MINISTERUL EDUCATIEI SI INVATAMINTULUI
INSTITUTUL POLITEHNIC "TRAIAN VUIA" TIMISOARA
FACULTATEA DE ELECTROTEHNICA

ing. Jian Ionel

CONTRIBUTII PRIVIND SISTEMELE DE CALCUL CU MULTIACCES

- teză de doctorat -

Conducător ştiinţific
Prof.dr.ing.Alexandru Rogojan

Prof.dr.ing.Mircea Petrescu

- Timişoara -
1984

C U P R I N S

1. INTRODUCERE

Dezvoltarea tehnicii de calcul în ultimele două decenii, ca urmare a inovaţiilor tehnologice apărute, s-a făcut după o exponenţială cu o dublare a performanţelor, la acelaşi preţ de cost, la fiecare doi ani. S-a ajuns astfel ca performanţele celor mai mari calculatoare ale anului 1965 să fie depăşite de sisteme microprocesoare din clasa 32 biţi, ca IAPX 432 produs de firma INTEL [Cox 83] [May 82][Bel 84]. Arhitectura sistemelor de calcul a cunoscut modificări substanţiale în mai multe etape, care au permis o explotare intensivă a performanţelor unităţii centrale, memoriilor externe de mare capacitate şi periferiicelor. Odată cu creşterea complexităţii echipamentelor a crescut şi fiabilitatea generală a sistemelor de calcul.

In paralel şi direct legat de modificările de arhitectură şi performanţe ale calculatoarelor, s-au dezvoltat sistemele de operare, care au devenit componente inseparabile ale sistemelor de calcul, cu o pondere în costul total de concepţie ce depăşeşte astăzi 80 %. Noi modificări de structură ale calculatoarelor au fost impuse pentru implementarea unor concepţii noi apărute în proiectarea sistemelor de operare : protecţia memoriei centrale pe pagini, instrucţii privilegiate şi regim supervizor, canale multiplexare cu program memorat, sistem de întreruperi, divizarea timpului unităţii centrale între utilizatori, principiul memoriei virtuale, etc.

Sistemele de operare sînt colecţii de programe care răspund de alocarea şi monitorizarea utilizării echipamentelor şi programelor dintr-un sistem de calcul. De performanţele lor depinde productivitatea şi flexibilitatea întregului sistem. Sistemele de operare se pot clasifica în patru clase cu trăsături specifice :

- multiprogramare ;
- multiacces ("time-sharing");
- cu memorie virtuală ;
- cu resurse distribuite.

Multiprogramarea presupune divizarea memoriei centrale, în care se încarcă la un moment dat mai multe programe, (< 15) ce lucrează aparent simultan, împărţind între ele timpul de unitate centrală şi perifericele disponibile.

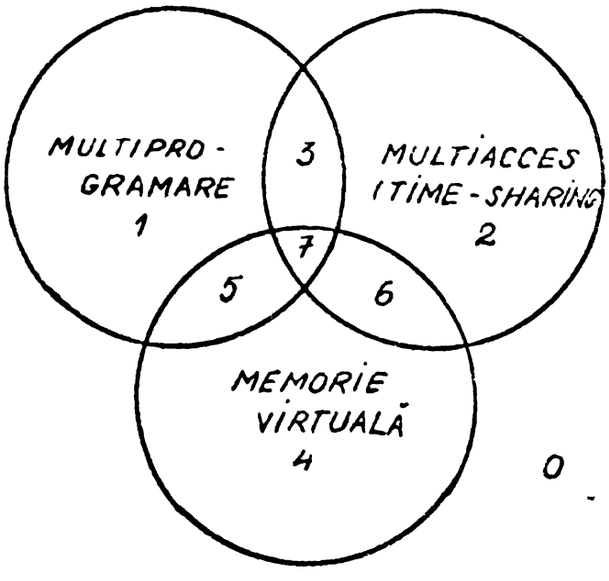
Multiacesul permite accesul conversaţional simultan şi de la distanţă a unui număr mare de utilizatori, (< 64) care folosesc terminale de teletransmisie. Ei utilizează pentru rularea programelor

acceaşi zonă de memorie centrală în care se încarcă pe rînd de pe un fişier disc programele care devin active un timp dat.

Memoria virtuală presupune existenţa unei ierarhii de memorii cu proprietăţi şi viteze diferite (pe ferite, pe tambur, pe discuri), pentru care se asigură prin "hardware" şi "software" o gestiune automată, astfel încît utilizatorul vede o singură memorie logică de capacitate mult mai mare.

Sistemele de operare cu resurse distribuite [Dav 83] au apărut în ultimii ani, legat de crearea reţelelor de calculatoare. Ele presupun distribuirea puterii de calcul şi de memorare în nodurile unei reţele pe calculatoare de mărimi, tipuri şi performanţe diferite. Orice utilizator legat la reţea trebuie să aibă acces controlat la toate resursele reţelei, simultan cu alţi utilizatori [Kum 83] [Kis 83] [Nof 81] [Dav 81].

Sistemele de operare moderne conţin trăsături caracteristice diferitelor clase prezentate. Luînd intersecţiile primelor trei clase obţinem 8 tipuri de sisteme de operare (fig. 1), cărora le corespund sisteme concrete comercializate în decursul anilor, prezentate în tabelul de mai jos.



0. Sisteme cu monoprogramare pentru microcalculatoare (SFDX, CP/M).

1. Sisteme moderne cu prelucrare în loturi (IBM OS/360/370 MVT, MULTIX 256/512 SIRIS-3).

2. Sisteme singulare (MIT/CTSS).

3. Cele mai multe sisteme cu multiacces (IBM OS/360 TSO, IBM OS/360 APL, MULTIX C 256/512 ARIEL).

4. Rar există asemenea sisteme deoarece sistemele cu memorie virtuală au şi multiprogramare.

5. IBM/370 VS lansat în 1972.

6. Există rar sisteme cu memorie virtuală fără multiprogramare (ministeme).

7. Sistem mari, GE/MIT MULTICS, IBM 360/67 TSS, IBM CP-67, IBM/370 VM.

Fig. 1. Clase de sisteme de operare şi combinaţiile lor.

Tipul 2, 4 şi 6 de sisteme de operare sînt rar întîlnite, deoarece sistemele cu memorie virtuală lucrează întotdeauna în regim de multiprogramare, iar sisteme care să lucreze numai în multiacces nu se proiectează. Multiaccesul este în general o facilitate suplimentară a sistemelor cu multiprogramare, care cunoaşte o largă extindere la sistemele cu memorie virtuală, unde problemele specifice ale acestui mod de lucru se pot rezolva în condiţii optime. Se confundă uneori în literatură [Dan 74] din motive comerciale, sistemele multi-user pentru minicalculatoare (UNIX, RSX-11) cu sistemele cu multiacces (time-sharing). Sistemele multi-user, conform clasificării date, sînt sisteme cu multiprogramare, care dispun de un limbaj de comandă conversaţional. Ele se implementează pe minicalculatoare moderne care pot să aibă anumite facilităţi de memorie virtuală şi pot accepta mai multe console de comandă (4 - 14).

Se foloseşte deseamenea în literatură [Mel 75] abuziv termenul de "time-sharing" (divizarea timpului) pentru a desemna sistemele cu multiacces. In cazul multiaccesului se partajează între programe mai multe resurse (memorie centrală, memorie auxiliară, periferice, compilatoare şi alte module ale subsistemului). Unitatea centrală (UC) este numai una din aceste resurse, iar divizarea timpului U C poate apare şi în multiprogramarea simplă la unele sisteme de operare.

Sistemele de operare distribuite ocupă o arie care se suprapune peste toate tipurile de sisteme prezentate în fig.1. Ele se implementează pe calculatoare foarte mari [Dav 81] [Len 84] [Lic 84] , care formează scheletul unei reţele de calculatoare, având componente de interfaţă pe fiecare calculator din celelalte noduri ale reţelei. Calculatoarele pot fi de tipuri şi capacităţi diferite lucrînd sub sisteme de operare locale din tipurile prezentate.

In lucrarea de faţă se face o analiză generală a sistemelor de operare cu multiacces, care prezintă o mare varietate de forme de

implementare. Se propune o clasificare sistematică a produselor "software" destinate multiacccesului, indicîndu-se principiile de realizare, performanţele obţinute, limitările existente şi domeniile de aplicabilitate. Se acordă o atenţie mai mare sistemelor cu multiprogramare şi multiacces care sînt larg răspîndite la calculatoarele de capacitate medie.

In cap.1. se analizează pe scurt, în prima parte (1.1) caracteristicile fundamentale ale sistemelor de operare cu multiprogramare, deoarece pe această structură de bază se adaugă noi componente, care vor realiza facilităţile de multiacces. Limitările în această direcţie vor fi determinate de restricţii introduse de sistemele cu multiprogramare concepute să asigure o exploatare optimă a resurselor la prelucrarea în loturi (batch) . In partea a doua (1.2 se analizează moduri de implementare a multiaccesului prin subsisteme specializate, care extind posibilităţile limbajului de comandă pentru lucrul conversaţional. După funcţiile realizate aceste subsisteme sînt clasificate astfel :

1). Subsisteme bazate pe limbaje de programare conversaţionale, care utilizează interpreter pentru traducerea programului sursă ;

2). Subsisteme bazate pe limbaje conversaţionale pentru gestiunea bazelor de date, care utilizează un interpreter de comenzi pentru selectarea unor proceduri specializate ;

3). Subsisteme pentru testarea conversaţională a programelor, prevăzute cu editor de texte şi comenzi de lansare în execuţie într-o partiţie serie a unor lucrări introduse de la terminal ;

4). Subsisteme pentru execuţia interactivă a programelor compilate şi depuse în bibliotecă ;

5). Subsisteme complexe cu multiacces, care realizează combinat funcţii ale diferitelor tipuri de subsisteme (IBM/360 TSO).

Notăm cu C lucrul conversaţional şi cu B lucrul în loturi (batch) şi considerînd principalele operaţii executate asupra programelor :

E D - editare program sursă (creare, modificare, listare
C O - compilare program sursă cu generare program obiect
E X I - execuţie interpretativă a programelor sursă;
E X - execuţie programe compilate.

Cu aceste notaţii în tabelul de mai jos se prezintă concis operaţiile pe care le poate efectua fiecare tip de subsistem şi în ce mod se execută operaţia respectivă, conversaţional sau "batch".

Contribuţii personale substanţiale ale lucrării sînt în domeniul subsistemelor interactive pentru execuţia programelor.

ED - C EXI- C	Subsisteme cu limbaje conversaţionale Subsisteme conversaţionale pentru baze de date
ED - C CO - B EX - B (EXI-C)	Subsisteme pentru testare conversaţională de programe Subsistemul IBM/360 T S O
EX - C	Subsisteme pentru execuţia interactivă a programelor
ED - C CO - C EX - C	Sisteme de operare cu multiacces generalizat

In cap.2. se prezintă pe scurt metode de realizare a unor programe cu multiacces sau subsisteme specializate cu multiacces pentru anumite aplicaţii. Se analizează ce facilităţi trebuie să ofere sistemul de operare pentru a se implementa ulterior sisteme cu multiacces performante.

Cap.3. se ocupă de sistemele de operare cu multiacces generalizat, implementate pe calculatoare mari, care lucrează cu memorie virtuală şi dispun de mecanisme sofisticate hardware şi software pentru realizarea unor protecţii sigure. Se prezintă principiile clasice de paginare şi segmentare a programelor în memoria virtuală şi protecţia prin inele pentru ierarhizarea proceselor (MULTICS). Se analizează anumite metode moderne de gestionare a resurselor şi protecţie ierarhică folosind noţiunea de capabilitate. O parte din aceste mecanisme şi principii aplicate la calculatoarele mari, pot fi extinse la calculatoarele de capacitate medie. Aceasta ar duce la posibilităţi noi de implementare a unor subsisteme cu multiacces cu performanţe superioare, fără a complica prea mult structura calculatoarelor şi costul fabricaţiei (2.3.)

Cap.4 prezintă principii noi şi eficiente de implementare a unui subsistem cu multiacces pentru execuţia interactivă a programelor, propuse de autor şi experimentate cu bune rezultate pe calculatoarele de capacitate medie FELIX 256/512, sub sistemul de operare cu multiprogramare SIRIS-3 V16. Contribuţiile personale ale autorului sînt :

- concepţia originală globală a subsistemului;
- modul de dialog în multiacces cu utilizatorii de la terminale;
- soluţia de încărcare dinamică a programelor în format IMT;
- modul de tratare a informaţiilor S G F necesare programului;
- modul de transmitere eficientă spre programul utilizator a parametrilor S G F păstraţi în tabele ale subsistemului;
- modul de rezolvare a încărcării programelor segmentate;
- modul de salvare şi reîncărcare a programelor aflate în lucru, în perioadele inactive;
- modul unitar de tratare a erorilor de diferite tipuri şi asigurarea fiabilităţii sistemului şi a protecţiei între programele utilizate care se execută în acelaşi timp;
- se propun soluţii eficiente de divizare a timpului între utilizatori, care asigură un timp de răspuns acceptabil şi o reducere a transferurilor de programe între memoria centrală şi discul magnetic;
- soluţii originale simple de comunicare între programele utilizator şi subsistem;
- modul de modificare rapidă a programelor testate în loturi care urmează să ruleze în multiacces;
- compatibilitatea completă la nivel de program sursă între lucrul în loturi şi conversaţional;
- asigurarea posibilităţii unor programatori neprofesionişti să-şi scrie şi să-şi testeze singuri programele care ar putea fi executate conversaţional, utilizînd limbaje de nivel înalt (FORTRAN, COBOL);
- reducerea considerabilă a timpului de testare a programelor conversaţionale prin testarea în loturi, cu o reducere corespunzătoare a preţului de cost a proiectelor, faţă de cazul testării direct conversaţionale sau utilizării unor simulatoare de regim conversaţional;
- asigurarea flexibilităţii întregului subsistem prin posibilitatea de modificare în orice moment a programelor utilizate disponibile, fără a afecta funcţionarea programelor în lucru;
- asigurarea unui dialog simplu şi eficient cu utilizatorii;
- realizarea întregului subsistem fără nici o modificare în sistemul de operare rezident şi folosirea în exclusivitate numai a posibilităţilor normale de programare, conform manualelor de utilizare ; se conferă astfel compatibilitatea

subsistemului cu versiunile ulterioare ale sistemului de operare;
- utilizarea tuturor protecţiilor sistemului de operare prevăzute pentru prelucrarea în loturi, care oferă o siguranţă mare în exploatarea programelor şi fişierelor ;
- asigurarea compatibilităţii complete cu fişierele utilizate în prelucrarea în loturi ;
- utilizarea eficientă a spaţiului de memorie centrală (subsistemul ocupă 20K).

Subsistemul realizat (SCOT) este în stare de funcţionare din 1979 şi a demonstrat valabilitatea soluţiilor utilizate la proiectarea sa, care pot fi folosite şi pentru alte sisteme de operare. Performanţele obţinute sînt foarte bune. Subsistemul a fost realizat în cadrul contractelor de cercetare cu Institutul Central pentru Conducere şi Informatică, Întreprinderile Electrotimiş şi IAMI Timişoara. Se exploatează curent din 1980 în Institutul Politehnic Timişoara, Întreprinderea Electrotimiş Timişoara, Întreprinderea Optica Română - Bucureşti, Centrul de Calcul al MA. Bucureşti şi alte centre de calcul din ţară.

Prin punerea în funcţiune a subsistemului SCOT (subsistem conversaţional de teleprelucrare), se completează gama produselor software destinate teleprelucrării în multiacces pentru calculatoarele FELIX 256/512/1024. Alături de subsistemul AMBL pentru testarea conversaţională a programelor şi subsistemul SCOMAR pentru gestiunea conversaţională a bazelor de date, subsistemul SCOT va contribui la extinderea teleprelucrării în informatica românească. Teleprelucrarea datelor constituie o etapă superioară în exploatarea sistemelor de calcul, care cunoaşte o extindere rapidă în ţara noastră ca urmare a diversificării echipamentelor de teletransmisie, a creşterii performanţelor, fiabilităţii şi a reducerii preţului lor de fabricaţie. Avînd în vedere importanţa teleprelucrării în creşterea eficienţei sistemelor informatice din economie, documentele de partid şi de stat prevăd sarcini de extindere largă a acestor sisteme pînă în 1990 în întreaga economie naţională. Subsistemul SCOT este o unealtă eficace la îndemîna informaticienilor pentru a dezvolta rapid programe de teleprelucrare pentru cele mai diverse tipuri de aplicaţii.

Pentru experimentarea diferitelor tipuri de echipamente de teleprelucrare şi pentru dezvoltarea unor noi produse program pentru aceste aplicaţii, autorul a realizat o reţea de teletransmisie a datelor, în cadrul Institutului Politehnic din Timişoara, care cuprinde Centrul de Calcul, Facultatea de Electrotehnică,

Facultatea de Mecanică, Facultatea de Construcţii şi Atelierul Şcolă şi de Prototipuri. Sînt prevăzute legături telefonice speciale în laboratoarele şi atelierele de proiectare ale catedrelor. In prezent sînt în funcţiune 16 terminale lucrînd în mod caracter prin subsistemele ARIEL şi SCOT, două concentratoare de date (tip TELEROM-3V1) lucrînd în mod mesaj cu 6 terminale fiecare şi un microcalculator satelit (TELEROM P). Există posibilitatea de extindere a sistemului cu încă 50 de terminale prin completarea cuplcarelor de teletransmisie. Se asigură astfel mijloace moderne de acces la calculator pentru cadre didactice şi studenţi, contribuind în acest fel la ridicarea calităţii procesului de instruire a studenţilor şi la creşterea eficienţei cercetării ştiinţifice.

x

x x

Autorul aduce un cald omagiu prof.dr.ing. Alexandru Rogojan, personalitate marcantă a şcolii româneşti de calculatoare, sub a cărui îndrumare competentă s-a format ca specialist, care prin sfaturile şi sprijinul acordat permanent a contribuit la finalizarea lucrărilor teoretice şi practice cuprinse în prezenta teză de doctorat.

Deasemenea autorul mulţumeşte prof.dr.ing. Eugen Pop pentru sprijinul acordat la finalizarea tezei şi pentru realizarea reţelei de telepreluorare, pe care s-a experimentat subsistemul elaborat.

Autorul aduce mulţumiri colegilor mat. Mihai Petru şi ş.l. ing. Ştefan Holban pentru colaborarea la realizarea practică a subsistemului şi pentru ideile sugerate la soluţionarea unor probleme de implementare.

Pe această cale autorul ţine să mulţumească tuturor colegilor Catedrei de Automatică şi Calculatoare şi în mod special ş.l. dr. ing. Ioan Jurca pentru sprijinul documentar, pentru discuţiile purtate, observaţiile făcute şi încurajările primite.

1. METODE DE ORGANIZARE A ACCESULUI LA SISTEMELE DE CALCUL DE CAPACITATE MEDIE

1.1. SISTEME DE OPERARE CU MULTIPROGRAMARE

1.1.1. Principiul multiprogramării

Sistemele de operare cu multiprogramare sînt caracteristice calculatoarelor de capacitate medie din generaţia III. Complexitatea structurii acestor sisteme de calcul a crescut considerabil, odată cu performanţele unităţii centrale (U C), memoriei centrale (M C) şi memoriilor auxiliare (discuri şi benzi magnetice). S-au diversificat considerabil echipamentele periferice folosite pentru introducerea şi extragerea datelor. Pentru a satisface unitar şi simultan cererile de transfer de date ale diferitelor periferice cu memoria centrală s-au introdus canale multiplexoare de mare viteză (unităţi de schimb multiplu) care sînt legate direct la M C şi lucrează în paralel cu U C, ca procesoare de I/E cu programe şi comenzi proprii. Se poate realiza astfel o mai bună încărcare a U C care lucrează cu date numai din M C, a cărei debit este corelat cu cel al U C. Pentru sincronizarea lucrului pe U C cu transferurile făcute prin canalele de I/E s-au introdus sistemele de întreruperi controlate prin programele de sistem.

Pentru a se asigura protecţia sistemului de operare rezident faţă de programele utilizator şi a diferitelor programe, încărcate simultan în M C, între ele s-a introdus protecţia memoriei prin chei de protecţie. O cheie de protecţie (C P), în general de 4 biţi, este asociată fiecărei pagini de memorie. Toate paginile aparţinînd aceluiaşi program au aceeaşi cheie de protecţie, care corespunde cu cheia de acces (CA) a programului, încărcată în registrul de stare program în momentul rulării. Instrucţiile programului au acces la datele din paginile care au C P = C A, condiţie verificată prin hard şi sancţionată în caz de încălcare prin întrerupere de violare de protecţie memorie. Sistemul de operare lucrează cu C A = 0, care îi asigură accesul în orice

zonă de memorie.

Pentru protecţia transferurilor de date solicitate din diferite programe, instrucţiile referitoare la periferice sînt instrucţii privilegiate şi pot fi utilizate numai de supervisor, care are în registrul de stare program un bit care indică acest regim. Tot privilegiate sînt şi instrucţiile de încărcare a registrului de stare program, de încărcare a cheilor de protecţie şi de control a sistemului de întreruperi.

Pentru a se putea încărca programele obiect începînd de la orice adresă de memorie, unde există spaţiu liber, s-au introdus registrele de bază. La compilare se calculează adresele relativ la începutul programului. La încărcarea programului se încarcă în registrul de bază adresa de început a programului, care va fi adunată la adresa relativă din instrucţie la orice adresare a memoriei. Se realizează astfel translatarea automată a programelor.

Existenţa canalelor multiplexoare care lucrează în paralel cu U C, sistemul de întreruperi, protecţia memoriei prin chei de protecţie, existenţa instrucţiilor privilegiate şi a registrelor de bază sînt inovaţii care au permis realizarea conceptului de multiprogramare la calculatoarele din generaţia III-a.

Avînd suportul fizic menţionat mai sus, sistemele de operare (S O) cu multiprogramare urmăresc o cît mai completă încărcare a resurselor sistemului de calcul, asociată cu o productivitate ridicată (numărul de lucrări rulate în unitatea de timp). Pentru aceasta memoria centrală este decupată în partiţii (regiuni) a căror număr poate fi fix sau variabil (fig.1.1.) In fiecare partiţie se încarcă cîte un program, avînd o cheie de protecţie proprie, egală cu numărul partiţiei. Pentru o cheie de protecţie de 4 biţi pot exista maxim 16 partiţii din care cel puţin una este rezervată monitorului. S O asociază cîte o prioritate fiecărei partiţii, pentru a ordona accesul la U C.

Oricare program încărcat în M C poate fi lansat în execuţie dacă este pregătit şi nu există un program mai prioritar pregătit. Lansarea în execuţie a programelor în ordinea priorităţii este asigurată de dispecerul S O. Dacă un program prioritar este în aşteptarea terminării unui transfer de date de pe un periferic, acest program este trecut în aşteptare şi este activat următorul pregătit care are prioritatea maximă. Cînd programul mai prioritar a terminat transferul, el este reactivat şi programul temporar

activat este trecut în aşteptare.

In acest fel se asigură o bună încărcare a U C, care deserveşte aparent simultan mai multe programe (la un moment dat numai unul este activ), ce sînt încărcate în diferite partiţii în M C. Numărul de partiţii utilizate efectiv depinde de capacitatea M C şi dimensiunea programelor. Fiecare program are acces numai la spaţiul de memorie din partiţia în care este încărcat, asigurîndu-se astfel protecţia programelor între ele. Un program încărcat în memorie într-o partiţie, rămîne acolo pînă la terminarea sa, cînd va fi înlocuit de un alt program din şirul de aşteptare.

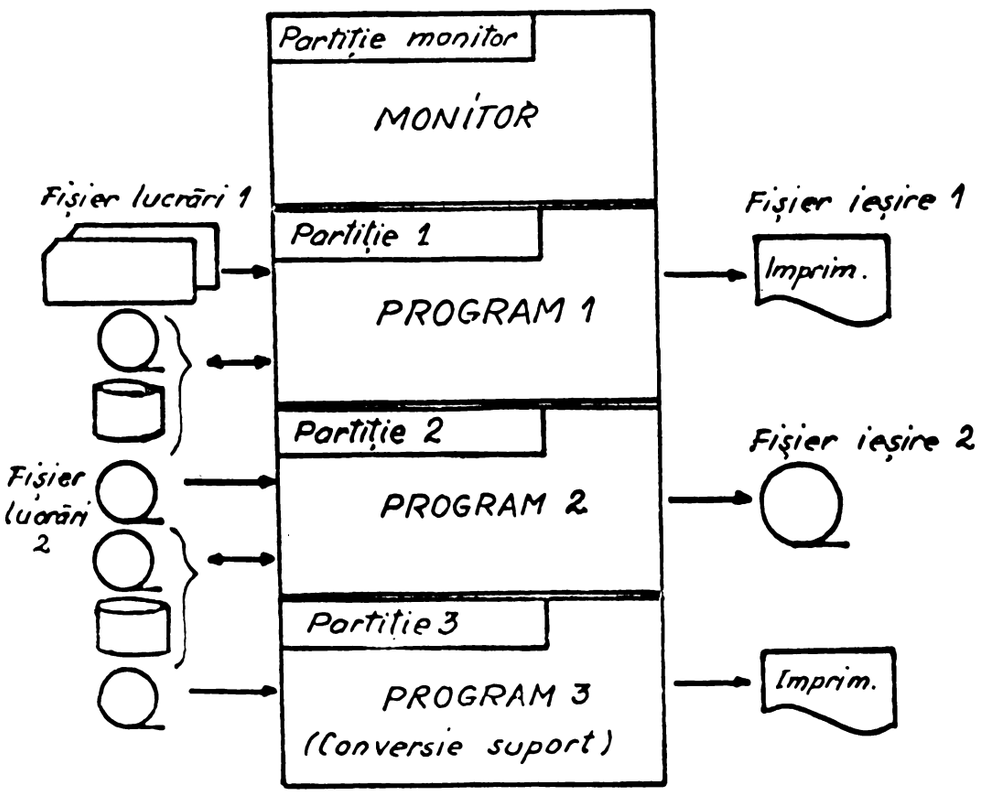


Fig.1.1. Structura memoriei centrale într-un sistem cu multiprogramare simplă.

Programele obiect gata compilate sînt translatabile şi se încarcă din biblioteci în oricare partiţie. Corecţia adresei relative din instrucţie se face în momentul execuţiei prin adunarea registrului de bază, care conţine adresa de început a programului. Există totuşi un număr de adrese absolute, din programele unităţii de schimb (canal), deoarece acestea sînt procesoare de I/E simple care nu au adresare cu bază. Aceste adrese se păstrează în bibliotecă în formă relativă şi se corectează în momentul încărcării pe baza unui tabel. Din acest motiv este dificil şi riscant ca un program încărcat în M C să fie deplasat dintr-o partiţie în alta.

Fiecărei partiţii i se asociază un fişier de lucrări la intrare (şir de aşteptare) şi un fişier de rezultate la ieşire, care pot fi pe suportul clasic (cartele respectiv imprimantă), sau pe un suport magnetic (disc sau bandă). Trecerea cartelelor de program pe un suport magnetic, respectiv a rezultatelor la imprimantă se face prin programe utilitare în partiţii speciale mai mici.

Pentru a putea lucra cu fişiere, fiecărei partiţii i se alocă după necesităţi un număr de periferice în exclusivitate sau în comun. Se asigură astfel protecţia fişierelor faţă de programele din alte partiţii, care lucrează simultan. Numărul periferice-lor disponibile poate limita numărul de partiţii posibile.

Intr-o partiţie lucrările din fişierul de intrare se execută secvenţial. Înlănţuirea este asigurată de un monitor de înlănţuire, care tratează cartelele de comandă intercalate, comandă executarea funcţiilor solicitate (compilare, editare de legături, execuţie) şi alocă perifericele necesare.

Sistemele de operare cu multiprogramare sînt orientate pe prelucrarea secvenţială (batch) a lucrărilor avînd funcţii speciale de gestionare a resurselor :

- gestionarea procesorului central,
- gestionarea memoriei centrale,
- gestionarea perifericelor,
- gestionarea şirurilor de lucrări la intrare şi ieşire,
- gestionarea fişierelor sistem de manevră.

După ce s-a prezentat aspectul cel mai simplu mod de gestionare a resurselor în S O cu multiprogramare, se vor da în continuare unele elemente mai evoluate de gestiune, existente în diferitele sisteme de calcul concrete, care măresc flexibilitatea exploatării şi conduc la o încărcare mai bună a sistemului.

1.1.2. Gestiunea memoriei centrale

Memoria centrală este o resursă vitală şi costisitoare pentru a cărei gestiune s-au adoptat în S C metode diverse, caracteristice fiind la calculatoarele din generaţia III alocarea statică şi alocarea dinamică.

