are un pronunţat caracter particular, determinat de aplicaţia concretă şi de cerinţele acesteia, şi depinde în mod esenţial de priceperea, experienţa şi intuiţia celui care o efectuează. Se mai precizează că proiectarea şi realizarea proceselor aplicaţiei nu are un caracter liniar ci unul iterativ presupunînd numeroase reveniri în fazele anterioare şi reluarea etapelor implementării. Volumul de muncă necesar a fi investit în această activitate se apreciază în funcţie de valoarea aplicaţiei şi de performanţele preconizate a fi atinse.

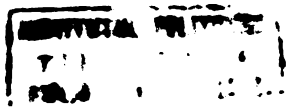
4.6. Extinderea arhitecturilor ierarhice pentru sisteme timp-real multi/uP

In cadrul acestui paragraf, se propune extinderea arhitecturii software preconizate, pentru sisteme multi/uP asimetrice, local distribuite, slab cuplate prin intermediul unei memorii comune (§ 2.2.3).

Faţă de arhitecturile implementate pe sisteme de calcul monoprocesor, caracteristica esenţială a sistemelor multiprocesor este aceea că ele înlocuiesc parţial paralelismul logic al execuţiei proceselor, cu paralelismul fizic, real (§ 2.4.1). Astfel, din punct de vedere conceptual arhitecturile software diferă fundamental de cele ale sistemelor monoprocesor; ele presupun însă completarea acestora cu un nucleu destinat a rezolva aspectele legate de comunicarea interprocesoare.

4.6.1. Arhitectura sistemelor de operare TR destinate unor sisteme multi/uP

Arhitectura care se propune pentru astfel de sisteme se încadrează în categoria sistemelor de operare distribuite prin replicare. Aceasta presupune multiplicarea executivului sistemului şi plasarea sa în „replică" pe fiecare dintre uP-le sistemului.



Astfel, fiecare µP are propriul său sistem de operare local, care este o componentă autonomă a sistemului distribuit. Această componentă are arhitectura unui sistem ierarhic TR şi îndeplineşte funcţia de executiv pentru procesorul pe care lucrează. Această arhitectură trebuie însă completată cu nucleul distribuit al sistemului multiprocesor şi cu componenta distribuită a schimbului de mesaje, componente care rezolvă problemele de interconectare a µP-lor din punct de vedere software. Ele de fapt prelungesc funcţiile componentelor sistemelor autonome, astfel încît să extindă domeniul lor de acţiune asupra sistemului multiprocesor. O astfel de arhitectură principială pentru un sistem cu trei µP apare în figura 4.6.1.

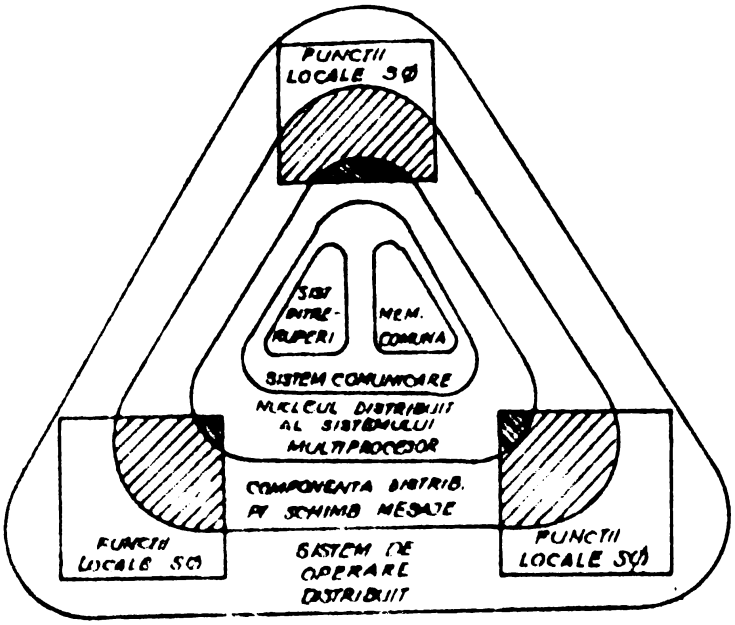


Fig.4.6.1. Arhitectura de principiu pentru un sistemul multi µP.

Integrînd această concepţie în coordonatele arhitecturii preconizate pentru o componentă autonomă, pentru un sistem multi µP rezultă necesitatea completării acesteia la bază cu un nivel destinat interacţiunilor dintre µP. O astfel de arhitectură completă, apare în fig. 4.6.2. [CR82],[CR82],[CG85],[GG84] .

Precizînd că atît pentru structura executivului timp-real, cît şi pentru implementarea sa în cadrul fiecărei componente autonome, rămîn valabile toate cele precizate pînă în prezent în acest capitol, în continuare se trece la analiza componentelor de interconectare.

4.6.2. Nucleul distribuit al sistemului multi µP

A. Funcţii. Nucleul distribuit al sistemului multiprocesor îndeplineşte funcţiile coordonării unor procese aparţinînd unor procesoare diferite. El conţine două subniveluri:

a) Nucleul distribuit inferior care implementează mecanismul de acces la memoria comună.

b) Nucleul distribuit superior care implementează componenta distribuită pentru coordonare.

Mecanismul de acces la memoria comună, rezolvă problemele excluziunii mutuale a acceselor la memorie, din acest punct de vedere el reprezentînd de fapt o extindere a conceptului de monitor monolitic.

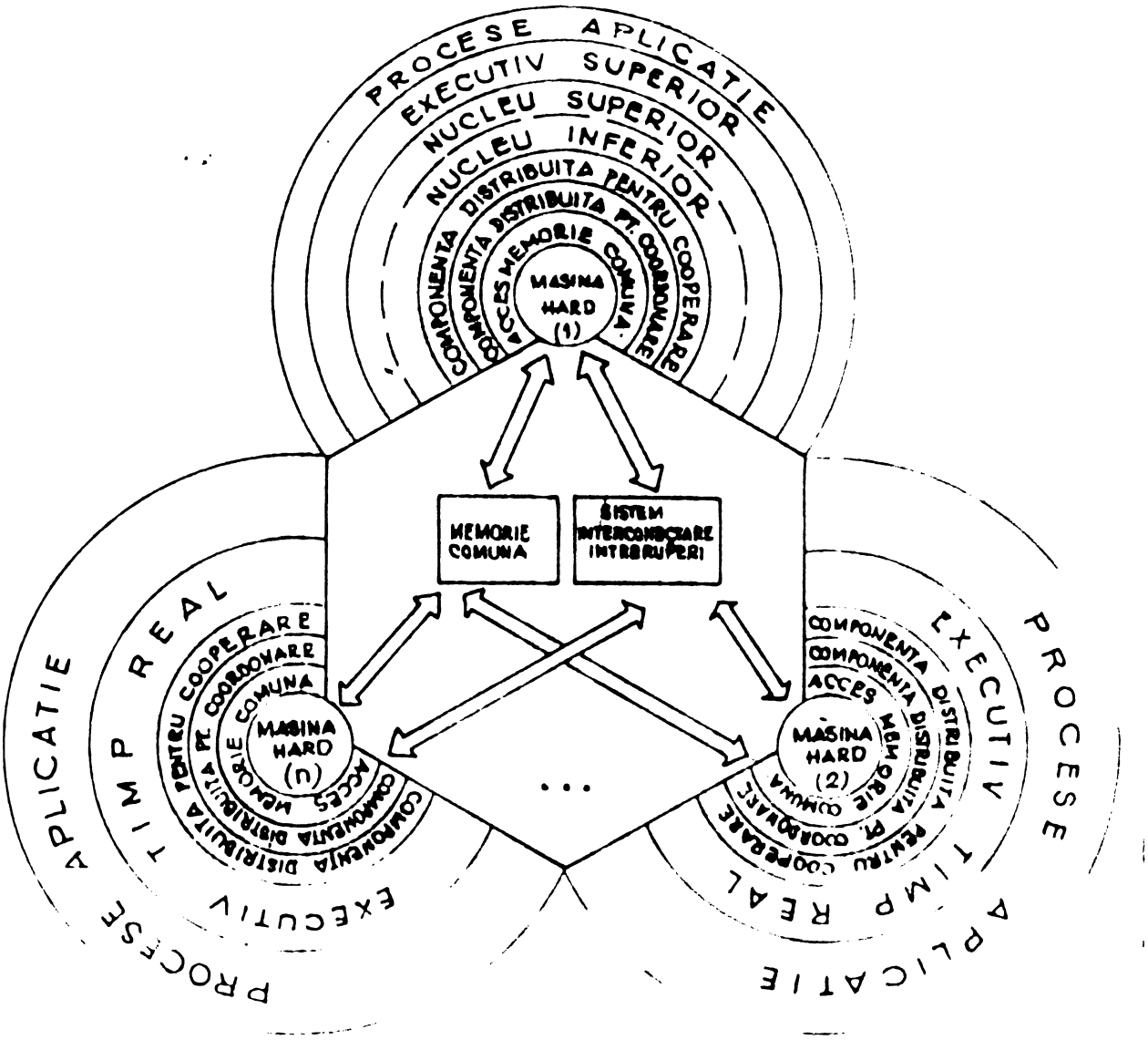


Fig.4.6. Arhitectura software ierarhică pentru un sistem de calcul multi/ul.

Maniera propusă de implementare este cea prezentată în § 3.3.3 sub denumirea de mecanism PHS (M.3.3.2). Mecanismul se poate implementa în forma a două rutine nucleu numite ACCES și ELIB a căror structură de principiu a fost de asemenea prezentată. Aceste funcții sînt apelate de către rutinele executivului, în momentul solicitării respectiv al renunțării la accesul de modificare al conținutului memoriei comune, acces care se realizează în regim special.

Consultarea locațiilor memoriei comune, nu necesită prezent acestui mecanism, arbitrajul cererilor simultane fiind realizat prin circuite hardware.

Rutinele ACCES și ELIB sînt utilizate exclusiv de către nucleul executivului în implementarea semafoarelor speciale și a mecanismului transmiterii mesajelor interprocesoare.

Componenta distribuită pentru coordonare, cuprinde rutinele necesare implementării semafoarelor speciale localizate în memoria comună. Aceste semafoare permit sincronizarea unor procese aparţinînd unor µP diferite. După cum s-a precizat şi în § 3.3.5, aceste semafoare au o serie de particularităţi de implementare, provenite din faptul că nici un µP nu are acces în memoria proprie a unui alt µP.

Astfel, deoarece într-un semafor special care lucrează în regim de blocare se pot bloca procese aparţinînd mai multor procesoare, înlănţuirea lor în lista de aşteptare a semaforului nu se poate face ca la semafoarele normale, utilizînd cîmpul de înlănţuire din tabela procesului. Pentru acest tip de semafoare, lista conţinînd procesele blocate se păstrează în memoria comună.

Funcţiile operatorilor P şi V sînt preluate de către două rutine specifice PS şi VS care aparţin componentei distribuite pentru coordonare.

In ceea ce priveşte funcţiile rutinei PS acestea coincid cu cele ale rutinei P normale, cu excepţia faptului că semaforul se află în memoria comună şi că diferă maniera de înlănţuire. Deoarece o operaţie P are efect numai asupra procesului care o apelează, procesele implicate aparţin în mod obligatoriu µP care execută fizic operaţia, deci sînt procese proprii (interne) ale acestuia. Drept urmare, rutina PS va realiza actualizările de stare în baza de date proprie a executivului µP pe care se execută, iar înlănţuirile la semafor, în memoria comună la care de asemenea are acces.

Operaţia V este mai complexă, deoarece execuţia ei asupra unui semafor special, are drept consecinţă eliberarea unui proces aparţinînd unui procesor diferit de cel care a executat operaţia. Prin urmare actualizarea informaţiei privitoare la starea procesului eliberat nu se poate face de către rutinele executiv ale µP care a executat V-ul, deoarece acestea nu au acces în memoria unui alt µP. Ca atare, este necesar ca operaţia V să fie „pasată" µP căruia îi aparţine procesul eliberat, împreună cu informaţia necesară identificării acestui proces. Pasarea V-ului se face prin intermediul sistemului de întreruperi, special conceput în acest scop, care permite ca executivul unui µP (cel pe care se execută operaţia V) să întrerupă activitatea unui alt µP (cel care conţine procesul eliberat), activizînd în acest fel rutinele specifice pentru actualizarea stării procesului eliberat. Transmiterea informaţiei privind identitatea procesului eliberat, se face printr-o zonă specifică a memoriei comune, prin mecanismul căsuţelor poştale (box-uri) (§ 6.2).

In acest context, rutinei VS îi revin următoarele sarcini:

- eliberarea procesului din lista de aşteptare a semaforului special

- pregătirea identităţii procesului de eliberat în căsuţa poştală şi lansarea unei IT către procesorul căruia îi aparţine procesul eliberat. Modul de realizare al sincronizării dintre µP întrerupă şi µP întrerupt depinde de modul de implementare al sistemului de întreruperi interprocesoare.

Secvenţa de tratare a IT provocate, se execută în nucleul µP vizat şi are drept scop achitarea întreruperii şi actualizarea stării procesului eliberat în tabela de definiţie a acestuia.

Operaţiile P şi V asupra semafoarelor pot fi concepute în mod unitar, astfel încît utilizatorul să nu sesizeze nici o diferenţă între un semafor obişnuit şi unul special. Această sarcină revine executivului care recunoaşte tipul semaforului şi în funcţie de aceasta apelează rutinele P şi V normale din executiv sau PS şi VS din nucleul distribuit.

B. Structurile de date utilizate de nucleul distribuit sînt următoarele:

(1) - structuri de tip semafor special care diferă cu mod de implementare de semafoarele normale.

(2) - zona cutiuţelor poştale (box-uri de întreruperi). O zonă specifică fiecărui µP, utilizată de către celelalte µP ale sistemului în operaţia de „pasare" a V-ului în vederea transmiterii identităţii procesului ce trebuie activat (eliberat).

4.6.3. Componenta distribuită pentru cooperare

A. Funcţii. Această componentă asigură realizarea schimbului de informaţii între procese aparţinînd unor procesoare diferite prin intermediul unor mesaje numite „externe". Mecanismul transmiterii acestor mesaje este o extensie a mecanismului transmiterii mesajelor normale şi a fost precizat conceptual în § 3.4.6. Acest mecanism se poate implementa în forma a două rutine SENDE şi RECVE care reprezintă alternativele funcţiilor SEND şi REC pentru mesajele externe. Selecţia acestor rutine este realizată automat de către sistem, astfel încît din punctul de vedere al programatorului, utilizarea celor două tipuri de mesaje (normale sau externe) este perfect identică.

Din motivele expuse în paragraful anterior cu privire la realizarea accesului la zonele de memorie ale µP, transmiterea unui mesaj extern se realizează în două etape.

a) In prima etapă, la iniţiativa procesului emiţător, funcţiile distribuite ale schimbului de mesaje plasează mesajul în portul specific al µP destinaţie, aflat în memoria comună. Accesul la port se realizează sub protecţia unui semafor specific de acces, aflat de asemenea în memoria comună, alături de semafoarele de sincronizare ataşate portului. In acelaşi timp, rutina SENDE efectuează o operaţie V asupra unui semafor specific µP care conţine procesul destinaţie, aflat de asemenea în memoria comună, numit semafor poştaş.

b) In etapa II-a, un proces sistem special numit distribuitor de mesaje sau poştaş, situat în executivul superior al µP destinaţie, este deblocat din semaforul poştaş specific de către operaţia V executată în etapa anterioară. Sarcina sa constă din a extrage mesajul din portul specific aflat în memoria comună, din a actualiza parametrii portului şi din a transmite printr-un SEND normal mesajul extras procesului destinaţie. Acest lucru este posibil întrucît se acţionează între procese aparţinînd aceluiaşi µP. In final, procesul poştaş se autoblochează în semaforul poştaş specific, aşteptînd un nou semnal de deblocare.

B. Structurile de date utilizate de componenta distribuită a schimbului de mesaje se află în memoria comună a sistemului multi µP şi cuprind:

(1) Zona de mesaje comune (zona porturilor) alcătuită din mai multe porţiuni de memorie (cîte una pentru fiecare µP) în care procesele interne ale µP pot primi mesaje externe.

(2) Zona semafoarelor de protecţie care cuprinde semafoarele de acces la fiecare port. Aceste semafoare asigură accesul în regim de excluziune mutuală la conţinutul portului.

(3) In funcţie de implementare pot exista semafoare care previn depăşirea dimensiunilor maxime alocate porturilor. Desigur, structurile de date utilizate variază în funcţie de metodele de implementare preconizate.

4.6.4. Arhitectura aplicaţiei pentru un sistem multiprocesor TR

In spiritul celor prezentate în acest subcapitol, din punctul de vedere al arhitecturii aplicaţiei, un sistem multi µP poate fi privit ca o sumă de subsisteme distribuite pe µP sistemului, între procesele cărora pot exista interacţiuni.

Sarcina cea mai dificilă care stă în faţa proiectantului constă în repartizarea statică a proceselor pe diferite µP. In această activitate pot interveni mai mulţi factori:

- factori concreţi legaţi de locul şi rolul unui anumit µP în cadrul aplicaţiei,
- gradul de încărcare al µP,
- considerente legate de timpul de răspuns,
- asigurarea unui nivel cît mai redus al interacţiunilor între procese aparţinînd unor µP diferite, etc.