Alocarea statică presupune împărţirea memoriei în partiţii de lungime fixă şi continue cuprinzînd un număr întreg de pagini (2 K), protejate prin aceeaşi cheie de protecţie. In fiecare partiţie se poate încărca un singur program, iar unitatea centrală este folosită succesiv (dar aparent simultan) de programele încărcate (vezi 1.1.1). Acest mod de alocare folosit iniţial la calculatoare din seria IBM/360 în sistemul de operare OS/360 M F T (Multiprogramming With a Fixed number of Tasks), este cel mai folosit la calculatoarele de capacitate medie din generaţia III şi cel mai simplu de implementat. Metoda permite partajarea celorlalte resurse şi scăderea timpului mediu de răspuns. Partiţiile fiind fixe şi programele de lungime variabilă, se pierde un spaţiu apreciabil de memorie ce rămîne disponibilă în fiecare partiţie. Se poate ameliora încărcarea prin ordonarea lucrărilor pe partiţii funcţie de cerinţele de memorie.

Alocarea dinamică urmăreşte o folosire mai eficientă a M C prin crearea dinamică de partiţii de lungime variabilă conform necesităţilor programelor şi eliberarea spaţiului după terminarea programului. Un exemplu tipic de alocare dinamică se face şi în OS/360 M V T (Multiprogramming with Variable Number of Tasks) [Hel75].

In OS/360 M V T se admit maxim 15 partiţii variabile, fiecare cu proprie cheie de protecţie. Lucrările din şirul de aşteptare de la intrare sînt introduse în sistem în ordine şi li se alocă cîte o partiţie de minim 52 K, pînă există spaţiu de memorie liber. Lansarea în execuţie se face ca la alocarea statică, cu deosebirea că programele în aşteptare pot fi depuse pe o memorie auxiliară pe disc şi în locul lor se încarcă alte programe de pe disc sau din şirul de intrare. La reluarea unui astfel de program el poate fi încărcat în altă partiţie (fig. 1.2). In acest fel în lucru pot exista mai multe programe decît partiţii active la un moment dat şi M C e mai bine folosită şi se reduce timpul mediu de trecere.

Totuşi alocatorul sistemului se complică şi apar probleme de relocare la o nouă încărcare pentru adresele absolute.

In urma alocării dinamice poate apare fragmentarea M C, prin apariţia unui număr mare de partiţii mici care nu permit încărcarea unor programe de dimensiuni mai mari şi care trebuie să aştepte un timp nejustificat. Se aplică în acest caz tehnici de compactare sau algoritmi "cu suspendă" [DIA 81] care interzic lansarea altor programe mici pînă nu se eliberează M C care să permită lansarea programelor mari care aşteaptă deja.

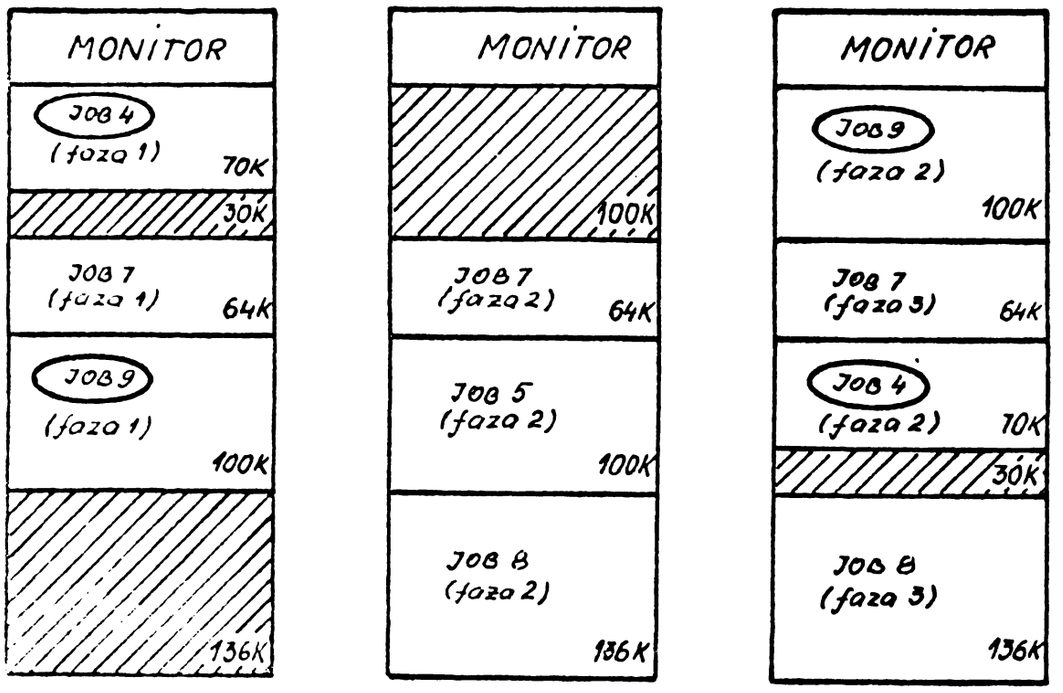


Fig. 1.2. Alocarea dinamică a memoriei centrale.

La depunerea unui program în memoria auxiliară (M A) şi la compactarea programelor în M C trebuie să se ţină seama de adresele absolute şi să se verifice dacă nu sînt operaţii de I/E în curs de execuţie. In general alocarea dinamică prin software nu este rentabilă din cauza costului ridicat în timp de U C şi a timpului de depunere şi reîncărcare (swaping) mare.

S-au propus soluţii de alocare dinamică care să reducă timpul de swaping prin depunerea şi încărcarea parţială a programelor. Algoritmul " Union Skin " (aplicat la M I T pe I B M/7094)

se poate aplica plecînd de la partiţii fixe şi în fiecare partiţie să ruleze aparent mai multe programe, care să se găsească parţial sau total în M C sau în M A. Programul activ va fi complect în M C de la începutul partiţiei. Părţile acoperite din programele în aşteptare vor fi depuse pe M A, iar cele neacoperite vor rămîne în memoria centrală. Programului activ i se va limita accesul pînă la sfîrşitul său, modificînd registrul limită C P al partiţiei. Se protejează astfel părţi din programele în aşteptare rămase în M C (fig. 1.3).

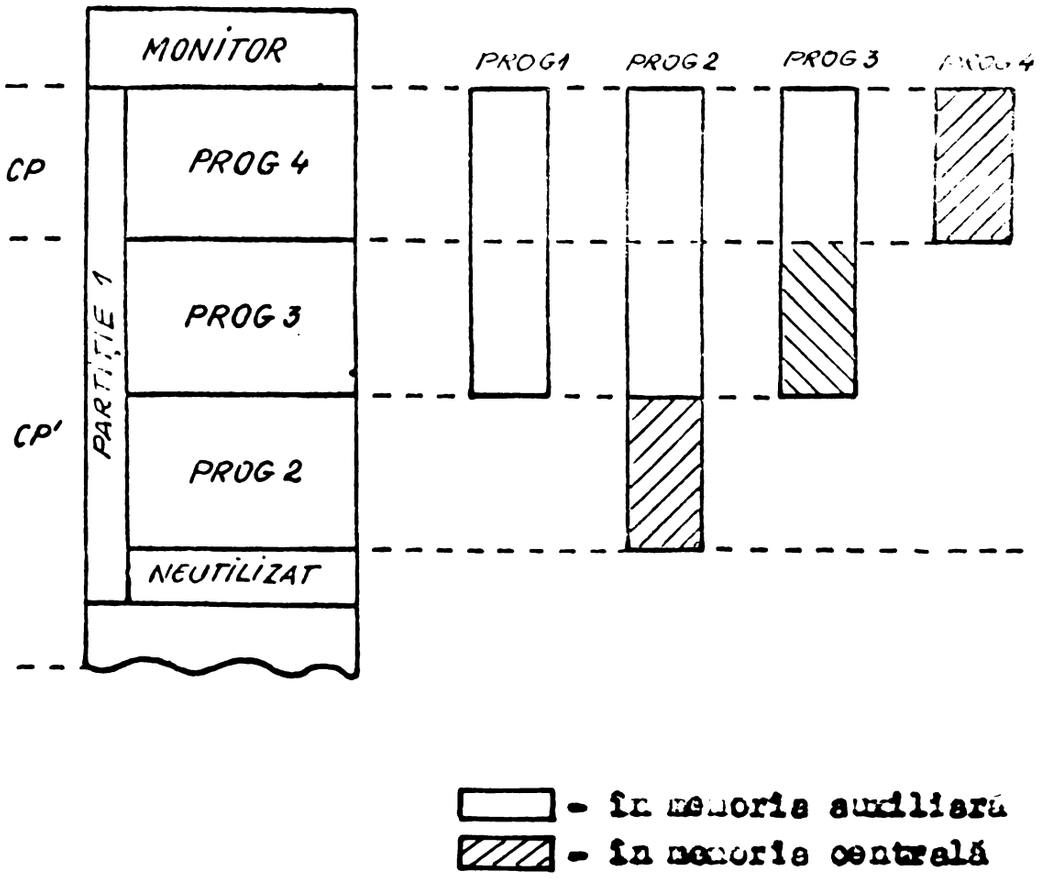


Fig. 1.3 Algoritmul "omon Khin" la alocarea dinamică.

Algoritmul suprapunerii minime ("minimum overlay") rezolvă alocarea dinamică cu swoping redus plasînd programele în M C în aşa fel încît suprapunerea lor medie să fie minimă. La activarea unui program se va depune şi încărca din M A numai partea suprapusă a programelor. Pentru aceasta, se ţine o tabelă de acoperire căreia îi corespunde un contor pentru fiecare pagină a M C. Tabelul cheilor de protecţie va indica programul ale cărui instrucţii sînt încăr-

este la un moment dat în pagină (fig. 1.4). Metoda a fost aplicată la Princeton pe un calculator C D C 1604 [Pău 82].

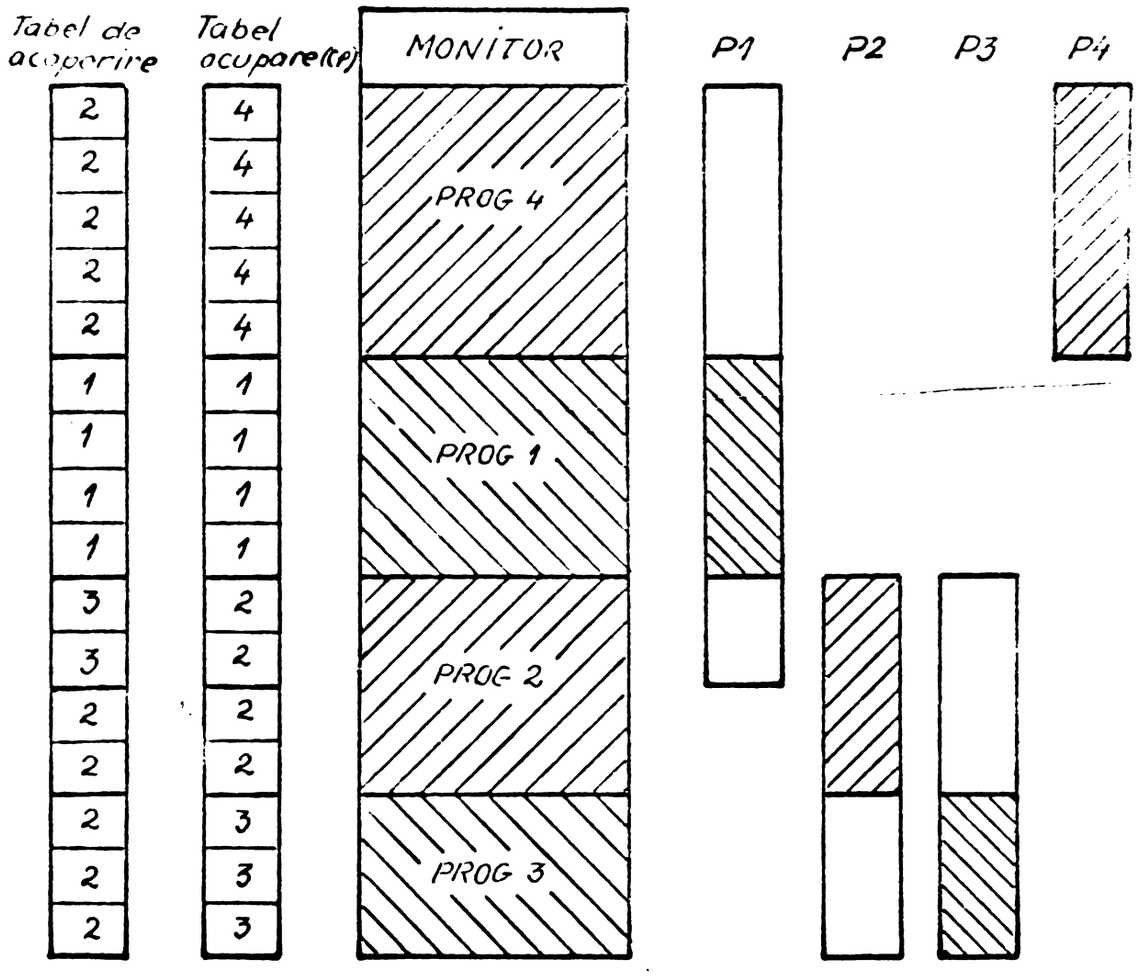


Fig. 1.4. Algoritmul suprapunerii minime.

Algoritmul acoperii întîrziate (Delayed Cover) completează algoritmul suprapunerii minime, încărcînd iniţial, parţial, mai multe programe lansîndu-le în execuţie în regim de multiprogramare. La fiecare adresare se verifică dacă pagina respectivă este în memorie, prin consultarea tabelului de ocupare. Dacă nu există, se lansează un swaping pentru acea pagină şi se lansează următorul program pregătit. Algoritmul a fost folosit pe sistemul de operare TITAN la Univ. din Cambrige.

La alocarea dinamică, pentru a evita fragmentarea excesivă a M C şi a asigura un timp mediu de trecere cît mai bun, fără compactarea zonelor, s-au propus mai mulţi algoritmi [Mad 74], [Hoe 78], [Dod 81], [Knu 74], [Pau 82], [Hir 73], [Sho 75].

Algoritmul prima potrivire F F ("firstfit") alocă unui program ce trebuie încărcat în M C prima zonă de memorie a cărei lungime LM este mai mare decît lungimea programului LP. Algoritmul e simplu dar duce la o fragmentare pronunţată.

Algoritmul cea mai bună potrivire B F ("best fit") alege din cele K zone de memorie libere cea cu lungimea L_{M_i} cea mai apropiată de lungimea programului :

$$L_{M_i} - L_P = \min_{j=1,K} (L_{M_j} - L_P)$$

Pentru optimizare se mai pot lua în considerare următoarele criterii, dacă există unele informaţii suplimentare :

- se evită alocarea zonelor care dau rezidu prea mic în raport cu lungimea medie a programelor, pentru ca golurile să nu rămînă nealocate ;

- nu se alocă zonele disponibile ale căror zone învecinate au termene de eliberare apropiate, favorizîndu-se crearea de zone libere de dimensiune mare ;

- dacă rezidul obţinut după o alocare este sub o anumită limită se alocă toată zona, pentru a nu rămîne goluri neutilizabile şi greu de recuperat ;

- în sistemul OS/360 M V T dimensiunea minimă alocată este impusă la 52 K de dimensiunea modulului "iniţiator".

Comparînd cei doi algoritmi a rezultat că algoritmul F F este mai rapid şi poate fi mai eficace decît B F [Knu 74]. Se recomandă [Sho 75] algoritmul F F dacă se cer dimensiuni mari) şi cu frecvenţe mari, iar B F se comportă mai bine la cereri de dimensiuni mici. Uneori se poate aloca o parte din memorie după B F şi cealaltă parte după F F. Raportul dintre cele două părţi va ţine cont de frecvenţă şi dimensiunea cererilor mici şi mari. Acest algoritm se numeşte F B F şi are cele mai bune performanţe ajungînd la o utilizare de 90 - 95 % a M C.

S-au propus şi algoritmi de complexitate mai mare care divizează zonele libere în multipli de doi (B S) sau după şirul lui Fibonacci (F S). Aceşti algoritmi nu se justifică însă practic. Metodele propuse pot fi combinate cu compactarea periodică a zonelor de dimensiune redusă. Aceasta este însă costisitoare şi experienţa arată că dacă alocatorul nu poate satisface o cerere în scurt timp după compactare se ajunge la aceeaşi situaţie din cauza încărcării excesive.

1.1.3. Gestiunea unităţii centrale

Prelucrările care se execută într-un program constau dintr-o succesiune de calcule aritmetice sau logice efectuate în unitatea centrală (U C), urmate de schimburi de date,

care se fac cu perifericele, printr-o unitate de schimburi multiple (U S M). Deoarece viteza perifericelor este mult mai redusă decît capacitatea de prelucrare a U C, se încarcă în M C mai multe programe care vor lucra în regim de multiprogramare. La început U C va lucra pentru un program pînă cînd continuarea sa este condiţionată de terminarea unor operaţii de I/E. In acest moment U C va fi pusă la dispoziţie altui program care e pregătit de lansare.

Pentru a realiza comutarea U C spre unul din programele încărcate, există în S C un modul dispecer (distribuitor), care utilizează algoritmi specifici de gestiune a U C. In general fiecărei partiţii i se afectează o prioritate şi o tabelă de control, care va indica în fiecare moment starea programului încărcat (activ, în aşteptare, cauza aşteptării, întrerupt, adresa de continuare, conţinutul registrelor generale, etc). Dacă mai multe programe sînt simultan pregătite se va lansa cel mai prioritar, urmînd ca celelalte să fie lansate cînd cel prioritar este în aşteptarea unei I/E. Momentul terminării aşteptării este sesizat printr-o întrerupere. După tratarea întreruperii, dacă programul prioritar este pregătit (nu mai aşteaptă o altă întrerupere) el va fi relansat şi cel mai puţin prioritar activ va fi pus în aşteptare. Algoritmul prezentat este cel mai simplu mod de gestiune a U C în sistemele de operare cu multiprogramare şi este folosit în majoritatea calculatoarelor din gen.III. (IBM 360/370, FELIX 256/512, etc). In cazul în care sînt încărcate multe programe în M C şi volumul de calcule de efectuat este mare (calcule ştiinţifice), programele prioritare pot "monopoliza" U C, iar timpul de răspuns al programelor mai puţin prioritare creşte nejustificat. Pentru a înlătura acest neajuns s-au propus şi implementat diferiţi algoritmi de gestiune a unităţii centrale, pornind de la diferite criterii de optimizare ce pot să fie contradictorii. Dacă se urmăreşte un timp de răspuns scurt, trebuie ca cererea de U C să fie satisfăcută rapid, lucru posibil numai în partiţiile cu prioritate mare sau dacă există disponibilitate mare (utilizare slabă a U C).

In alegerea algoritmilor de gestiune a U C se urmăresc în general criterii globale ca timp mediu de răspuns, număr de lucrări efectuate în unitate de timp, detectarea programelor în ciclu infinit, respectarea unor priorităţi. Unele informaţii sînt cerute în acest caz de la utilizatori (prioritatea, timpul maxim de execuţie),

Lăsînd la o parte algoritmii banali primul venit primul servit, ultimul venit primul servit (stivă), şir de aşteptare unic cu priorităţi, vom prezenta principalii algoritmi folosiţi la gestiunea U C.

Algoritmul celui mai scurt timp de execuţie S X F S (" Shortest eXecution First Service ") dă prioritate maximă programelor cu timp de execuţie redus. Dacă numărul sosirilor este mare, timpul de răspuns pentru lucrările lungi va fi inacceptabil . Creşte numărul de lucrări terminate în unitate de timp, dar timpul de execuţie trebuie dat la început şi erorile de estimare pot deprecia calitatea serviciilor.

Algoritmul cuantei de timp ("round robin") este foarte echitabil şi poate deservi simultan multe programe chiar dacă se foloseşte alocarea dinamică a memoriei cu swaping. Se recomandă la U C de mare viteză pentru o bună încărcare a U C. Se foloseşte divizarea timpului între utilizatori, utilizînd o cuantă unică C. Programele se lansează în execuţie în ordinea sosirii. Dacă un program nu s-a terminat în timpul C el este întrerupt şi trecut ultimul în şirul de aşteptare. Vor fi favorizate programele scurte, fără a afecta prea mult timpul mediu de răspuns al programelor lungi, care vor trece de mai multe ori prin execuţie (fig. 1.5.).

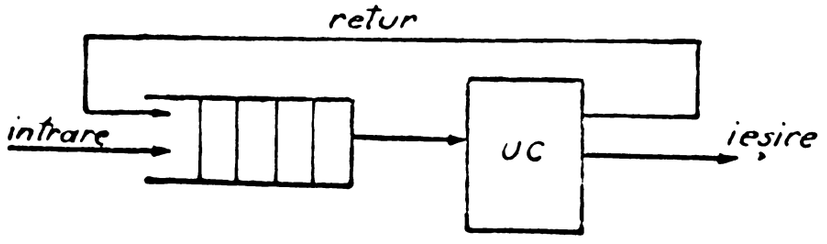


Fig. 1.5 Algoritmul "round robin"

Algoritmul carusel multiplu presupune existenţa mai multor şiruri de aşteptare în conformitate cu timpul de execuţie al programului care nu se cere încheiat. Fiecărui şir i se stabileşte o cuantă $C_i = C_0 \cdot 2^i$ sec [HEL 75] unde $i = 0 \div 7$. Prioritate mai mare au lucrările din primele şiruri (Fig. 1.6). Lucrările din şirul j vor fi lansate numai dacă toate şirurile anterioare sînt vide.

La început programele se introduc în şirul S_0 unde li se alocă o cuantă C_0. Dacă nu s-au terminat la expirarea cuantei vor fi întrerupte şi depuse în şirul S_1 unde li se va aloca la lansare cuanta $C_1 = 2\,C_0$, ş.a.m.d. Algoritmul a fost aplicat la MIT în sistemul C T S S pe un calculator IBM 7090 în 1962 cu $C_0 = 0{,}5$ sec. Valoarea lui C_0 depinde de viteza U C folosite şi timpul de răspuns dorit pentru programele scurte. Algoritmul reduce swapingul pentru programele lungi, dar acesta se face astăzi în paralel cu lucrul U C pentru alte programe şi nu deranjează. Complexitatea modului de gestiune creşte considerabil. Un program lung va urca progresiv în şirurile superioare şi fiecare nouă lansare se va face cu o cuantă dublă faţă de cea anterioară, până la nivelul 7 unde va fi rulat până la terminare cu cuanta $C_7 = 2^7 \cdot C_0 = 128\,C_0$.

Algoritmul se poate simplifica fără a defavoriza programele lungi folosind **carusel-simplu cu cuante variabile de timp** pentru a reduce timpul de swaping la sistemele cu multiacces într-o partiţie fixă. Prin toţi algoritmii prezentaţi se obţine şi o decongestionare a firelor de aşteptare prin servirea prioritară a programelor scurte.

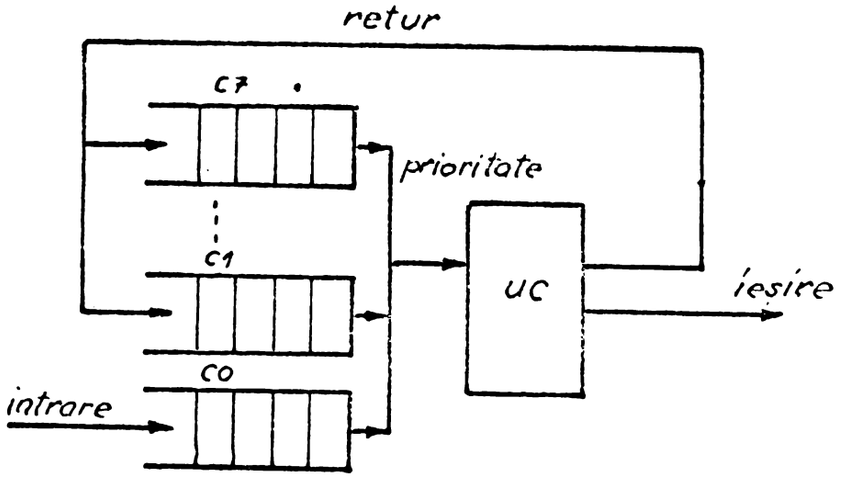


Fig. 1.6. Algoritmul carusel multiplu.

1.1.4. Gestiunea perifericeloz

Calculatoarele de capacitate medie din gen.III. se caracterizează printr-o diversitate mare de periferice, dintre care un rol important îl au discurile magnetice, care devin suportul obligatoriu al sistemului de operare. Pentru a asigura alocarea perifericeloz pentru diferite partiţii, care lucrează în regim de multiprogramare şi protecţia lor faţă de programele din alte partiţii, s-au introdus componente speciale în sistemul de operare. La gestiunea şi protecţia perifericeloz concură supervizorul de I/E, sistemul de gestiune a fişierelor (S G F) şi monitorul de înlănţuire. Componentele hardware indispensabile sînt unitatea de schimburi multiple (U S M) cu acces separat la memorie, sistemul de întreruperi şi ceasul de timp real. U S M asigură multiplexarea perifericeloz alocate diferitelor partiţii pentru a transmite simultan date spre memoria centrală şi în paralel cu lucrul în U C. Sincronizarea operaţiilor din U C cu cele de transfer se realizează prin sistemul de întreruperi de către supervizorul de I/E. Acesta este sesizat printr-o întrerupere de intrare-ieşire, controlează corectitudinea transferului, corectează dacă e cazul erorile şi anunţă programul care a lansat transferul (poziţionează un semafor în blocul de comandă). Ceasul de timp real este declanşat la fiecare lansare de operaţie de I/E şi va da o întrerupere de avarie (watch-dog), dacă perifericul defect nu a răspuns cu întrerupere de terminare după un timp maxim admis. Avaria e tratată de supervizorul de I/E.

Pentru a se proteja perifericeloz afectate unei singure partiţii în exclusivitate, se ţine o tabelă de adrese logice (LUB) în supervizor. Programele cer afectarea unui periferic (ASSIGN) din partiţia în care lucrează dînd adresa logică, care este un punct de intrare în LUB din zona partiţiei. Perifericeloz neafectate partiţiei nu vor avea punct de intrare în LUB în zona partiţiei date. Celor afectate pentru mai multe partiţii (partajabile) vor avea cîte o adresă pentru fiecare partiţie în LUB, toate indicînd aceeaşi intrare în tabela de adrese fizice P U B (Physical unit bloc).

Deoarece instrucţiunile de I/E sînt privilegiate, ele nu pot fi folosite în partiţiile utilizator şi sînt folosite în exclusivitate de supervizorul de I/E. Programul utilizator

pregăteşte programul unităţii de schimb (PUS) şi cere lansarea lui pe un anumit periferic, dat prin adresa logică. Supervizorul de I/E înregistrează cererea, determină adresa fizică a periferiocului din P U B (legătura L U B - P U B) şi o pune în şirul de aşteptare al periferiocului (fig. 1.7).

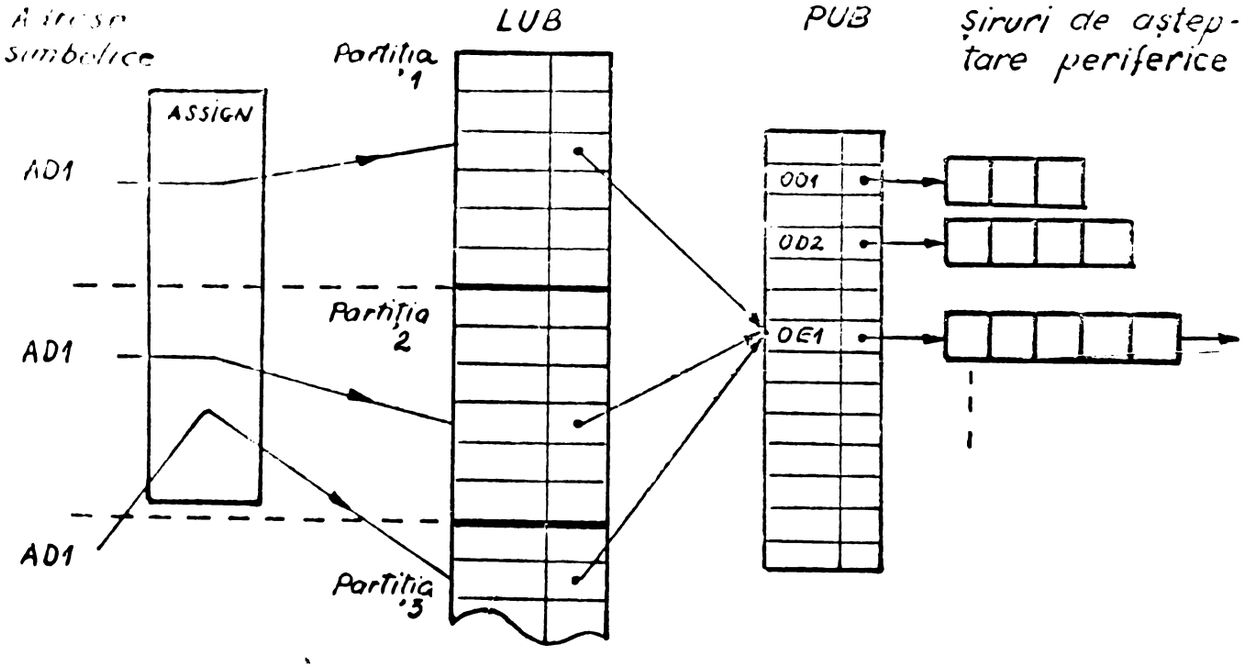


Fig.1.7. Tabele ale supervizorului de I/E

Lansarea operaţiilor de I/E la periferice se face de supervizorul de I/E strict în ordinea intrării în şirul de aşteptare, fără a ţine cont de prioritatea partiţiilor care le-a solicitat. În şirul de aşteptare se memorează adresa P U S corespunzătoare cererii (adresă C B).

In acest fel, dacă în momentul solicitării unui transfer periferiocul e ocupat de un transfer din altă partiţie, se interzice oprirea acelui transfer şi în plus se fac înainte transferurile cerute anterior. Pentru a se ţine cont de prioritatea partiţiei la nivelul supervizorului ar trebui multiplicate şirurile de aşteptare ale periferiocelor cu numărul partiţiilor, complicînd gestiunea.

In sistemele de operare mai noi (UNIX) se combină tot mai mult gestiunea periferiocelor cu gestiunea fişierelor, considerînd periferiocele cazuri particulare de fişiere [Den 74]. Se îmbină astfel protecţia şi alocarea periferiocelor cu protecţia informaţiilor. Acest lucru este foarte important pe discuri magnetice, care sînt partajabile şi la nivel fizic sînt mai multe programe care au simultan

acces, dar la zone diferite. Protecţia zonelor pe disc se asigură prin S G F, care la fiecare deschidere de fişier pe disc verifică drepturile de acces ale programului şi indică zone accesibilă din acea partiţie, într-o tabelă a supervizorului (JIB). Înainte de a lansa o operaţie de transfer pe disc, supervizorul de I/E verifică dacă zona respectivă a fost validată prin S G F pentru acea partiţie [Jur 84] . Se asigură astfel o bună protecţie a discului, importantă în multiprogramare, dar care pune probleme serioase de complexitate la implementarea unor programe (subsisteme) care lucrează în multitasking.

Pentru a asigura o mare flexibilitate în specificarea tîrzie (pînă înaintea deschiderii) a parametrilor externi ai fişierelor (adresă periferic, nume, versiune, etc), aceste informaţii date în limbajul de comandă (ASSIGN , LABEL) sau dinamic în program, sînt păstrate în tabele de comunicaţie S G F, în biblioteci alături de textul programului obiect. La încărcarea programelor în M C se pot modifica aceste tabele dinamic (RFCOM, WFCOM) prin program. Ele sînt efectiv utilizate prin interpretare în momentul deschiderii fişierelor.

Această flexibilitate de tratare a informaţiilor SGF este importantă în multiprogramare, dar deranjează în programele ce lucrează în multitasking, cu procese ce folosesc fişiere specifice. Modificarea dinamică a acestor informaţii cere timp, este incomodă deoarece fişierul DCZ e protejat şi în plus dimensiunea cerută pentru DCZ creşte considerabil. Zonele din DCZ folosite de anumite procese terminate trebuie distruse (eliberate) deoarece ele sînt asociate partiţiei şi nu procesului. Eliberarea lor globală (pe partiţie) se face de monitorul de înlănţuire la terminarea unei faze a lucrării. Facilităţi suplimentare de modificare a fişierului DCZ de către procese se găsesc în sistemul de operare MELIOS, dezvoltat de ITC.

Pentru a optimiza timpul mediu de acces la disc se poate ordona şirul de aşteptare al supervizorului în ordinea cilindrilor faţă de cilindrul curent. Se reduce astfel timpul mediu de poziţionare, pe cilindru (t_c) care este cea mai importantă componentă a timpului de răspuns (t_r) la disc :

$$t_r = t_c + t_s + t_t$$

unde t_s - timpul de poziţionare pe sector este în medie 1/2 din timpul de rotaţie (t_s = 12,5 ms).

t_t - este timpul de transfer şi depinde de lungimea blocului transferat şi pentru discurile de 50 Mo variază între 0 şi 400 msec corespunzător unui cilindru complect (120 K).

Pentru operaţiile de swaping de programe, t_s devine important chiar şi faţă de t_e = 12 - 80 ms. La discurile de 50 Mo, la un debit de 315 Ko/sec, pentru o lungime medie a programelor de 64 K rezultă un timp de transfer pentru swaping

$$t_t = 2 \cdot \frac{64\ K}{315\ K} = 400\ ms.$$

Pentru a compara diferiţi algoritmi de acces la disc se defineşte accesibilitatea [Dod 81], [Bar 71] ca raportul :

$$a = \frac{t_r}{t_t + t_c + t_s}$$

Dintre aceşti timpi se poate optimiza timpul de poziţionare pe cilindru. Pentru N cilindri şi o distribuţie aleatoare a cererilor deservite în ordinea sosirii, se demonstrează că t_c mediu este cel corespunzător parcurgerii a N/3 cilindri. Algoritmul celui mai apropiat cilindru de cel curent şi algoritmul baleierii cilindrilor reduc acest timp. Ultimul evită servirea cu întîrziere a cilindrilor depărtaţi de cel curent.

Modul de gestionare a periferiicelor influenţează considerabil productivitatea sistemului, deoarece fiecărei partiţii trebuie să i se aloce complect sau partajat un număr de periferice. Numărul acestora fiind limitat se limitează numărul de partiţii active iar partajarea şi protecţia discurilor devine vitală. La implementarea unor subsisteme cu multiacces se complică mai mult gestiunea periferiicelor pe procese, iar tratarea periferiicelor ca fişiere şi protecţia ierarhică a fişierelor pe utilizatori adoptată în UNIX, simplifică şi rezolvă mult mai bine probleme.

1.1.5. Gestiunea şirurilor de lucrări

La primele sisteme de operare lucrînd în multiprogramare, fiecărei partiţii trebuia să i se asocieze un fişier de intrare, de pe care se citeau programele şi cartelele de comandă şi un fişier de ieşire pe care se imprimau listingurile programelor sursă şi rezultatele de tipărit (Fig. 1.1). Aceste fişiere puteau fi pe un suport magnetic dar în format cartelă, obţinut printr-un program de conversie (Reader la IBM, YIRAVI la MELIX) ce rula în

altă partiţie. Tot printr-un program de conversie se tipăreau într-o partiţie redusă rezultatele de pe fişierul de ieşire la imprimantă. Această tehnică numită SPOOL ("Simultaneous Peripheral Operation On Line) asigură o exploatare intensivă a imprimatelor şi cititoarelor sistemului, care sînt periferice lente. Se reduce substanţial timpul de trecere a programelor deoarece citirea fişierului de intrare şi scrierea fişierului de ieşire se făcea cu o viteză de peste 10 ori mai mare. A crescut productivitatea sistemului de calcul deoarece numărul de partiţii active a putut fi crescut peste numărul cititoarelor de cartele şi a imprimantelor disponibile [Jia 74].

Odată cu creşterea numărului de partiţii posibile, prin creşterea capacităţii MC, tehnica SPOOL a devenit incomodă deoarece fiecărei partiţii trebuia să i se afecteze un fişier de intrare şi unul de ieşire, în general pe suport magnetic. Gestiunea acestor fişiere a complicat şi a limitat numărul de partiţii utilizate din lipsă de fişiere, iar lucrările erau executate fără prioritate în ordinea introducerii în fişierul de intrare.

O gestiune mult mai eficace a şirurilor de lucrări s-a obţinut prin introducerea unui modul unic al sistemului de operare, care reglementează intrările şi ieşirile din sistem utilizînd flexibil toate cititoarele de cartele şi imprimantele (SYMBIONT la calculatoarele FELIX). Lucrările pot fi introduse prin SYMBIONT, de pe oricare cititor de cartele, specificînd pe cartela de JOB, clasa, categoria şi prioritatea lucrării. SYMBIONT-ul depune lucrările într-un fişier unic partajat pe disc şi ţine evidenţa lor prin şiruri de aşteptare numite clase. Ordinea lucrărilor într-o clasă este funcţie de prioritatea indicată pe cartela JOB. Fiecărei clase îi corespunde un tabel de ordine (fig. 1.1.). Operatorul afectează unei clase de lucrări o anumită partiţie. Listingurile programelor rulate sînt preluate de SYMBIONT şi ordonate în categorii conform cererii din JOB, indiferent de partiţia în care au fost rulate. Operatorul afectează fiecărei categorii o anumită imprimantă funcţie de cerinţele de editare. SYMBIONT-ul ocupă o partiţie de 30 K şi asigură o mare flexibilitate de gestiune a fişierelor de intrare şi ieşire.

Pentru a permite introducerea unor lucrări de la distanţă, provenite de la calculatoare satelit sau de la concentratoare de date s-a prevăzut un modul al SO numit TbLo-SYMBIONT.

Acesta preia lucrările, gestionînd liniile de teletransmisie, de la stațiile plasate la distanță și le transmite SYMBIONT-ului care le inserează în clasele cerute. Rularea lor se va face normal (batch) într-o partiție de multiprogramare. Rezultatele obținute vor putea fi listate local, dacă se specifică o categorie normală, sau returnate SYMBIONT-ului, într-o categorie specială, care le va transmite pentru listare la stația aflată la distanță.

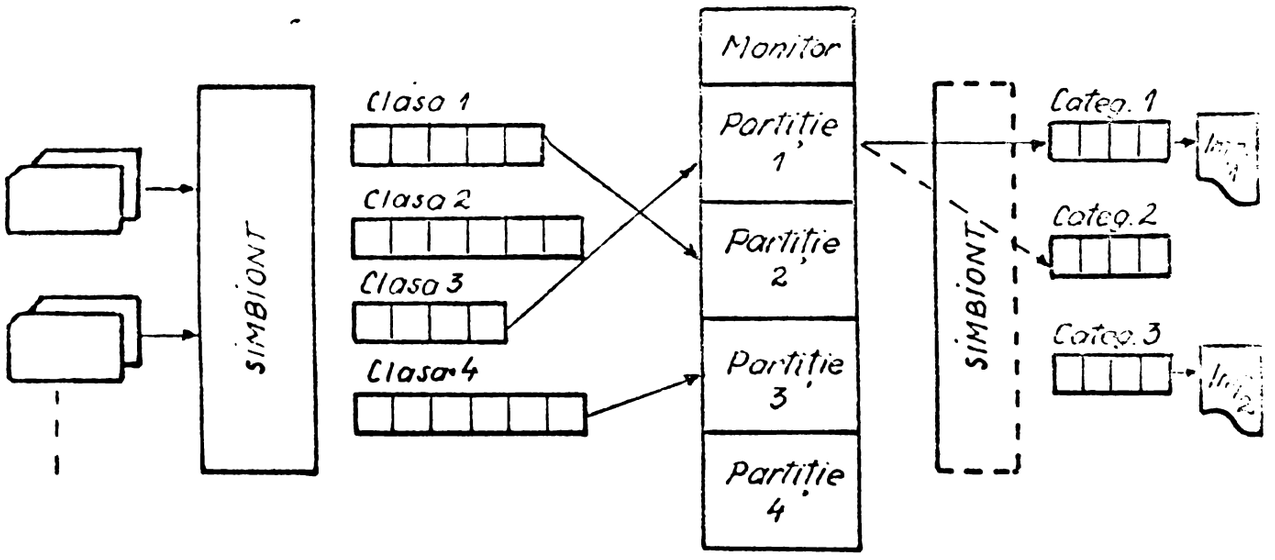


Fig. 1.8. Gestionarea fișierelor de intrare/ieșire prin SYMBIONT.

Dacă gruparea lucrărilor pe clase se face funcție de necesitățile de memorie centrală, periferice și prioritate, clasele pot fi asociate optim unor partiții care dispun de asemenea resurse. Crește în acest caz gradul de încărcare a resurselor și se îmbunătățește calitatea serviciilor. Totuși sistemul limitează restrictiv accesul în sistem, acceptînd rularea programelor numai în "batch" din clasele SYMBIONT. Concepția unor sisteme interactive în multiacces devine dificilă și se reduce la editarea programelor, introducerea lor în clasele SYMBIONT și recuperarea rezultatelor din categoriile SYMBIONT pentru a putea fi afișate pe terminal (ARIEL la sistemul MELIX C 256/512).

1.1.6. Gestiunea fișierelor sistem

Componentele sistemului de operare folosesc în timpul lucrului un număr de fișiere sistem care trebuie să fie alocate

pentru fiecare partiţie, care lucrează cu înlănţuire automată de lucrări şi permite lucrul compilatoarelor. O partiţie serie (background) poate să efectueze orice tip de lucrări. In afara fişierelor utilizatorilor, partiţie trebuie să aibă permanent alocate :

- fişierul de manevră al compilatoarelor în care se citesc cartelele de program sursă şi în care se ţin informaţiile intermediare (tabele de variabile) necesare în compilare (ZPS MANS);
- fişierul de ieşire al compilatoarelor, în care se depun modulele biner translatabile rezultate şi care este fişierul de intrare al editorului de legături (FILMIT);
- fişierul de ieşire al editorului de legături în care se depune programul "imagine memorie translatabilă" (IMT) rezultat (LOAD GO) acest program poate fi încărcat într-o partiţie de memorie pentru a fi rulat (comanda RUN), sau poate fi catalogat într-o bibliotecă.
- fişierul în care la o terminare anormală a programului se depune conţinutul partiţiei şi registrelor generale pentru a putea fi scrise pe listing dacă se cere vidaj (DUMP).

Dacă se cere existenţa mai multor partiţii serie (universale) atunci pentru fiecare trebuie asigurate aceste fişiere şi în plus o copie a monitorului de înlănţuire pentru tratarea cartelelor de comandă. Dacă se cere ca în partiţie să se execute cu înlănţuire automată numai programe de execuţie chemate din biblioteci IMT, atunci este necesară numai o copie a monitorului de înlănţuire. Aceste partiţii se numesc "paralele cu înlănţuire" (forground) şi nu admit compilări şi editări de legături Partiţiile paralele simple admit numai execuţie unor programe, lansate de la consolă sau din alte partiţii, care au fost catalogate în prealabil în biblioteca utilizator standard (BUS). Ele nu admit tratarea de cartele de comandă, deoarece nu au o copie a monitorului de înlănţuire. Dacă acesta ar fi realizat ca reentrant, toate partiţiile paralele ar putea avea înlănţuire fără a consuma spaţiu pe discul sistem pentru copiile monitorului de înlănţuire.

Un fişier sistem comun tuturor partiţiilor este DCZ (disc communication zone), care conţine informaţiile comenzi de ASSIGN şi LABEL pentru toate fişierele utilizate de programele ce sînt încărcate la un moment dat în partiţiile sistemului. Dimensiunea se depinde la un moment dat de numărul partiţiilor folosite şi numărul mediu de fişiere utilizate pe program (vezi 1.1.4).

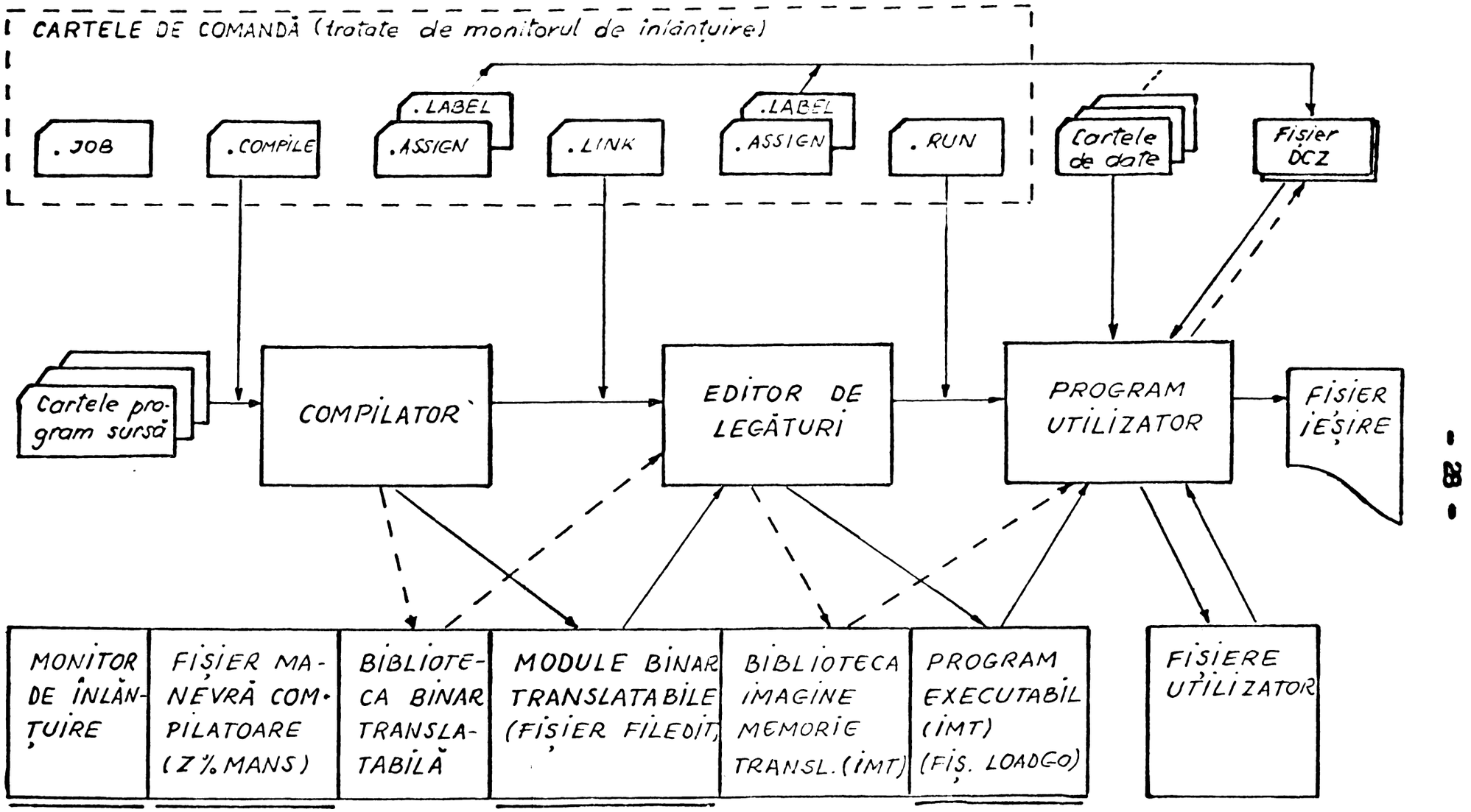


Fig. 1.9. Utilizarea fișierelor sistem.

Aceste restricţii existente pentru partiţii în sisteme cu multiprogramare pun probleme dificile la implementarea unor subsisteme interactive cu multiacces. In general în aceste sisteme nu este posibilă compilarea programelor prin utilizarea compilatoarelor obişnuite, deoarece aceasta ar necesita multiplicarea inadmisibilă a fişierelor sistem, care nu se justifică nici pentru partiţiile paralele la prelucrarea în loturi. Din acest motiv cele mai perfecţionate subsisteme cu multiacces interactive, cum este OS/360/370 TSO (time Sharing option) permit numai editarea programelor sursă, lansarea lor în execuţie în prelucrarea în loturi, recuperarea listingului ce se afişează la terminal şi lansarea unor compilatoare speciale interpretative conversaţionale [Hel 75] .

1.1.7. Limbaje de comandă

Programele sursă ce intră în sistem sînt scrise în limbaj de asamblare sau într-un limbaj de nivel înalt. Ele vor suferi prelucrări variate funcţie de dorinţele utilizatorului şi posibilităţile SO. Pentru a facilita utilizatorului specificarea directă a prelucrărilor ce le vor suferi cartelele ce intră în sistem şi caracteristicile fişierelor solicitate, calculatoarele din gen. III. au fost prevăzute cu limbaje de comandă (JCL = job control language). Comenzile limbajului de comandă se dau pe cartele cu o sintaxă anumită, identificate cu semne speciale pe prima coloană ("." la MULTIX şi //. la IBM). Funcţiile realizate sînt diverse şi se referă în special la delimitarea lucrărilor şi evidenţa lor (JOB), comanda compilării, editării legăturilor sau execuţia programului (COMPILE, LINK, RUN), chemarea unor programe din bibliotecă (FETCH), alocarea spaţiului pentru fişiere, specificarea adresei periferice şi a numelor externe ale fişierelor (ALLOC, ASSIGN, LABEL), etc.

Cartelele de comandă sînt tratate de module ale monitorului de înlănţuire, (interpretor de comenzi) asociat partiţiilor cu înlănţuire. Unele comenzi sînt imediate, iar altele (cele referitoare la fişiere) completează doar tabele ale SO fiind folosite ulterior (DCL).

La mini şi microcalculatoare limbajele de comandă sînt conversaţionale. Comenzile se pot da în anumite momente de către utilizator de la consolă conform unei sintaxe date. Funcţiile lor sînt mai reduse şi mai simple. Sînt prevăzute comenzi şi pentru funcţiile de editare a textului sursă, de

inventariere, scriere sau ştergere de fişiere. Cele mai cunoscute limbaje de comandă sînt cele din sistemele de operare SFDX şi CPM pentru microcalculatoare, RSX - 11 şi UNIX pentru minicalculatoare. La minicalculatoarele mai puternice se permite lucrul în multiprogramare (multi-user). Pot exista mai mult console (4 - 14) de la care utilizatorii pot lucra independent, afectîndu-se fiecăruia o zonă de memorie centrală sau virtuală protejată (partiţie) în care lucrează programele sale. Rezultă că sistemele de operare "multi-user" realizează o multiprogramare cu limbaje de comandă conversaţionale. Sistemul UNIX implementat pe minicalculatoare din gama PDP-11/45, 11/50 [Den 74] are facilităţi deosebite în asigurarea protecţiei periferice lor între utilizatori, folosind fişiere ierarhizate ordonate pe utilizatori în care periferice le sînt cazuri particulare (acces fizic direct). Se utilizează şi virtualizarea memoriei folosind o memorie auxiliară pe disc.

Limbajele de comandă au o deosebită importanţă în dezvoltarea sistemelor de operare. Se pot adăuga noi funcţii prin completarea setului de comenzi şi argumente şi scrierea de module corespunzătoare.

Limbajul de comandă poate fi foarte bogat în comenzi şi poate fi extins după necesităţi. O extindere excesivă a limbajului de comandă îl face greoi şi heterogen. Se recomandă adăugarea unor subsisteme de operare specializate, care sînt lansate prin limbajul de comandă şi dispun de un subset propriu de comenzi, adaptate aplicaţiei (subsisteme pentru multiacces conversaţional, gestiunea bazelor de date, etc). Pentru o utilizare şi o însuşire mai simplă se utilizează în ultimul timp limbaje de comandă de tip meniu, în care se prezintă utilizatorului la început setul de comenzi existente şi modul de apelare iar pentru fiecare comandă utilizată se dau pe display argumentele, modul de plasare şi sensul lor. Acestea presupun legături de mare viteză, cu terminalele şi display cu ecran adresabil .

Un caz particular îl constituie limbajele de comandă destinate subsistemelor conversaţionale cu multiacces, care se asemănă cu limbajul de comandă de bază, care este completat şi adaptat cerinţelor. Un exemplu tipic îl constituie limbajul conversaţional IBM OS/360/370 TSO (time sharing option), care a fost implementat (1971) pe calculatoarele OS/360 model 67 prevăzute cu memorie virtuală şi care va fi prezentat în 1.2.5.

1.1.8. Multiprogramarea complexă

Calculatoarele din gen.III-a. sînt universale, dar inițial sistemele de operare s-au proiectat pentru prelucrarea în loturi locală (batch), urmînd ca ulterior să fie extinse.

Nucleul și arhitectura acestor SO urmăresc o flexibilitate ridicată la scrierea programelor prin posibilitatea utilizării diferitelor limbaje pentru secțiunile unui program, la care se pot adăuga module existente în biblioteci ale sistemului sau utilizatorului. Aceste module se reunesc într-un program unitar de către editorul de legături într-o fază separată. Se asigură o protecție sigură a datelor între partiții în MC și pe fișiere, o gestionare eficientă a periferiecelor și UC. O partiție lucrează ca o linie separată de prelucrare.

Sistemele de operare cu multiprogramare realizează folosirea eficientă a sistemului de calcul prin încărcarea intensivă a tuturor resurselor și asigurarea de facilități variate pentru utilizatori, la depanarea programelor, la exploatarea lor și un preț redus al prelucrării. Aceasta presupune însă rularea secvențială, locală a programelor în centrul de calcul, care implică manipulări importante de informație (în general pe cartele) între locul de utilizare și centrul de calcul. Corecțiile datelor și programelor eronate se operează cu întîrzieri mari, fără a se permite intervenție imediată a utilizatorului. Aceste inconveniente conduc la nemulțumirea utilizatorilor care preferă achiziționarea unor mini și microcalculatoare, a căror preț a scăzut foarte mult în ultimii 10 ani și care pot rezolva o mare gamă din problemele practice. Totuși rămîn o mare parte din problemele de proiectare și cercetare complexe, care necesită o capacitate de calcul importantă și problemele de gestiune a bazelor de date care necesită volume de memorie externă indisponibile pe minisisteme.

Pentru a reduce timpul de acces la sistemele de calcul și a asigura un confort sporit de exploatare a programelor s-au extins în anii '70 sistemele de teleprelucrare a datelor, implementate pe sistemele cu multiprogramare existente. Această orientare a fost stimulată de reducerea considerabilă a costului terminalelor și interfețelor de teletransmisie (de cca. 6 - 10 ori) în ultimii 10 ani. Cea mai simplă soluție de teleprelucrare interactivă a programelor o constituie legarea unui terminal conversațional (display, teletype) printr-o linie telefonică

la calculator şi afectarea lui ca periferic de intrare şi ieşire la o partiţie. Se pot asigura astfel simplu unui utilizator aflat la distanţă toate facilităţile unui utilizator local. Numărul maxim al utilizatorilor admişi la un moment dat pe sistem (locali şi la distanţă) se limitează însă la numărul partiţiilor utilizatori (în general < 15).

Această soluţie nu este însă acceptabilă din punct de vedere al încărcării resurselor, deoarece se blochează o partiţie şi un număr de periferice, pentru o lucrare în care introducerea datelor se face cu o viteză de maxim 1 - 2 car/sec iar extragerea cu 10 - 120 car/sec. Toate perfecţionările aduse sistemelor cu multiprogramare, pentru introducerea şi extragerea datelor (SPOOL, SYMBIONT) sînt astfel compromise, iar preţul prelucrării conversaţionale creşte de zeci de ori faţă de prelucrarea în loturi. Această soluţie este acceptabilă numai în sistemele de calcul mari prevăzute cu SO orientate pe lucrul conversaţional, care asigură virtualizarea MC pe spaţiul unei memorii auxiliare mai ieftine şi dispune de mari resurse de periferice. Pentru o anumită categorie de terminale plasate la distanţă, cum sînt mini sau microcalculatoare satelit, concentratoare de date, staţii dotate cu cititoare de cartele şi imprimante, se poate asigura accesul de la distanţă, prin linii telefonice lucrînd în mod sincron la viteză mai ridicată (4800 biţi/sec). Gestiunea acestor terminale de intrare/ieşire se asigură prin modulul TELESYMBIONT al SO, care introduce lucrările primite în clasele SYMBIONT-ului (conform indicaţiilor de pe JOB), urmînd să se ruleze în acelaşi regim ca şi lucrările locale. Datele de ieşire pot fi plasate în categorii speciale, de unde TELESYMBIONT-ul le va recupera şi le va transmite la distanţă unde vor fi imprimate (remote - batch).

Pentru rezolvarea majorităţii problemelor de teleprelucrare, în condiţii acceptabile, SO cu multiprogramare nu oferă soluţii. Se cere accesul la sistemul de calcul în regim conversaţional, local sau de la distanţă, a unui mare număr de utilizatori (în general < 64), de la terminale foarte lente (1 - 120 oct/sec). Pentru a extinde posibilităţile sistemelor de calcul în domeniul teleprelucrării s-au conceput subsisteme de operare cu multiacces conversaţional, care sînt pachete de programe ce lucrează într-o partiţie de multiprogramare asigurînd partajarea resurselor disponibile între un număr mare de utilizatori aflaţi la distanţă sau locali.

Se asigură gestiunea terminalelor de teletransmisie (cu problemele lor specifice), partajarea memoriei centrale şi protecţia programelor sau datelor cu salvarea lor pe fişiere disc, divizarea timpului de UC între utilizatori. Un asemenea subsistem este văzut de SO ca un singur program normal (fig. 1.10). Caracteristicile diferitelor subsisteme cu multiacces vor fi prezentate în Cap. 1.2.