Odată stabilită această prerepartizare se poate aplica pentru fiecare subsistem care aparţine cîte unui µP al sistemului metodologia de proiectare precizată în § 4.5, utilizînd un sistem de dezvoltare corespunzător.

Se precizează că activitatea de implementare propriu-zisă a proceselor nu este în nici un fel influenţată de distribuirea pe µP, deoarece mecanismele de coordonare şi cooperare sînt unitare şi transparente din punctul de vedere al utilizatorului. Astfel, trecerea statică a unui proces de pe un µP pe altul, nu presupune din punctul de vedere al implementării decît modificarea numelui procesului.

4.6.5. Dezvoltarea de aplicaţii TR pentru sisteme multi µP

Dezvoltarea, testarea şi adaptarea aplicaţiilor TR destinate sistemelor multi µP este o activitate complexă care, în funcţie de condiţiile materiale concrete poate fi abordată în mai multe feluri. Se precizează însă faptul că, indiferent de modul de abordare, elementul central al acestei activităţi îl constituie mediul de programare, definit în sensul celor precizate în § 4.4. În funcţie de natura acestui mediu şi de tipul de sistem de dezvoltare folosit se disting două categorii de metode.

A. Dezvoltarea de aplicaţii TR pentru sisteme multi µP utilizînd sisteme de dezvoltare de tip monoprocesor. În cadrul acestei metode se disting mai multe posibilităţi.

M.4.6.1. O primă posibilitate în acest sens o reprezintă dezvoltarea, testarea şi adaptarea pe rînd a subsistemelor caracteristice fiecărui µP în parte, utilizînd în acest scop un sistem de dezvoltare complet, ca cel preconizat în § 4.4.3 spre exemplu. Această metodă nu permite însă verificarea interacţiunilor interprocesoare, respectiv între procesele aparţinînd unor µP diferite. Cu toate aceste limitări, din aplicarea funcţiilor de măsurare pe fiecare subsistem în parte, proiectantul de sisteme, poate percepe ansamblul din punctul de vedere al intercondiţionărilor sub aspect temporal şi poate adopta măsuri potrivite pentru rezolvarea unor situaţii deosebite. Odată dezvoltat, un subsistem este implementat direct pe

/uP specific, prin intermediul memoriilor EPRØM.

M.4.6.2. O a doua posibilitate o reprezintă simularea execuţiei proceselor unui sistem multi /uP pe un sistem de dezvoltare mono /uP.

Acest lucru este posibil din următoarele două motive.

1° Paralelismul logic şi cel fizic sînt identice din punct de vedere conceptual; diferenţa dintre ele rezidînd în maniera de implementare.

2° Din punctul de vedere al testării, interesează în ultimă instanţă constrîngerile care afectează execuţia proceselor. Aceste constrîngeri se materializează în sincronizări şi interconditionări care se manifestă identic atît pe sisteme mono cît şi pe sisteme multi /uP. Ceea ce diferă este doar viteza de execuţie a proceselor.

Cu atare, funcţionarea de ansamblu a unui sistem multiprocesor poate fi testată pe un sistem monoprocesor. Desigur, o astfel de testare nu evidenţiază toate aspectele pe care le ridică sistemul real, în special aspectele legate de timp. Aceste dificultăţi pot fi însă depăşite printr-o gestionare atentă a ceasului sistem şi printr-o interpretare corespunzătoare a timpilor pe care procesele îi petrec în sistem. In funcţie de condiţiile concrete, în vederea obţinerii de rezultate superioare se pot combina cele două metode care practic se completează reciproc.

B. Dezvoltarea de aplicaţii TR pe sisteme multi /uP similare sau asemănătoare sistemului fizic real.

M.4.6.3. O primă posibilitate în acest sens o reprezintă implementarea cîte unui mediu de programare pe fiecare /uP al sistemului (MP replicat), manieră de lucru care permite dezvoltarea, testarea şi adaptarea în cele mai bune condiţii a aplicaţiei TR întrucît procesele interacţionează în condiţii reale. Această metodă, de regulă, nu se justifică din punct de vedere economic, decît în condiţii cu totul speciale [CG85].

M.4.6.4. O a doua posibilitate o reprezintă dotarea unui singur /uP al sistemului multi cu facilităţi de dezvoltare şi completarea „mediului de programare" implementat pe acest sistem cu funcţii de transbordare a proceselor de pe un sistem pe altul şi de apel a unor funcţii de tip executiv ale celorlalte /uP. Acest lucru se poate realiza spre exemplu prin intermediul unui proces sistem cu cu rol de interpreter de comenzi, activizat prin intermediul unor mesaje externe specifice. In această manieră se permite testarea la un moment dat a unui singur subsistem al sistemului TR (cel aflat pe sistemul de dezvoltare) testare care se execută însă în condiţii reale. Prin permutarea pe rînd a subsistemelor pe sistemul de dezvoltare se poate pune la punct în mai multe etape întreaga aplicaţie.

5. Implementarea unor arhitecturi de sisteme de operare TR pe sisteme de calcul mono µP din familia I8080/85

Concluziile teoretice referitoare la concepţia şi proiectarea unor arhitecturi de SØTR pentru sisteme mono µP, prezentate pînă acum în lucrare, au fost aplicate şi verificate în practică prin implementarea unor astfel de arhitecturi TR şi a MP aferente, pe mai multe sisteme de calcul aflate în dotarea Catedrei de Calculatoare a Facultăţii de Electrotehnică sau a Centrului de calcul al institutului. Scopul capitolului de faţă este de a prezenta succint aceste realizări şi de a sublinia unele detalii specifice de implementare [Cr80],[Cr81a],[Cr84a],[CC84] .

5.1. Descrierea generală a sistemelor de calcul utilizate

In vederea implementării s-au utilizat trei sisteme de calcul a căror descriere succintă este furnizată în continuare.

Primul microsistem utilizat, are la bază un µP 8085 şi a fost realizat în cadrul Catedrei de Electronică şi Măsuri. El dispune de 5 Ko memorie EPRØM şi de 8 Ko memorie RAM. Sistemul conţine în structura sa periferice standard : display (DAF 1001), lector de bandă perforată (LB 50), perforator de bandă (P50) şi periferice specializate în vederea prelucrării semnalelor analogice (fig.5.1.1).

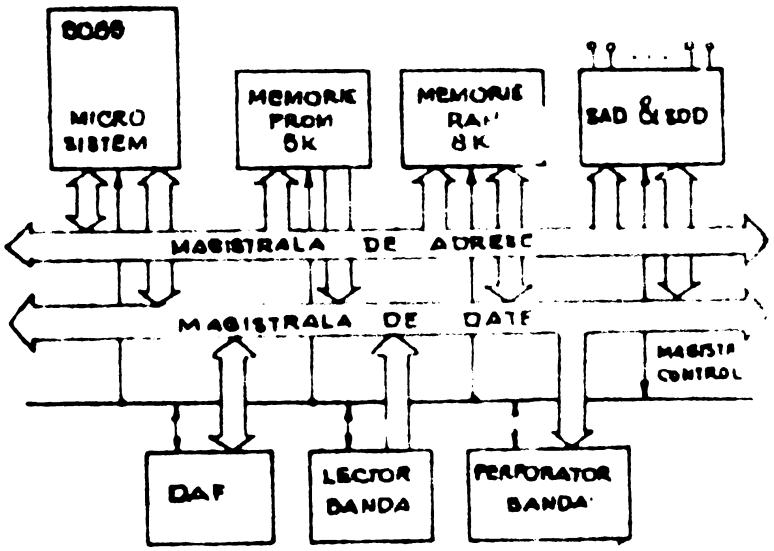


Fig.5.1. Sistem de calcul bazat pe µP 8085.

Un al doilea sistem mai complex, este bazat pe un µP 8080, avînd 64 Ko de memorie (32 Ko RAM nevolatil cu miezuri de ferită şi 32 Ko RAM MØS dinamic). Sistemul de întreruperi are 15 niveluri şi este realizat cu integratele specializate I8259 şi 8214 [GF83] . Periferia standard este formată dintr-un display (DAF 1001 P), un lector de bandă perforată (LB 50), un perforator de bandă (DT 105), o maşină de scris RØBØTRØN (SM 4000) şi o imprimantă.

Ca memorie externă de masă se utilizează un casetofon obişnuit. Sistemul dispune de un orologiu de timp programabil bazat pe

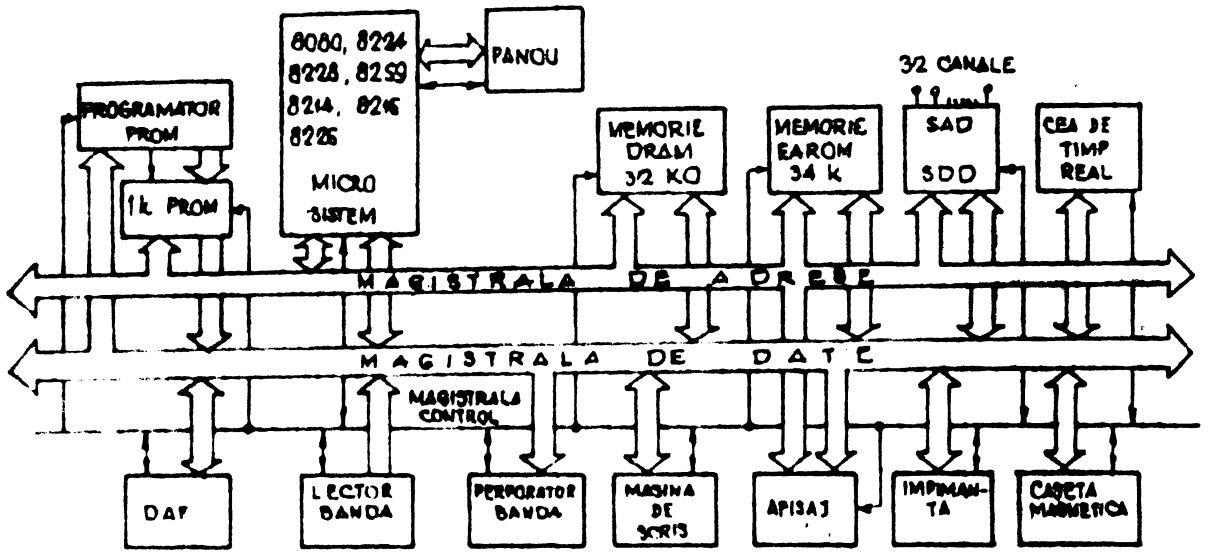


Fig.5.1.2. Sistem de calcul bazat pe μP 8080.

integratul I8253. In vederea utilizării microsistemului ca şi calculator de proces, acesta a fost echipat cu un sistem pentru achiziţia şi distribuţia datelor cu o rezoluţie de 12 biţi şi o rată de transfer de 50 kHz, avînd 24 de borne de intrare/ieşire (fig.5.1.2).

Cel de al treilea sistem utilizat a fost sistemul ECARØM 800 aflat temporar în dotarea C.C.E, sistem destinat comenzii şi supravegherii proceselor industriale. Dintre facilităţile timp-real pe care le implementează amintim ceasul de timp-real programabil care asigură două baze de timp T1 şi T2, furnizînd trenuri de impulsuri dreptunghiulare cu perioadele variind între 100-1000s respectiv 10-100s. Atît T1 cît şi T2 pot fi divizate în frecvenţă cu 1, 10, 100 sau 1000 selecţia factorului de divizare făcîndu-se prin program (fig.5.1.3.(a)).

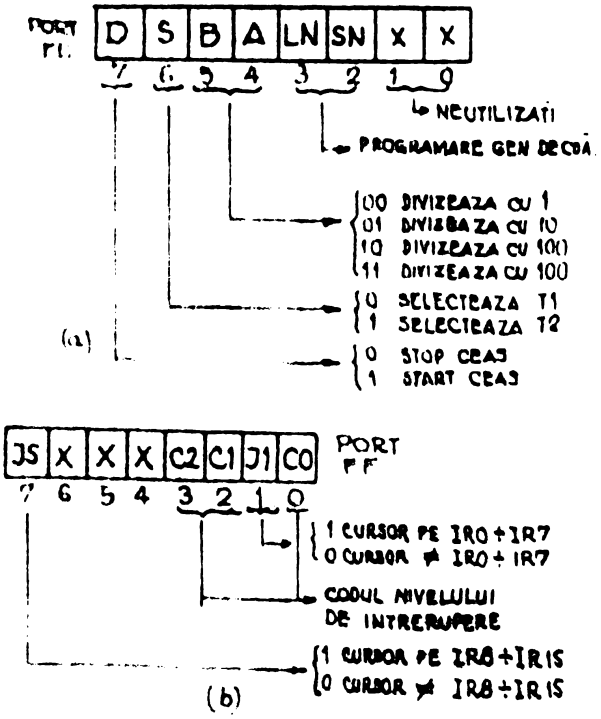


Fig.5.1.3. Programarea ceasului timp-real şi a sistemului de întreruperi la sistemul ECARØM 800.

De asemenea se remarcă şi sistemul de IT care asigură tratarea a 16 cereri. Ierarhizarea cererilor de IT este asigurată de 2 circuite I8214 şi de utilizarea unui "cursor" programabil transmis prin portul de ieşire OFFH. Structura cursorului apare în figura 5.1.3.(b). In acelaşi context apare şi modulul NGI (numărător generator de impulsuri) care este pre-

văzut cu 5 numărătoare programabile I8253 şi care permite atît contorizarea evenimentelor externe declanşate asincron cît şi declanşarea unor activităţi specifice la intervale precizate de timp, prin conectarea sa la sistemul IT. Pentru cele trei sisteme prezentate, au fost realizate trei executive timp-real cu mediile de programare aferente respectiv SFIT85, SISTV şi SIECAR. In continuarea acestui capitol unele detalii de implementare pentru aceste sisteme de operare.

5.2. Detalii de implementare a nucleului inferior al unui SØTR

In proiectarea şi implementarea nucleului inferior al unui SØTR punctul de pornire îl constituie stabilirea configuraţiilor structurilor de date aferente. Astfel, pentru sistemele în discuţie pentru structurile de date de bază precizate în § 4.3.2 (tabelă proces, tranzacţie procesor şi semafor) au fost stabilite configuraţiile din figurile 5.2.1 şi 5.2.2. Sistemele pot gestiona un număr maxim de 256 de procese care se identifică prin număr. Fiecare proces al sistemului dispune de o tabelă de definiţie, dispusă fizic în zona TDBCP în locul corespunzător numărului procesului. Sistemul păstrea

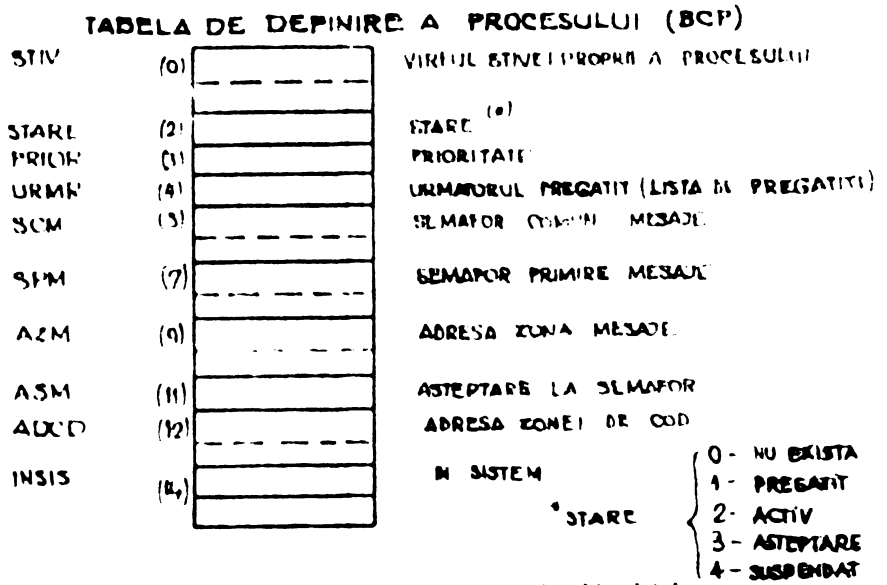


Fig.5.2.1. Structura tabelei de definiţie a procesului.

ză evidenţa procesului actual (NRPA), adresa tabelei acestuia(PRCA) şi numărul real de procese din sistem (NRPRC). Se precizează că zona de memorie a fiecărui proces, începe cu o tabelă proprie (TABPR) care conţine adresa stivei, prioritatea şi zona sa de mesaje, codul începînd de la adresa TABPR+6 (fig.5.2.6). Această tabelă are o semnificaţie deosebită în activitatea de iniţializare a sistemului. De regulă, codul procesului începe cu o secvenţă de iniţializare a zonelor de date proprii urmată de secvenţa propriu-zisă, de natură

ciclică, a codului procesului.

Implementarea structurii de semafor nu ridică probleme deosebite, ea fiind multiplicată în cîte un exemplar pentru fiecare din semafoarele necesare sistemului.