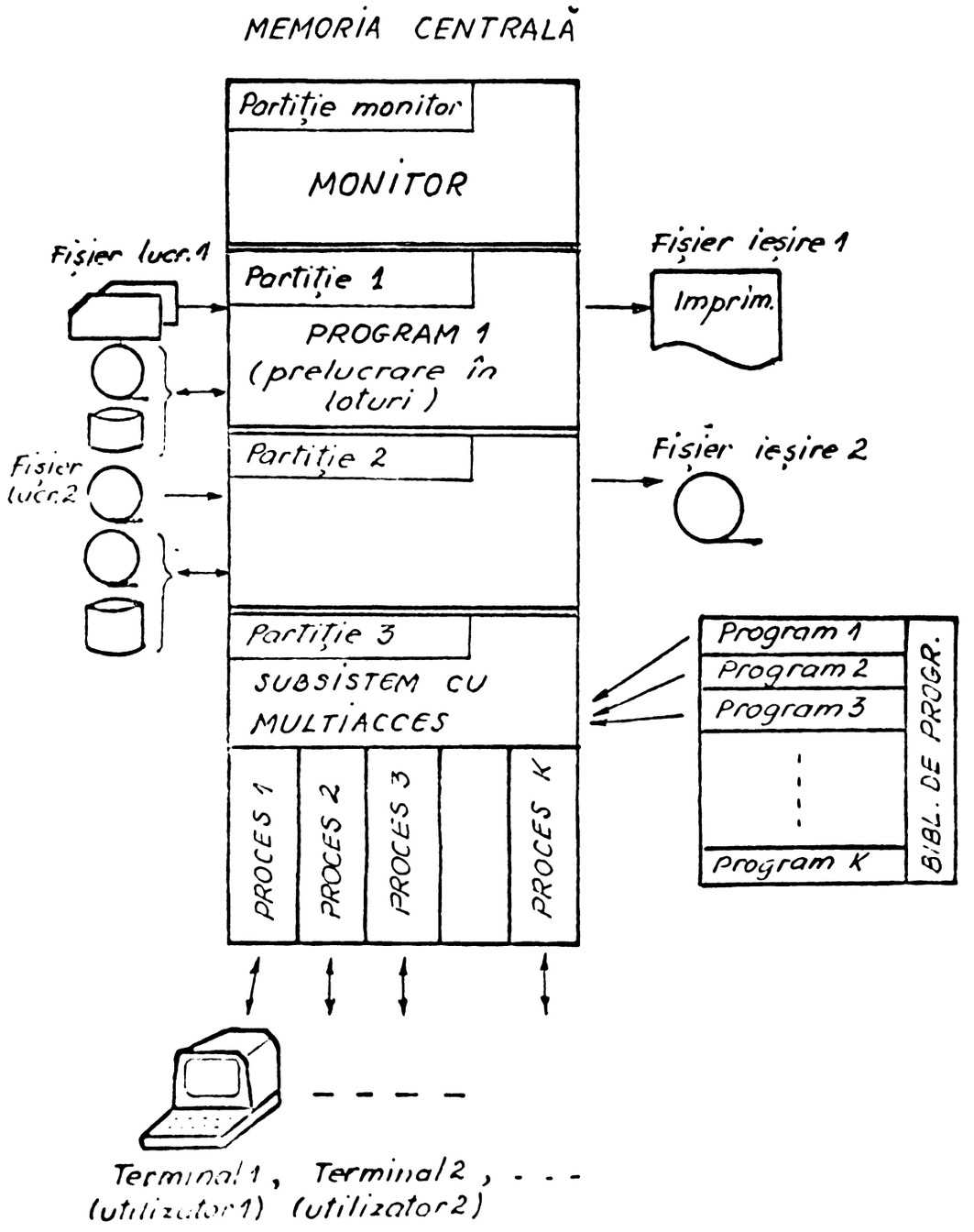


Fig. 1.10. Multiprogramare complexă.

Lucrînd într-o partiţie normală subsistemele cu multisecces trebuie să accepte restricţiile impuse de SO pentru programele obişnuite în privinţa protecţiei şi accesului la resursele MC, UC, periferice şi fişiere. Din acest motiv şi facilităţile acestor subsisteme sînt limitate şi rezolvă numai un grup de probleme specifice. O rezolvare globală a problemelor de multisecces ar impune schimbarea profundă a concepţiei sistemului de operare, care vine în contradicţie cu principiile multiprogramării şi practic nu este posibil din cauza volumului de muncă necesar. Desvoltarea facilităţilor sistemelor de operare se poate rezolva acceptabil prin adăugarea de subsisteme specializate, [Wu 74][Kah 81] cu limbaje de comandă proprii, dacă la proiectarea SO s-ar prevedea unele macroinstrucţii speciale, care să înlesnească accesul la unele resurse şi să permită preluarea unor sarcini de gestiune şi protecţie a MC , periferice şi fişierelor de către subsistem (vezi 2.1). S-ar realiza astfel SO puternice cu o multiprogramare complexă, care să asigure atît avantajele de eficienţă ale multiprogramării, cît şi timpul de răspuns redus şi interacţiunea cu programul oferite de sistemele conversaţionale. Pentru partiţia subsistemului cu multisecces se va prevedea o prioritate mai mare faţă de partiţiile afectate prelucrărilor locale. Programele rulate local nu vor fi întîrziate mult de cele conversaţionale la o încărcare medie a subsistemului cu multisecces. Timpi de UC nefolosiţi de programele conversaţionale vor fi afectaţi programelor locale. Subsistemele existente se comportă relativ izolat de celelalte componente ale SO.

1.2. SUBSISTEME DE OPERARE CU MULTIACCES

1.2.1. Limbaje conversaţionale pentru multiacces.

Limbajele de programare dezvoltate pentru prelucrare în loturi (FORTRAN, COBOL, PL/1, etc) nu pot fi folosite în forma standard în sistemele cu multiacces, datorită restricţiilor impuse de acest mod de lucru. Compilatoarele existente pentru aceste limbaje necesită fişierele sistem descrise în 1.1.6., care nu pot fi asigurate în numărul cerut de către utilizatorii de teleprelucrare. Compilatoarele lucrează în mai multe faze plus editarea legăturilor şi ocupă în general o partiţie. Partajarea lor între utilizatori la un moment dat ar presupune variante de compilatoare reentrante şi ca toţi utilizatorii subsistemului să lucreze în acelaşi limbaj. Accesul dinamic la fişierele disponibile la prelucrarea în loturi nu este posibil fără restricţii în conversaţional.

Din motivele enunţate, dacă se doreşte utilizarea conversaţională a unor programe în format sursă se folosesc limbaje specializate, ca BASIC, APL, sau subseturi ale limbajelor clasice FORTRAN, COBOL, PL/1, adaptate noilor cerinţe. Pentru traducerea programelor sursă se utilizează interpretoare reentrante care lucrează ca subsisteme cu multiacces într-o partiţie a MC. Fiecărui utilizator i se vor aloca spaţii de memorie centrală pentru schimbul de date cu terminalul şi spaţii de memorie pe disc pentru păstrarea programului sursă, a datelor aferente, a tabelelor de adrese, etc. Subsistemul va dispune de un modul de gestiune a terminalelor de teletransmisie în regim de multiacces, un modul de divizare a timpului între utilizatori, un modul de asigurare a schimbului între MC şi disc pentru informaţiile unui utilizator, un modul de gestiune a fişierelor, un editor de text şi de un interpretor de instrucţii sursă (fig. 1.11).

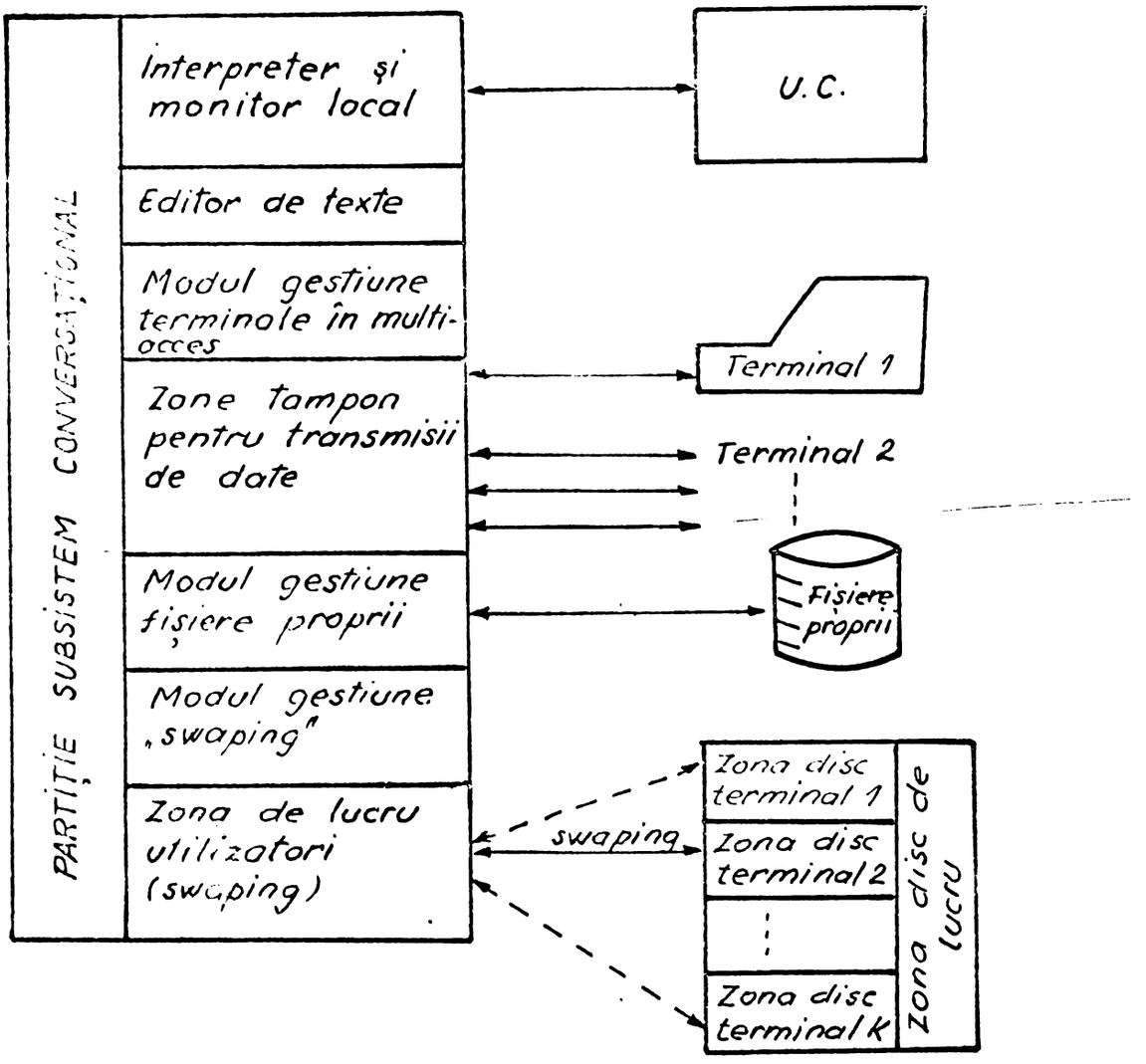


Fig. 1.11. Subsistem cu limbaj conversațional și multiacces.

Modulul de gestiune a terminalelor ține legătura "on line" cu utilizatorii aflați la terminal, preluînd liniile sursă de program sau comenzile linie cu linie în zonele tampon pentru transmisii date. Fiecare linie de program este preluată de editorul de texte care o depune în zona alocată programului, pe un fișier disc. Comenzile vor fi transmise monitorului local.

Se permite astfel crearea de către fiecare utilizator a unui program sursă, care poate fi uşor corectat utilizînd editorul de texte. Fiecărui program i se poate crea, tot prin editor, unul sau mai multe seturi de date. Dacă programul e corect, utilizatorul poate cere lansarea sa în execuţie cu un set de date anumit. Interpretorul va citi, analiza şi executa programul sursă linie cu linie, pentru fiecare linie se va genera secvenţa de instrucţii necesară, care se va executa imediat. Secvenţa de instrucţii pentru o linie, tabelele de adrese şi variabilele necesare în program vor fi alocate în zona de lucru. Dacă execuţia unui program este întreruptă, pentru a activa programul altui utilizator, zona de lucru este salvată într-o zonă de lucru pe un fişier disc de unde se va încărca zona de lucru a programului activat (Swaping). La reluarea programului, întreaga zonă se va restaura de pe disc şi se va putea continua execuţia.

Dacă se întîlnesc erori la analiza unei linii de program, aceasta va fi întreruptă, utilizatorului i se va comunica eroarea şi o va putea corecta cu editorul de texte. După corectare, programul va fi executat de la început. In limbajele conversaţionale interpretative nu se generează un program obiect, ci numai o secvenţă de instrucţii în cod maşină, pentru fiecare linie de program, ce se execută imediat. Acest mod de lucru impune restricţii de complexitate a programului de tehnici de programare folosite, de utilizare a variabilelor tablourilor şi fişierelor. Se acceptă utilizarea numai a unor fişiere speciale ale subsistemului şi cu o organizare simplă (secvenţiale şi nedefinite). Aceste limitări restrîng aria de utilizare a limbajelor conversaţionale la proiectarea asistată de calculator şi la instruirea în programare.

Interpretoarele de limbaj satisfac cerinţele accesului conversaţional, dar lucrează mult mai încet decît compilatoarele. In plus la fiecare rulare a programului se face şi compilarea. Dacă numărul de instrucţii în program creşte mult, timpul de rulare creşte mult mai rapid faţă de programele rulate cu compilatoare normale în prelucrarea în loturi. Ultimele presupun un timp de compilare, care creşte încet în raport cu numărul de instrucţii al programului sursă şi nu se repetă la fiecare rulare a programului, dacă s-a păstrat programul obiect într-o bibliotecă. In figura 1.12. se prezintă creşterea timpului de execuţie a programelor, funcţie de

numărul de instrucţii executate cu interpreter şi compilator [Hsl 75]

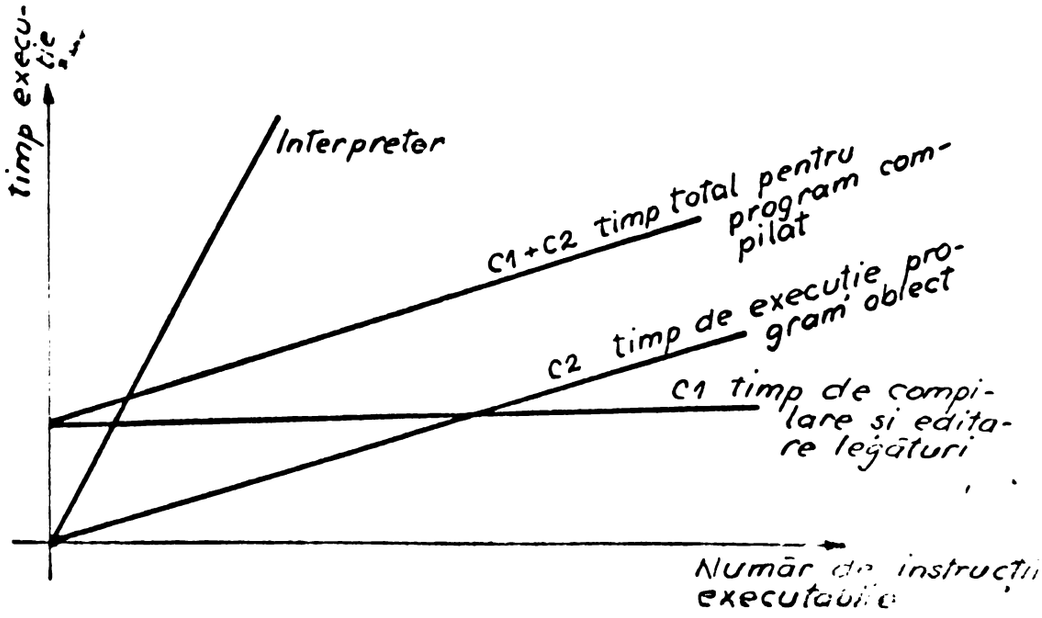


Fig.1.12. Timpul de execuţie pentru interpreter şi compilator.

Se observă că pentru programele scurte interpretorul satisface acceptabil cerinţele, dar pentru programele lungi timpul de execuţie este inacceptabil. Utilizarea unor programe lungi sau ciclice în regim conversaţional poate bloca şi pe ceilalţi utilizatori şi impune existenţa unui nodul de divizare a timpului UC. Toate componentele unui subsistem conversaţional trebuie să fie reentrante şi ele sînt protejate între utilizatori. Spaţiul ocupat de un astfel de sistem este destul de mare, ajungînd la limbajul APL implementat pe IBM/360 la 100 K din care 70 K interpretorul şi monitorul local.

Pentru a ameliora performanţele unui astfel de subsistem şi a extinde domeniul de aplicaţie, în sistemul IBM RAX se utilizează un compilator FORTRAN rapid, fără editare de legături, care compilează un program complect, fără a fi întrerupt. La execuţia programului obiect rezultat, se aplică divizarea timpului între utilizatori, cu salvarea programelor inactive pe disc. Programul obiect rezultat de la un compilator conversaţional nu poate fi rulat într-o partiţie cu prelucrare în loturi.

Limbajul BASIC [Kod 78] este cel mai răspîndit limbaj conversațional utilizat inițial pe minicalculatoare, cu un interpretor reentrant. Limbajul este orientat pe calcule științifice, este simplu de învățat și are diferite variante implementate în ultimul timp pe calculatoarele personale. Permite lucrul cu variabile și tablouri aritmetice și șiruri de caractere, are funcții standard pentru calcule științifice și funcții grafice pentru ecranul terminalului. Posibilitățile limbajului sînt totuși mai reduse decît ale limbajului FORTRAN. Este indicat pentru calcule științifice și instruiree în programare unde rezultatele obținute sînt mai superioare celor obținute la prelucrarea în loturi.

Limbajul APL, conceput special pentru lucrul conversațional în multiacces, este cel mai puternic limbaj de acest tip. Definit de K.E. Iverson în cartea "A Programming Language apărută în 1963, [Iv 63][Hel 75] limbajul APL permite implementarea unei mari varietăți de algoritmi printr-o descriere clară și concisă neîntîlnită în alte limbaje. Are instrucțiuni speciale pentru lucrul cu vectori, matrici și structuri multidimensionale. Admite variabile de tip bit, numere binare sau virgulă flotantă, șiruri de caractere și o varietate mare de operatori speciali aplicabili la variabile simple sau indexate.

Prima implementare a limbajului APL s-a făcut în 1964. O implementare extinsă s-a făcut în 1966 pe calculatoare IBM/360, recomandîndu-se modele mari cu MC de capacitate mare (250 K) și UC rapidă. Tipic se consideră modelul IBM/360 model 50 cu MC de minimum 500 K care lucrează în multiprogramare complexă cu mai multe partiții în prelucrarea în loturi și o partiție APL. La o încărcare de 50 de terminale se obține prin APL un timp mediu de răspuns de 3 sec. [Hel 75] Translatorul pentru limbajul APL implementat pe IBM/360 este un interpreter, reentrant, prevăzut cu modul de multitasking pentru gestiunea terminalelor, un limbaj de comandă simplu, modul de gestiune a MC prin swaping pe disc și posibilități limitate pentru utilizatori de a avea acces la o zonă de disc. Zona de lucru utilizată este de 32 K și este accesibilă pentru un singur utilizator la un moment dat. La schimbarea

utilizatorului se salvează prin swaping pe disc. Pentru a lucra subsistemul APL are nevoie de o partiţie de peste 100 K. O variantă mai nouă, anunţată în 1973 pentru IBM/370, permite şi utilizarea unor fişiere şi mai multe zone de lucru (32 K). Aceasta permite simultaneitatea depunerii sau încărcării de pe disc a unei zone de lucru (max. 6 zone) şi lucrul în UC pentru un alt utilizator (unul singur activ).

Subsistemul APL asigură o divizare a timpului între utilizatori cu o cuantă medie de 0,1 sec şi un timp de swaping mai mic decît 0,25 sec. Pentru evaluarea performanţelor există implementat un modul special. Pentru programele lungi timpul de răspuns creşte foarte mult şi performanţele scad considerabil per total subsistem, datorită interpretorului.

1.2.2. Limbaje conversaţionale pentru gestiunea în multiacces a bazelor de date.

Pentru organizarea şi regăsirea informaţiilor la gestiunea unor volume mari de date, sistemele de operare cu multiprogramare din gen. III-e. pun la dispoziţia utilizatorilor un sistem de gestiune a fişierelor ("file management system "), utilizabil prin macroinstrucţii din limbajele de programare. Se asigură astfel organizarea şi protecţia fişierelor standard (secvenţiale, secvenţial indexate, selective), realizîndu-se interfaţa cu supervizorul de intrare/ieşire din nucleul sistemului de operare. Extinderea mare a utilizării calculatoarelor în gestiunea economică, gestiunea bancară, rezervări de locuri, evidenţa populaţiei, etc, a impus realizarea unor sisteme informatice complexe, în care organizarea neredondantă, flexibilă şi multifuncţională a unor volume mari de informaţie reprezintă o cerinţă obligatorie şi un criteriu de eficienţă. Realizarea unor structuri complexe de date şi mai ales actualizarea se face greu prin limbajele de programare şi necesită un efort de programare considerabil.

Acestea au fost motivele care au impus în ultimii 15 ani apariţia unor subsisteme de gestiune a bazelor de date (BD). Variantele de structură, metode de acces,

limbaje de descriere şi interogare folosite sînt diverse, materializate în peste 100 de sisteme de gestiune a bazelor de date comercializate de firmele producătoare. Deoarece cerinţele şi soluţiile propuse pentru bazele de date diferă şi nu s-a ajuns la o standardizare unanim acceptată, sistemele de gestiune a BD s-au realizat cu subsisteme, relativ independente de sistemele de operare. Interfaţa cu SO se realizează prin sistemul de gestiune al fişierelor, parte componentă a oricărui SO. S-a ajuns astfel ca pe anumite sisteme de calcul să fie implementate mai multe subsisteme de gestiune a BD, de complexitate şi mod de organizare diferite [Ui 77], [Ui 82], [Ad 78], [La 75].

Pentru descrierea structurilor BD şi pentru interogarea BD s-au definit limbaje specializate, care pot fi considerate ca extinderi ale limbajelor de programare, sau extinderi ale limbajului de comandă. Dacă limbajul de descriere şi interogare are ca limbaj gazdă un limbaj de nivel înalt ca PL/1 sau COBOL, el este o extindere a limbajului şi trebuie modificat corespunzător compilatorul pentru a recunoaşte noile directive şi macroinstrucţii, tratate prin module speciale de acces adăugate programului obiect (IMS/IBM, Codasyl, TOTAL). Ca suport logic al BD se folosesc fişierele standard, la ale căror posibilităţi de organizare şi protecţie se adaugă facilităţi noi. Utilizarea acestor BD se face numai în prelucrarea în loturi.

Subsistemele independente de gestiune a BD, au apărut mai tîrziu, ele au limbaj propriu de descriere şi interogare a BD, care se poate considera ca o extindere a limbajului de comandă a SO pentru acest gen de aplicaţii. In general subsistemul lucrează în prelucrarea în loturi, fiind chemat din biblioteca standard a sistemului şi încărcat într-o partiţie utilizator. Descrierea BD la creare şi interogare se face conform limbajului propriu pe cartelele tratate în timpul rulării. In sistemele de calcul mai noi subsistemul de gestiune a BD este integrat organic, ca o componentă a sistemului de operare, şi ca o formă superioară de gestiune a informaţiilor (în locul SGF).

Dezvoltarea sistemelor de teletransmisie în ultimul deceniu a impus, din considerente practice, accesul conversaţional la BD de la distanţă şi din puncte diferite în regim de multiacces. Complexitatea ridicată a BD face aproape imposibilă interogarea şi mai ales actualizarea acestora conver-

seţională prin subsistemele cu multiacces existente. Din acest motiv s-au conceput în ultimul timp module de interfaţă conversaţională pentru subsistemele de gestiune a BD. Acest modul asigură dialogul în multitasking cu terminalele conversaţionale, păstrează mesajele primite în zone tampon speciale şi le dirijează secvenţial spre modulele de acces ale sistemului de gestiune a BD (aceleaşi module folosite şi la prelucrarea în loturi). Datele primite din BD sînt dirijate spre terminalul care le-a cerut.

Sistemele de gestiune a BD sînt printre cele mai numeroase şi mai importante implementări ale subsistemelor cu multiacces întîlnite în practică. In fig. 1.13. se prezintă modul de realizare a accesului conversaţional în BD de tip SOCRATE [So 75] unde se folosesc aceleaşi comenzi atît în prelucrarea în loturi pe cartele, cît şi în conversaţional. Se utilizează acelaşi interpreter care apelează modulele de acces. Se schimbă numai modulul de intrare/ieşire. Dacă se dispune de linii de transmisie rapide şi display-uri cu posibilităţi de adresare a ecranului, se pot utiliza şi limbaje de interogare de "tip meniu", foarte simplu de utilizat şi recomandate la BD de tip relaţional. In acest mod se pot interoga şi actualiza rapid BD aflate la mare distanţă şi simultan de mai mulţi utilizatori, asigurîndu-se protecţia şi securitatea informaţiei.

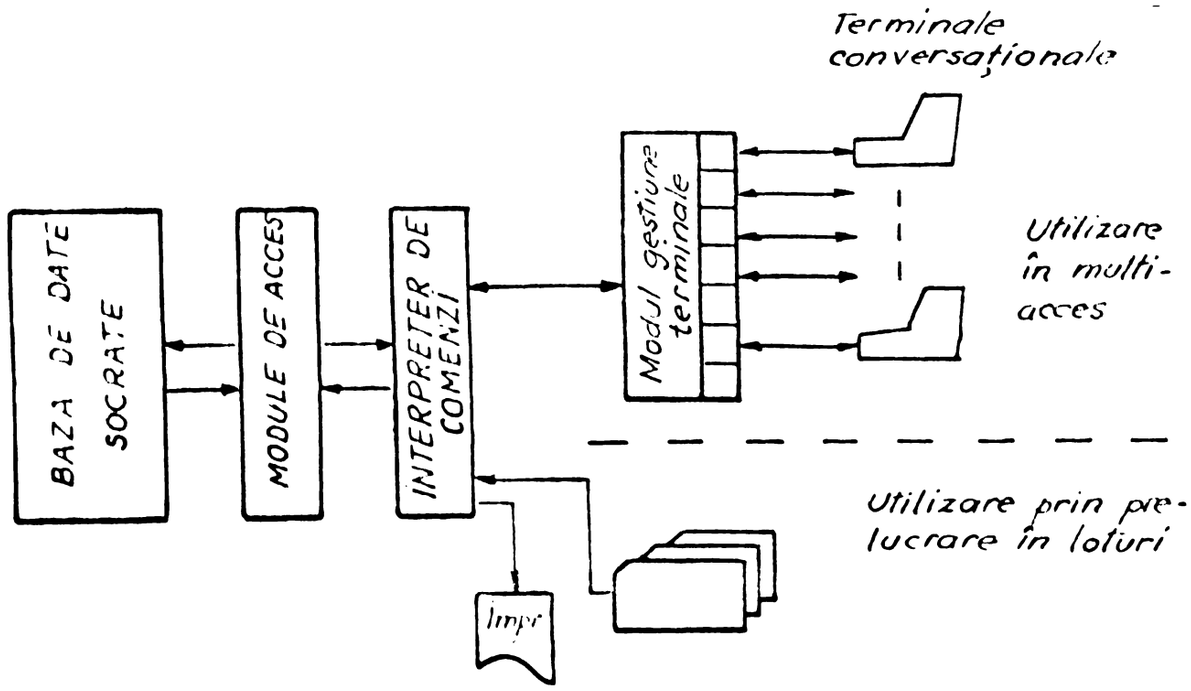


Fig.1.13. Multiaccesul conversaţional la BD tip SOCRATE.

1.2.3. Subsisteme cu multiacces pentru testarea conversaţională a programelor.

Deoarece limbajele conversaţionale pot fi folosite eficient numai pentru programe scurte şi de complexitate redusă, care în general nu utilizează fişiere (limitări impuse de interpreter), s-au conceput subsisteme cu multiacces, care permit crearea şi modificarea unor programe sursă în mod conversaţional. După ce lucrările în format sursă au fost create, ele pot fi transmise symbiontului, care le plasează într-o clasă de multiprogramare şi le execută într-o partiţie serie alături de lucrările din prelucrarea în loturi. Rezultatele obţinute într-o categoria a symbiontului sînt recuperate de subsistemul conversaţional şi afişate la terminale. Se permite astfel accesul conversaţional de la distanţă, pentru mai mulţi utilizatori, care dispun de toate facilităţile de prelucrare existente în prelucrarea în loturi : compilatoare puternice şi variate, editor de legături, biblioteci, inventare de fişiere, etc.

Lucrul în aceste subsisteme este numai parţial conversaţional. Execuţia programelor se face în prelucrarea în loturi, fără a permite intervenţia operatorului. Ei îi cunoştinţă de rezultate şi erori după terminarea programului, care se face mai repede sau mai tîrziu funcţie de prioritatea lucrărilor, de numărul de utilizatori care folosesc acest serviciu şi de timpul de execuţie al acestor lucrări. Subsistemul lucrează în multiacces numai pentru punerea la punct a programelor, dar nu şi în execuţie. Vom prezenta din această categorie subsistemul ARIEL.

Subsistemul ARIEL [Ar 80] este o dezvoltare a subsistemului SIT (Soumission Interactiv du traveux) [Si 75] realizat în cadrul I.C.I. Bucureşti şi destinat calculatoarelor ELIX 256/512, care lucrează sub sistemul de operare SIRIS-3. El presupune existenţa symbiontului pentru planificarea şi introducerea lucrărilor în sistem în regim de multiprogramare şi o reţea de terminale de teletransmisie, care lucrează în mod caracter (există şi o variantă pentru mod mesaj). Subsistemul ARIEL dispune de un subset de comenzi, accesibile utilizatorilor aflaţi la distanţă, care constituie o extindere a limbajului de comandă pentru modul de lucru conversaţional şi pentru intersecţiunea cu symbiontul.

Sistemul de calcul lucrează normal în multiprogramare cu mai multe partiţii de execuţie serie (regiuni), o partiţie pentru subsistemul ARIEL (> 70 K) şi o partiţie pentru symbiont. Symbiontul asigură introducerea şi extragerea lucrărilor din sistem, asigurînd planificarea lucrărilor care se ordonează la intrare pe clase (şiruri de aşteptare) şi la ieşire pe categorii. Symbiontul controlează toate cititoarele de cartele şi imprimantele existente în sistem local. Terminalele conversaţionale sînt gestionate de subsistemul ARIEL printr-un modul de dialog ce lucrează în multitasking. (fig.1.14).

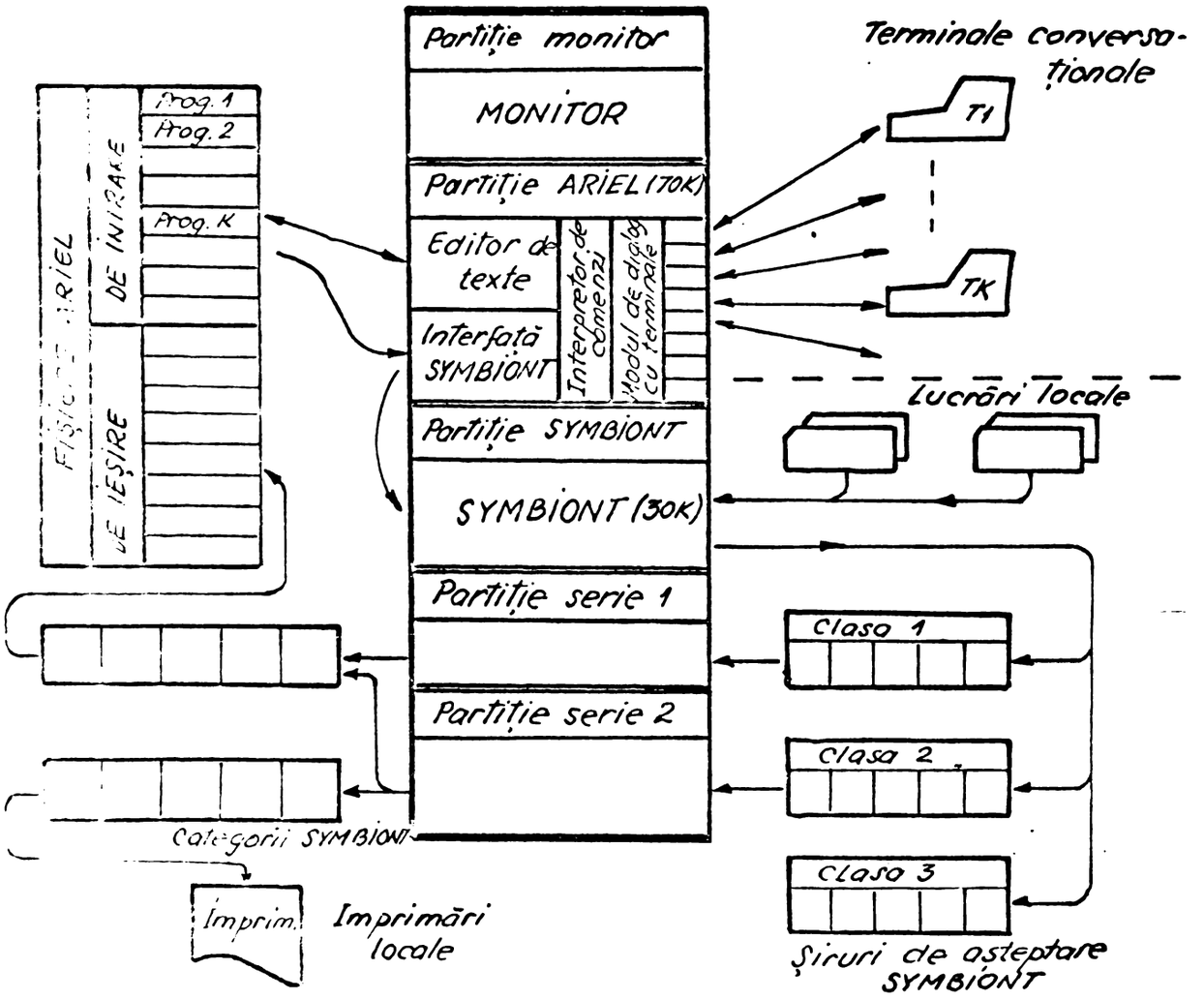


Fig.1.14. Introducerea interactivă de lucrări prin subsistemul ARIEL.

Acest modul transmite pe rînd mesajele primite de la utilizatori spre interpreterul de comenzi, care le analizează sintactic şi semantic. Comenzile de editare sînt tratate de către editorul de texte, care permite fiecărui utilizator să creeze unul sau mai multe programe sursă (inclusiv cartele de comandă) pe un fişier dintr-o zonă partajată proprie. Aceste programe pot fi ulterior completate, modificate, listate, sau şterse prin editorul de texte, care asigură identificarea unui program după nume, căutarea unei linii după număr sau prin context.

După terminarea editării utilizatorul poate cere ca programul să fie transmis symbiontului, care îl pune într-un şir de aşteptare (clasă), cu o anumită prioritate, pentru a fi rulat într-o partiţie serie. In timpul rulării utilizatorul nu poate interveni asupra programului, dar el poate cere informaţii despre starea sa (în aşteptare, în execuţie, terminat) şi despre lungimea şirului de aşteptare în care se găseşte. Interfaţa cu symbiontul este asigurată de un modul special al subsistemului ARIEL. La terminarea programului listingul rezultat se va recupera din fişierele symbiont şi va fi trecut într-un fişier de ieşire ARIEL accesibil utilizatorului de la terminal. El poate cere afişarea lui totală sau parţială. Dacă au fost greşeli în programul sursă se pot opera noi corecturi cu editorul de texte şi se poate cere o nouă rulare. Fişierele de intrare şi ieşire care nu mai sînt necesare pot fi şterse.

Lucrul conversaţional în subsistemul ARIEL este numai pe perioada pregătirii programului sursă, utilizatorul nu poate "vedea" evoluţia programului său în timpul execuţiei şi nu poate interveni în această fază, cum este posibil la utilizarea limbajelor conversaţionale. El poate totuşi să activeze de la distanţă programe complexe, care necesită resurse variate, care nu pot fi altfel exploatate conversaţional. O limitare a subsistemului constă în imposibilitatea trecerii programelor sursă ARIEL în biblioteci sursă prin comandă conversaţională. Ele pot fi copiate în biblioteci standard numai cu o variantă a subsistemului care lucrează în "batch" (CARIEL). Subsistemul este deosebit de util în faza de punere la punct a programelor în mod conversaţional asigurînd confort şi productivitate.

1.2.4. Subsisteme cu multisecces pentru execuţia interactivă a programelor

Datorită complexităţii interacţiunilor dintre program şi sistemul de operare în timpul execuţiei, majoritatea subsistemelor cu multisecces evită faza de execuţie a programului, dacă el a fost obţinut prin compilare normală. Faza de execuţie se face în general printr-o lansare într-o prelucrare în loturi. Dificultăţile apar la încărcarea programului IMT, tratarea informaţiilor SGF, dialogul continuu cu utilizatorul, tratarea segmentelor, tratarea erorilor, protecţia fişierelor, divizarea timpului UC, asigurarea "swaping-ului" programelor în aşteptare. Majoritatea contribuţiilor personale din lucrarea de faţă sînt în acest domeniu şi vor fi prezentate în cap.4 unde se prezintă subsistemul SCOT, realizat după o concepţie originală.

Unul dintre primele sisteme de operare cu multisecces, pentru execuţie interactivă a programelor a fost implementat în 1962 la Institutul Tehnologic din Massachussetts pe un calculator IBM 7090, sub numele CTSS (Compatible Time Sharing System) [Hal 75]. Orice program ce putea rula în "batch" putea rula fără modificări în "time sharing". Calculatorul IBM/7090 avea 32 K memorie pentru supervizor şi 32 K memorie pentru programele utilizator. La un moment dat în memorie exista un singur program iar celelalte se găseau pe disc sau tambur (fig. 1.15). Sistemul lucra numai în "time sharing", pentru utilizatori aflaţi la terminale conversaţionale multiplexate printr-un calculator de comunicaţie IBM/7750. Swaping-ul programelor se făcea parţial, numai pentru partea acoperită de programul încărcat(" onion skin "). Divizarea timpului între utilizatori se făcea după metoda "Carusel multiplu" utilizînd 8 şiruri de aşteptare funcţie de timpul de lucru utilizat deja (vezi 1.1.3) Programele scurte aveau astfel prioritate mai mare. Primul şir avea prioritate maximă şi i se aloca o cuantă de timp C. Dacă nu se termina programul la prima rulare, trecea în şirul de nivel următor cu prioritate mai mică dar cu o cuantă 2C ş.a.m.d. Sistemul realiza astfel divizarea timpului între utilizatori, dar nu accepta lucrări locale şi nici multiprogramare. Se asigura compatibilitatea cu programele rulate normal. Problemele ridicate la concepţie au fost de dificultate redusă deoarece

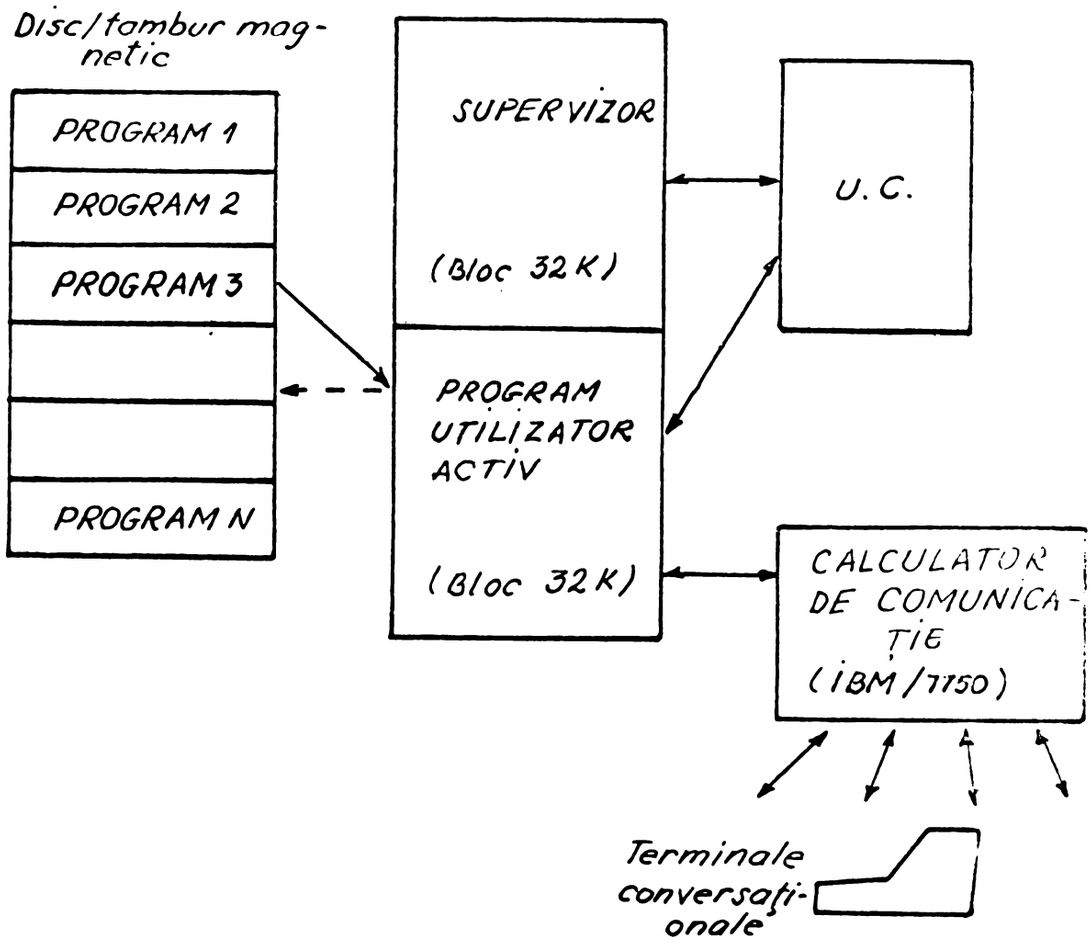


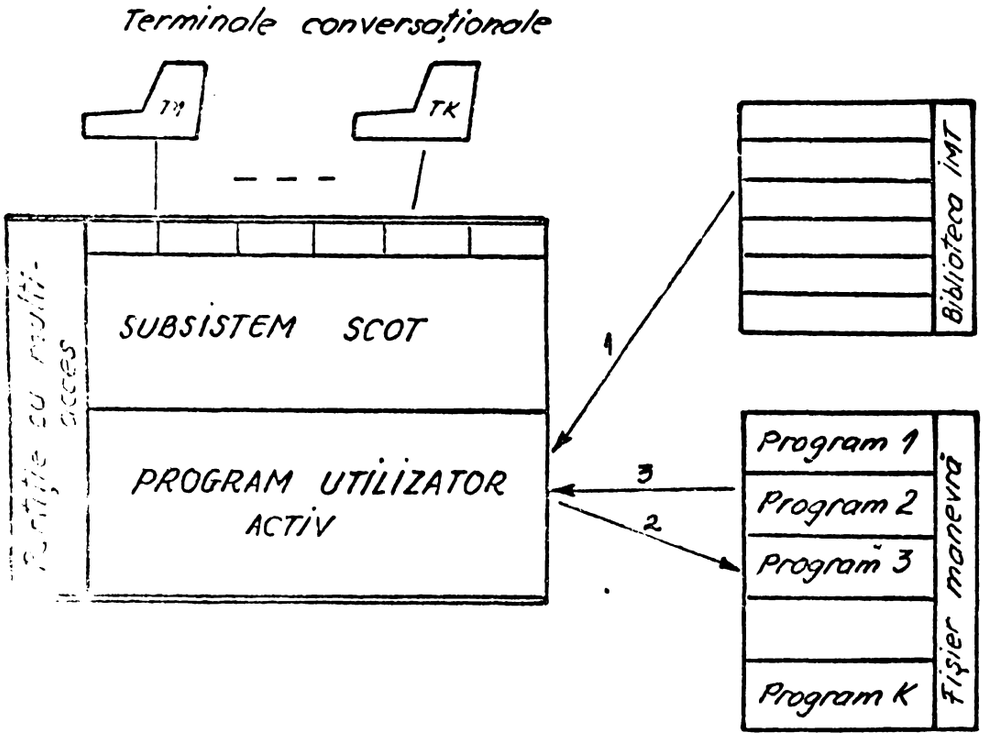
Fig. 1.15. Structura sistemului CTSS - MIT.

configurația sistemului era simplă și protecțiile realizate de sistemul de operare privind gestionarea resurselor erau aproape inexistente. Nu se divizează spațiul de memorie între programe și nu se realocă dinamic programele.

Odată cu apariția sistemelor de operare cu multiprogramare, la gen.III de calculatoare, configurația sistemelor și arhitectura sistemului de operare s-au complicat substanțial. S-au introdus mai multe partiții protejate între ele, protecția monitorului și instrucțiii privilegiate, protecția și gestiunea perifericeloor, protecția fișierelor, compilatoare puternice cu fază de editare de legături, limbaj de comandă. Toate aceste facilități permiteau o încărcare puternică a resurselor și o siguranță sporită în exploatarea programelor în prelucrarea în loturi. Implementarea unor subsisteme care să permită în multiacces execuția intere-

tivă a programelor a devenit foarte dificilă.

Subsistemul SCOT (subsistem conversaţional de teleprelucrare) realizat de autor cu sprijinul ing. Holban St. şi mat. Petru P., permite execuţia interactivă a programelor IMT, rezultate din compilarea în prelucrare în loturi a programelor sursă scrise în limbajul FORTRAN sau COBOL pe calculatoarele FELIX 256/512/1024. Pentru a se putea scrie şi exploata programe conversaţionale de către programatori obişnuiţi, care cunosc numai limbaje de nivel înalt, s-a plecat de la ideea modificării modulelor de acces la fişierele sistem de intrare/ieşire din programele utilizator în momentul editării legăturilor. Aceste module modificate vor constitui interfaţa între programele utilizator şi subsistemul SCOT (fig. 1.16). Subsistemului SCOT i se alocă o partiţie în care se lucrează în execuţie (ocupă 30 K) indicînd terminalele care vor putea fi activate. In partiţie se lasă un spaţiu liber egal cu lungimea celui mai lung program ce se va rula. In acest spaţiu la un moment dat se va găsi programul utilizator activ.



1. încărcare program din bibliotecă
2. salvare program în aşteptare pe fişierul de manevră
3. încărcare program activat de pe fişierul de manevră

Fig. 1.16. Organizarea subsistemului SCOT.

Programele utilizator se găsesc depuse într-o bibliotecă în format LMT. Orice utilizator aflat la terminal poate începe un dialog cu subsistemul, în multiacces, cerînd lansarea în execuţie a programului indicat. Programul indicat va fi selectat, încărcat în spaţiul liber din partiţie şi va fi lansat în execuţie. Cînd programul aşteaptă terminarea unei operaţii de I/E de la terminal, el va fi trecut pe un fişier pe disc, iar în locul lui se va încărca un alt program care este pregătit, exploatat de alt utilizator. In acest mod toate programele se activează pe rînd. Datele cerute pe cartele vor fi introduse de la terminal, iar rezultatele ce ar fi trebuit listate se afişează pe terminal. Utilizatorul poate interveni oricînd în desfăşurarea programului pe care o urmăreşte pe ecranul terminalului. Datele care doreşte să le reţină, se pot copia pe imprimante recopistoare de ecran. Programele utilizator pot conţine fişiere şi pot fi segmentate. In caz de eroare se va termina numai programul care a produs eroarea.

Realizarea subsistemului pentru a lucra sub sistemul de operare SIRIS-3, în regim de multiprogramare complexă a necesitat rezolvarea unor probleme dificile dintre care amintim :

- dialogul în multitasking cu utilizatorii ;
- încărcarea programelor IMA la adrese de memorie variabile ;
- tratarea informaţiilor SOF la încărcarea programului şi la fiecare închidere sau deschidere de fişiere ;
- tratarea unitară a erorilor din programele utilizator pentru a nu afecta subsistemul şi ceilalţi utilizatori ;
- divizarea timpului între utilizatori ;
- tratarea specială a programelor segmentate ;
- salvarea şi reîncărcarea programelor pe disc.

Modul concret de rezolvare a acestor probleme va fi prezentat pe larg în cap. 4, constituind contribuţii personale. Nu s-au încărcat mai multe programe simultan, deoarece spaţiul de memorie în general nu permite, iar acestea nu pot fi protejate separat, lucrînd în aceeaşi partiţie. S-ar permite astfel simultaneitatea "swepingului" cu lucrul în UC. Timpul de sweping este folosit de SO în folosul altei partiţii mai puţin prioritare, care lucrează în prelucrarea în loturi, asigurîndu-se şi pentru aceste programe un timp de răspuns acceptabil. Calculatoarele de capacitate medie nu se folosesc numai pentru multiacces ci în regim

de multiprogramare complexă, asigurându-se prioritate mai mare programelor conversaţionale.

1.2.5. Subsistemul cu multiacces IBM/360/370 TSO

Subsistemul IBM Time Sharing Option (TSO), lansat în 1971, asigură extinderea sistemului de operare OS/360 în domeniul teleprelucrării în multiacces. Dispune de un limbaj de comandă conversaţională, asemănător cu cel folosit în prelucrarea în loturi (JCL), căruia i s-au adăugat facilităţi noi [Hel 75] . Este cel mai complet subsistem conversaţional realizat pentru calculatoarele de capacitate medie, care nu presupune memorie virtuală. A necesitat un efort de concepţie evaluat la 1.000 ani x om faţă de 5.000 ani x om cerut de întregul sistem de operare OS/360. Se îmbină serviciile oferite de subsistemele cu multiacces analizate anterior, dar cu modificări substanţiale ale componentelor sistemului de operare.

TSO asigură accesul interactiv al utilizatorilor la anumite limbaje, pentru care s-au conceput interpretare cu facilităţi speciale. În 1975 erau disponibile ITF/BASIC, ITF/PL/1 şi ITF/FORTRAN . În plus se permite lansarea de la terminal a unor programe sau lucrări ce se vor executa într-o partiţie serie, ce lucrează în paralel cu subsistemul TSO.

Utilizatorul de la terminal apelează funcţiile subsistemului prin limbajul de comandă, extins cu facilităţi conversaţionale, care include :

- Comenzi de editare pentru introducere, memorare, afişare şi modificare a informaţiilor (programe sursă, date sau seturi de comenzi) din fişierele proprii. Facilităţile de editare includ referirea prin număr de linie şi prin context.
- Comenzi de specificare a numelor simbolice şi periferiilor asignate fişierelor (ASSIGN, LABEL).
- Comenzi de execuţie a programelor în varianta interactivă sau într-o partiţie serie.

Se permite definirea unor proceduri de comandă, ca o secvenţă de comenzi parametrizate, care pot fi apelate prin nume. În momentul apelului se indică valoarea parametrilor efectivi ca la subrutine . Pe lîngă limbajele conversaţionale se pot folosi în rulare serie compilatoarele normale FORTRAN, COBOL, PL/1, ALGOL şi ASSEMBLER, activate de la terminal. Programele rulate în serie pot fi urmărite de la terminal prin comanda TEST, care permite oprirea şi repornirea programului, cu indicarea adresei unde a ajuns (trace). Utilizatorul va face corec-

pondenţe dintre adresele fizice şi adresele simbolice. El nu poate modifica datele în timpul execuţiei în serie şi nici nu cunoaşte rezultatele parţiale. Listingul programului îi va fi accesibil după terminarea rulării într-un fişier special.

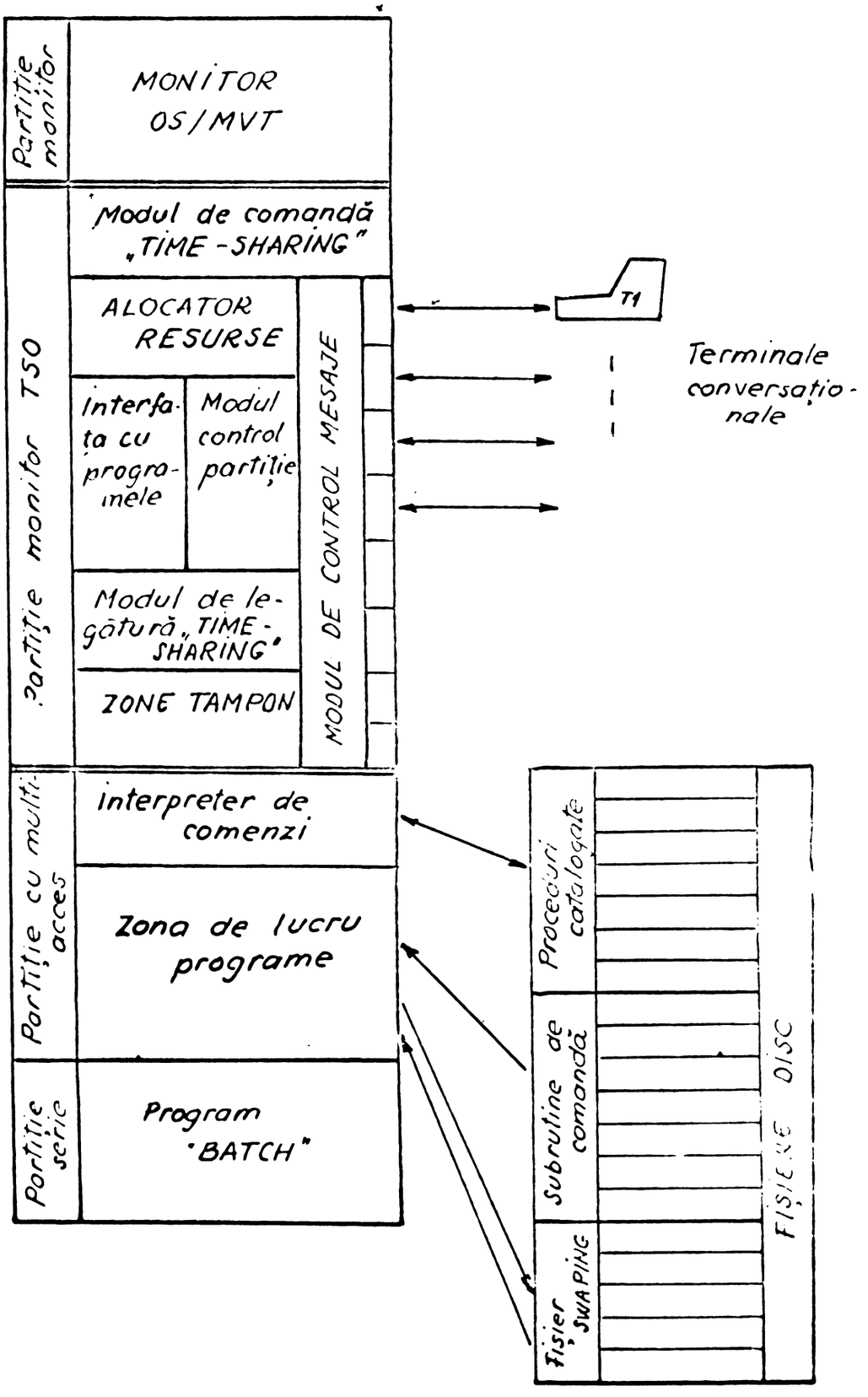


Fig.1.17. Organizarea subsistemului în /360 "TI...-SHARING OPTION" (TSO)

Structura subsistemului IBM-TSO este prezentată în Fig.1.17. Lucrînd sub sistemul de operare cu partiţii variabile OS/MVT, ocupă o partiţie pentru modulele de comandă şi una sau mai multe partiţii paralele unde se execută programe în regim de multisecces. Fiecărei partiţii de multisecces îi corespunde cîte un modul de control a partiţiei şi un interpreter de comenzi rezident în cadrul partiţiei. Interpreterul de comenzi primeşte mesajele de la terminale, le analizează sintactic, verifică drepturile de acces ale utilizatorului şi selectează subprogramele de comandă, care vor executa funcţiile cerute şi vor lucra în zona de lucru. Utilizatorul poate cere execuţia unor proceduri catalogate anterior, care conţin secvenţe de comenzi parametrizate. Fiecare utilizator poate dispune de una sau mai multe proceduri pe care interpreterul le va executa la cerere numai pentru utilizatorul căruia îi aparţin. Comenzile pot cere execuţia unor programe cărora li se vor aloca la rulare anumite resurse.

Toate resursele necesare pentru lucru în multisecces sau într-o partiţie serie, pentru utilizatorii de la terminale, se alocă printr-un modul special "alocatorul de resurse". Se poate cere utilizarea de fişiere sau partiţii serie care se alocă dinamic prin OS/MVT. Programele care lucrează în multisecces folosesc pentru rulare aceeaşi zonă de memorie, care este gestionată prin "swaping" utilizînd un fişier disc special. Controlul operaţiei de "swaping" este asigurat de către modulul de comandă "time-sharing", care decide mişcarea programelor între MC şi fişierul de swaping. Se utilizează unul sau mai multe şiruri de aşteptare pe baza cărora fiecărui program pregătit i se alocă o cuantă de timp, după care este întrerupt. Depunerea programului nu se face înainte de terminarea tuturor operaţiilor de I/S în curs de execuţie.

Pentru toate programele în lucru se păstrează în MC permanent cîte o zonă tampon de comunicaţie cu terminalul, care se poate face simultan pentru toţi utilizatorii chiar dacă programul este depus pe disc în aşteptare. Toate operaţiile de dialog cu terminalele se fac centralizat în multitasking de către "modulul de control mesaje", care comunică cu programele utilizator printr-un modul de interfaţă şi prin interpreterul de comenzi. Pentru anumite funcţii ale subsistemului mai puţin utilizate, este prevăzută o zonă de memorie, "modul de legătură time-sharing", în care subrutinele corespunzătoare sînt încărcate şi

lansate în execuţie. Diferitele module pot folosi în comun un număr de zone tampon.

Planificarea resurselor foloseşte două cuante de timp :

- cuanta de timp majoră, este timpul dintre o încărcare a programului în MC şi următoarea depunere ;
- cuanta de timp minoră, este timpul maxim continuu de UC alocat unui program încărcat în MC.

Planificarea se face după cuanta de timp majoră utilizînd algoritmul "carusel simplu" ("round robin") pentru fiecare şir de lucrări în aşteptare. Unii parametri de planificare au valori stabilite de caracteristicile constructive ale sistemului de calcul, iar alţii pot fi modificaţi dinamic. Parametrii modificabili pentru un şir q sînt :

A_q - timpul mediu de aşteptare pentru lucrări în şir, exprimat prin timpul dintre intrarea şi ieşirea din şir ;

N_q - numărul de lucrări pregătite de rulare din şir ;

M_q - minimul cuantei majore de timp, care arată că anumite lucrări din şir stau în memorie mai mult decît lucrează efectiv;

SL_q - este spaţiul maxim de memorie pe care lucrările din şir îl pot cere (swap-load limit);

IL_q - timpul limită de lucru în memorie solicitat de utilizator (interaction time limit);

SC_q - numărul de cicluri de deservire ce se efectuează pe şir înainte de a trece la alt şir;

T_q - cuantă de timp majoră alocată pentru lucrările din şir.