Odată stabilită configuraţia structurilor de date de bază, este necesar să se stabilească stările specifice prin care pot să treacă procesele, tranzacţiile acestora, precum şi influenţa pe care funcţiile implementate o au asupra diagramei de tranziţie a stărilor. Această diagramă se completează pe măsură ce se introduc noi funcţii. Forma ei finală pentru sistemele realizate apare în fig.5.2.3.

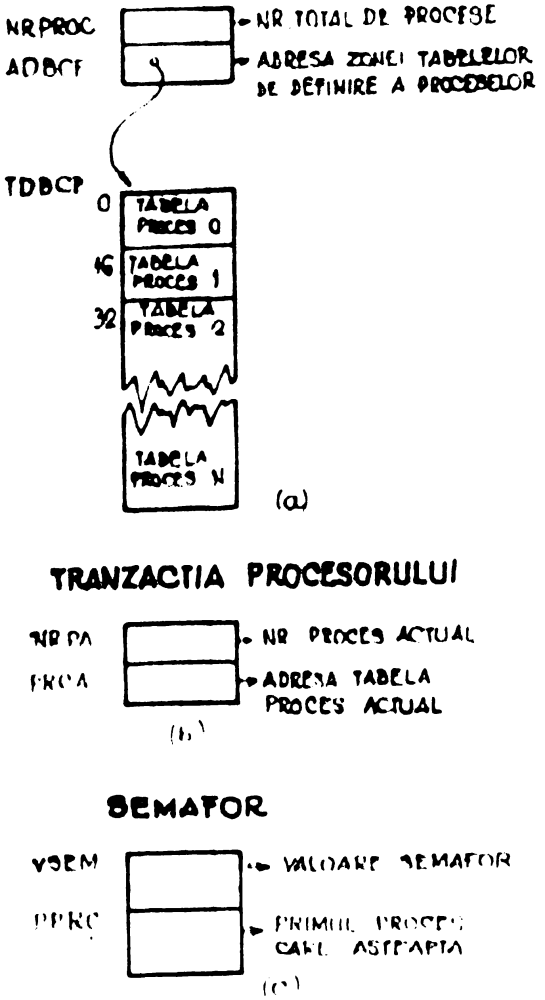


Fig.5.2.2. Tranzacţia procesorului şi semaforul.

Revenind la implementarea funcţiilor nucleului inferior (§ 4.3.2) se fac următoarele precizări. Mecanismul de comutare a fost implementat prin intermediul a două rutine specifice denumite SLAV şi RETUR care se apelează ori de cîte ori se intră, respectiv se iese din sistem. Funcţiile acestor rutine au fost descrise în § 4.3.2. Se precizează în plus că ele acţionează asupra stivei proprii a procesului aflat în execuţie în UC. In vederea rezolvării unor apeluri sistem încuibate, a fost conceput un mecanism de înlănţuire a adreselor corespunzătoare vîrfului stivei de lucru pentru diferitele apeluri, astfel încît în condiţii excepţionale de părăsire fortuită a unui apel, returul are loc în mod corect, chiar dacă stiva nu a fost complet eliberată. In tabela de definiţie a procesului se păstrează adresa vîrfului stivei proprii, iar în vîrful stivei se păstrează adresa fostului vîrf (fig.5.2.4).

In ceea ce priveşte gestionarea timpului UC, între cele trei sisteme realizate apar diferenţe. Sistemul SFIT85 este un sistem fără ceas de timp în care divizarea timpului UC se realizează la iniţiativa proceselor, fie din considerente de sincronizare (blocare într-un semafor) fie din iniţiativa programatorului care a prevăzut în codul procesului un apel sistem special CMT (comută). Implementarea acestei funcţii este simplă, sarcina ei fiind aceea de a salva starea procesului în cur-

execuţie în stiva sa proprie şi de a da comanda dispecerului spre a selecta pentru execuţie un alt proces pregătit.

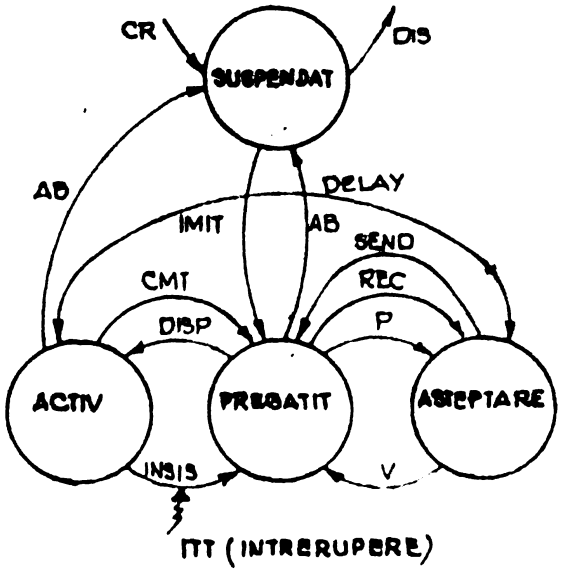


Fig.5.2.3. Diagrama tranziţiei stărilor proceselor.

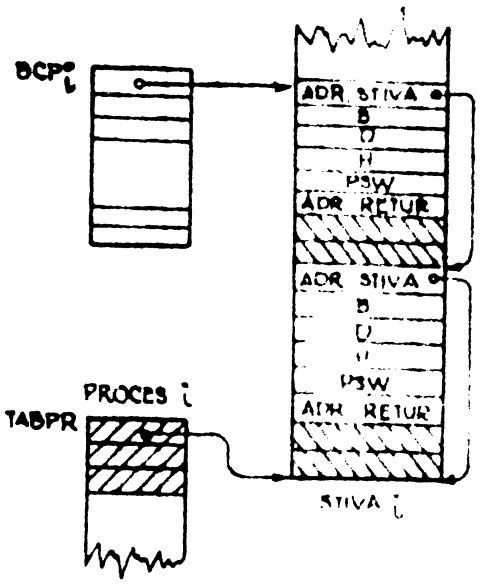


Fig.5.2.4. Gestiunea stivei pentru apeluri încuibate.

La sistemele SISTV şi SIECAR care dispun de ceasuri de timp intern s-a utilizat metoda cuantizării timpului UC, prin distribuirea acestuia între procesele pregătite pentru execuţie la un moment dat. Comutarea UC de la un proces la altul are loc fie din condiţii de sincronizare, fie datorită epuizării cuantei de timp UC alocate. Aceasta se realizează prin faptul că atît în cazul sincronizării, cît şi în cazul cuantizării, funcţia P care execută blocarea procesului curent, respectiv secvenţa de tratare a IT de timp dau în final comanda dispecerului.

În ceea ce priveşte politica de planificare în execuţie a proceselor, fiecare dintre dispecerele implementate conţine cîte două intrări: una pentru selectarea procesului următor într-o manieră circulară (disciplina carusel simplu), cealaltă pentru priorităţi fixe (baleierea listei de la început procesele fiind aşezate în listă în ordinea descrescătoare a priorităţilor). Concepţia modulară a sistemului permite însă implementarea facilă a oricărei politici de planificare în execuţie a proceselor modificînd doar modulul dispec

Mecanismul de sincronizare bazat pe semafoare este implementat în forma a două rutine (P şi V) care se execută în regim de monitor monolitic. Apelul acestor funcţii se realizează prin instrucţiuni simple CALL, după ce în prealabil a fost încărcată adresa semaforului în registrele B şi C. Rutinele prelucrează o structură de date de tip semafor în care se pot bloca oricîte procese; în semafor

apare numărul primului proces blocat (cîmpul PPRC - fig.5.2.2.c), iar în continuare procesele blocate se înlănţuie prin intermediul cîmpului ASM al tabelei procesului (fig.5.2.1). Fiecare cîmp conţine numărul procesului următor blocat în şir, cîmpul corespunzător ultimului proces blocat conţinînd OFFH. Mecanismul implementat este identic în toate cele trei sisteme.

Trecerea proceselor dintr-o stare în alta, reflectată în modificarea cîmpului STARE din tabela procesului, se realizează prin module specifice implantate în rutinele care iniţiază această activitate (P,V,DISP,etc.). Ele sînt realizate în acord cu conceptul de obiect.

Problema tratării IT interne ale nucleului a fost abordată în mod diferenţiat în cele trei sisteme datorită pe de o parte obiectivelor distincte de proiectare iar pe de altă parte posibilităţilor hardware specifice ale sistemelor. Cu toate acestea există o serie de elemente comune. Spre exemplu, fiind vorba în general despre sisteme dedicate, numite în ultima perioadă „la cheie", o importanţă deosebită revine tratării IT interne de iniţializare, realizată în concret prin nivelul corespunzător lui RST 0. Rutina INS implementată realizează următoarele:

- iniţializează zona temporară de stivă a sistemului,
- iniţializează zona tabelelor proceselor, după care apelează funcţia de iniţializare (INIT) pentru fiecare din procesele care figurează în tabela de procese a sistemului TSIST.

Activitatea de iniţializare a proceselor este o activitate relativ complexă care presupune pentru fiecare proces, completarea tabelei, a stivei acestuia şi a zonei de mesaje pornind de la tabela TABPR (fig.5.2.5). Procesele se trec în starea pregătit.

- în funcţie de configuraţia concretă a sistemului, rutina INS iniţializează unele circuite specifice (ceasurile I8253, circuitele de IT 8259 - în cazul lui SISTV, sau baza de timp, cursorul întreruperilor şi ceasurile de timp în cazul lui SIECAR),
- în final INS dă comanda dispecerului spre a lansa sistemul în execuţie.

Toate iniţializările au loc în zona RAM a memoriei, avînd drept punct de plecare informaţiile memorate în tabelele TSIST şi $TABPR_i$ memorate în zona EPROM (fig.5.2.6).

Alte întreruperi tratate la nivelul nucleului inferior sînt cele de timp şi cele de dispozitive periferice. Dintre acestea tratarea IT de timp este mai laborioasă (§ 5.3), secvenţa de tratare a IT de periferic reducîndu-se la execuţia unei operaţii V asupra semaforu-

lui specific de sincronizare.

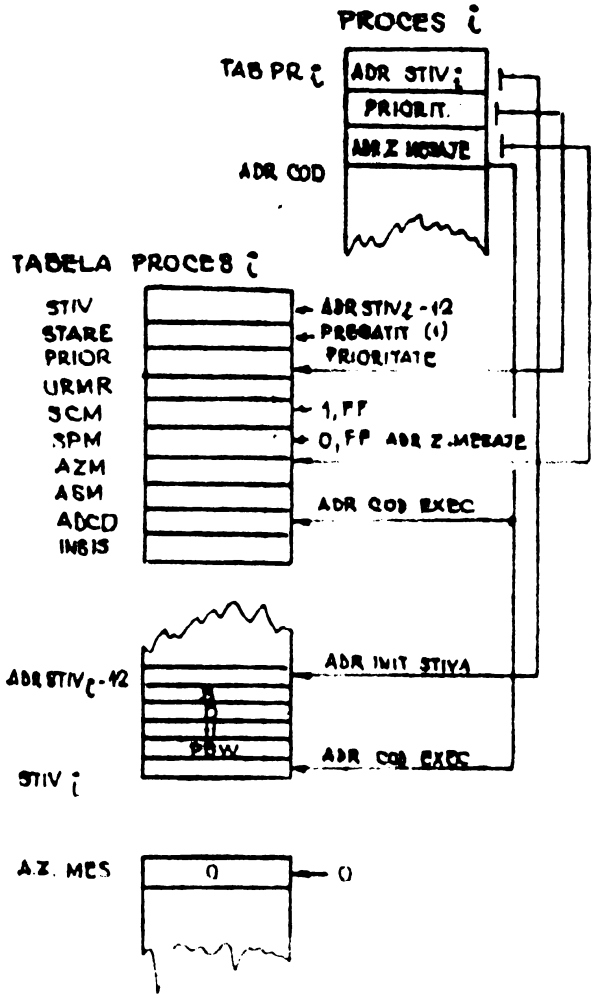


Fig.5.2.5. Iniţializarea structurii de date corespunzătoare unui proces.

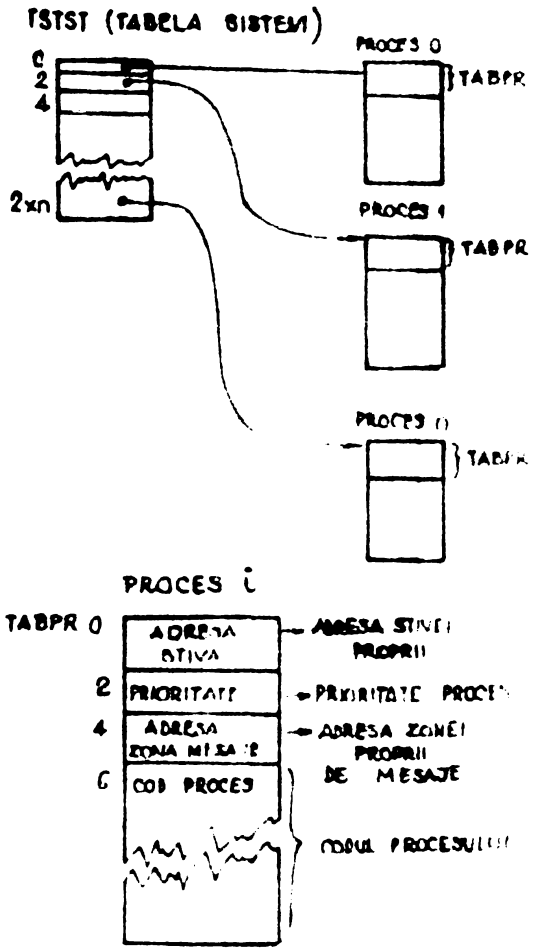


Fig.5.2.6. Tabela sistem TSTST şi tabelele TABPR corespunzătoare proceselor

O menţiune specială trebuie făcută în legătură cu implementarea „monitorului monolitic".

Regula generală adoptată este următoarea: la intrarea în sistem (funcţia SALV) se dezactivează IT iar la ieşirea din sistem (funcţia RETUR) ele se reactivează. Apar însă două probleme:

1° - Există funcţii care nu se execută decît parţial în regim de monitor monolitic (SEND, REC).

2° - Funcţiile sistem se pot apela unele pe altele (apeluri încuibate).

Rezolvarea acestor situaţii a necesitat completarea tabelei procesului cu un octet denumit INSIS (fig.5.2.1) asupra căruia acţionează în acord cu conceptul de obiect, rutinele INSI (punere) şi SUTSI (repunere) prin intermediul căruia se poate indica sistemului păstrarea sau părăsirea regimului de monitor monolitic la execuţia funcţiei RETUR. Au fost adoptate următoarele norme interne:

a) SALV şi RETUR încep prin a dezactiva IT (DI).

b) funcţiile care nu se execută în regim de monitor monolitic după SALV reactivează IT (EI).

c) funcţia RETUR verifică cîmpul INSIS şi autorizează întreruperile doar dacă el nu este poziţionat.

d) poziţionarea lui INSIS se face prin funcţiile INSI şi ØUTSI precizate, care se utilizează pentru a forţa regimul de monitor monolitic unor secvenţe care altfel nu s-ar putea executa într-un astfel de regim datorită unor apeluri sistem încuibate (spre exemplu TRITT,DELAY,etc.).

Pe lîngă cele precizate, nucleul inferior implementează o serie de rutine auxiliare necesare căutării în tabele, determinării de adrese (ABCP, ASCM), mutării şirurilor (MUTAD, MUTAI), etc.

5.3. Detalii de implementare a nucleului superior al unui SØTR

In proiectarea şi implementarea nucleului superior al executivului se porneşte tot de la precizarea structurilor de date. Pentru sistemele prezentate în cadrul lucrării au fost stabilite următoarele structuri: mesaj, zona mesaje, ceas sistem şi tabelă ceas (fig.5.3.1).

Mecanismul de cooperare şi comunicare interprocese bazat pe mesaje, a fost implementat în forma a două rutine sistem intreruptibile (SEND şi REC). Tehnica utilizată a fost aceea a porturilor materializate în zone de mesaje caracteristice proceselor, fiecărei astfel de zone asociindu-i-se un semafor pentru sincronizarea primirii mesajelor (SPM-fig.5.2.1) şi un semafor pentru protecţia zonei (SCM-fig.5.2.1).

A fost implementată funcţia RECEIVE cu blocare, iar depăşirea zonei alocate mesajelor se semnalizează în vederea adoptării de măsuri. De asemenea au fost implementate şi mesajele prioritare precizate prin valoarea 1 a cîmpului TIPM (fig.5.3.1(a)). Apelul funcţiei SEND respectiv REC se face după ce în prealabil în registrele B şi C s-a încărcat adresa mesajului. Selecţia procesului destinatar se face cu ajutorul cîmpului PRDST din structura mesajului (fig.5.3.1(a)), poziţionat de către procesul emiţător. Dacă se doreşte transmiterea unui mesaj la care se solicită răspuns, în cazul implementării de faţă, prin convenţie, textul mesajului va conţine pe prima poziţie caracterul ``R'', iar pe poziţia următoare identitatea procesului solicitant.