Folosind aceşti parametri lucrările sînt ordonate în mai multe şiruri de aşteptare (clase). Cuanta de timp majoră pentru o clasă se calculează cu formula :

$$T_q = \text{Max}\left(M_q, \frac{A_q}{N_q}\right)$$

Fiecare program din şirul q este păstrat în memoria centrală un timp T_q, după care este trecut în şir conform algoritmului "carusel simplu". După SC_q cicluri de servire în şirul q se trece la servirea şirului următor. Parametrii SL_q şi IL_q servesc la detectarea programelor care depăşesc posibilităţile clasei q (memorie prea mare) şi care vor fi trecute

în altă clasă care îndeplineşte condiţiile cerute. Şirul trebuie să fie în aceeaşi partiţie (regiune) deoarece OS/360 nu permite alocare dinamică şi nici trecerea unui program lansat în altă partiţie. Programele mai lungi vor fi lansate mai rar, dar având cuante de timp mai lungi pentru a reduce timpul inactiv de "swaping".

La un moment dat UC este afectată unui program dintr-o partiţie cu multiacces pentru o cuantă de timp minoră, de către dispecerul TSO. Aceste partiţii au prioritate faţă de cele serie, dar trebuie să asigure celor serie un procentaj minim de UC. Timpul disponibil pentru partiţiile paralele se împarte în cuante minore după trei procedee (selectate de programatorul de sistem):

a). Distribuirea simplă consideră cuanta minoră egală cu timpul pentru partiţiile paralele şi îl alocă programului mai prioritar. Metoda reduce "swaping-ul" şi se utilizează întotdeauna când se lucrează cu o singură partiţie paralelă.

b). Distribuţia egală împarte timpul pentru partiţiile paralele la numărul de programe pregătite încărcate în partiţii paralele.

c). Distribuţia ponderală calculează cuante de timp minoră pentru fiecare program ţinînd cont de comportarea sa anterioară , de procentul şi lungimea aşteptărilor de I/E.

2. TEHNICI DE PROIECTARE A SUBSISTEMELOR CU MULTIACCES

2.1. PROGRAMAREA CONCURENTA IN LIMBAJE DE ASAMBLARE

Subsistemele cu multiacces sînt programe de complexitate mai mare, care necesită metode de programare speciale, ce permit execuţia simultană a mai multor procese în aceeaşi partiţie, comunicarea între procese şi sincronizarea lor. Procesele sînt activităţi secvenţiale , care în anumite condiţii pot fi executate în paralel şi pot avea acces la resurse comune partajabile, dar la un moment dat numai unul din procese poate utiliza acea resursă. Celelalte procese care solicită aceeaşi resursă vor fi puse într-un şir de aşteptare şi deservite în ordinea cererilor. Se spune în acest caz că se execută sincronizarea proceselor. Ordinea de execuţie a proceselor se poate reprezenta printr-un graf, care exprimă intercondiţionarea proceselor, deoarece desfăşurarea unui proces poate fi condiţionată de terminarea altuia, care îi furnizează date (citire - scriere, citire - prelucrare, prelucrare - scriere, etc.) Două sau mai multe procese care pot solicita simultan utilizarea aceleaşi resurse (UC, zonă de memorie, procedură, acces la un periferic, la un disc, la un fişier, la un terminal, etc) şi pot coopera între ele se numesc procese concurente. Metoda de programare care rezolvă problemele proceselor concurente se numeşte programare concurentă.

Limbajele de nivel înalt clasice (COBOL, FORTRAN, PL/1, ALGOL) nu permit rezolvarea concurenţei proceselor şi ele exprimă activităţi pur secvenţiale care se succed. In execuţie ele vor reprezenta un proces unic. Pentru scrierea programelor concurente se utilizează limbajele de asamblare, care oferă toate posibilităţile de programare de care dispune sistemul de calcul. Programarea în limbaj de asamblare dă programe de performanţă maximă (viteză ridicată, consum redus de memorie, facilităţi speciale), dar necesită un efort de programare important şi programatori cu multă experienţă. Din acest motiv în ultimii 10 ani au apărut limbaje de programare concurentă de nivel înalt (PASCAL CONCURENT, FORTAL, ADA), care permit gestionarea proceselor, rezolvarea concurenţei proceselor şi cooperarea lor prin construcţii speciale. Apariţia limbajelor concurente este legată de dezvoltarea teleprelucrării şi aplicaţiilor calculatoarelor utilizate în conducerea proceselor industriale.

Posibilităţile oferite de limbajele de asamblare pentru scrierea unor programe sau subsisteme concurente depind de la un sistem de operare la altul. Sistemele de operare clasice cu multiprogramare (IBM/OS/360 MVT, FELIX C256/512 SIRIS 2,3) oferă posibilităţi reduse doar la scrierea unor programe care utilizează un dialog în multiacces cu terminale de teletransmisie (SGT) sau gestiunea unor procese simple (SGA). In sistemul de operare SIRIS-3 există un subsistem de gestiune a teletransmisiilor (SGT), care asigură un multitasking restrîns numai pentru terminalele în mod caracter, sau terminale în mod mesaj legate pe linii directe (fără concentrator).

SGT-ul asigură gestiunea proceselor (taskuri), care sînt asociate cîte unei linii de teletransmisie, prevăzută cu o tabelă de control (LCB), utilizată de modulul de multitasking pentru comunicarea cu programul şi pentru a salva informaţiile proprii procesului (registre generale, adresă de reluare, etc). Fiecărui task i se asociază o procedură şi o prioritate. Procedurile asociate taskurilor trebuie să fie reentrante şi mai multe taskuri pot utiliza aceeaşi procedură. Procedura poate conţine macroinstrucţii de intrare-ieşire, care pun temporar în aşteptare taskul. Pe această perioadă se lansează un task, mai puţin prioritar, pregătit, care poate să lucreze chiar pe aceeaşi procedură cu taskul întrerupt. La reluarea taskului întrerupt se refac registrele generale şi se comunică procedurii care este taskul activ, prin încărcarea în registrul 13 a adresei tabelei lui de control.

Pentru a asigura reentranţa procedurilor, într-un mod cît mai simplu se recomandă [Jia 82] plasarea zonelor de date proprii taskului imediat după tabela de control (LCD). Pentru fiecare procedură se va descrie o secţiune fictivă, corespunzătoare structurii unei zone de date, avînd în faţă o tabelă de control fictivă (fig. 2.1.). Adresa procedurii asociate unui task se dă la activarea taskului (ACTP). Activizarea multitaskingului se face prin macroinstrucţia BEGP, iar terminarea procedurii pentru un task se indică prin ENDP. Celelalte taskuri active lucrează în continuare pînă cînd toate au atins sfîrşitul procedurilor, moment în care multitaskingul s-a terminat.

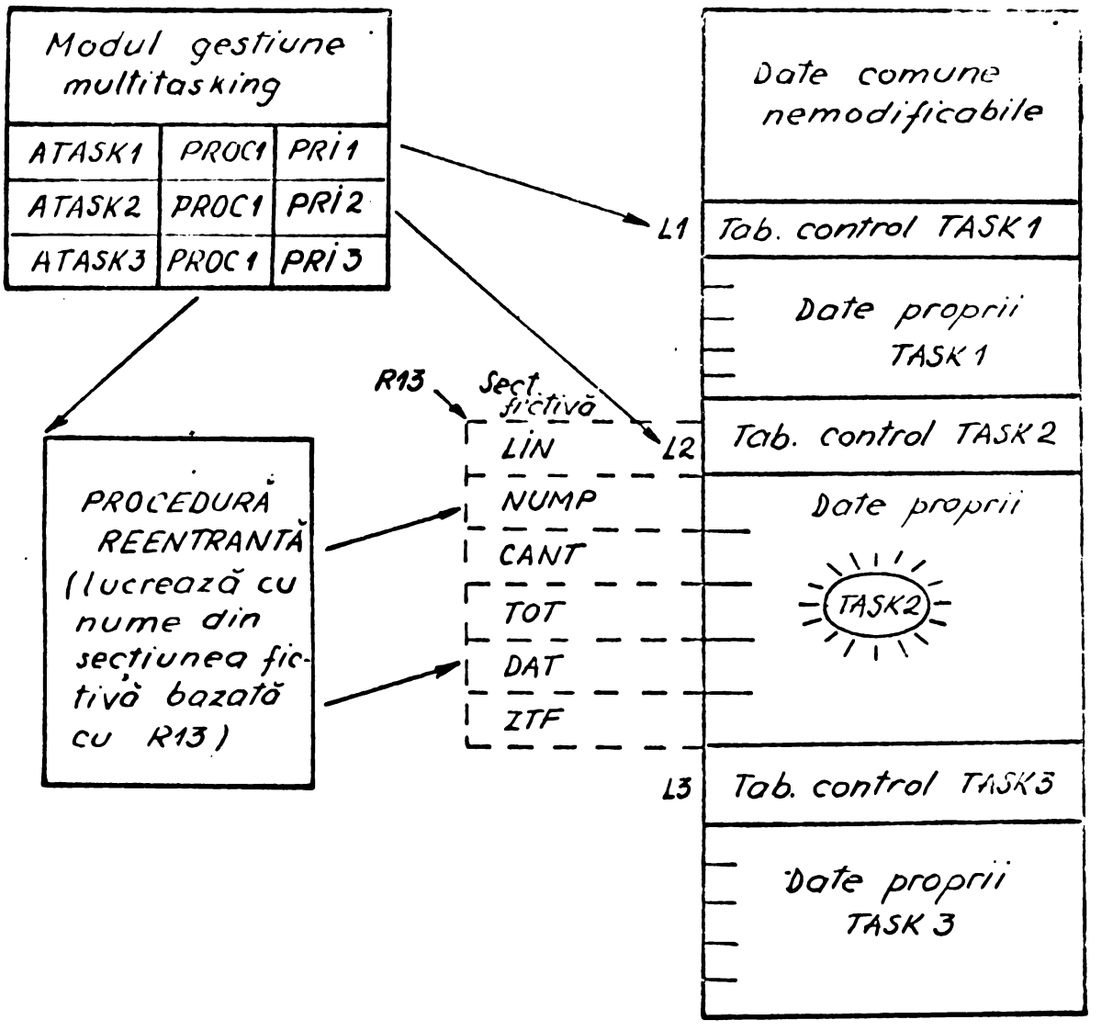


Fig.2.1. Structura unui program ce lucrează în multiacces.

Dăm în continuare o schiță de program corespunzătoare fig.2.1 ce conține 3 taskuri și o singură procedură comună în care se deservesc utilizatori de teletransmisie ce au acces la un fișier nedefinit pe disc. Operațiile de trimitere (SEND) și recepție de mesaje (RECEIVE) sînt urmate de instrucțiuni CHKT, care dau comanda modulului de multitasking în perioada de așteptare a terminării transferului, pentru a lansa alt proces pregătit. Transferurile în curs sînt deservite simultan prin multiplexare de către un modul HARD... de teletransmisie din supervisorul de intrare-ieșire.

```
*        SECTIUNEA FICTIVA
        DSECT LDSCT
LIN     ...     164
NUMP    ...,4     2
CANT    ...,4     1
TOT     ...,4     1
DAT     ...       12
ZTF     RES     1024

*PROGRAM PRINCIPAL *
        CSECT P
        TRANSFER TASK:3
```

```
* PROCEDURA ...ENTRANTA
PROC 1  LD4,14 *
        ...,DSECT 14
        SEND CH,P...G,BUF:LJ...
        CHKT
        :
        RECEIVE CH,P...G:CANT,
                LGH:20
        CHKT
        :
```

```
FIS    DFD    A1, DVT:AD,....          SEND   CN,PRG,BUF:TOT
GRUP   LGD    TCU:A02,DEV:TY01,        CHKT
               LCB:L1,L2,L3             :
          \    Date comune              .
           \   nemodificabile           BRSBRT      CITD
L1     LCB  TT01,LGD:GRUP              RECEIVE CN,PRG:DAT,LGH:12
       RES  144+28+1024                CHKT
L2     LCB  TT02,LGD:GRUP               :
       RES  144+28+1024                 .
L3     LCB  TT03,LGD:GRUP    CITD      DPSBRT  CIT
       RES  144+28+1024                LD4I,6  ZTP
                                       READ    FIS,CCB:1,BFS:1024
START  OPEN GRUP,FIS                   CHKL    FIS
       ACTP L1,PRO:PROC1,PRI:5          :
       ACTP L2,PRO:PROC1,PRI:2         RTSBRT
       ACTP L3,PRO:PROC1,PRI:4
*LANSARE MULTITASKING                  ENDP
       BEGP
       CLOSE  GRUP,FIS          *     SFIRSIT PROCEDURA
       TERM
```

Subrutine protejate

Deoarece şi citirea unui bloc de pe disc produce o întîrziere, timpul de transfer e utilizat în folosul altui task realizat prin macroinstrucţia CHKL. Noul task lansat poate ajunge şi el să ceară citirea discului, înainte de a se termina citirea precedentă. Aceasta ar duce la conflict în accesul la resurse comune (eroare). Pentru a se evita această situaţie, secvenţa de citire disc se protejează printr-o subrutină protejată definită prin DPSBRT, apelată prin BRSBRT şi terminată prin RTSBRT. Apelarea subrutinei protejate o blochează şi nu permite accesul altui task pînă nu s-a efectuat RTSBRT, care o eliberează. Această metodă reglementează accesul la resurse critice, ca un semafor simplu.

Comunicarea între taskuri este posibilă prin sistemul "cutie poştală". Fiecărui task i se poate adăuga la tabela de control un şir de aşteptare, care conţine adrese de mesaje. Fiecare task poate pune în şirul de aşteptare a altuia adresa unui mesaj prin instrucţia ENQ şi poate lua secvenţial (FIFO) adresele mesajelor transmise de alte taskuri. Ele pot fi prelucrate sau afişate la terminal.

Accesul la fişiere în regim de multitasking este limitat la fişierele cu acces direct, la care adresa înregistrării este dată fie prin adresă fizică (fişiere nedefinite) fie prin adrese simbolice (fişiere selective şi secvenţial indexate). Această limitare apare deoarece există un singur modul de acces la un fişier pentru toţi utilizatorii. La accesul secvenţial aici se ţine evidenţa adresei curente. La un nou apel din program se ia înregistrarea următoare, fără a şti care proces a cerut citirea. Fişierele secvenţiale se pot exploata în multiacces dacă se tratează ca fişiere nedefinite, ţinînd cont de modul de blocare al articolelor. Dacă se actualizează fişiere standard în acces direct trebuie prevăzut argumentul MDF : SbC (security), pentru ca orice modificare să se opereze imediat şi nu la terminarea tuturor modificărilor pe bloc. Procesul următor lansat poate distruge complet zona tampon a fişierului şi poate folosi altă zonă tampon. Utilizarea mai multor fişiere diferite de către fiecare utilizator, de la terminale, pune probleme de număr de intrări disponibile în fişierul de comunicaţie SCF şi suporturi de fişier disponibile. Se recomandă în acest caz exploatarea dinamică a fişierelor, care se realizează cu dificultăţi (vezi cap.4).

Facilităţi de implementare a subsistemelor cu multiacces în sistemele de operare moderne

Subsistemele de operare cu multiprogramare proiectate mai recent [Hel 81] ,cum este HELIOS pentru calculatoarele FELIX 8010/20, ţin cont de necesitatea existenţei unor macroinstrucţii care să permită implementarea unor subsisteme cu multiacces performante. In plus toate <u>componentele sistemului de operare sînt reentrante</u>, putînd fi uşor partajate între procese. Din acest motiv toate partiţiile pot fi cu înlănţuire utilizînd acelaşi interpreter de comenzi. Partiţiile sînt văzute ca formînd <u>grupe de procese</u> care se pot crea şi distruge din alte procese. Rădăcina programului încărcat conţine procesul rădăcină, din care se pot crea ierarhic alte procese. Partiţia este controlată de distribuitor printr-o tabelă de control a lucrării (TCT), iar fiecare proces are o tabelă aparte de control a procesului PCB (Process Control Block). Din orice proces se pot crea alte procese în aceeaşi partiţie prin macroinstrucţia CREA, se pot distruge procese prin DSTR sau activa procese prin ACTV. Procesele reprezentînd activităţi diverse pot rula în paralel, dar cel mai adesea sînt interdependente.

SEMAFOARE.

Pentru sincronizarea şi comunicarea proceselor s-au prevăzut macroinstrucţii speciale care realizează funcţiile P şi V pe semafoare pentru utilizarea resurselor critice într-o anumită ordine serială. Accesul la resursă se face printr-o operaţie P, care ocupă resursa dacă e liberă, iar dacă nu procesul este pus într-un şir de aşteptare al semaforului. Resursa se eliberează printr-o operaţie V, permiţînd următorului proces din şirul de aşteptare să ocupe resursa şi să devină pregătit [Sed 74] . Utilizarea semafoarelor şi operaţiilor P şi V cu mesaje permit uşor sincronizarea proceselor de tip " producător - consumator ".

In sistemul RELIOS s-au implementat mai multe variante de funcţii P şi V pe semafoare cu sau fără mesaj. Un semafor (fig. 2.2.) cuprinde în general 2 cuvinte. Pe primii doi octeţi (VAL) este valoarea semaforului (numerică). Tipul semaforului se specifică pe biţii 4,5 din octetul 2 :

00 - semafor fără mesaje
10 - semafor cu mesaje şi modul de acces LOCATE
11 - semafor cu mesaje

Organizarea şirului de aşteptare a semaforului se dă în biţii 6 şi 7 :

0 0 - Ordonare după momentul sosirii (FIFO)
01 - Ordonare după prioritatea proceselor (PRTP)
10 - Ordonare tip stivă (LIFO)
11 - Ordonare după prioritatea elementelor (mesaje sau procese)

In cuvîntul 2 se dă adresa primului mesaj al şirului.

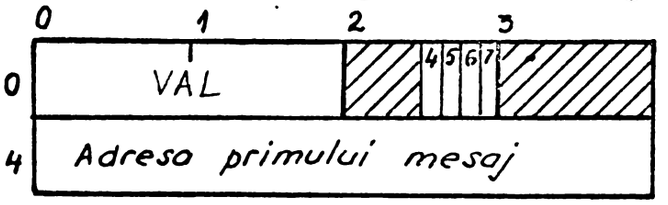


Fig.2.2. Structura unui semafor.

Macroinstrucţia P are următorul efect asupra semaforului unei resurse :

```
VAL = VAL - 1
IF VAL < 0 THEN   INSERT (P, SEM)
                  SET STATUS (P) = "WAIT"
```

Decrementează semaforul şi dacă VAL ≥ 0 procesul continuă execuţie putînd ocupa resursa, iar dacă VAL < 0 procesul trece în aşteptare şi se înregistrează în şirul semaforului.

Macroinstrucţia V serveşte la eliberarea unei resurse de către un proces. Ea are următorul efect :

```
VAL  =  VAL + 1
IF   VAL ≤ 0    THEN  REMOVE  (P, SEM)
                      SET  STATUS (P) = "READY" ;
```

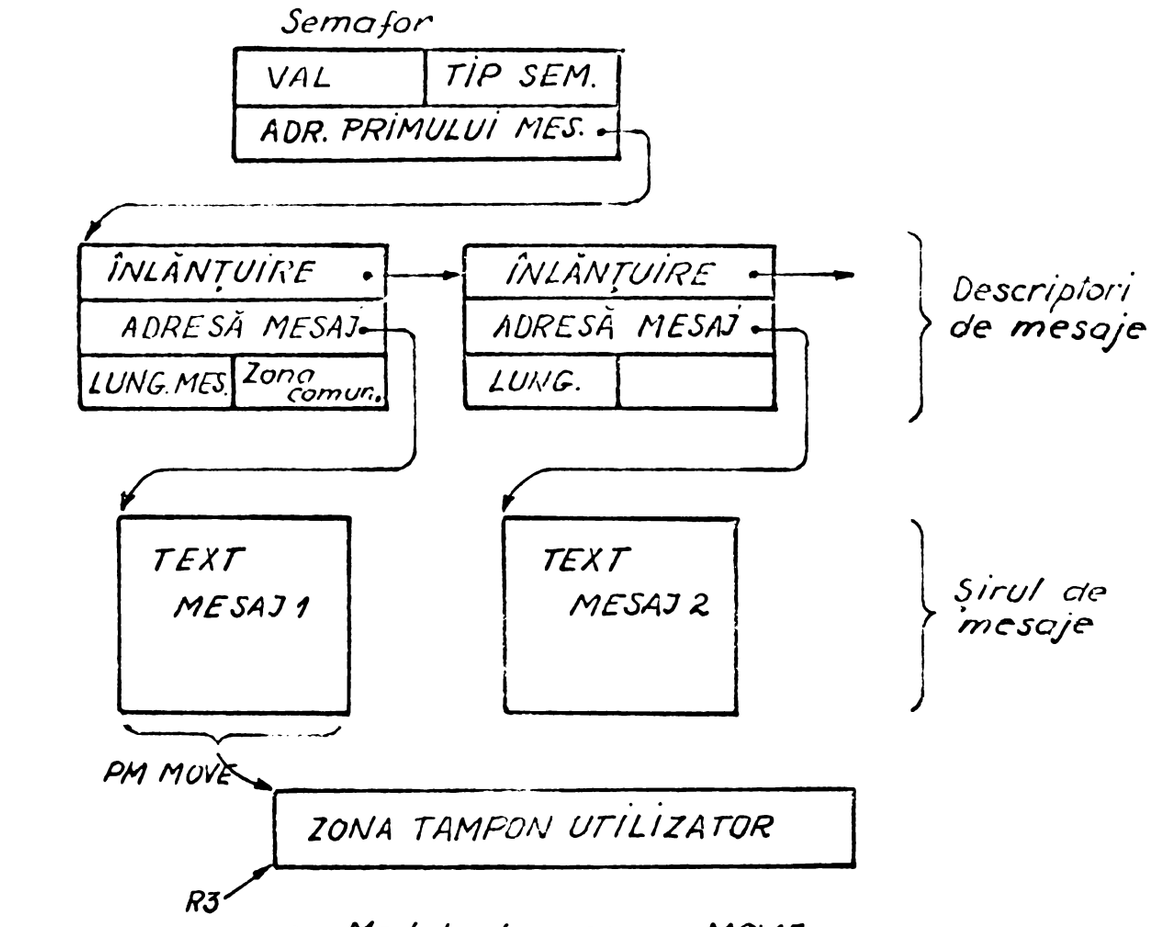
Se incrementează contorul VAL şi dacă VAL ≤ 0 (există procese în şirul de aşteptare) se ia următorul proces din şir şi se activează. El poate deci ocupa resursa semaforului şi o va elibera tot printr-o macroinstrucţie V.

Macroinstrucţia P V efectuează operaţia P pe un semafor şi V pe altul.

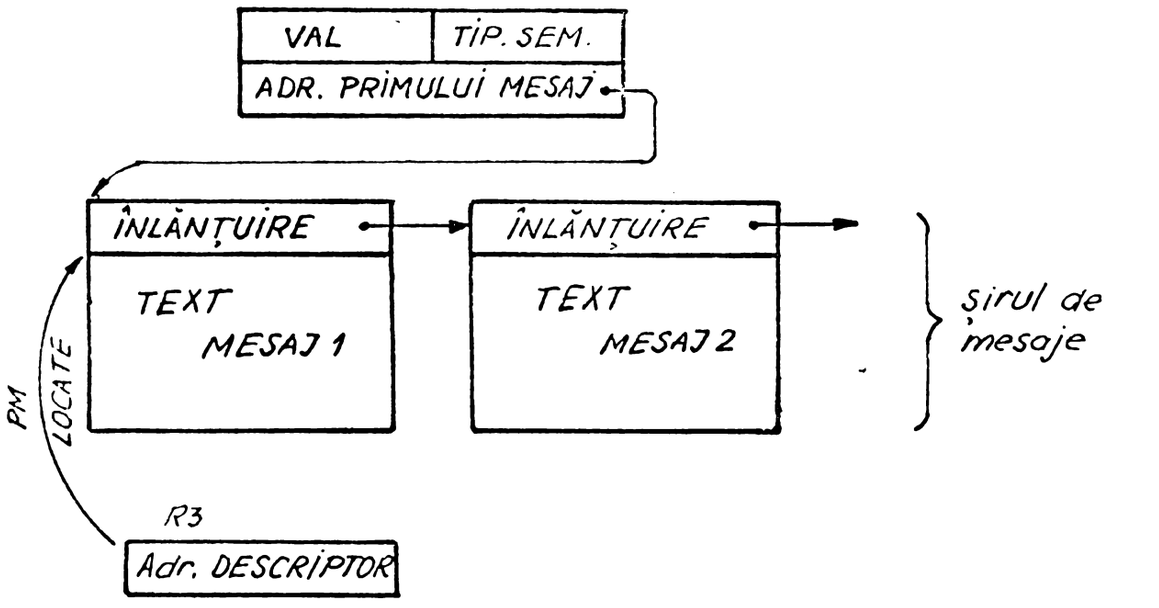
Macroinstrucţie VALL validează activarea tuturor proceselor din şirul de aşteptare al unui semafor (operaţia V ciclic pînă VAL > 0).

Macroinstrucţiile PM şi VM realizează funcţia P şi V pe un semafor cu mesaje. Operaţiile PM sînt "consumatoare" de mesaje (REQUEST), iar VM sînt "producătoare" de mesaje, (RELEASE) pe un semafor. Mesajele produse prin VM se pun într-un şir de aşteptare incrementînd contorul VAL. Fiecare operaţie PM preia din şirul semaforului cîte un mesaj şi decrementează contorul VAL. (fig.2.3.) Dacă acesta a ajuns la valori negative, înseamnă că în şirul semaforului nu mai sînt mesaje, procesele care fac operaţia PM sînt puse în aşteptare şi înserate în şirul semaforului. Deservirea lor se va face în ordinea sosirii, cînd alte procese produc mesaje. Operaţia VM, care depune un mesaj dacă sînt procese în aşteptare le activează şi le trimite mesajul. Mesajul poate fi transmis spre procesul consumator în mod " LOCAL " (adresă mesaj în R 3) sau în mod " MOVE " la adresa indicată în macroinstrucţia PM în R3. Se asigură astfel colaborarea proceselor prin ordonarea "cererilor" şi "depunerilor", care se efectuează asincron. Prin directive COM şi utilizînd cîteva subprograme se pot implementa şi sub SIRIS-3 pe calculatoarele HELIX-C 256/512 semafoare şi macroinstrucţii P şi V similare cu cele prezentate.

Sincronizarea accesului la anumite resurse între partiţii se poate face în sistemul HELIOS prin macroinstrucţii. Ele se bazează pe crearea unor şiruri de aşteptare pentru anumite re-



a. Modul de acces MOVE



b. Modul de acces LOCATE

Fig. 2.3. Semafoare cu mesaje

surse definite la iniţializarea sistemului. Programul cere acces la resursă prin ENQR şi poate fi cu sau fără punerea în aşteptare a partiţiei. Cererea de acces poate fi în exclusivitate (E) sau partajabilă (S = Shared). O cerere partajată poate fi satisfăcută dacă şi celelalte partiţii au cerut-o partajabilă. Dacă resursa este ocupată cererea este pusă în şirul de aşteptare al resursei. Eliberarea resursei se face prin macroinstrucţia DEQ, care permite atribuirea ei altei partiţii din şirul de aşteptare al resursei.

Comunicarea între partiţii se poate face într-un sistem de "cutie poştală" care conţine mesaje plasate într-o zonă specială interpartiţii. Fiecare partiţie poate să-şi creeze una sau mai multe "cutii poştale", unde să primească mesaje prin macroinstrucţia DECL şi poate să le distrugă prin SUPP. Fiecare partiţie poate lua mesaje din propria cutie, fiind informată de originea lor prin MACL. Poate transmite mesaje spre altă partiţie punându-le în "cutia" ei prin ENQ. Sistemul poate fi folosit dacă se creează un subsistem cu multiacces care lucrează pe 2 partiţii. In a doua se execută compilări şi execuţii, iar în prima editarea programelor prin dialog cu terminalele.

Încărcarea dinamică a unor programe specificate prin nume simbolice, prin macroinstrucţia LOAD, este o facilitate deosebit de importantă pentru extinderea funcţiilor subsistemelor cu multiacces. Din păcate în HELIOS numele bibliotecii nu se poate da dinamic, lucru posibil în SCOT (vezi 4.3).

Prezintă importanţă macroinstrucţia de scriere în fişierul de contabilitate al sistemului şi cea de contorizare a timpului de UC pe fiecare proces din momentul creării (PTS). Se poate ţine astfel evidenţa lucrului pe terminale în multiacces. Pentru a evita terminarea, dintr-o eroare într-un proces, a programului se pot trata prin secvenţe de SXK M şi macroinstrucţiile ABORT, TERM şi WAIT. Tratarea WAIT-ului permite realizarea unor subsisteme concurente de înaltă performanţă, care să folosească şi timpul de aşteptare la citirea fişierelor executată prin SGF.