In ceea ce priveşte păstrarea evidenţei timpului real au fost rezolvate probleme legate de implementarea unui ceas de timp intern care guvernează activitatea sistemului precum şi a unei funcţii care permite utilizatorului să cunoască timpul curent. Ceasul sistem este un contor pe 3 octeţi, care pentru o cuantă de timp de 1 ms poate reprezenta intervale pînă la 4h şi 37'. Conform conceptului de obiect, asupra

ceasului acționează rutine specifice de incrementare (TCLK), de test de nul (TZR) sau complementare (CMP3). Gestionarea ceasului este strîns legată de tratarea IT de timp provenită de la orologiul hardware al sistemului (I8253 la SISTV sau ceasul de timp la SIECAR programat a da IT la intervale de timp dependente de aplicație).

Cunoașterea timpului curent este realizată de funcția întreruptibilă MARK, căreia în momentul apelului, în registrele B și C i se furnizează adresa zonei în care se va memora valoarea ceasului sistem.

Mecanismul de lansare întîrziată în execuție, cu o întîrziere prescrisă, este cel prezentat în § 4.3.3, și a fost implementat în forma unei funcții numite DELAY. In acest scop în tabela TCEAS (fig.5.3.1(d)), s-au prevăzut intrări pentru toate procesele care se pot întîrzia simultan. Rolul funcției DELAY este de a completa o intrare în tabela TCEAS, pentru procesul întîrziat și de a realiza o funcție P pe semaforul specific. Ca o particularitate a implementării, un proces se poate întîrzia doar pe sine însuși.

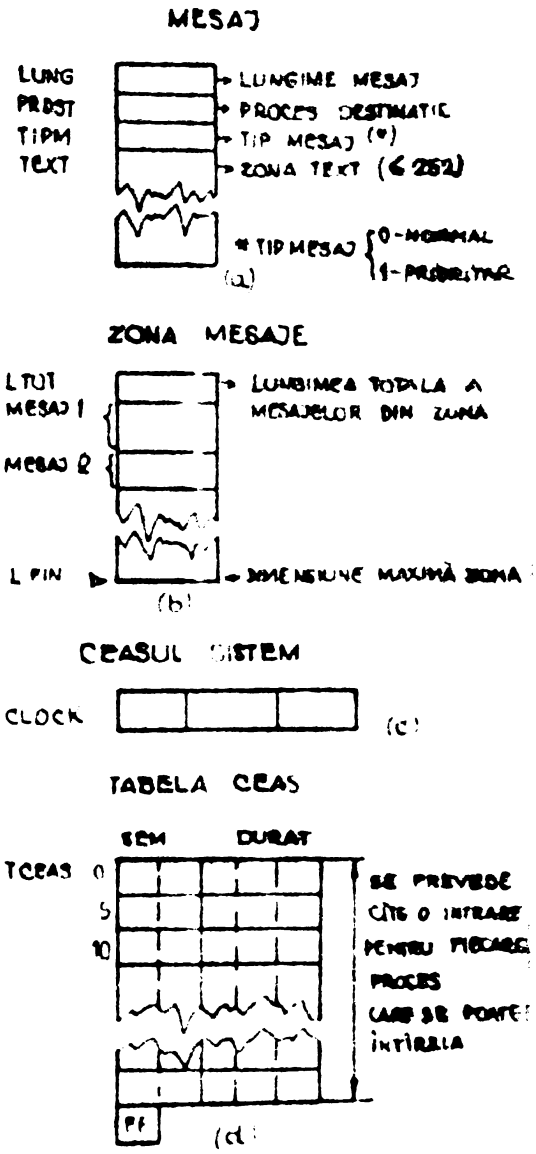


Fig.5.3.1. Structuri de date utilizate în nucleul superior al unui SDNR.

In cadrul acestor sisteme, un rol central revine rutinei TRITT care tratează întreruperea de timp. Atribuțiile ei esențiale sînt: actualizarea ceasului sistem și a contoarelor pentru duratele de întîrziere ale proceselor, respectiv de divizare în cuante a timpului UC, cuante care în funcție de aplicație pot să difere radical de cuanta ceasului sistem. La epuizarea duratei de întîrziere, se baleiază tabela TCEAS, se decrementează contoarele pentru toate intrările active ale tabelei și se deblochează procesele a căror durată de întîrziere a fost realizată, prin funcția V (§ 4.3.3).

Părăsirea rutinei TRITT are loc în două moduri: dacă s-a consumat cuanta de devizare a timpului UC, se apelează modulul DISPECER, în caz contrar se revine în procesul întrerupt. In această formă rutina TRITT a fost implementată în SISTV; în sistemul SIECAR apare o formă simplificată a sa în care cele trei cuante de ceas, de întîrziere şi de divizare coincid ca valoare.

In ceea ce priveşte alocarea resurselor, în sistemele implementate nu au apărut aspecte care să necesite o tratare specială astfel încît nu s-a pus problema implementării unui mecanism de alocare a resurselor fixe. Din acest motiv, nu au fost abordate în mod distinct nici problemele interblocării proceselor. Problema resurselor consumabile a fost rezolvată implicit prin implementarea mecanismului de cooperare şi comunicare prin mesaje (§ 4.3.3), în contextul unei comunicări ierarhice.

5.4. Detalii de implementare a executivului superior

Din aspectele referitoare la executivul superior, prezentate în § 4.3.5, a rezultat că pentru simplificarea şi sistematizarea activităţii de proiectare şi implementare, funcţiile acestuia se pot realiza în forma unor procese sistem activate prin mecanismul întreruperilor externe, semafoarelor sau mesajelor. In general aceste procese sînt strîns legate de aplicaţia pe care o deservesc şi au sarcini în domeniul tratării unor IT externe, dispozitive periferice, erori, redirectări, reconfigurări, etc. In acest subparagraf se vor face însă referiri succinte la o singură categorie de procese, cu un caracter mai general şi anume acelea care se referă la tratarea operaţiilor I/E.

In cadrul sistemelor realizate, fiecărui dispozitiv periferic i se asociază un proces specific sincronizat prin mesaje. Complexitatea unui astfel de proces numit şi driver depinde de natura periferiocului tratat precum şi de maniera de desfăşurare din punctul de vedere al echipamentului, a operaţiei I/E (transfer programat, aşteptare activă, întreruperi, acces direct la memorie, etc.). Mecanismul general de funcţionare al acestor drivere a fost precizat în § 4.3.5. Din punctul de vedere al proiectantului software există în general două categorii de procese: procese care citesc informaţii de la un DP şi procese care scriu informaţii la un DP.

Pentru procesele care citesc informaţii driverul lucrează în trei etape: recepţia solicitării din partea procesului apelant (SEND transmis de solicitant), citirea propriu-zisă a datelor, şi transmiterea datelor solicitantului (SEND către acesta). Pentru a realiza acest lucru în implementare s-a utilizat tehnica mesajului cu răspuns (§ 5.3). Procese

clasice care realizează citiri sînt driverele pentru cititorul de bandă perforată, intrare DAF, intrare casetă magnetică, etc. Driverele sînt astfel implementate încît terminarea unei operaţii de citire are loc fie la epuizarea unui număr precizat de caractere, fie la depistarea unui caracter specific (CR).

Pentru procesele care scriu, evenimentele se derulează în două etape: recepţionarea solicitării de scriere de către driver şi scrierea propriu-zisă a textului. In unele cazuri, dacă solicitantul doreşte poate cere un mesaj de răspuns care să ateste terminarea operaţiei de scriere solicitate. Acest lucru se realizează tot prin tehnica mesajului cu răspuns. Exemple de procese care scriu sînt: driverul perforatorului de bandă, driverul imprimantei, al casetei magnetice, scriere DAF, etc.

5.5. Implementarea unor „medii de programare"

Consideraţiile generale care stau la baza abordării arhitecturii „mediului de programare" (MP), pentru dezvoltarea de aplicaţii TR au fost prezentate în § 4.4. In cadrul paragrafului de faţă, se vor face cîteva referiri concrete la maniera de implementare a unor astfel de medii. Varianta adoptată a fost cea corespunzătoare unui MP implementat pe un sistem de dezvoltare ca şi o componentă de sine stătătoare, slab cuplată cu restul arhitecturii.

Conform celor precizate în § 4.4.3, pentru cele 3 sisteme amintite „mediul de programare" a fost implementat în forma unui program interactiv, realizat în jurul unui interpretator de comenzi. Au fost dezvoltate următoarele comenzi:

- CREA care creează un proces cu un număr precizat, a cărui tabelă TABPR se află la o adresă specificată. In acest scop pentru procesul nou creat se completează tabela de definiţie a procesului, stiva şi zona mesaje conform figurii 5.2.5, apelînd secvenţele de program corespunzătoare ale funcţiei de iniţializare a sistemului. Procesele create se trec în starea suspendat.

- DIST-distruge procesul specificat, radiindu-l din tabelele sistemului.

- MONY - apelează monitorul sistemului. In legătură cu această funcţie trebuie făcută o menţiune specială: mediile de programare implementate lucrează în strînsă legătură cu o serie de programe care permit dezvoltarea efectivă a proceselor. Pentru sistemele realizate şi prezentate în lucrarea de faţă, aceste programe au fost grupate într-un subsistem care conţine un monitor, un asamblor şi un editor de fişiere, autorul avînd o bună experienţă în realizarea şi implementarea unor astfel de sisteme. Subsistemul de dezvoltare

este tot un program interactiv, în care se ajunge apelînd comanda MONY a MP, funcţiile pe care el le realizează fiind numai enumerate, implementarea lor neconstituind obiectul lucrării de faţă. Este vorba despre funcţia de monitor (vizualizarea conţinutului unor zone de memorie, introducerea unor valori în locaţiile de memorie, vizualizarea şi poziţionarea conţinutului registrelor, mutarea unor zone de memorie, lansarea în execuţie a codului obiect, implantarea de puncte de întrerupere), despre funcţia de asamblare (asamblor în două treceri, cu posibilitatea de relocare a codului) şi despre funcţia de editor de fişiere (creare, distrugere fişier în memorie, vizualizare director, adăugare/ştergere de linii din fişierul curent, listare linii sursă).

- INIT - iniţializează un proces precizat, creat anterior, prin trecerea sa în starea „pregătit". Sînt iniţializate toate structurile de date aferente procesului.

- LIST - se listează starea tuturor proceselor aplicaţiei.

- INIS - start aplicaţie. Se iniţializează pe rînd toate procesele create, trecîndu-se în starea pregătit. Se dă comanda dispecerului care le va lansa în execuţie, demarînd execuţia aplicaţiei.

- REXC - reluarea execuţiei după o întrerupere cu refacerea stării program, fără iniţializare.

- AZSI - preiniţializare sistem. Se iniţializează tabelele de definiţie ale proceselor, tabela TSIST şi TCEAS.

- CØNT - contorizează trecerile printr-un punct de program precizat prin adresa sa (punct de numărare).

- CØNS - şterge un punct de numărare situat la o adresă precizată. Pentru implementarea perechii de comenzi CØNT şi CØNS, MP dispune de o structură de date specifică - tabela TNUM (fig.5.5.1.(a)) în care se contorizează numărul de treceri (CØNTØR) printr-o anumită adresă (ADR CØD). Dimensiunea tabelei se alege funcţie de numărul de puncte de contorizare care se doresc a funcţiona simultan (în cazul implementării - 8).

Maniera de tratare a punctului de numărare este bazată pe instrucţia RST 3 care se înserează la detectarea comenzii CØNT, într-un punct de măsurare prevăzut special de către utilizator printr-un NØP.

Simultan se completează şi o linie în tabela TNUM. Comanda CØNS şterge punctul de măsurare introducînd un NØP şi radiază linia corespunzătoare din TNUM. Tratarea lui RST 3 presupune determinarea adresei punctului de măsurare, căutarea acestei adrese în TNUM, incrementarea contorului aflat în tabelă şi revenirea în programul întrerupt.

Inafara acestor funcţii cu caracter general, pe sistemele care dispun de facilităţi de timp (SISTV şi SIECAR) au fost implementate funcţiile de măsurare TIMP şi TIMN.

- TIMP pune la dispoziţie timpul curent la trecerea printr-un punct precizat din program. Necesită doi parametri: adresa punctului de măsurare şi adresa la care se depune timpul determinat.

- TIMN este o comandă mai complexă care permite ridicarea unor histograme de timp. Comanda permite înregistrarea unui număr precizat de timpi succesivi corespunzători trecerii printr-un punct de măsurare indicat. In acest scop, TIMN completează o comandă TIMP dată anterior cu numărul dorit de măsurători. Pentru a realiza aceste obiective, funcţiile TIMP şi TIMN folosesc o structură de date specifică, tabela TTIME (fig.5.5.1.(b)), în care pentru fiecare comandă TIMP se completează adresa punctului de măsurare (ADR CØD) şi adresa locaţiei (tabelei) în care se depun valorile determinate (ADR TIMPI). Dacă se dă în plus şi o comandă TIMN, pentru punctul de măsurare deja înregistrat printr-o comandă TIMP se mai completează în TTIME şi contorul (CØNT).

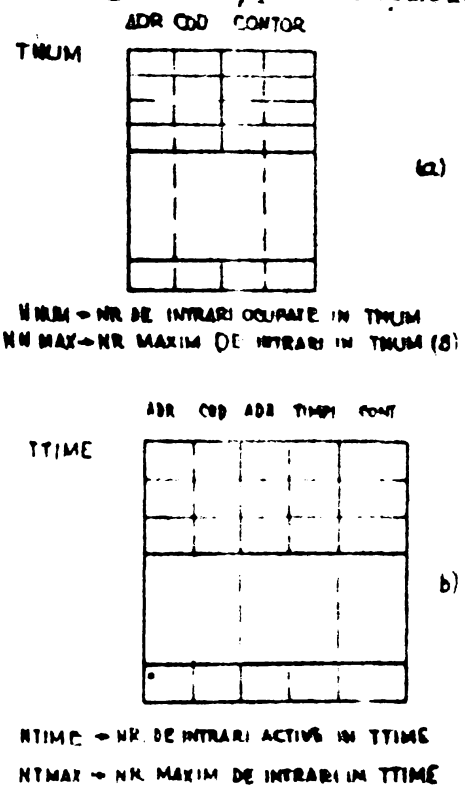


Fig.5.5.1. Structura tabelelor TNUM şi TTIME.

Maniera concretă de implementare aleasă, similară, comenzii CØNT, se bazează pe înserarea unei instrucţii RST 4 în punctul de măsurare prevăzut iniţial cu NØP de către utilizator. În acest context, tratarea lui RST 4 presupune: determinarea adresei punctului de măsurare, căutarea intrării corespunzătoare acestuia în TTIME, apelul funcţiei MARK şi depunerea timpului determinat la adresa precizată. Cu această ocazie se decrementează şi contorul; cînd valoarea acestuia devine nulă se şterge instrucţia RST 4 precum şi linia corespunzătoare ei în tabela TTIME.

Se precizează că implementarea eficientă a mediului de programare presupune utilizarea unor funcţii cu caracter general aparţinînd sistemului de dezvoltare, cu deosebire cele referitoare la regimul de activitate interactiv (citirea comenzilor, a parametrilor, teste de corectitudine, rutine pentru afişare, conversii, etc.). Utiliza-

rea acestor rutine simplifică mult implementarea mediului de programare, interfaţa cu ele realizîndu-se la nivelul unei tabele de echivalenţe care precizează adresele lor de implantare.

5.6. Concluzii referitoare la activitatea de implementare a sistemelor

Aplicarea metodologiilor de proiectare indicate în capitolele 3 şi 4, conform detaliilor precizate în capitolul de faţă au condus la realizarea unor SØTR avînd următoarele dimensiuni exprimate în Kocteţi (fig.5.6.1).

Se precizează că:

a) în ceea ce priveşte executivele TR, diferenţele de dimensiune provin din numărul şi maniera de tratare a IT.

	EXECUTIV	MEDIU PROG	DATE PROPRII
SFITOS	0,65	0,8	0,4
SISTV	1,00	0,7	0,7
SIECAR	1,30	1,2	0,8

Fig.5.6.1. Dimensiuni ale sistemelor realizate.

b) mediile de programare implementate în toate cazurile presupun funcţiile de bază ale unui monitor interactiv, de care sînt legate prin intermediul unei tabele de echivalenţe.

c) nu au fost luate în considerare procesele sistem care tratează dispozitivele periferice.

d) pentru fiecare proces (sistem sau utilizator), baza de date a sistemului creşte cu 18 octeţi (16 oct. pentru Tabela de definiţie a procesului şi 2 oct. pentru Tabela sistem - TSIST). Inafară de aceasta, fiecare proces presupune: o tabelă proces (6 octeţi), o zonă proprie de mesaje, o zonă pentru stiva proprie şi codul propriu-zis. Zonele amintite s-au dimensionat în funcţie de necesităţi.

Punerea la punct a SØTR s-a realizat pe un microsistem M18, după care codurile generate au fost transbordate cu ajutorul benzii perforate pe sistemele destinaţie, în vederea testării lor. Se poate aprecia că dezideratele impuse prin condiţiile de proiectare precizate în capitolul 2, au fost atinse în cadrul SØTR realizate, demonstrînd valabilitatea metodologiei utilizate.

6. Implementarea unei arhitecturi de sistem de operare TR pe un sistem de calcul m/μP

Pornind de la aspectele teoretice prezentate în § 2.2.3, pentru aplicaţii, a fost dezvoltată în cadrul Catedrei de Calculatoare o arhitectură de sistem multiprocesor asimetrică avînd μP componente slab cuplate prin intermediul unei memorii comune. Pentru această structură hardware a fost proiectată şi implementată de către autor o arhitectură software corespunzătoare. Prezentul capitol îşi propune să prezinte unele detalii de implementare ale acestei arhitecturi [CP82],[Cr82],[CG83],[PP83a],[GC84] .

6.1. Descrierea sistemului de calcul m/μP

Sistemul m/μP utilizat este un sistem de tip MIMD construit cu un număr redus de μP de 8 biţi. El a fost realizat prin cuplarea unor microsisteme autonome mono μP. Elementele de legătură între μP ai sistemului sînt un bloc comun de memorie şi o reţea de interconectare între sistemele de IT ale microsistemelor.

Sistemul de calcul poate conecta în structura implementată cel mult 4 μP, pentru început lucrîndu-se cu 2 microsisteme bazate pe circuite din familia I8080/85 (fig.6.1.1), microsisteme deja prezentate în cadrul capitolului precedent.

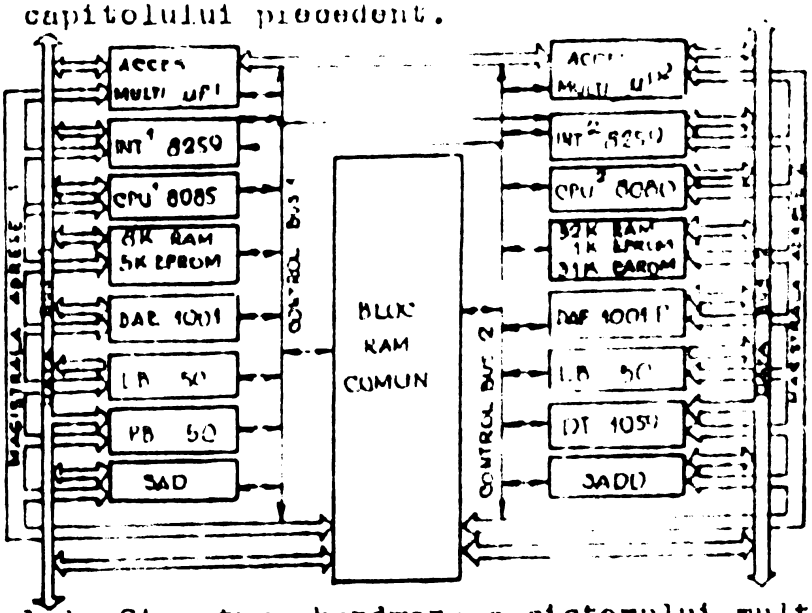


Fig.6.1.1. Structura hardware a sistemului multiprocesor.

Primul microsistem bazat pe μP 8085 este înzestrat cu 5 Ko de EPRØM şi 8 Ko de RAM. Pe lîngă periferia standard (DAF 1001, LB 50, P 50) mai este echipat cu un sistem pentru achiziţia datelor. Al doilea microsistem conţine un μP 8080 şi are o memorie de 64 Ko (32 Ko EPRØM şi 32 Ko RAM MØS dinamic). Acest sistem conţine în structura sa un display (DAF 1001 P), un lector de bandă perforată (LB50)

un perforator de bandă (DT105S) şi o maşină de scris (RØBØTRON SM4000). Ca memorie de masă este utilizat un casetofon comercial. Acest microsistem poate fi conectat la procese timp-real prin intermediul unui sistem pentru achiziţia şi distribuţia datelor realizat cu circuite hibride şi integrate BURR BROWN. Conectarea periferiecelor la unitatea centrală se face prin intermediul sistemului de întreruperi realizat cu circuitele 8259 şi 8214.

Cel de al doilea sistem este echipat cu un orologiu de timp-real realizat cu circuitul dedicat 8253.

Legătura dintre microsistemele componente este asigurată prin intermediul unei zone comune de memorie de 2Ko care poate fi utilizată de către toate µP. Accesul fiecărui microsistem la această zonă poate avea loc în două regimuri distincte: normal sau exclusiv. In prima situaţie fiecare procesor are acces la blocul de memorie comun cîte un ciclu. In cazul unei concurenţe a cererilor de acces, o schemă de arbitraj serveşte beneficiarii în ordinea unei priorităţi cablate. Intrucît nici unul dintre utilizatori nu generează în fiecare stare a maşinii o cerere de acces este exclusă utilizarea continuă a memoriei doar de către cel mai prioritar utilizator. Pînă la terminarea ciclului de memorie curent, dacă celălalt µP cere acces la aceeaşi zonă de memorie el este menţinut în WAIT.

In cel de al doilea regim, cel exclusiv, oricare din cele două procesoare poate bloca accesul celuilalt, atîta timp cît doreşte să lucreze neîntrerupt în memoria comună. Pînă la terminarea acestui regim exclusiv impus de unul din cele două µP, în caz că celălalt are acces în aceeaşi zonă, el este menţinut în WAIT pînă la eliberarea memoriei. Intrarea în regimul exclusiv poate fi comandată de oricare din cele două procesoare cu ajutorul unei instrucţii de scriere în memorie la o adresă dedicată acestui scop. Ieşirea din acest regim este admisă doar sub controlul procesului activ, adică a celui care are acces la memoria comună în momentul respectiv.

Pe lîngă această modalitate de transfer a informaţiilor între cele două µP, sistemul m/µP este prevăzut cu legături între sistemele de IT ale µP componente. Aceste conexiuni permit sincronizarea activităţii microsistemelor, deoarece prin intermediul lor, fiecare µP poate cere o întrerupere la celălalt µP printr-o instrucţie de intrare/ieşire. Nivelul de IT pe care este legat µP care cere întreruperea, este tratat de către acesta ca un port.

6.2. Detalii de implementare a nucleului distribuit al sistemului m/μP

Pornind de la arhitectura hardware realizată, a fost proiectă o arhitectură software corespunzătoare de SØ distribuit prin replicare (vezi § 4.6.1). Conform celor precizate în acest paragraf, fiecare /μsistem are propriul său SØ organizat ierarhic, ce se constituie într-o componentă autonomă a sistemului distribuit. Interconectarea software a sistemelor este realizată prin intermediul unor componente specializate (nucleul distribuit şi componenta distribuită pentru cooperare), de asemenea distribuite prin replicare, care fac apel la facilităţile hardware anterior prezentate: memoria comună şi sistemul de întreruperi. Sistemul m/μP timp-real implementat a fost denumit MULTIX.

Subliniind faptul că marea majoritate a aspectelor prezentate în capitolul precedent rămîn valabile pentru fiecare din componentele autonome ale STR distribuit, în continuare se fac referiri la cîteva din detaliile de implementare ale componentelor distribuite ale arhitecturii software m/μP. Pentru început, în paragraful de faţă se abordează problemele nucleului distribuit al sistemului.

După cum s-a precizat în § 4.6.2, nucleul distribuit este divizat în două subniveluri care implementează funcţiile mecanismului de acces la memoria comună respectiv funcţiile coordonării proceselor. Şi în acest caz elementul de pornire al implementării îl constituie stabilirea structurilor de date. Se subliniază faptul că toate structurile indicate în capitolul anterior rămîn valabile pentru fiecare componentă autonomă (microsistem) în parte. Pentru componentele distribuite ale nucleului se precizează în plus structurile de semafor special - fig.6.2.1 şi zona cutii poştale (box-uri de întreruperi)- fig.6.2.2.

În cadrul sistemului MULTIX, procesele se identifică prin numere cuprinse între (0 - 255); prin convenţie primii trei biţi codifică numărul /μP (0 - 7) iar restul biţilor numărul intern al procesului în cadrul /μsistemului.

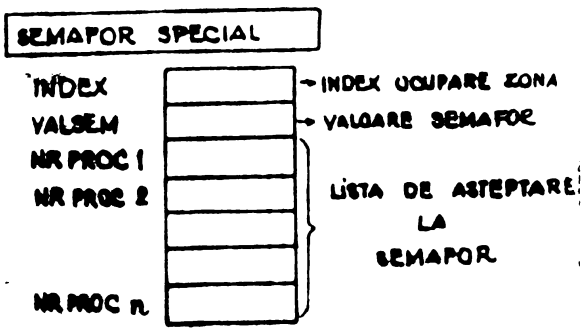


Fig.6.2.1. Structura semaforului special.

Mecanismul de acces la memoria comună care rezolvă problemele excluziunii mutuale, este mecanismul PHS prezentat deja în cadrul lucrării (§ 3.3.3, § 4.6.2) care a fost implementat în forma rutinelor ACCES şi ELIB. Fiecare

zonă de memorie comună ce va fi protejată,are asociat un index de ocupare a zonei, materializat în cadrul implementării, prin primul octet al zonei. In cadrul rutinei ACCES se verifică acest index în regim de aşteptare activă şi dacă se găseşte liber, se cere acces exclusiv la memorie (fig.6.2.3). Este important de remarcat faptul că, testul indexului se repetă după obţinerea accesului exclusiv la memoria comună, deoarece între momentul solicitării şi cel al obţinerii accesului, cererea curentă poate fi întîrziată de către arbitrul de prioritate şi un alt acces să poziţioneze între timp indexul zonei. Se mai precizează că la nivel fizic,solicitarea şi părăsirea regimului special, se realizează prin instrucţii simple de scriere în memorie prin poziţionarea pe 1 respectiv pe 0 a bitului cel mai puţin semnificativ al unei adrese de memorie fixe (OE7FH pentru 8080 respectiv OE7FEH pentru 8085).

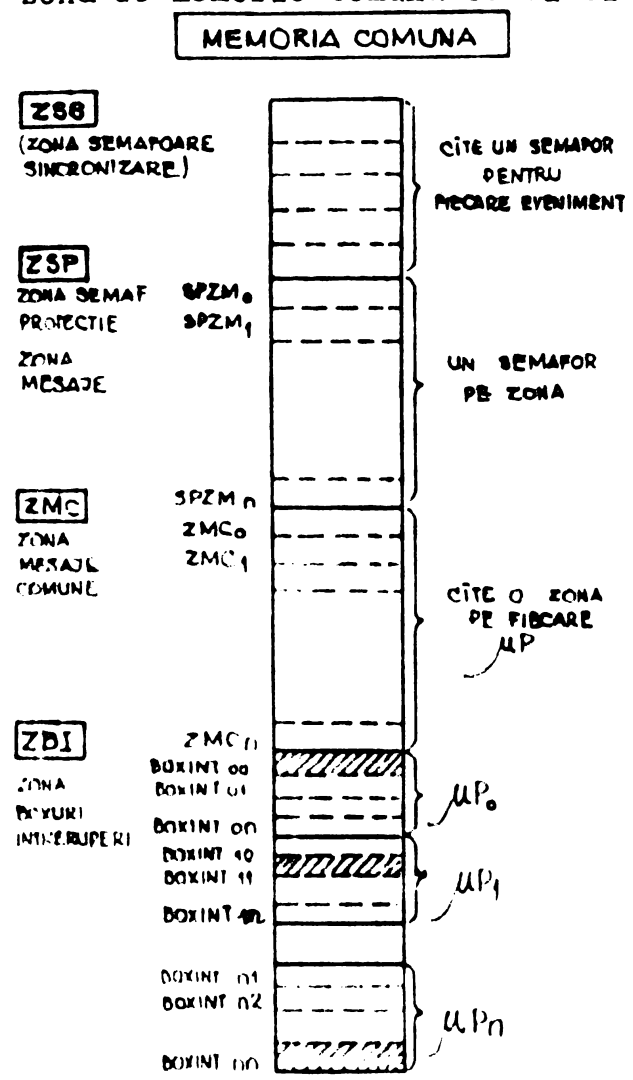


Fig.6.2.2. Structura zonelor ce aparţin memoriei comune.

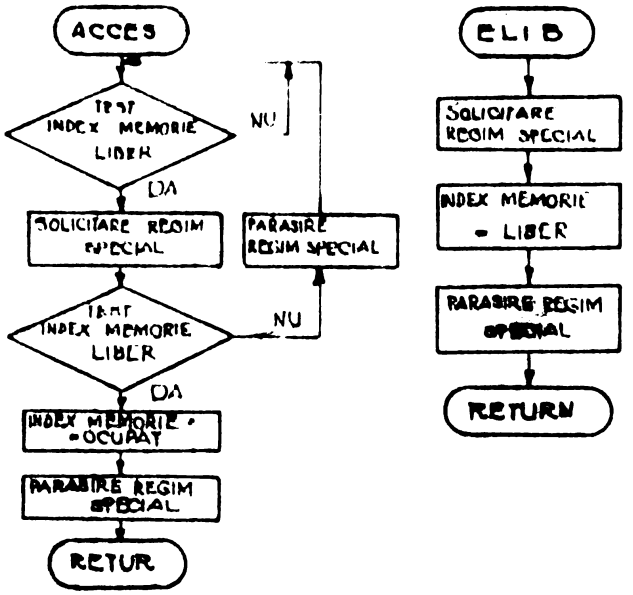


Fig.6.2.3. Ordinograma de principiu a funcţiilor ACCES şi ELIB.

Una din problemele pe care le ridică acest mecanism se referă la dimensiunea şi natura zonelor de memorie comună cărora este necesar a li se asocia indexul. In cazul implementării de faţă s-a ajuns la concluzia că este suficient ca numai semafoarele speciale să fie protejate prin intermediul acestui mecanism,ele putînd asigura în continuare protecţia altor structuri de date (§ 6.4). Acest minim necesar adoptat în implementare, nu împiedecă însă utilizatorul să protejeze în mod direct şi alte zone, în acest caz revenindu-i în

mod direct consecinţele unor tratări defectuoase. Se recomandă însă în astfel de cazuri utilizarea semafoarelor speciale.

Componenta distribuită pentru coordonare cuprinde rutinele PS şi VS care acţionează asupra semafoarelor speciale. In cazul implementării, aceste rutine devin componente ale rutinelor generale P şi V, ele fiind selectate automat în cazul semafoarelor speciale detectabile prin adresele lor situate în memoria comună. Secvenţa PS are funcţii identice cu cele ale operaţiei P cu excepţia utilizării în plus a mecanismului de protecţie PHS şi a modificării manierei de înlănţuire la semafor, motiv pentru care implementarea ei nu ridică probleme deosebite.

După cum s-a precizat în § 4.6.2, datorită faptului că o operaţie V executată de un proces aparţinînd unui μP, asupra unui semafor special, eliberează un proces aparţinînd unui alt μsistem, execuţia operaţiei V are loc în două etape distincte:

1° - eliberarea procesului din lista de aşteptare a semaforului special, lucru care se realizează de către SØ al μsistemului care a iniţiat operaţia V, sub protecţia mecanismului PHS.

2° - pasarea operaţiei V μsistemului căruia îi aparţine procesul eliberat în vederea actualizării stării acestuia (din „aşteptare" în „pregătit"). In cazul sistemului MULTIX pentru a ne implementa acest lucru ne utilizează zona boxurilor de întreruperi (a cutiilor poştale - fig.6.2.2). Fiecare μP al sistemului are în memoria comună o zonă proprie partajată în locaţii corespunzătoare tuturor μP sistemului numite cutii poştale. Astfel există zona BOXINT corespunzătoare μP_0 (cu cutiile $BOXINT_{00}$ şi $BOXINT_{01}$) şi BØXINT$_1$ corespunzătoare μP_1 (cu cutiile având indicii 10 şi 11). Desigur la fiecare BOXINT cutiile cu indicii ii nu ne utilizează .

Pasarea V-ului presupune introducerea identităţii procesului „de eliberat" în cutia $BOXINT_{k\ell}$ unde k este numărul procesorului destinaţie iar ℓ numărul procesorului care a iniţiat operaţia V. Identitatea procesorului destinaţie (k) se determină din numărul procesului „de eliberat". După introducerea numărului procesului cutia poştală corespunzătoare, se lansează o IT către procesorul destinaţie. Pentru implementarea sistemului de întreruperi interprocesoare, s-a utilizat tehnica memory-mapped: fiecărui μP i s-a alocat cîte o locaţie în memoria comună prin care i se poate solicita o IT. μP care doreşte să lanseze IT, testează în regim de aşteptare activă locaţia de memorie corespunzătoare μsistemului destinaţie şi în momentul în care o găseşte 0, o poziţionează pe 1 declanşînd

lansarea IT. In sistemul destinaţie semnalul este legat la un nivel de IT al µP. Adresele utilizate în cadrul implementării au fost OE7FDH pentru 8080 şi OE7FCH pentru 8085. Sînt valabile următoarele convenţii:

a) completarea unei cutii poştale se realizează numai dacă ea este găsită goală. Aceasta rezolvă problema secvenţializării cererilor provenind de la acelaşi procesor.

b) o IT se lansează doar cînd locaţia corespunzătoare ei este nulă. Aceasta rezolvă problema sincronizării microsistemului care lansează IT cu µsistemul întrerupt.

c) secvenţa de tratare a IT provenind de la un alt µP presupune recuperarea numărului procesului din cutia poştală, trecerea sa în stare pregătit, introducerea valorii 0 în cutia poştală utilizată, şi poziţionarea lui „0" în locaţia de memorie afectată întreruperii, pentru a marca tratarea acesteia şi a permite solicitarea unei noi IT.