Utilizarea dinamică a fişierelor este posibilă sub sistemul HELIOS în condiţii optime, respectînd toate protecţiile SGF. Componentele SGF sînt utilizabile şi modificabile prin macroinstrucţii. Modulele SGF monitor de OPEN/CLOSE sînt apelabile prin CALL SGF.

Macroinstrucţia FMI (File management interface) permite crearea unei interfeţe SGF pentru un proces dat. O interfaţă SGF se compune din zonă de informaţii generale (ZIG), resumat fişiere (RSGF), zonă de reentranţă SGF, modul de legătură (ML) şi bloc de comandă pentru accesul la ML.Procesele pot utiliza concurent un ML care trebuie protejat (operaţii EPS, OPS, RFCOM, WFCOM). Macroinstrucţia FCI (File Communication Interface) permite realizarea dinamică de :

- tabele de descriere a fişierelor;
- blocuri de comandă şi programe de I/E pentru fişiere;
- zone tampon;
- elemente de FILCOM în fişierul de comunicare (DCA);
- încărcare a modulelor de acces SGF.

Se pot executa astfel dinamic funcţii ale încărcătorului (FILCOM şi ML), compilatoarelor (generare TDF), editorului de legături (încărcare module de acces, CB, PUS, zone tampon). Se extind facilităţile de modificare dinamică a programelor.

Toate componentele sistemului de operare fiind reentrante se permite apelarea lor din program. Se pot executa la cererea programului :

- macroasamblare (ASSM);
- compilare FORTRAN, COBOL, PL/1;
- editarea legăturilor pentru un set de module BT;
- modificarea programelor din bibliotecă;
- conversii de suport;
- inventare de fişiere;
- întreţinere de fişiere.

Activarea compilatoarelor necesită fişiere de manevră, dar se pot executa simultan mai multe compilări cu un singur compilator. Utilizînd toate aceste facilităţi ale sistemelor de operare moderne se pot dezvolta subsisteme de operare specializate puternice, unitare şi rapide. Cele mai multe facilităţi sînt în folosul subsistemelor cu multiacces, care pot fi restrînse ca varietate şi integrate într-un sistem unitar cu funcţii multiple, care să cuprindă ca posibilitate şi compilarea programelor sursă "semiconversaţionale".

2.2. LIMBAJE DE PROGRAMARE CONCURENTA

[Fre 81] [Jur 84] (PASCAL CONCURENT, MODULA, PORTAL, ADA)

Limbajele de programare concurentă au fost concepute special pentru a realiza sincronizarea şi comunicarea între procesele concurente, care în general sînt interdependente. Metodele folosite sînt diferite şi constau în introducerea unor construcţii sintactice speciale destinate rezolvării concurenţei în general. Descrierea acestor limbaje depăşeşte domeniul acestei lucrări, dar sînt amintite deoarece reprezintă o alternativă importantă în scrierea subsistemelor cu multiacces şi mai ales a programelor cu multiacces.

Cel mai răspîndit este limbajul PASCAL CONCURENT definit de P.Brinch Hansen [Bri 75] [Cio 84], care a completat limbajul PASCAL definit de N. Wirth cu conceptele de proces, monitoare şi clase. Este un limbaj structurat, modular, flexibil şi portabil. Permite introducerea de tipuri de date abstracte cu proprietăţi particulare, articole structurate şi proceduri specializate, care pot asigura accesul simplu la terminale de orice tip. Pe această cale se pot introduce în nucleul PASCAL, module de acces la terminalele de teletransmisie, care vor fi gestionate în procesele din program cu tehnicile disponibile. Implementarea limbajului pe un calculator dat se face prin crearea unui nucleu, care asigură legătura dintre codurile virtuale PASCAL şi sistemul de operare existent. Compilatorul generează module BT compatibile, care vor fi asamblate împreună cu modulele BT ale nucleului existent într-o bibliotecă.

Se consideră că există <u>variabile comune</u> la care procesele au acces secvenţial. Secvenţa de program în care un proces are acces la o variabilă comună se <u>numeşte regiune critică.</u> Pentru a rezolva concurenţa proceselor la variabile comune se propune în PASCAL <u>conceptul de monitor,</u> care grupează toate secvenţele de acces posibile la o variabilă comună. Un monitor se defineşte împreună cu structura de date comune pe care operează şi conţine procedurile de prelucrare a acestora. Accesul proceselor la datele comune se face numai prin procedurile monitor. Pe perioada cît un proces execută o procedură monitor ea este considerată parte a codului procesului. Apelul se face indicînd numele monitorului, procedura şi parametrii actuali sub forma:

nume monitor . nume proc (parametrii actuali procedură)

Declararea unui monitor în PASCAL CONCURENT are forma generală :

```
TYPE  nume monitor = MONITOR (parametrii formali
      monitor);
      declarare date comune monitor ;
      PROCEDURE ENTRY nume proc (parametrii
      formali procedură)
        BEGIN corpul procedurii END ;
        ...declararea altor proceduri de monitor ;
        ...declararea procedurilor locale ;
        BEGIN operaţie iniţială END ;
```

Un monitor nu poate apela propriile proceduri de monitor (apel recursiv interzis). Monitorul rezolvă probleme excluderii reciproce dintre procese. Procesele care găsesc la apel monitorul ocupat, sînt puse în şirul de aşteptare al monitorului şi se lansează la eliberarea monitorului în ordinea sosirii.

Limitările utilizării limbajelor concurente sînt date de mai mulţi factori.

- Fiind limbaje de nivel înalt programele obiect rezultate vor fi mult mai lente decît cele obţinute din limbajele de asamblare, lucru important la subsistemele cu multiacces.

- Posibilităţile de acces la resursele sistemului (periferice, fişiere, MC, macroinstrucţii speciale) sînt limitate de performanţele nucleului şi restricţiile sistemului de operare pe care se implementează.

- Scrierea directă a unor programe conversaţionale pentru diferite aplicaţii, nu este accesibilă direct utilizatorilor, care îşi însuşesc mai greu tehnicile de programare concurente.

Importanţa limbajelor concurente este incontestabilă, contribuind la dezvoltarea unor subsisteme şi programe cu multiacces la care performanţele de viteză nu sînt obligatorii. Aplicaţiile lor sînt mult mai largi la realizarea unor componente ale sistemelor de operare şi la programarea în timp real.

2.3. SUBSISTEME PENTRU GENERAREA DE PACHETE DE PROGRAME CU MULTIACCES

Numărul redus al programatorilor care pot scrie pachete de programe în limbaje concurente şi apariţia recentă pe piaţă a compilatoarelor corespunzătoare, a determinat apariţia unor subsisteme ce oferă module de programe cu multiacces, scrise în limbaj de asamblare, care pot fi combinate cu module scrise în limbaje clasice. Rezultă astfel subsisteme cu multiacces. Din această categorie face parte subsistemul STRATEGE [Str 76] destinat calculatoarelor IRIS şi FELIX pentru a realiza subsisteme de teleprelucrare.

Subsistemul STRATEGE realizează o interfaţă între programele de aplicaţie scrise în limbaje clasice (FORTRAN, COBOL), monitorul SIRIS-3 şi o reţea de terminale diferite acceptate de SGT. Subsistemul asigură următoarele funcţii :

- gestiunea procedurilor utilizator într-o zonă de memorie alocată dinamic în cadrul partiţiei ;
- gestiunea fişierelor utilizator ;
- gestiunea transacţiilor de I/E executate cu terminalele.

Procedurile utilizator se testează în prelucrare în loturi. Ele vor fi interclate între module STRATEGE, pe care le vor apela prin parametrii şi nume, date în manualul de utilizare. Descrierea liniilor de telatransmisie şi a fişierelor se face în module STRATEGE, care cuprind şi funcţiile de gestiune a multitaskului, modulele de acces la fişiere, modulul de legătură SGF. Modulele utilizator conţinând apelurile la subsistem vor suferi editarea legăturilor împreună cu modulele subsistemului referite prin cartele MATCHB. Va rezulta un program unic, care pentru a fi adaptat condiţiilor dinamice de multiacces se supune unei post-link-editări STRATEGE şi se comportă ca un subsistem.

Subsistemele obţinute prin STRATEGE conţin limitatori de utilizare a fişierelor, interzicându-se accesul secvenţial. La un moment dat numai un fişier utilizator poate fi deschis şi o procedură poate trata un singur mesaj de la terminal. După generare nu se mai pot adăuga sau modifica proceduri. Aceasta implică o nouă generare completă. Subsistemul care obligatoriu să existe în lucru este SYMBIONT (30 K) şi SYMBIONTA. Necesită numai pentru partea de module STRATEGE 100 K de memorie, la

care se adaugă spaţiul pentru procedurile utilizator. Nu există posibilităţi de testare interactivă a subsistemului rezultat. Utilizarea funcţiilor [illegible] şi generarea programului este complicată.

3. SISTEME DE OPERARE CU MULTIACCES ŞI MEMORIE VIRTUALĂ

3.1. METODE DE IMPLEMENTARE A MEMORIEI VIRTUALE

Structura calculatoarelor clasice bazate pe protecţia memoriei la nivel de pagină prin chei de protecţie, depuse într-o memorie cu acces rapid, limitează posibilităţile subsistemelor cu multiacces. Folosirea generală a cheilor de protecţie cu 4 biţi permite maxim 16 combinaţii, asociate la tot atîtea partiţii, din care cel puţin 2 sînt normal destinate monitorului. In sistemele cu multiprogramare în fiecare partiţie se poate încărca un program, deci pot lucra simultan maxim 12 - 14 programe. In multiacces lucrează simultan zeci de programe pentru care nu se poate asigura protecţia directă. Subsistemele cu multiacces lucrînd în multiprogramare complexă folosesc una sau două partiţii (două chei de protecţie) pentru toate programele gestionate, care se păstrează pe un fişier disc. Numai programele active sînt încărcate în MC. Chiar dacă există loc în partiţie pentru a încărca mai multe programe, aceasta nu este practic posibil deoarece nu se poate asigura protecţia lor reciprocă. Subsistemul şi grupul de programe gestionate sînt văzute de sistemul de operare ca un singur program (lucrare) cu o singură cheie de protecţie. Deşi unele sisteme de operare (MULTICS) permit generarea şi gestionarea dinamică de procese, iar partiţia e văzută ca un grup de procese, ele au o singură cheie de protecţie fizică şi nu sînt protejate reciproc împotriva erorilor de programare posibile. Protecţia logică a proceselor este insuficientă practic şi singura metodă sigură în multiacces este salvarea pe disc a programelor inactive. Execuţia paralelă a lucrului în UC a unui program, cu swapingul altui program, în zone diferite, este nesigur. O greşeală de programare ar putea duce la modificarea accidentală a programului încărcat sau salvat.

Gestionarea programelor în multiacces prin "swaping" pierde mult timp cu operaţiile de depunere/încărcare executate prin subsistem. Utilizarea conceptului de memorie virtuală înlătură aceste limitări şi oferă următoarele facilităţi :

- se asigură accesul utilizatorilor la un spaţiu de adresare mult mai mare:

- se adresează unitar o ierarhie de memorii cu două nivele, cu raport de viteză de 1/1000, ca o singură memorie logică;
- se realizează automat prin hardware şi software transferul de informaţii dintr-un nivel de memorie pe altul;
- se asigură o gestiune eficientă a programelor pentru a reduce la maxim transferul între cele două nivele de memorie;
- se asigură protecţia individuală pentru maxim 256 programe (mai mult în sistemul MULTICS).

In aceste condiţii se pot realiza subsisteme cu multiacces care să gestioneze flexibil un număr mare de programe, cu protecţie individuală şi să folosească eficient şi dinamic resursele de memorie. Unele sisteme de operare bazate pe memorie virtuală au fost concepute special pentru a lucra în multiacces (MULTICS), prelucrarea în loturi fiind considerată un caz particular. Există mai multe metode de virtualizare a memoriei.

3.1.1. Memorii virtuale cu paginare

In conceptul de memorie virtuală există un spaţiu logic de adresare AS (address space) căruia îi corespunde un spaţiu pe disc şi un spaţiu fizic de adresare în memoria centrală (MC). Numai o parte din AS are un corespondent direct în MC. (fig. 3.1.)

Adresarea memoriei din program se face pe spaţiul virtual unde se presupune că există programele şi datele memorate. Unitatea centrală lucrează însă numai în MC pe zone în care datele solicitate au fost copiate. Prin hardware şi software se asigură un mecanism automat de încărcare a programelor de pe disc în MC şi translaţia adresei virtuale într-o adresă de memorie centrală. Realizarea virtualizării memoriei numai prin software nu este aplicabilă. Ea reduce viteza programelor de cel puţin două ori. Prin hardware se pot asigura mecanisme de translaţie rapide şi relativ ieftine.

Sistemele virtuale cu paginare consideră atât memoria centrală cât şi cea virtuală divizate în pagini de egală lungime. Transferul între cele două memorii se face prin

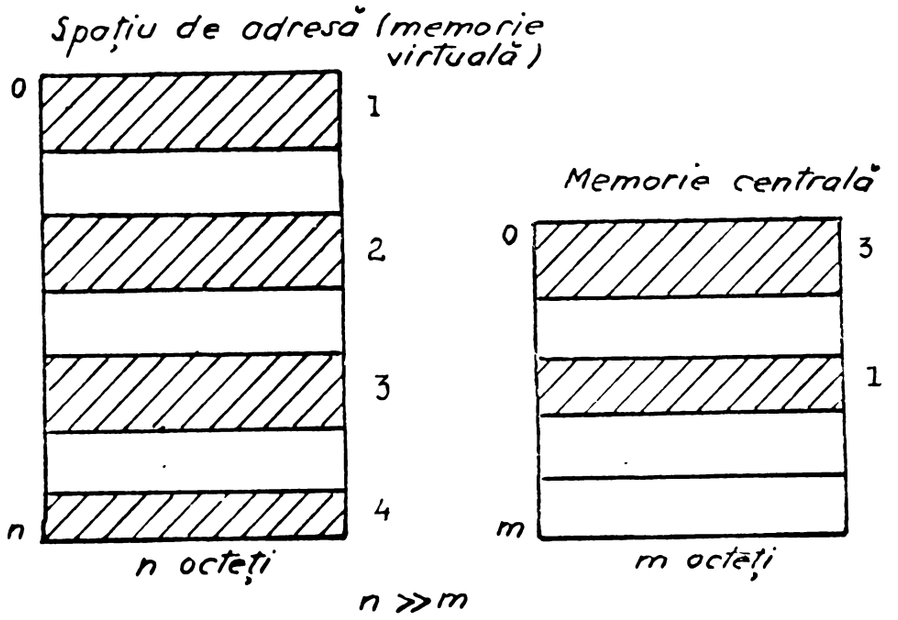


Fig. 3.1. Spațiul de adresă virtual

pagini întregi. Programele ocupă în memoria virtuală spații continue, care le putem numi partiții virtuale. Un program ocupă un număr întreg de pagini. Alocarea memoriei virtuale se face dinamic cu partiții de lungime fixă sau variabilă. Paginile unui program pot fi dispersate în MC. Corespondența dintre adresa paginilor în MV și MC se face printr-un tabel în MC pentru toate paginile virtuale.

Parțial această corespondență se ține într-o memorie asociativă de capacitate mică și de mare viteză. Memoria asociativă se adresează prin conținutul unui cîmp. Dacă valoarea dată la intrare (adresa paginii virtuale) este găsită într-un cuvînt, atunci la ieșire se va obține valoarea celuilalt cîmp al cuvîntului (adresa paginii din MC corespunzătoare. La calculul adresei într-o instrucțiune rezultă o adresă virtuală formată din numărul paginii virtuale și adresa relativă în pagină. Cu adresa paginii virtuale se adresează memoria asociativă care furnizează la ieșire, în registrul de date adresa paginii din MC. La această adresă se adaugă prin concatenare adresa relativă în pagină și rezultă adresa completă de MC. În fig. 3.2. se prezintă mecanismul de translație a paginilor pentru o memorie centrală de 1 Mo și o memorie virtuală de 16 Mo, cu pagini de 4 Ko. MC conține 256 pagini, iar MV conține 4 K pagini. Pentru deplasamentul în pagină sînt necesari 12 biți, pentru adresa paginii virtuale 12 biți, iar pentru adresa paginii din MC 8 biți.

Un registru din memoria asociativă conţine 3 octeţi. Ultimii 2 biţi sînt folosiţi pentru a indica pagină prezentă în MC (P) şi pagină activă (A).

Dacă pagina virtuală adresată nu se găseşte în memoria asociativă, se provoacă o derută care activează o procedură de căutare în tabela de pagini din MC, deoarece memoria asociativă nu are de regulă stiteă cuvinte cîte pagini sînt în MC. Dacă pagina e încărcată în MC se obţine adresa ei şi se poate completa adresa, pentru a obţine operandul din MC. Cu cît este mai mare memoria asociativă, creşte eficienţa translaţiei. Deoarece programele execută majoritatea instrucţiilor secvenţial, iar datele sînt într-o zonă compactă, majoritatea adresărilor se fac prin memoria asociativă. Ea conţine adresa ultimelor 64 pagini (în ex. dat) adresate. La adresarea unei pagini ce nu se găseşte în memoria asociativă, dar este în MC, adresa ei va fi completată în memoria asociativă în locul unei pagini inactive (A = 0). Pentru protecţia programelor pot fi adresate la un moment dat numai paginile programului activ pentru care la lansare se pune A = 1. Dacă o pagină adresată nu se găseşte în MC, se lansează o procedură de încărcare a paginii după următoarea secvenţă :

1). - Se caută o pagină liberă în MC prin tabela de ocupare a MC.

2). - Dacă se găseşte, se încarcă pagina şi se actualizează tabela de pagini şi memoria asociativă.

- Dacă nu se găseşte, se aplică un algoritm de înlocuire a unei pagini existente în MC. La fiecare adresare de pagină se poziţionează bitul R = 1, în tabela de pagini (la copierea în memoria asociativă). Paginile care sînt prezente (P = 1), inactive (A = 0) şi nu au fost de mult timp referite (R = 0) pot fi înlocuite. Cînd toate paginile încărcate ajung cu R = 1 se forţează la toate R = 0.

3). - Dacă se înlocuieşte, se memorează pe disc pagina înlocuită.

4). Se citeşte de pe disc pagina adresată şi se actualizează tabelele.

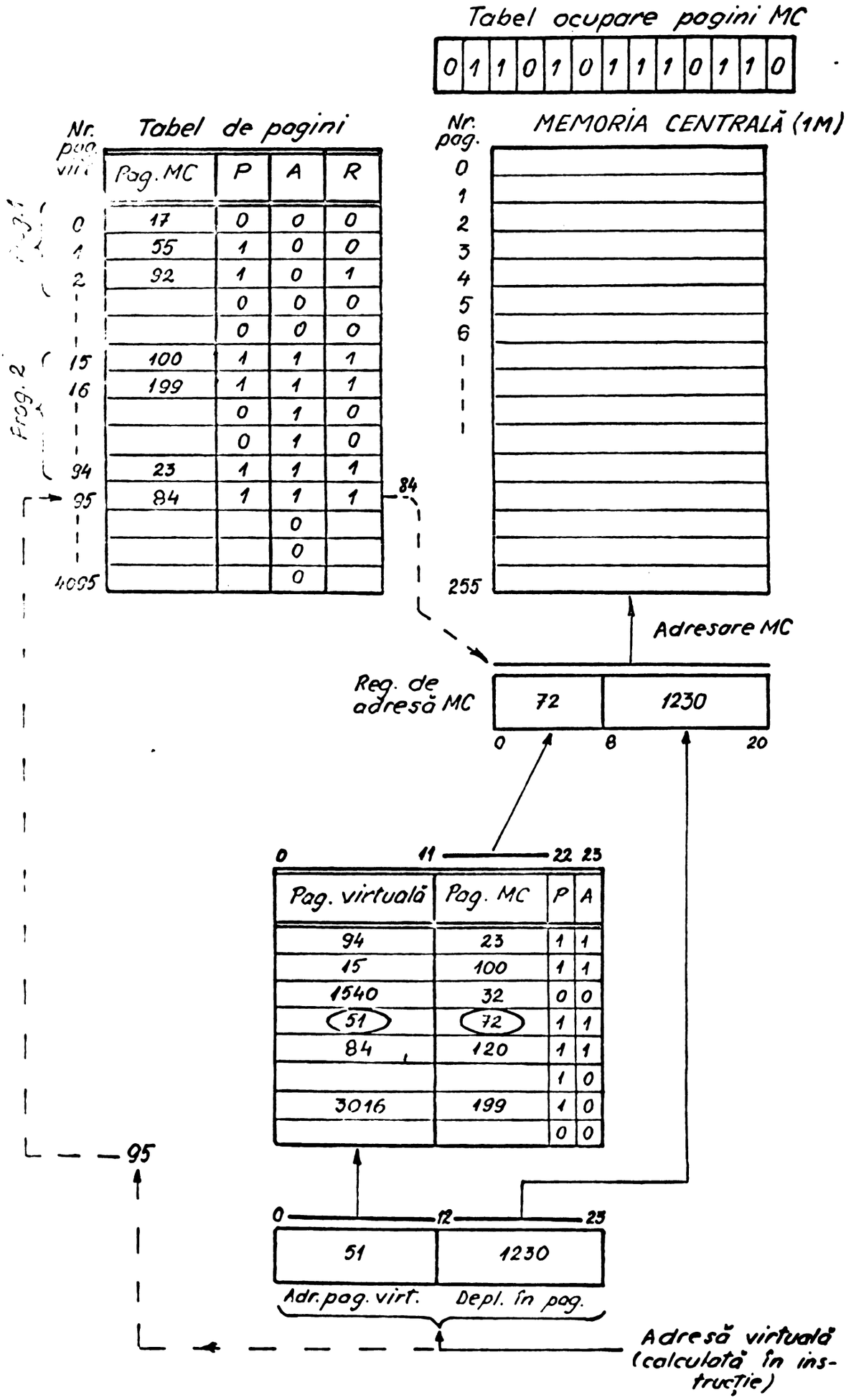


Fig. 3.2. Memorie virtuală cu paginare pură memorie asociativă

Pentru programele terminate se şterg paginile în tabela de ocupare a MC.

Virtualizarea memoriei prin paginare cu memorie asociativă este cea mai simplă, mai ieftină şi mai simplu de implementat prin software. Se utilizează în ultimul timp şi pe minicalculatoare de capacitate mare care lucrează în multiprogramare. [illegible] permite o gestionare simplă a transferului de programe şi date între MC şi disc, o folosire economică a MC. Scăderea de viteză, datorită adresării unor pagini care nu sînt indicate în memoria asociativă, se compensează prin transferul pe disc numai a unor pagini şi nu a unor programe. Rămîn în MC numai paginile cele mai solicitate din toate programele. Gradul de multiprogramare creşte mult.

3.1.2. Memorii virtuale cu segmentare

Modularitatea scrierii programelor şi posibilitatea de adresare prin instrucţie a maxim 64 Ko (16 biţi partea de adresă), au dus la segmentarea programelor. Segmentele pot fi de program,de date sau comune. Pentru a reduce spaţiul de memorie folosit unele segmente se încărcau succesiv în aceeaşi zonă de memorie direct din bibliotecă (segmente paralele), întotdeauna în forma iniţială. In ele nu se puteau memora rezultate intermediare, deoarece la încărcarea altui segment paralel nu se puteau salva şi se pierdeau. Necesitatea segmentelor paralele dispare la memoria virtuală şi restricţiile impuse de aceasta. Suportul de păstrare a segmentelor care nu încap în MC, pe timpul execuţiei este luat de fişierul disc, suport al memoriei virtuale. Un segment înlocuit în MC va fi înainte salvat pe memoria virtuală. El va păstra pentru o viitoare reîncărcare toate modificările făcute anterior. Un program va fi format în memoria virtuală dintr-un număr de segmente puse cap la cap.

In aceste condiţii numărul de segmente al programelor scade simţitor. Segmentarea va fi impusă numai de dimensiunea maximă (64 K) şi modularitatea programelor. Avînd în vedere importanţa segmentelor în structura programelor s-a propus o metodă de implementare a memoriei virtuale bazată pe segmente. Se asigură astfel compatibilitatea cu sistemele de operare clasice. Segmentele se consideră încărcate în memoria virtuală pe spaţii continue, dar segmentele unui program nu trebuie plasate alăturat. Se asigură prin sistemul de operare o gestiune dinamică a memoriei virtuale la nivel de seg-

ment. Programele pot fi considerate ca o colecţie de segmente. Anumite segmente de program (de regulă reentrante) sau de date pot fi folosite în comun de mai multe programe.

Primele calculatoare de mare capacitate au adoptat memoria virtuală cu segmentare, cu sau fără paginare la mijlocul anilor 60 (B 5500, IBM 360/67). Ea s-a păstrat şi pe calculatoarele moderne IBM 370 model 158 şi 168 lansate în 1973. Trebuie remarcat că IBM 360/67 a fost obţinut din IBM 360/65 de care diferă prin memorie virtuală şi a fost conceput special pentru multiacces. S-a conceput pentru acesta un sistem de operare special CP/CMS (Time Sharing System), care a servit ca bază pentru sistemele CP/VM implementate pe IBM/370.

Segmentele sînt formate dintr-un număr întreg de pagini. Încărcarea segmentelor în MC se face la cerere prin gestiune dinamică a spaţiului MC. Corespondenţa dintre adresa segmentului în memoria virtuală şi cea din MC se face automat prin adresarea unei memorii rapide, care are rol de tabelă de segmente (fig. 3.3.). Tabela de segmente are atîtea elemente cîte pagini are memoria virtuală. Pentru un segment virtual se găseşte adresa segmentului în MC (în Ko), lungimea sa şi indicatorii de prezenţă (P), activitate (A) şi referinţă (R) paginã.

Adresa virtuală de 24 biţi poate adresa o memorie virtuală de 16 Mo şi este împărţită în număr de segment virtual (8 biţi) şi deplasament în segment (16 biţi). Memoria virtuală poate conţine maxim 256 segmente de maximum 64 Ko. Adresarea se face în felul următor : numărul segmentului din adresa virtuală este folosit ca adresă în tabela de segmente, de unde se citeşte adresa segmentului în MC dat în număr de pagini. Aceasta se adună cu deplasamentul şi rezultă adresa operandului. Pentru a nu se adresa un spaţiu mai mare decît lungimea segmentului (LS), se compară aceasta cu deplasamentul din adresă (DEPL). Dacă DEPL > LS se produce o derută ce indică eroare de adresare (EA = 1). Pentru a evita acceptarea unui număr de segment inexistent, se dă într-un registru de bază a tabelei de segmente lungimea acesteia (număr intrări), care indică numărul de segmente active în memoria virtuală. Un număr de segment mai mare decît lungimea dă eroare de adresare.

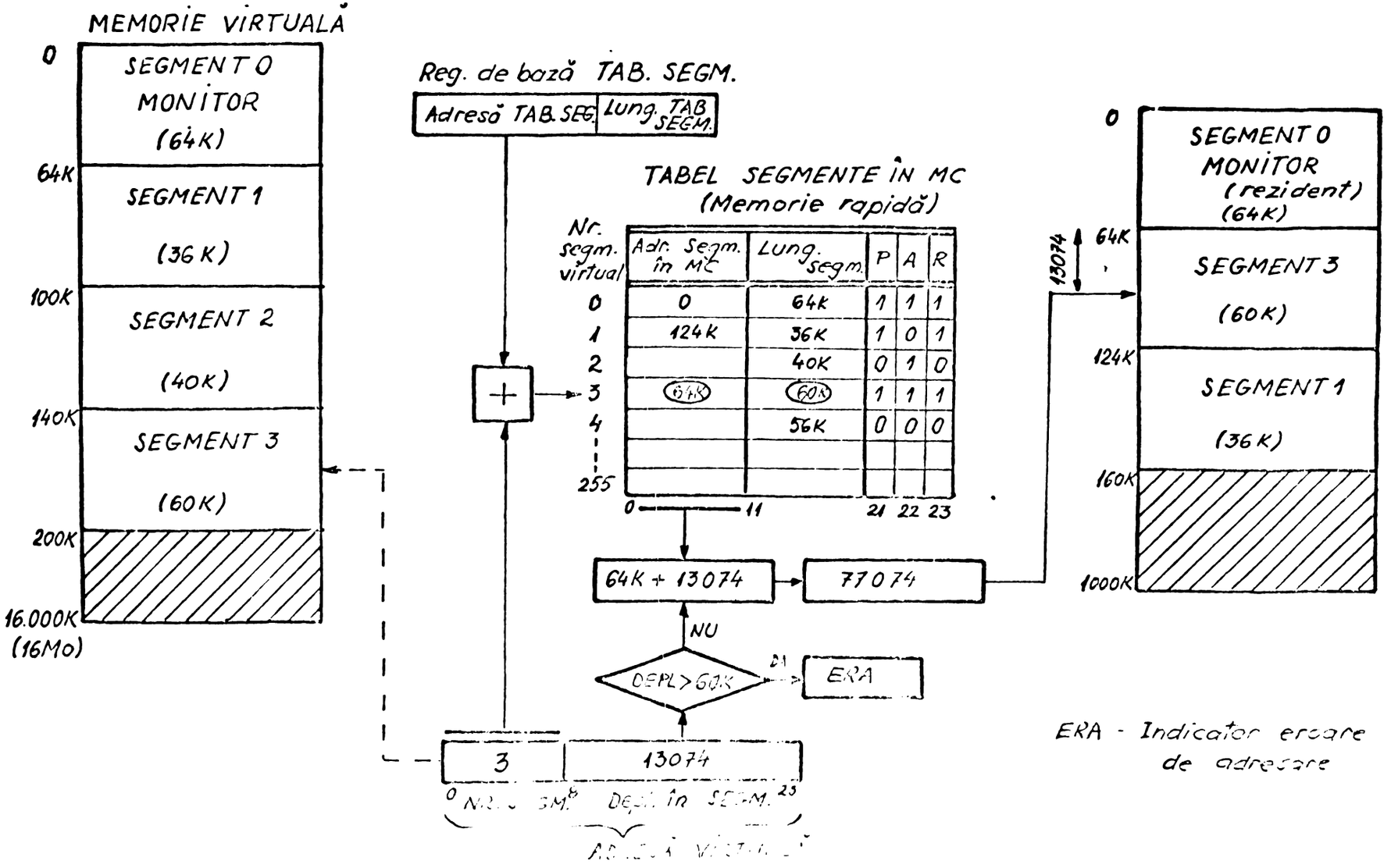


Fig. 3.3. Memorie virtuală cu segmentare.