6.3. Detalii de implementare a componentei distribuite pentru cooperare şi a mediului de programare

Componenta distribuită pentru cooperare asigură realizarea schimbului de informaţii între procesele aparţinînd unor procesoare diferite prin intermediul mesajelor externe care se transmit via memoria comună. Elementele specifice referitoare la mecanismul de funcţionare al acestei componente au fost prezentate detaliat în § 4.6.3. In paragraful de faţă se furnizează cîteva elemente de implementare.

Punctul de pornire al implementării acestei componente îl constituie şi în acest caz stabilirea structurilor de date specifice. Este vorba despre:

a) zona de mesaje comune (zona porturilor) fig.6.2.2, care cuprinde cîte o subzonă de 256 de octeţi pentru fiecare din cele două µP ale sistemului (zonele ZMC). In aceste zone se introduc mesajele care le sînt destinate.

b) zona semafoarelor de protecţie a zonelor de mesaje, cuprinde cîte un semafor special pentru fiecare zonă (zona ZSP).

c) zona semafoarelor de sincronizare, cuprinzînd semafoarele proceselor „poştaş" (vezi § 4.6.3), materializată în zona ZSS.

Se precizează că toate aceste elemente aparţin memoriei comune şi că excluziunea mutuală se rezolvă cu ajutorul semafoarelor de protecţie, care la rîndul lor sînt implementate utilizînd mecanismul PHS.

Secvenţele care rezolvă problemele transmiterii mesajelor, sînt cuprinse în rutinele SEND respectiv REC, selecţia lor realizîndu-se în mod automat la întîlnirea unui mesaj extern detectat prin numărul procesorului căruia îi aparţine procesul destinaţie.

O menţiune în plus trebuie făcută în legătură cu procesul sistem „poştaş" (distribuitor de mesaje) replicat pe ambele µsisteme. Acesta se autoblochează pe semaforul poştaş propriu din memoria comună şi este eliberat la transmiterea unui mesaj (§ 4.6.3). Procesul acţionează asupra unei zone precizate în memoria comună (zona de mesaje comune a procesorului în cauză) cu semafoare (poştaş şi de protecţie) specificate. Sarcina sa este ca sub protecţia semaforului să extragă mesajul din portul propriu şi să-l lanseze cu un SEND normal procesului destinaţie, care de această dată este un proces intern (ambele procese aparţin aceluiaşi µsistem).

In ceea ce priveşte „mediul de programare", în implementarea s-a utilizat metoda M.4.6.3. a „mediului de programare" replicat. Conform acesteia, pe fiecare din µsistemele componente a fost implementat un MP specific. Adoptarea acestei metode a fost determinată de mai multe considerente:

a) caracterul experimental al implementării,

b) necesitatea unor intervenţii prompte, măsurări şi ajustări care în alte condiţii ar fi fost greu de realizat,

c) existenţa unor MP pentru µsistemele ce compun sistemul m/µP care cu unele modificări au putut fi adaptate scopului dorit

6.4. Structura ierarhică a interacţiunilor proceselor

Aspectele legate de interacţiunile dintre procese prezentate în § 2.4.3, au obţinut o nouă dimensiune în urma implementării lor concrete. Astfel s-a constatat că utilizarea coordonării drept mecanism de sincronizare pentru cooperare este foarte utilă şi eficace. Acest lucru a fost impus direct de faptul că implementarea mecanismului de cooperare prin mesaje presupune în mod necesar utilizarea semafoarelor. Conducînd raţionamentul în aceeaşi manieră se constată că implementarea semafoarelor pe un sistem mono µP multiprogramat presupune mecanismul monitorului monolitic în vederea realizării accesului în regim de excluziune mutuală, semaforul fiind în ultimă instanţă o variabilă partajată.

Pentru sistemele m/µP, acest mecanism nu este suficient, el trebuind să fie completat cu mecanismul PHS. In această viziune se remarcă o structurare ierarhică a mecanismelor de sincronizare şi protecţie, structurare pusă în evidenţă în fig.6.4.1 şi reflectată parţial şi în structurile de date aferente (§ 4.3.4).

Pornind de la această abordare se observă că interacţiunile se pot structura pe niveluri:

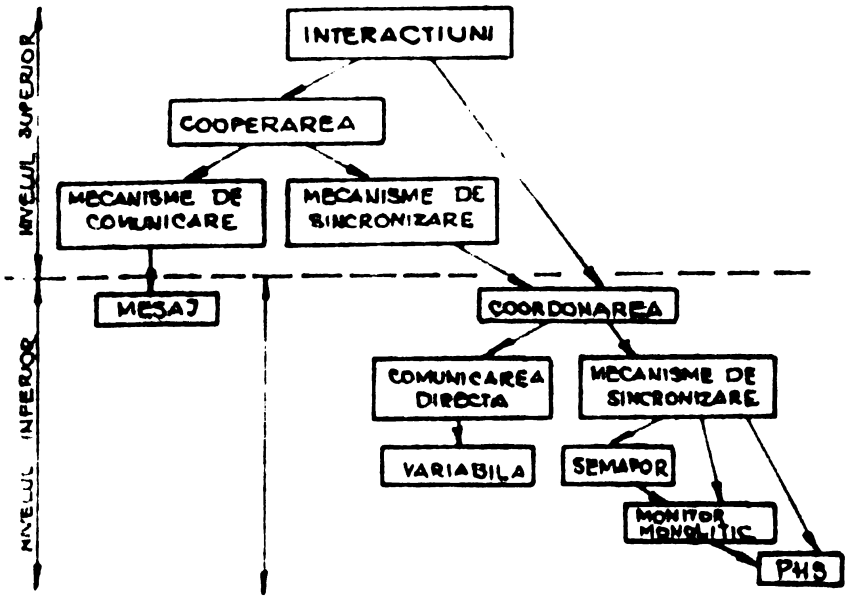


Fig.6.4.1. Structura ierarhică a interacţiunii proceselor.

- interacţiuni de nivel superior sau cooperare avînd drept mediu de acţiune procesele şi drept mijloc de acţiune mesajele
- interacţiuni de nivel inferior sau coordonare avînd drept mediu variabilele partajate situate în memoria proprie şi drept mijloc semafoarele
- un subnivel al coordonării impus de extinderea conceptului de semafor la sisteme m/μP, bazat pe mecanismul PHS (fig.6.4.2).

Această abordare teoretică structurală a interacţiunii proceselor pe lîngă avantajele clarităţii şi sistematizării, simplifică şi ordonează activitatea de implementare.

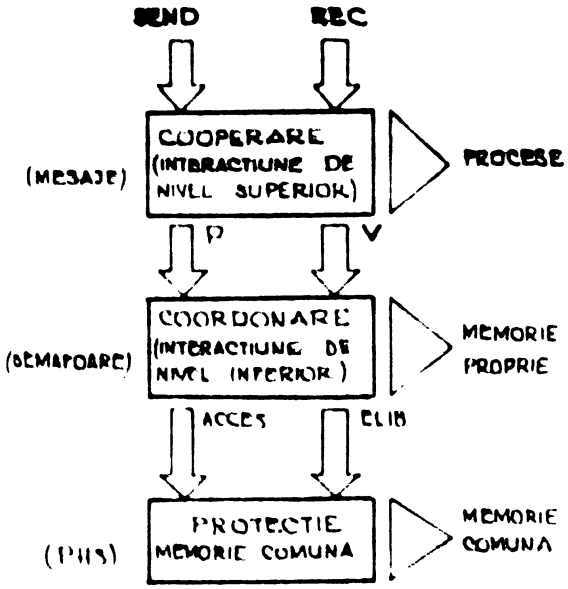


Fig.6.4.2. Niveluri de interacţiunii proceselor.

6.5. Concluzii referitoare la activitatea de implementare a sistemului

Aplicarea conceptelor şi metodologiilor de proiectare precizate pe parcursul lucrării pentru sisteme m/μP slab cuplate au condus la realizarea unui executiv avînd dimensiunea de 1,2 Kocteţi de memorie (ca parte rezidentă replicabilă). Dimensiunea zonei de memorie comună utilizate pentru două microprocesoare a fost de 0,6 Kocteţi. In ceea ce priveşte componentele autonome ale sistemului m/μP, rămîn valabile concluziile prezentate în § 5.6, cu precizarea că pentru fiecare sistem în parte executivul TR este cel de tip MULTIX.

Dezvoltarea componentelor autonome şi a celor distribuite pentru sistemele care alcătuiesc sistemul m/μP, s-a realizat pe un microsistem de calcul M18. Trecerea lor pe sistemul destinaţie, în vederea testării în condiţii reale, s-a efectuat prin intermediul

benzii perforate, ambele sisteme componente dispunînd de facilităţi hardware în acest sens. Comportarea corespunzătoare a componentelor implementate pe parcursul testării, demonstrează valabilitatea metodei utilizate.

7. Concluzii

După cum s-a precizat în primul capitol, scopul acestei orări este acela de a-şi aduce într-o manieră originală contribuţia la abordarea problematicii timpului-real din punctul de vedere al proiectantului de sisteme de operare în cadrul restrîns al sistemelor mici mono şi multiprocesor bazate pe µP. In acest context s-au stabilit următoarele obiective:

1) Selecţia şi sistematizarea cunoştinţelor domeniului TR.

2) Particularizarea unor aspecte teoretice ale teoriei generale a sistemelor de operare la caracteristicile TR.

3) Furnizarea unor metodologii de lucru utile proiectantului software în concepţia şi realizarea unor SØTR pentru sisteme de calcul mono şi m/µP.

7.1. Contribuţii originale

Pornind de la scopul şi obiectivele acestei lucrări se pot scoate în evidenţă următoarele contribuţii originale ale autorului:

- Definirea termenilor de timp-real şi ambianţă TR prin precizarea elementelor caracteristice.

- Studiul arhitecturilor sistemelor de calcul mono şi multiprocesor, abordat prin prisma cerinţelor unei ambianţe TR şi apreciarea din acest punct de vedere a principalelor tipuri de arhitecturi. In concluzie, se propune o structură originală de sistem m/µP, slab cuplat, asimetric, distribuit local.

- Precizarea caracteristicilor hardware care trebuie să stea în atenţia proiectantului software TR.

- Elaborarea unei concepţii originale asupra STR concretizată în definirea noţiunii de SØTR prin separarea sa în două componente - „Executivul TR" şi „Mediul de programare TR", cu precizarea factorilor care îi oferă un statut aparte faţă de sistemele de operare universale clasice.

- Definirea sistematică a funcţiilor componentelor unui SØTR.

- Reconsiderarea noţiunii de proces şi de interacţiune interprocese în contextul ambianţei TR. S-a propus introducerea termenilor de coordonare şi cooperare pentru a desemna tipurile specifice de interacţiuni dintre procese.

- Elaborarea unui model pentru studiul sistemelor de procese implementat în limbajul PASCAL care cuprinde algoritmi pentru determinarea închiderii tranzitive a unei relaţii definite pe mulţimea proceselor unui sistem, determinarea grafului minim de precedenţă şi a grafului paralel maximal al unui sistem dat. Pornind de la acest model s-au elaborat metodologii de lucru pentru determinarea grafului minim de precedenţă, pentru studiul propriu-zis al determinării şi pentru determinarea grafului paralel maximal.

- Sistematizarea aspectelor teoretice referitoare la activitatea de coordonare a proceselor, urmată de precizarea şi analiza celor mai potrivite mecanisme pentru STR: „monitorul monolitic" şi semafoarele. Se propune extinderea originală a acestora pentru sisteme m/μP slab cuplate în forma mecanismului PIS şi a semafoarelor speciale, pentru care se furnizează maniere concrete de implementare. Se prezintă de asemenea cîteva metodologii de utilizare a acestor mecanisme în implementarea excluziunii mutuale, a sincronizării de condiţie, sau a alocării resurselor, necesare proiectantului de STR.

- Pornind de la studiul relaţiei producător/consumator se realizează definirea teoretică a unui mecanism de cooperare pentru sisteme de procese concurente bazat pe transmiterea mesajelor. Pentru acest mecanism se prezintă posibilităţi concrete de implementare, furnizîndu-se metode pentru dimensionarea bufferului, pentru implementarea diferitelor tipuri de mesaje (prioritare, cu răspuns, etc.). Se scot în evidenţă o serie de posibilităţi de utilizare a mecanismului propus sintetizate în forma unor metodologii referitoare la implementarea diferitelor tipuri de cooperări interprocese, a relaţiilor de precedenţă dintre ele şi legat de aceasta a rezolvării problemei proceselor partajate. Se propune extinderea mecanismului cooperării prin mesaje şi pentru sisteme m/μP.

- Elaborarea unui model pentru studiul planificării sistemelor de procese închise, implementat în limbajul PASCAL pe baza unui studiu al aspectelor caracteristice ale planificării în execuţie a proceselor în cadrul unor SØTR. Utilizînd acest model, s-au propus o serie de metode de planificare specifice unor STR, formulîndu-se unele concluzii practice pentru proiectantul de SØTR.

- Elaborarea unui model generalizat pentru studiul stării resurselor unui sistem de procese de asemenea implementat în PASCAL, care permite abordarea problemei interblocării proceselor. Se analizează în acest context problemele prevenirii, detecţiei şi evitării interblocărilor furnizîndu-se o serie de metode şi algoritmi cu caracter aplicativ pentru rezolvarea acestor aspecte.

- Abordarea structurală a noţiunii de arhitectură de SØ şi definirea unei arhitecturi de SØTR structurată pe niveluri şi subniveluri ierarhice.

- Particularizarea unor metode generale de proiectare şi implementare a SØ la nivelul SØTR: metoda construcţiei ierarhice şi metoda adaptării.

- Elaborarea de metodologii de proiectare şi realizare pentru fiecare din nivelurile unei arhitecturi de SØTR. In acest context se precizează structurile de date specifice fiecărui nivel al arhitecturii precum şi interrelaţiile dintre acestea într-o manieră structurată ierarhic. In proiectarea şi implementarea acestor structuri se realizează conceptul de obiect.

- Extinderea conceptului de arhitectură ierarhică pentru SØTR implementate pe sisteme m/µP din clasa celor propuse în lucrare (mai cuplate asimetrice). Se propune o astfel de arhitectură distribuită prin replicare pentru care se furnizează metodologiile de proiectare şi realizare a componentelor distribuite specifice.

- Aplicarea în practică a aspectelor teoretice prezentate anterior în realizarea unor SØTR pentru sisteme de calcul mono/µP pentru care se prezintă detaliile de implementare ale fiecărei componente în parte. De asemenea se prezintă detaliile de implementare ale unui SØTR, destinat unui sistem m/µP.

7.2. <u>Valorarea aplicativă şi direcţii de dezvoltare viitoare</u>

Aspectele prezentate în lucrare au fost valorificate în cadrul a 10 lucrări ştiinţifice publicate şi a 3 contracte de cercetare ştiinţifică. Pe viitor se conturează însă noi perspective de valorificare ţinînd cont de interesul crescînd pe care îl manifestă industria şi institutele de cercetări faţă de sistemele TR, interes stimulat pe de o parte de proliferarea sistemelor de calcul bazate pe µP în cele mai diverse domenii, iar pe de altă parte de creşterea cererii de software adecvat diverselor aplicaţii.

In acest context, din mulţimea direcţiilor posibile de dezvoltare viitoare a domeniului SØTR se prezintă doar trei mai importante: domeniul „sistemelor la cheie", domeniul roboticii şi domeniul limbajelor concurente.

A. „Sistemele la cheie" sînt sisteme strict dedicate unor aplicaţii concrete. Varietatea acestor aplicaţii pe de o parte, iar pe de altă parte cerinţele sporite de performanţă şi eficienţă ridică probleme serioase proiectantului software în implementarea unor astfel de sisteme. In acest context, aspectele prezentate în lucrarea de faţă reprezintă o contribuţie la rezolvarea unora dintre ele, cu atît mai mult cu cît în ţara noastră există preocupări, deocamdată în fază incipientă în acest domeniu, preocupări cuprinse în cadrul mai larg al creării unei industrii naţionale de programe. In acest sens pot fi amintite preocupările şi realizările Laboratorului software al Filialei I.T.C. Timişoara care prefigurează posibilităţile unei largi colaborări.

B. Robotica reprezintă un domeniu deosebit de dinamic, cu largi perspective de dezvoltare în ţara noastră, în care Scoala politehnică timişoreană are o serie de realizări. Se apreciază că pentru anumite tipuri de aplicaţii ale roboţilor, se pretează în condiţii deosebit de avantajoase, conducerea acestora de către sisteme TR mono şi multi /uP, domeniu în care aspectele prezentate în lucrare sînt foarte utile. Mai mult chiar, arhitectura sistemului m/uP prezentat a fost concepută avînd în vedere şi cerinţele conducerii unor roboţi industriali. Toate acestea conferă perspective certe SØTR în domeniul roboticii.

C. Domeniul limbajelor concurente, larg abordat în ultima perioada, ca o etapă nouă, evoluată a activităţii de programare, acordă o atenţie cu totul particulară execuţiei paralele şi sincronizării activităţilor (proceselor, taskurilor). In acest context, implementarea unor limbaje concurente pe sisteme mono sau multiprocesor (/uP) presupune funcţii adecvate pentru rezolvarea acestor aspecte. Una din alternativele de rezolvare o reprezintă executivul TR prezentat în lucrare, care prin funcţiile cu care este înzestrat este în măsură să satisfacă exigenţele unei astfel de implementări. Se remarcă în acest sens preocupările existente la Centrul de calcul al Institutului politehnic în acest domeniu, care de asemenea conturează premisele unei colaborări mai îndelungate. Nu este lipsit de interes ca în acest context, să se sublinieze faptul că, la ora actuală în lume există o serie de preocupări legate de concepţia şi realizarea unor limbaje de programare de nivel înalt, cu facilităţi TR, care să determine o creştere substanţială a productivităţii şi siguranţei activităţii de programare în acest domeniu.

In toate domeniile prezentate, necesitatea creşterii performanţelor şi scăderii timpilor de răspuns va impune utilizarea unor sisteme m/uP respectiv a unor structuri de astfel de sisteme. In această

direcţie, dezvoltarea unor structuri de tipul celei propuse în lucrare, bazate pe cuplarea slabă a sistemelor componente reprezintă una din alternativele promiţătoare de abordare a acestui domeniu.

In final se poate aprecia că aspectele prezentate în această lucrare sînt ele însele susceptibile de a fi aprofundate, perfecţionate şi dezvoltate în sensul domeniilor anterior precizate, extinse şi asupra altor categorii de sisteme de calcul şi eventual consolidate într-o teorie unitară a domeniului TR, în care aspectele structurate pot să ocupe un loc foarte important.

8. Bibliografie

Prescurtări utilizate:

AFIPS	: American Federation of Information Processing Societies.
BSIPTVT	: Buletinul Stiinţific IPTV Timişoara.
CACM	: Communications of the Association for Computing Machinery.
CS	: ACM Computing Surveys.
NHPC	: North Holland Publishing Company.
P	: proceedings.
SIGØPS	: ACM SIGØPS (Special Interest Group on Operating Systems).
SØSP	: Symposium on Operating Systems Principles.
SPE	: Software, Practice and Experience.
TC	: IEEE Transaction on Computers.
TCS	: ACM Transaction on Computer Systems.
TSE	: IEEE (Institute of Electrical and Electronics Engineers). Transactions on Software Engineering.

AB78 E.A.Akkoyunlu, A.J.Bernstein, F.B.Schneider, A.Silberschatz: "Conditions for the Equivalence of the Synchronous and Asynoronous Systems", TSE,VOL.SE-4,No.6,Nov,1978, pag. 507-516.

Al77 M.W.Alford: "A Requirements Engineering Methodology for Real-Time Processing Requirements", TSE,VOL.SE-3,No.1, January, 1977, pag. 60-69.

An79a G.Andler: "Synchronization Primitive and the Verification of Concurrent Programs", Operating Systems, Theory and Practice, N.H.P.C, 1979, pag.67-99.

An79b G.R.Andrews : "The Design of a Message Switching System: An Application and Evaluation of Modula", TSE,VOL.SE-5, March, 1979, pag.138-147.

AS83 G.R.Andrews, F.B.Schneider: "Concepts and Notations for Concurrent Programming, CS,Vol.15,No.1,March,1983, pag.3-43.

Ba74 V.Baltac: "Optimizarea sistemelor de operare ale calculatoa-relor numerice",Ed.Facla,Timişoara,1974.

Ba78 J.Backus: Can Programming be Eliberated from the Von Neumann Style ? A Functional Style and its Algebra of Programs", CACM,Aug.1978,Vol.21,No.8.pag.613-641.

Ba81 J.F.Bartlett: "A Non Stop Kernel", SIGØPS, VOL 15,No.5, December 1981,P.of the Eights SOPS,pag.22-29.

Ba83 V.Baltac: "Tehnica electronică de calcul şi dezvoltarea Romaniei".Vol.„Noile tehnologii de vîrf şi societatea", EP,1983, pag.51-75.

BB76 C.Bass,D.Brown: "A Perspective on Microcomputer Software", P.of the IEEE,Vol.64,No.6,June,1976,pag.905-909.

BD84 V.Baltac,ş.a: "Sisteme interactive şi limbaje conversaţionale", ET, Bucureşti,1984.

BE72 C.G.Bell, J.L.Eggert, J.Grason, P.Williams: "The Description and the Use of Register Transfer Modules (RTMs)", TC,Vol.C-21,No.5,May,1972,pag.495-500.

BH81 A.Bernstein, P.K.Harter: "Proving Real-Time Properties of Programs with Temporal Logic", SIGØPS,Vol.15,No.5, December, 1981,pag.1-11.

BI79 R.E.Birney: "Processor Architecture Anticipates Future Performance Requirements", Computer Design,April 1979, pag.(71-79).

BJ79 R.R.Bate,D.S.Johnson: "Pascal Software Support Real-Time multiprogramming on Small Systems", Electronics, June 7,1979, pag.117-121.

BK80 C.Bass,J.S.Kennedy,J.M.Davidson: "Local Network Gives New Flexibility to Distribute Processing",Electronics, September 25,1980,pag.114-122.

BL82 A.D.Birnel,R.Levin,R.N.Needham,M.D.Schroeder: "Grapevine. An Exercise in Distributed Computing",CACM,April 1982, Vol.25,No.4,pag.260-274.

Ca73 G.Cain: "The Minicomputer as a System Element", Minicomputers in Instrumentations and Control,1973,pag.11-25.

CA82 T.C.K.Chou,J.A.Abraham: "Load Balancing in Distributed Systems", TSE,Vol.SE-8,No.4,July 1982,pag.401-411.

CC83 G.W.Cox,W.M.Corwin,K.K.Lai,F.J.Pollak: "Interprocess Comunication and Processor Dispatching on the Intel 432",TCS,Vol.1,No.1,Feb.1983,pag.45-66.

CC84 V.Cretu,V.Coltuc: "A Real-Time Devolopement System Implemented on the ECARØM-800 Process System", ICIPTVT, (în curs de publicare).

CD73 E.G.Coffman,P.Denning: "Operating System Theory", Prentice-Hall,Inc,Englewood Cliffs,1973.

Ce79 N.Ceauşescu: "Raport la cel de-al XII-lea Congres al PCR", EP, 1979, pag.40-41.

CG83 V.Cretu,V.Groza: "Sistem multiprocesor destinat dezvoltării unor aplicaţii timp-real", Buletinul Sesiunii de Comunicări Ştiinţifice „Tehnic 2000", Timişoara 1983, pag.75-78.

Ch79 D.R.Cheriton: "TØTH a Portable Real-Time Operating System", CACM,Feb.1979,Vol.22,No.2,pag.105-112.

Ch80 Y.P.Chien: "Multitasking Executive Simplifies Real-Time Microprocessor System Design", Computer Design, January,1980,pag.110-117.

Ch82 E.J.H.Chang: "Echo Algorithms: Depth Parallel Operations on General Graphs", TSE,July 1982,Vol.SE-8,No.4, pag.391-400.

CL82 J.P.Collins, M.M.Lewitt: "An Object Oriented Operating System for Microcomputers", Computer Design,Vol.21, June 1982, pag.165-172.

Co78 D.Del Corso: "An Experimental Multiprocessor System with Improved Internal Communication Facilities", Euromicro Journal, 4 (1978),pag.326-332.

Co79 M.Conrad: "Pascal-A High Level Language for Micros and Minis", Datamation, July 1979, pag.153-156.

CP78 V.Creţu,M.Petru,A.Furtunescu,E.Văcărescu: "Probleme ale tratării operaţiilor de I/E într-un sistem de operare timp-real", BSIPTVT,Tom 23(37),Faso.2 iulie-dec.1978, pag.270-276.

CP81 V.Creţu,D.Petriu: "Minisisteme de operare în timp-real pentru sisteme de calcul bazate pe microprocesoare",Lucrările colocviului de cibernetică,Timişoara 1981,Preprint. Litografia IPT,pag.35-39.

CP82 V.Creţu,D.Petriu: "Sistem de operare multiprocesor", Al II-lea Simpozion de teoria sistemelor,Vol.III,Secţiile a III-a şi a IV-a,Craiova 1982, pag.315-320.

Cr80 V.Creţu: "Proiectarea şi realizarea unor sisteme de operare timp-real", Referat doctorat 1980.

Cr81a V.Creţu: "Nucleul unui sistem de operare timp-real" BSIPTVT, Tom 26(40),Faso 1 ian-iunie 1981,pag.183-188.

Cr81b V.Creţu. "Stadiul actual al dezvoltării limbajelor de programare pentru mini şi microsisteme de calcul",Referat doctorat 1981.

Cr82 V.Creţu: "Sistem de operare distribuit în timp-real pentru conducerea unor roboţi industriali", Conferinţa Nat. Electr.Autom,Telecom şi Calc, Bucureşti 1982,Vol. Secţia 13, Secţia 15, pag.177-184.

Cr84a V.Creţu: "Consideraţii privind proiectarea şi realizarea unor sisteme de operare timp-real pentru sisteme de calcul cu microprocesor",Lucrările sesiunii de comunicări ştiinţifice „Electromotor 84",Februarie 1984,pag.275-280.

Cr84b V.Creţu: "Arhitectura software a sistemelor de operare timp-real",Buletinul Sesiunii de comunicări ştiinţifice pentru tineret "Tehnic 2000", Ediţia III-a,Timişoara, aprilie 1984, pag.326-330.

Da79 I.Dancea: "Microprocesoare. Arhitectură internă.Programare. Aplicaţii", Ed.Dacia,1979.

Di68a E.W.Dijkstra: "The Structure of the THE Multiprogramming System", CACM.11,5(May 1968),pag.341-346.

Di68b E.W.Dijkstra: "Cooperating Sequential Processes in Programming Language", Ed.F.GENUYS,Academic Press London,1968.

Di75 E.W.Dijkstra: "Guarded Commands, Nondeterminancy and Formal Derivation of Programs", CACM,Aug.1975,VOL.18,No.8, pag.453-457.

DB82a R.D.Dutton,R.C.Brigham: "The Complexity of a Multiprocessor task assignment problem without deadlines", Theoretical Computer Science 17(1982),NHPC,pag.213-216.

DB82b M.Dubois,F.A.Briggs: "Performance of Synchronized Iterative Processes in Multiprocessor Systems", TSE,Vol.SE-8, No.4,July 1982,pag.419-431.

DM77 St.Dumitrescu,V.Marinoiu,I.Dumitrescu,M.Florovici: "Aplicaţii inginereşti ale calculatoarelor. Calculatoare de proces" EDP,Bucureşti,1977.

Do81 Gh.Dodescu: "Modelarea sistemelor de operare",Ed.Ştiinţifică şi Enciclopedică,Bucureşti 1981.

Du79 F.G.Duncan: "Microprocessor Programming and Software Development",Prentice Hall International 1979.

Ev82 S.Evanczuk: "Productivity Gains as 8-bit Operating Systems Move to 16-bit Machines. Standards are Set Up and Portability Grows",Electronics,October 20,1982,pag. 181-184.

FC79 A.Furtunescu,V.Creţu,M.Petru,E.Văcărescu: "Sistem de operare specializat în aplicaţii de proces implementat pe calculatorul FELIX C-32P",BSIPTVT,Tom 24(38).Fasc 1 ian-iunie 1979,pag.86-92.

FC81 A.Furtunescu,V.Creţu,M.Petru: "Sistem de operare specializat în aplicaţii timp-real",Vol. „Programarea calculatoarelor",Ed.Facla 1981,pag.205-223.

Fo78 W.S.Ford: "Implementation of a Generalized Critical Region Conutruct",TSE,Vol.SE-4,No.6.Nov.1978,pag.449-456.

Fr73 A.Freedman: "Minicomputers in Real-Time Systems",Minicomputers in Instrumentation and Control 1973,pag.27-36.

FW78 D.P.Friedman,D.S.Wise: "A Note on Conditional Expresions", CACM,Nov.1978,Vol.21,No.11,pag.931-933.

FW80 C.Fulton,R.Whiffen: "High-level Language Takes on Most of Real-Time System Software",Electronics,December 4,1980, pag.157-160.

GC84 V.Groza,V.Creţu: "Sistem multiprocesor pentru aplicaţii timp real",Lucrările Sesiunii de comunicări ştiinţifice „Electromotor 84",Feb.1984,pag.263-268.

Ge74 I.Georgescu: "Sisteme de operare pentru calculatoare numerice Ed.Tehnica,1974.

GG84 V.Groza,A.Furtunescu,L.Balea: "Microprocessor-based System for Digital Processing of Analogue Signals", BSIPTVT, Tom 28(42),Fasc 1-2,ianuarie-decembrie 1984.

GG78 P.Gordon,B.Stallard: "Microprogrammed CPU Architectures Offers User-Artorable Minicomputer Performance",Computer Design,June 1978,Vol 17,No.6,pag.91-100.

GW80 J.Gurd,J.Watson: "Data Driven System for High Speed Parallel Computing-Part 1: Structuring Software for Parallel Execution",Computer Design,June 1980,pag.91-100.

Ha70 P.Brinch Hansen: "The Nucleus of a Multiprograming System" CACM,V13,No.4,April 1970,pag.238-241.

Ha73 P.Brinch Hansen: "Operating System Principles",Prentice-Hall INC,Englewood Cliffs, New Jersey, 1973.

Ha76a P.Brinch Hansen: "The Solo Operating System: A Concurrent Pascal Program",SPE,Vol.6,1976,pag.141-150.

Ha76b P.Brinch Hansen: The Solo Operating System: Job Interface" SPE,Vol.6,1976,pag.151-164.

Ha76c P.Brinch Hansen: "The Solo Operating System: Processes, Monitors and Classes",SPE,Vol.6,1976,pag.165-200.

Ha77 P.Brinch Hansen: "Experience with Modular Concurrent Programming",TSE,SE-3,No.2,March 1977,pag.156-159.

Ha78 P.Brinch Hansen: "Distributed Processes: A Concurrent Programming Concept",CACM,Nov.1978,Vol.21,No.11, pag.934-941.

HA79 C.E.Hewitt,R.R.Atkinson: "Specification and Proof Techniques for Serializers",TSE,Vol.SE-5,No.1,Jan.1979, pag.10-23.

Ha81 L.Hartge: "Distributed data processing",International Systems, Sept.1981, pag. 57-59.

He79 J.H.Heine: "The Control Response Time in Multi-Class Systems", CACM, July 1979,Vol.22,No.7, pag.415.

He83 J.H.Herzog: "A Design Perspective for Real-Time Task Control in Distributed Systems",IEEE.Transaction on Industrial Electronics,Vol.IE-30,No.1,Febr.1983,pag. 46-51.

HH82 D.Horovitz, F.Holsworth, J.Warten:"Concurrency: Key to 16-bit Operating System Efficiency", Computer Design, Vol.21,No.6,June 1982, pag.183-192.

Ho74 C.A.R.Hoare: "Monitors: An Operating System Structuring Concept",CACM,Vol,17,No.10,Oct.1974,pag.549-557.

Ho78 C.A.R.Hoare: "Communicating Sequential Processes", CACM, Vol.21,No.8,Aug.1978, pag.666-677.

Ho83 R.C.Holt: "Concurrent EUCLID, the UNIX* SYSTEM and TUNIS", Addison-Wesley Publishing Company, 1983.

HS67 M.A.Harrison,A.Schwartz: "SHARER-a Time Sharing System for the CDC 6600",CACM, 10/Oct.1967/pag.659.

HS78 T.P.Hughes, D.H.Sawin: "Breakpoint Design for Debugging Microprocesor Software",Computer Design,Nov.1978.

HZ79 M.Hamilton, S.Zoldin: "The Relationship Between Design and Verification",The Journal of Systems and Software 1, 1979, pag.(29-56).

Ic79a J.D.Ichbiah : "Preliminary ADA Reference Manual", CACM, V14,No.6,June 1979, Part A.

Ic79b J.D.Ichbiah: "Rationale for the Design of the ADA Programming Language",CACM,Sigplan Notices,V14,No.6,June 1979, Part B.

JC77 A.K.Jones, R.J.Chansler, I.Durham, P.Feiler, K.Schwans: "Software Management of Cm* . A Distributed Multiprocessor", AFIPS 1977, Vol.46,pag.657-663.

Jo81 R.C.Johnson: "Special Report: ADA the Ultimate Language", Electronics, Feb 24,1981, pag.127-132.

JP79 M.Joseph, V.R.Prasad, K.T.Narayana: "Language and Structure in an Operating System", Operating Systems, Theory and Practice, NHPC, pag.347-357.

JP81a I.Jurca,D.Petriu,V.Creţu: "Tendinţe ale dezvoltării programării concurente",Vol"Programarea calculatoarelor", Ed.Facla 1981, pag.9-43.

JP81b I.Jurca,D.Petriu,V.Creţu: "A Study on Concurrent Programming Concepts Implemented in High Level Languages", 4-th International Conference on Control Systems and Computer Science,1981, Vol.IV, pag.91-97.

Ju77 I.Jurca: "A Multiprocessor System with Multitasking Facilities", teză doctorat, TH Delft (Olanda) 1977.

Ju84 I.Jurca: "Sisteme de operare". Curs partea I,II.Litografia IPTVT, 1984.

Ka74 H.Katzan: "Operating Systems", Ed.Van Nostrand-Reinhold New York 1974.

KC81 K.C.Kahn,W.M.Corwin,a.a: "i MAX: A Multiprocessor Operating System for an Object-Based Computer", P.of the 8-th SØSP, SIGØPS, Vol.15,No.5,Dec.1981,pag.127-136.

KJ82 I.Kaufmann, I.Jurca, D.Petriu, V.Creţu: "Programarea în limbajul ADA", Ed.Facla, 1982.

Kl75 L.Kleinrock: "Queueing Systems. Volume I: Theory" Ed.John Wiley & Sons. 1975.

Kn74 D.E.Knuth: "Tratat de programarea calculatoarelor" Ed.Tehnică, Bucureşti, 1974.

Ko78 A.Koestler: "Janus-A Summing up Hutchinson", 1978.

Ko81 A.Konstam: "A Method for Controlling Parallelism in Programming Languages",Sigplan Notices,Vol.16,No.9,Sept.1980

KT79 Kwok-Tung Fung,H.C.Torng: "On the Analysis of Memory Conflicts and Bus Contentions in a Multiple Microprocessor System",TC,Vol.C-27,No.1,Jan.1979,pag.28-37.

La81 P.E.Lauer: "Syncronization of Concurrent Processes without Globality Assumptions", Sigplan Notices,Vol.16,No.9, September 1981.

LD81 H.Lorin,H.M.Deitel: "Operating Systems". Addisson-Wesley Publishing Company, 1981.

Le82 A.Lewis: "16-bit Operating Systems, A Whole New Ball Game", Computer Design, Vol.21,No.6,June 1982,pag.197-204.

Li78 H.Lienhard: "The Real-Time Programming Language PORTAL. Introduction and Survey", Landis & Gyr Review,2-1978, pag.2-9.

Li81 B.Liskov: "Report on the Workshop on the Fundamental Issues in Distributed Computing", Operating System Review Vol.15,No.3,July 1981, pag.9-39.

LR80 B.W.Lampson,D.D.Redell: "Experience with Processes and Monitors in Mesa", CACM,Vol.23,No.2,Feb.1983,pag.105-117

LT78 P.Lenders,J.Tiberghien: "Debugging Aids for Real-Time Microprocessor Systems", Euromicro Journal 4(1978)pag.220-221

Ma80 R.N.Magee: "RTL/2 Multi-Task System (MTS).Documentaţie 1980.

MC81 J.Misra,K.M.Chandy: "Proof of Network of Processes", TSE, Vol.SE-7,No.4,July 1981, pag.417-426.

MD74 S.E.Madnik, J.J.Donovan: "Operating Systems", Mc Graw - Hill Book Company, NY, 1974.

MM79 G.Milne,R.Milner: "Concurrent Processes and Their Syntax", JACM,Vol.26,No.2,April 1979,pag.302-321.

Mu68 J.E.Murphy: "Resource Allocation with Interlock Detection in a Multi-task System", Fall Joint Computer Conference, 1968, pag.1169-1176.

Mu78a K.Mühlemann: "Communication Between Processors without Shared Memory", Joint Meeting ACM/IEEE, Nov.78, Preprint, pag.3-33.

Mu78b K.Mühlemann: "An Approach to Distributed Systems Design", Euromicro-78, NHPC, pag.48-55.

Ne81a G.Neumann: "An Operating System for a Microcomputer Network", Performance of Data Communication Systems and their Applications, NHPC, 1981.

Ne81b D.M.Nessett: "Identifier Protection in a Distributed Operating System", Operating System ReView ,Vol.16,No.1, January 1982, pag.26-31.

Ni77 E.Nicolau: "Analogie, Modelare, Simulare, Cibernetică", Ed. ştiinţifică şi enciclopedică, 1977.

NP70 E.Nicolau,A.Popovici: "Algoritmi. Automate finite. Calculatoare Electronice". Ed.Stiinţifică, Bucureşti 1970.

NP78 S.I.Novatchenko,V.A.Pavlov,E.I.Jurevich: "Specialized Modular Software System of Sensitized Robot Control Computer",International Symposium on Theory and Practice of Robots and Manipulators, Italy,1978,Preprint,pag.1-16.

Nu77 G.J.Nutt: "A Parallel Processor Operating System Comparison", TSE,Vol.SE-3,No.6,Nov.1977,pag.467-475.

Pa80 R.L.Patrick: "A Checklist for System Design", Datamation January 1980, pag.147-152.

Pa82 F.Paunescu: "Analiza şi concepţia sistemelor de operare", Ed.Stiinţifică şi Enciclopedică, 1982.

PC79 M.Petru,V.Creţu,A.Furtunescu,E.Văcărescu: "Sistemele de operare timp-real-sisteme orientate pe gestionarea taskurilor", BSIITVT,Tom 24(38),Fasc 1 ian-iun,1979,pag.78-86.

PC80 D.Petriu,V.Creţu: "Studiu comparativ asupra limbajelor de programare concurente pentru calculatoare de proces", BSIITVT,Tom 25(39),Fasc 1 ian-iun·1980,pag.175-186.

Pe81 D.Petriu: "Sisteme de operare pentru microprocesoare", Referat doctorat, 1981.

PG77 M.Petrescu,C.Giumale,P.Dumitru,T.Popescu: "Aspecte constructive privind realizarea unui terminal inteligent", Actualitatea în informatică, Ed.Dacia 1977.

PK81 F.J.Pollack,K.K.Kahn,R.M.Wilkinson: "The iMAX-432 Object Filing System",P. ofthe 8-th SOSP, SIGOPS,Vol.15, No.5,Dec.1981,pag.157-147.

PM84 A.Petrescu,ş.a: "Microcalculatoarele FELIX M18, M18B şi M118" Ed.Tehnică, Bucureşti 1984.

PN79 K.V.S.Prasad,N.Natarjan,M.K.Sinha: "Phisical and Logical Abstractions in a Kernel", Operating Systems Theory and Practice, NHPC,1979,pag.359-368.

Po81a V.Pop: "Arhitectura sistemelor multiprocesor", Lucrările Colocviului de cibernetică,Litografia IPTVT,Noe.1981, pag.17-27.

Po81 V.Pop: "Structura sistemelor de prelucrarea datelor numerice" Curs,Pl,Litografia IPTVT,1981.

Po84 N.Popescu: "Sisteme informatice cu funcţionare în timp-real", Ed.militară, Bucureşti, 1984.

PP83a E.Petriu,D.Petriu,V.Creţu: "Sistem de control multiprocesor pentru un robot experimental cu elemente elastice",Vol. "Cibernetica în slujba dezvoltării economico-sociale a ţării", Ed.Academiei,1983,pag.240-247.

PP83b E.Petriu,D.Petriu,V.Creţu: "Sisteme de comandă şi control pentru roboţi industriali în mod interactiv", Conf.Naţ. de Electron,Telecom,Autom şi Calc,Vol Secţia 13-15, Noem.1983,pag.227-235.

PS77 E.Pop,V.Stoica: "Principii şi metode de măsurare numerică" Ed.Facla 1977.

PS83 E.Pop,V.Stoica,I.Naforniţă,E.Petriu: "Tehnici moderne de măsurare", Ed.Facla 1983.

RF76 C.V.Ramamoorthy,T.F.Fox , Hon F.Li: "Scheduling Parallel Processable Tasks for a Uniprocesor", TC,Vol.C-25,No.5, May 1976, pag.485-495.

RG79 J.E.Rodriguez,S.J.Greenspan: "Directed Flowgraphs: The Basis of a Specification and Construction Methodology for Real-Time Systems", The Journal of Systems and Software,Vol.1,Number 1,1979,pag.19-27.

RK79 D.P.Reed,R.K.Kanodia: "Synchronization with Event Counts and Sequencers",CACM,Vol.22,No.2,Feb.1979,pag.115.

RL81 J.Ruttner,W.W.Latin: "ADA Determine Architecture of 32-Bit Microprocessor", Electronics, Feb.24,1981,pag.119-126.

RR81 R.F.Rashid,G.G.Robertson: "ACCENT: A Communication Oriente Network Operating System Kernel" P.of the 8-th SOSP, SIGOPS, Vol.15,No.5,Dec.1981,pag. 64-75.

Ro73 A.Rogojan: "Calculatoare numerice", Curs. Vol.1, Litografia IPTVT, 1973.

Ro74 A.Rogojan: "Calculatoare numerice", Curs. Vol.2, Partea 1. Partea 2, Litografia IPTVT, 1974.

Ro79 J.T.Robinson: "Some Analysis Techniques for Asynchrnous Multiprocessor Algorithms", TSE,Vol.SE-5,No.1,Jan.1979, pag.24-31.

RT74 D.M.Ritchie, K.Thompson: "The UNIX Operating System", CACM Vol.17, July 1974, pag. 365-375.

RS79 G.S.Rao, H.S.Stone, T.C.Hu: "Assignement of tasks in a Distributed Processor System with Limited Memory", TC, Vol.C-28,No.4, April 1979, pag.291-299.

Ru79 T.Rus: "Data Structures and Operating Systems", Ed.Academiei, John Wiley & Sons, 1979.

SA80 K.Shimizu, E.Aiyoshi: "Hierarhical Multi-Objective Decision Systems and Power-Descentralized Systems for General Resource Allocation Problem", KEIØ, Engineering Report Vol.33, No.2, 1980, pag. 13-29.

SB68 S.Stimler, K.A.Bions : "A Methodology for Calculating and Optimizing Real-Time System Performance", CACM, Vol.11 No.7, July 1968, pag. 509-516.

SB77 R.J.Swan, A.Bechtolsheim, K.W.Lai, J.K.Ousterhout: "The Implementation of the Cm * Multi-Microprocessor", AFIPS, Conference Proceedings, Montvale 1977, pag. 645-655.

So78 R.Schild: "Parallel Processes in Portal, Exemplified in a Group Project", Landis & Gyr, Reoiew, 2-78, pag.9-17.

So79 H.A.Schutz: "On the Design of a Language for Programming Real-Time Concurrent Processes", TSE,Vol.SE-5,No.3, March.1979, pag.248-255.

SF77 R.J.Swan,S.H.Fuller, D.P.Siewiorek: "Cm * - A Modular, Multi-Microprocessor", AFIPS, Conference Proceedings, Vol.46, Montvale 1977, pag. 637-644.

SG77 H.A.Seidel, G.Grebe: "A Kernel for a Concurrent Pascal Operating System", Operating System Theory and Practice, NHPC, 1977, pag. 333-344.

Sh73 D.N.Shorter: "Software Developement for Real-Time Minicomputer Applications", Minicomputer in Instrumentation and Control, 1973, pag. 81-114.

Sh74 A.C.Shaw: "The Logical Design of an Operating Systems", Prentice-Hall, INC, Englewood Cliffs, N.J.1974.

Sh77 K.Shimizu: "Two Level Planning for Multi-Objective Systems, KEIØ, Engineering Reports,Vol.30,No.8,1977, pag.69-83.

SK77 A.Silberschatz, R.Kieburtz, A.Bernstein: "Extending Concurrent Pascal to Allow Dinamic Resource Management", TSE,Vol.SE-3, No.3, May 1977, pag.210-217.

SL80 R.Schild, H.Lienhard: "Real-Time Programming in Portal", CACM, Sigplan Notices, Vol.15,No.4,April 1980,pag.79-93.

SM78 H.J.Siegel, P.T.Mueller, H.E.Smalley: "Control of a Partitionable Multimicroprocessor System", Proceedings of the 1978 International Conference on Parallel Processing, Aug.1978,pag.9-17.

St77 H.S.Stone: "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", TSE,VOL.SE-3,No.1,Jan.1977, pag.85-93.

St79 C.Strugaru: "Echipamente periferice şi transmiterea datelor" Curs. Litografia IPTVT, 1979.

St80 M.J.Stefik: "Planning with Constraint", teză de doctorat, Standford University, January 1980.

Su78 R.Suri: "Resource Management in Large Systems", Tehnical Report No.671, Harward University, Massachusetts, Dec. 1978.

Ta80 C.J.Tavora: "A Basic Technique for Real-Time System Design", Computer Design, Oct.1980, pag.147-152.

TC78 D.Towsley, K.M.Chandy, J.C.Browne: "Models for Parallel Processing Within Programs: Application to CPU: I/Ø and I/Ø Overlap", CACM, Vol.21,No.10,Oct.1978,pag.821-831.

Th79a K.J.Thurber: "Parallel Processor Architectures - Part 1: General Purpose Systems", Computer Design, Jan.1979, pag. 89-97.

Th79b K.J.Thurber: "Parallel Processor Architectures - Part 2: Special Purpose Systems", Computer Design, Feb.1979, pag.103-114.

TM81 A.S.Tanenbaum, S.Mullender: "An Overview of the AMOEBA Distributed Operating System", Operating Systems Review, Vol.15,No.3, July 1981, pag.51-64.

To77 D.A.Townsen: "A Task Scheduling Executive Program for Microcomputer Systems", Computer Design, June 1977.

TW82 R.Taylor, P.Wilson: "Process Oriented Language Meets Demands of Distributed Processing", Electronics/Nov.30, 1982, pag. 89-95.

VB78 I.Văduva, V.Baltac, V.Florescu, I.Floricică: "Programare structurată", Ed.Tehnică, 1978.

Ve75 W.A.Vervoort: "Concurrent Pascal and the Design of a Timesharing Operating System", Proceedings of the DEC Users Society, Sept 1975, pag.215-220.

We77a A.J.Weissberger: "Analysis of Multiple-Microprocessor System Architecture", Computer Design, Vol.16, No.6, June 1977 pag,151-163.

We77b H.Wettstein: "The Implementation of Synchronizing Operations in Various Environments", SPE, Vol.7,1977,pag.115-126

We82 T.C.Wesselkamper: "Computer Program Schemata and the Processes They Generate", TSE, Vol.SE-8,No4,July 1982, pag.412-418.

Wi76 N.Wirth: "Algorithms + Data Structures = Programs", Prentice Hall, INC, Englewood Cliffs, N.J.1976.

Wi77a N.Wirth: "Toward a Discipline of Real-Time Programming", CACM, Vol.20,No.8, Aug.1977. pag.577-583.

Wi77b N.Wirth: "MODULA: A Language for Modular Multiprogramming" SPE, Vol.7, 1977, pag.3-35.

Wi82 R.M.Weatherly, J.F.Leathrum: "Efficient Semaforo Management Using Read/Modify/Write Memory Cycles", Operating Sistems Review, Vol.16,No.1, Jan.1982,pag.10-15.

Wu74 W.A.Wulf,p.a: "HYDRA: The Kernel of A Multiprocessor Operating System", CACM,Vol.17,No.6,June 1974,pag. 337-345.

Zi80 C.P.Zing: "Devolopement System Put Two Processors on Speaking Terms",Electronics, July 31,1980,pag.93-97.

xxFE xxxx "FELIX M-18. Sistemul de operare cu discuri flexibile SFDX-18" ICE. Bucureşti.

xxIM xxxx "Self Contained System. Operating System. Text Editor. Assembler.

xxIS xxxx "ISIS-II User's Guide". Manual INTEL.

xx74 xxxx "Introduction to RSX-11M", DEC,Maynard, 1974.

xx76 xxxx 8080 Programming for Logic Design", Adams Osborne and Associates Incorporated. 1976.

xx79 xxxx "CP/M", Digital Research, 1979.

xx80 xxxx "Sistem de operare în timp-real pentru microprocesoare. RTOS-80. Manual de prezentare", ICI,Bucureşti,1980.

xx81 xxx "Introduction to the iAPX 432-Architecture", Intel Corporation, 1981.

xx82 xxx "Sistem de operare în timp-real pentru microcalculatoare.RTØS-80. Executiv. Manual de utilizare. I.C.I. Bucureşti,1982.

xx83 xxx "Program privind îmbunătăţirea nivelului tehnic şi calitativ al produselor, reducerea consumurilor de materii prime, de combustibili şi energie şi valorificarea superioară a materialelor în perioada 1983-85 şi pînă în 1990", Scînteia,30 Noe.1983, pag.4-5.

L I S T A

prescurtărilor utilizate în lucrare

I/E	- intrare/ieşire
IT	- întrerupere
MIMD	- instrucţii multiple - date multiple
MP	- mediu de programare
m/uP	- multi microprocesor
multi/uP	- multi microprocesor
PHS	- protecţie hard-software
SIMD	- o singură instrucţie - date multiple
SØ	- sisteme de operare
SØTR	- sisteme de operare timp-real
STR	- sisteme timp-real
TC	- timp curent
TE	- timp de execuţie
TR	- timp-real
TR_e	- timp restant
TT	- timp total (de trecere)
UC	- unitate centrală
/uP	- microprocesor