Dacă la adresare bitul P = 0, înseamnă că segmentul nu e încărcat în MC. Se produce o derută care lansează procedura de încărcare a segmentului. Se caută loc corespunzător lungimii L S în MC, se încarcă segmentul şi se adresează. Dacă nu există loc liber în MC se identifică un segment inactiv (A = 0), prezent (P = 1), care nu a fost de mult activat (R = 0) şi care are lungime apropiată de cea necesară. Se depune segmentul înlocuit în memoria virtuală şi se încarcă segmentul necesar. Adresele segmentelor virtuale şi toate datele despre ele se păstrează într-o tabelă generală de segmente în MC în zona monitorului. Monitorul ocupă de regulă primele două-trei segmente din memoria virtuală. Primul segment al monitorului este rezident la începutul MC.

La un moment dat un singur program este activ şi segmentele lui vor avea A = 1, chiar dacă nu sînt prezente în MC. La adresare se permite referirea la orice segment activ şi se interzice accesul în segmentele încărcate şi inactive, care aparţin altor programe (P = 1 şi A = 0). Indicatorul R se pune pe 1 la orice activare a segmentului. Segmentele care au R = 0 nu au fost recent activate şi vor fi înlocuite dacă în MC nu mai e loc şi trebuie încărcat un segment. Cînd toate segmentele au R = 1, se forţează la toate R = 0, ş.a.m.d. Această metodă de înlocuire are un mare coeficient de hazard şi poate duce la un trafic inutil de segmente. În subcapitolul 3.3. se va da o metodă simplă şi eficientă de înlocuire a segmentelor.

3.1.3. Memorii virtuale cu segmentare şi paginare.

Memoriile virtuale cu paginare au avantajul unei utilizări intensive a MC şi un trafic de informaţie redus între memoria externă şi MC. În MC pot exista în lucru mai multe programe decît la memoriile virtuale segmentate, deoarece programul poate rula numai cu cîteva pagini încărcate din segment. Celelalte se vor încărca în momentul cînd vor fi apelate, înlocuind eventual altele inactive. Aceasta este un mare avantaj pentru sistemele cu multiacces, în care lucrează simultan un număr foarte mare de programe, ce vor putea coexista într-o MC de dimensiuni reduse.

Deoarece memoria asociativă de translaţie costă mult şi pentru viteze mari are dimensiuni reduse (zeci de cuvinte), la calculatoarele cu MC mare (1 - 8Mo) apare o scădere apre-

cisibilă a vitezei de acces. Sînt dese cazurile de adresări în care paginile nu sînt în memoria asociativă şi trebuie căutate în tabelul din MC. Acest tabel nu poate fi ţinut într-o memorie rapidă, deoarece ocupă 4 - 8 Ko pentru o memorie virtuală de 16 M şi pagini de 4 K, indiferent de dimensiunea memoriei centrale. Pentru a respecta structurarea programelor în segmente şi a avea şi avantajele paginării , s-au realizat pentru calculatoarele de capacitate mare, memorii virtuale cu segmentare şi paginare. Tabelele de translatare a paginilor virtuale în pagini de MC sînt organizate pe două nivele, tabele de segmente care fac referire la tabelele de pagini (Fig.3.4). Pentru o memorie virtuală de 16 Mo cu pagini de 4 Ko şi segmente de 64 Ko, tabela de segmente va avea 256 de elemente. Fiecărui segment îi corespunde o tabelă de pagini, care are un număr de intrări egal cu numărul de pagini ocupate de segment (maxim 16). În acest fel se ocupă intrări în tabelele de pagini numai pentru paginile virtuale efectiv ocupate. La o memorie virtuală cu paginare simplă de 16 Mo ar fi necesară o memorie rapidă de capacitate:

256 segm x 16 pag. = 4000 intrări x 2 oct. = 8 K

La un coeficient de umplere medie a memoriei virtuale de 50 % (segmente mai puţine şi unele sub 16 pag.), virtualizarea cu segmentare şi paginare necesită doar 4 K memorie rapidă. Încărcarea de 50 % reprezintă 256 segmente de 32 K în lucru, ceea ce înseamnă un coeficient de multiprogramare foarte ridicat. Tabelele de pagini fiind completate dinamic prin software (supervizorul de paginaţie), pot fi plasate, la depăşirea coeficientului de umplere mediu prevăzut, în MC. Aceasta ar încetini viteza de lucru a unor segmente de prioritate redusă.

Dacă presupunem că tabela de segmente şi tabelele de pagini ocupă o memorie rapidă de 8 K şi elementele de tabel sînt la multiplu de 2 octeţi, pentru referirea unei tabele de pagini sînt suficienţi 12 biţi (2^{12} = 4 K x 2 = 8 K), iar pentru a indica lungimea ei, 4 biţi (16 intrări). Elementele tabelei de segmente pot avea o lungime de 2 octeţi (12 adresă + 4 lungime). Tabela de segmente va ocupa :

256 segm x 2 oct = 512 octeţi.

Restul pînă la 8 K va fi ocupat de tabelele de pagini. Un element din tabela de pagini conţine pe 2 octeţi

numărul paginii din MC, care corespunde paginii virtuale (12 ob), indicatorul de prezenţă (P), de activitate (A), (R) de

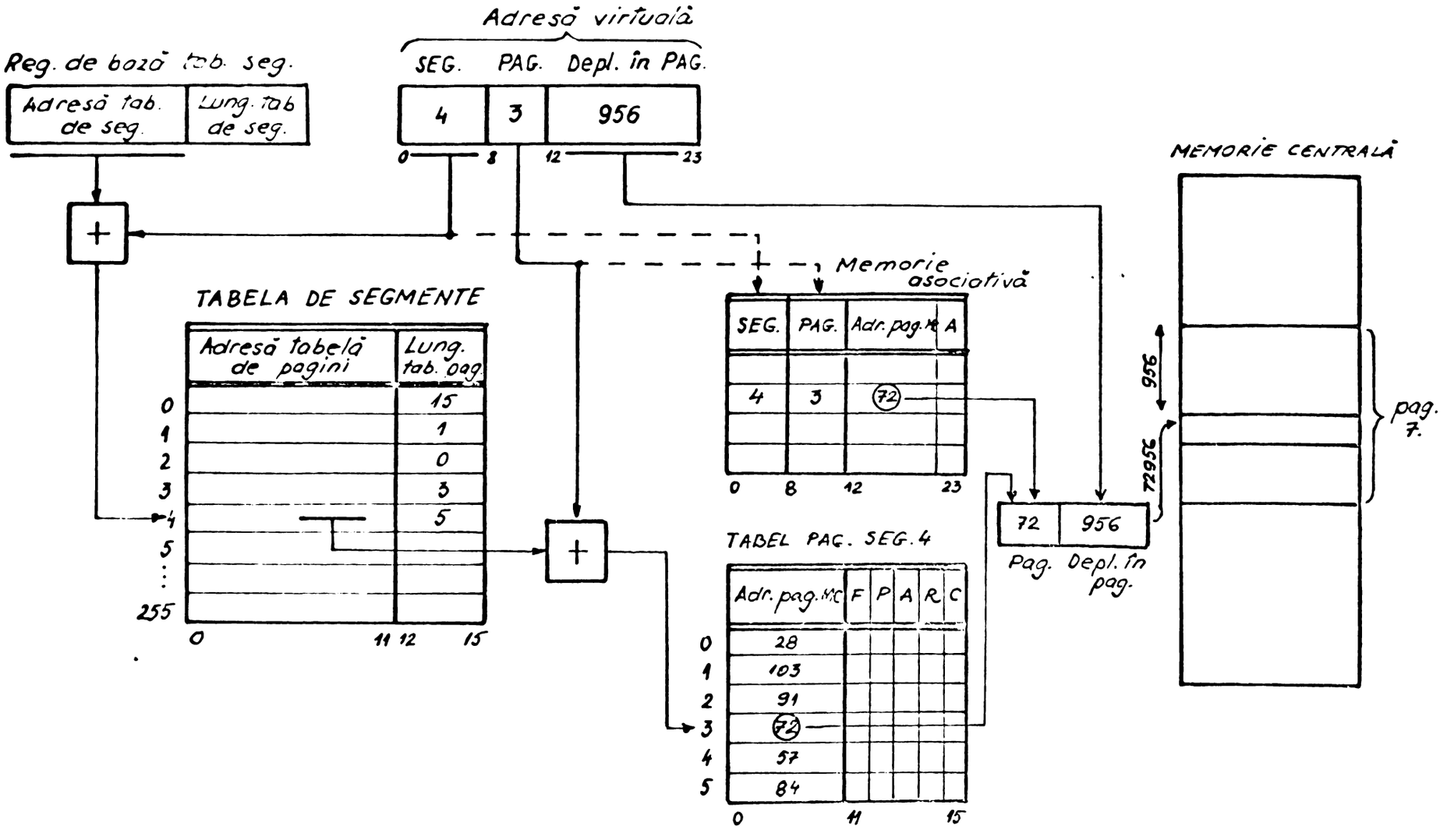


Fig. 3.4. Memorie virtuală cu segmentare și paginare.

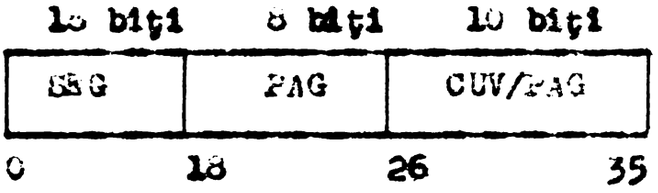
referire şi de pagină fixă (F). Bitul C = 0 indică pagină care nu a suferit modificări de la ultima încărcare. Ea nu trebuie rescrisă în memoria virtuală dacă se înlocuieşte. Dacă se fac memorări în pagină se forţează automat C = 1. Pentru a cunoaşte adresa segmentelor în memoria virtuală se foloseşte o tabelă de segmente în MC, care cuprinde şi alte date referitoare la segmente. O reducere la 2 Ko a lungimii paginilor dublează lungimea tabelelor de pagini.

Adresarea porneşte de la adresa virtuală (24 biţi), care cuprinde segmentul (8 biţi), pagina în segment (4 biţi) şi deplasamentul în pagină (12 biţi). Segmentul va indica intrarea în tabela de segmente, unde se găseşte adresa tabelei de pagini asociate segmentului şi lungimea acestei tabele (numărul de pagini care compun segmentul). Numărul paginii din adresa virtuală se compară cu lungimea tabelei şi dacă e mai mare se indică eroare de adresare. Dacă nu.pagina din adresa virtuală va constitui un index în tabela de pagini, unde se va găsi pagina din MC ce trebuie adresată. Adresa din MC a operandului va fi formată din această pagină completată cu deplasamentul în pagină. La translatarea paginii în acest caz apar două nivele de adresare în memoria rapidă şi calculul a două sume, care măresc timpul de translaţie faţă de memoria asociativă. Din acest motiv se poate folosi o memorie asociativă de dimensiune redusă, care va conţine adresa paginilor din MC mai recent adresate. Căutarea în tabela de pagini se va face numai dacă pagina nu este găsită în memoria asociativă.

Pentru a evita adresarea unui segment inexistent, registrul de bază al tabelei de segmente va conţine şi lungimea tabelei de segmente (numărul de segmente în memoria virtuală). Dacă numărul segmentului din adresa virtuală este mai mare decît această lungime, se indică eroare de adresare. Elementele tabelei de pagini conţin şi biţii de activitate (A), prezenţă (P) şi referinţă (R) a paginii indicate. Acest mecanism de translatare a paginilor este foarte performant, dar necesită hardware complex. Din acest motiv el se aplică numai pe calculatoarele de capacitate mare (IBM/370 model 185).

Acest sistem ierarhic de translatare a paginilor a fost implementat prima dată pe un calculator experimental GE645 pentru implementarea sistemului de operare MULTICS

dezvoltat la Institutul Politehnic din Massachusetts în 1965. Sistemul MULTICS propune adrese virtuală pe 36 biţi împărţiţi astfel :



Se folosesc cuvinte de 36 biţi, o pagină are 1 K cuvinte, într-un segment pot fi 256 pagini (1 Mo), iar memoria virtuală poate conţine 256 K segmente. Se obţine practic un spaţiu de adresare infinit, prin care se pot virtualiza toate memoriile externe cu acces direct. Implementările care s-au efectuat au considerat memorii virtuale mai reduse. Lungimea tabelelor de pagini depinde de lungimea segmentelor şi nu se apropie de limita admisă. Având în vedere generalitatea sistemului MULTICS el poate fi implementat pe calculatoare oricât de mari. Ultima implementare s-a făcut pe un calculator multiplexor de periferice pentru calculatoarele Chai [Lion 64] ,cele mai puternice comercializate.

3.2. Sistemele de operare IBM/370 OS/VS şi OS/Vm

Deşi firme ca Burroughs, Xerox (Scientific Data Systems), RCA, GE (Honeywell) au introdus memorii virtuale pe calculatoarele mari din 1960 - 65, firma IBM a lansat în 1965 doar un model experimental IBM/360 model 67 derivat din modificarea modelului clasic 65, căruia i s-a adăugat un hardware suplimentar Acesta a fost conceput special pentru multiacces şi a fost generalizat, lucrînd sub sistemul de operare OS/TSS (operating system/time sharing system). Experimentările au continuat mai mulţi ani (1965 - 1970) pe aceste calculatoare şi în 1972 s-au anunţat comercial sistemele cu memorie virtuală IBM/370 mod 158 şi 168 de mare viteză (au şi memorie tampon de 8 K cu ciclu de 230 ns). Pentru acestea s-a livrat sistemul de operare OS/VS (virtual storage), prevăzut cu posibilităţi speciale de multiacces.

Sistemul de operare OS/VS foloseşte conceptul AS (1) (address space), care consideră o memorie virtuală unică pentru toţi utilizatorii de 16 Mo, aplicînd asupra ei principiile ge-

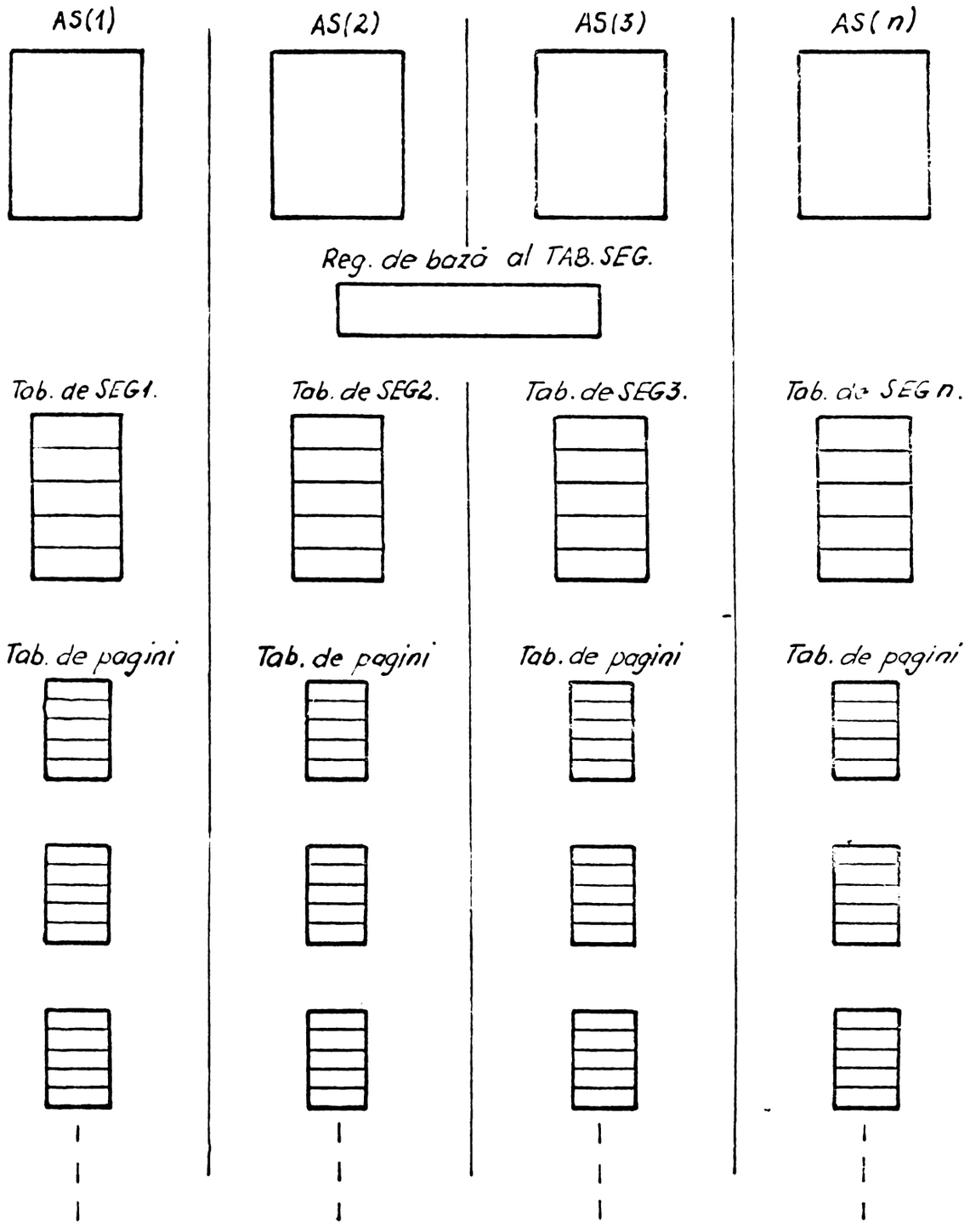


Fig.3.5. Structură cu n spații de memorie virtuală.

tiunii cu partiţii variabile alocate dinamic din O./VT (fig. 3.5). Pentru calculatoarele mari s-a elaborat sistemul de operare O./V. (virtual machine), în care pentru fiecare utilizator se oferă un spaţiu virtual de adrese notate A (1), A (2) . . . A (n). Această gestiune a spaţiului virtual A (n), pentru utilizatorul n se face ca pentru un calculator cu memorie virtuală simplă (alocare periferice, protecţii) încît utilizatorului i se oferă o maşină virtuală proprie IBM/370. În acest spaţiu el poate să lucreze în multiprogramare sau chiar în multiacces. Ţinul declarat urmărit la sistemul de operare IBM/370 O./V. a fost compatibilitatea completă cu produsele software şi programele existente pe sistemele IBM/360 O./V.. S-a extins corespunzător posibilităţile de multiprogramare şi multiacces şi cooperarea între programe. Gradul de multiprogramare a sistemului se indică de utilizatori. Algoritmul de gestiune a paginilor se bazează pe LRU (least recently used), utilizînd bitul de referinţă în elementele tabelei de pagini. Monitorul se încarcă fix în primele segmente ale memoriei virtuale, împreună cu programele partajabile şi cele de gestiune a lucrărilor la intrare şi ieşire (SYS.IN). Primele două segmente sînt rezidente în MC şi conţin tabelele supervizorului de paginaţie. Planificatorul de lucrări realizează şi afectarea periferice lor la partiţiile virtuale. Fiecare partiţie virtuală va avea un tabel propriu de segmente, care conţine referiri la tabela generală de segmente virtuale din MC.

Posibilitatea de a avea 256 segmente, permite protecţie individuală a unui număr maxim de 256 programe. Există astfel posibilităţi mari de implementare a subsistemelor specializate cu multiacces, faţă de sistemele cu gestiune a memoriei prin chei de protecţie, care permit maxim 12 - 14 programe utilizator cu protecţie individuală. Se realizează o încărcare intensivă a MC şi transfer automat a programelor între MC şi cea virtuală la nivel de segmente. Memoria virtuală poate exista pe mai multe discuri. La încărcarea unui program din bibliotecă el trece în memoria virtuală şi numai la activare, primul segment e încărcat în MC.

Problemele deosebite pun translatarea adreselor din programele de canal. Canalele sînt procesoare I/E specializate, care lucrează în paralel cu UC. Teoretic s-ar putea realiza

translaţie prin aceleaşi mecanisme ca la programele de UC. Natura secvenţială sincronă de mare viteză a transferurilor de informaţie cu periferice şi lucrul în multiplexare a canalelor, fac practic imposibilă implementarea unei translaţii hardware. Dacă pagina în care s-a ajuns cu transferul nu este prezentă, trebuie întreruptă operaţia de I/E, continuate celelalte, încărcată pagina şi reluarea transferului. Întreruperea transferului unui bloc de bandă magnetică nu e posibil. Din aceste motive translaţia adreselor din programele de canal se face în exclusivitate, prin software.

Înainte de lansarea unei operaţii de I/E trebuie încărcate în MC programul de canal şi zonele tampon necesare. Se translatează programul de canal, paginile utilizate se marchează ca fixe (F = 1) şi se efectuează transferul. Paginile aferente nu vor putea fi înlocuite până la terminarea transferului, când se pune F = 0. Aceste funcţii se adaugă în supervizorul de intrări-ieşiri. Ele vor putea fi utilizate de UC prin mecanismul normal de reamplasare. Din cauză că zonele tampon sînt mari, per total sistem se limitează numărul de I/E posibile simultan, deci paralelismele posibile în sistem. Limitarea depinde de dimensiunile zonelor tampon şi capacitatea MC.

Canalele transmit datele direct în MC în anumite zone, care nu pot fi controlate "on line" dacă sînt corecte. Pot apare transferuri în zone greşite dacă programele de canal se modifică dinamic. Acestea trebuie să fie detectate de mecanismul de protecţie al memoriei canalului. Sistemul trebuie să rezerve pentru programele de canal zone tampon în memoria virtuală, care să aibă adrese fixe în MC, încît adresele din programele de canal să fie cele din momentul încărcării şi să nu poată fi modificate dinamic. Protecţia prin bit de activitate nu se poate aplica la adresările făcute de canal numai la lansarea transferului.

Deoarece există posibilitatea unui grad înalt de multiprogramare, la un moment dat pot lucra multe programe activate pe rînd de UC. Deşi UC are capacitate de deservire, dacă programele sînt mari ele pot să ajungă la o mare consumenţă în utilizarea paginilor. Se ajunge la un schimb intens de pagini care scad mult performanţele sistemului numit "thrashing" (vînturare). Pentru a evita acest lucru se introduce în supervizorul de paginaţie, un modul de detectare a thrashing-

-ului". El se bazează pe numărarea paginilor înlocuite în unitate de timp. Limitarea "trashing-ului" se poate face automat prin punerea în aşteptare a lucrărilor cu prioritate redusă. Ele vor fi automat reactivate la scăderea "trashing-ului".

Există o legătură strînsă între gradul posibil de multiprogramare şi stilul de programare. Scrierea unor programe structurate, modulare, la care să existe puţine salturi între module permite un grad înalt de multiprogramare la execuţie, deşi o utilizare intensivă a sistemului de calcul. In fig.3.6 se prezintă rezultatele experimentale rezultate din rularea simultană a n programe identice de sortare a 10.000 de înregistrări de 40 octeţi fiecare, folosind un program oarecare şi unul care respectă un stil optim. S-au utilizat pagini de 1 K, cuante de timp UC de 0.1 sec, MC de 184 K şi programe de 129 pagini fiecare. Datorită adresărilor aleatoare creşte rapid timpul de răspuns. Curba 3 corespunde aceluiaşi program neperformant dar cu utilizarea unui modul de limitare trashing. [Hal 75]

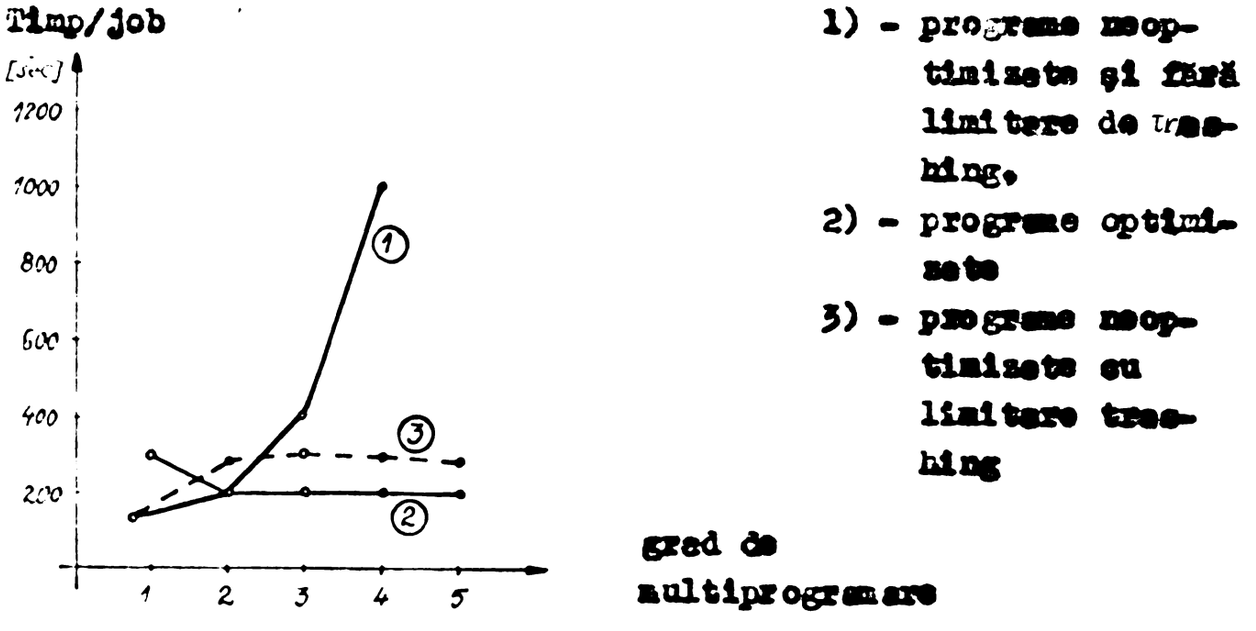


Fig.3.6. Limitarea traficului de pagini în multiprogramare.

Se reduce traficul de pagini dacă se utilizează bitul C din tabela de pagini. Nu se vor scrie paginile nemodificate (C = 0) în memoria virtuală la înlocuire. Aceasta necesită modificări hard pentru poziţionarea C = 1 la orice operaţie de memorare. Utilizarea virtualizării cu segmentare pentru memorii de capacitate redusă micşorează trashingul dacă se respectă un nivel anumit de multiprogramare. Pentru a evita fragmentarea memoriei virtuale, partiţiile virtuale se alocă în multipli de segment. Fiecărei partiţii i se alocă şi un segment modul de legătură.

Sistemul de operare OS/VS2 dispune de o opţiune de multiacces (TSO), care lucrează ca un subsistem cu multiacces TSO clasic (vezi 1.2.5.), dar cu facilităţi suplimentare. Mai mulţi utilizatori au programe protejate individual, care se execută în una sau mai multe partiţii virtuale. Traficul de pagini în multiacces creşte mult faţă de prelucrarea în loturi. Swapingul programelor se face automat pe blocuri de pagini. Paginile nemodificate nu trebuie salvate pe disc.

Sistemul de operare IBM/370 OS/VM (virtual machine) desvoltat după CP 67 (control program) experimentat pe IBM/360 mod 67, oferă un multiacces mai larg. Fiecare terminal utilizator e văzut ca o consolă centrală a unei maşini virtuale "IBM/370" căreia i se alocă periferice virtuale de toate tipurile şi dispune de un spaţiu virtual complet. Utilizatorul dispune de toate facilităţile unui sistem de operare OS/VS2 pentru lucrul în batch, sau poate cere ca maşina virtuală să lucreze în multiacces, sub comanda unui monitor special CMS (Conversational Monitor Systems). Sistemul de operare OS/VM se pretează numai pentru prelucrarea în multiprogramare extinsă, deoarece la o încărcare mare, ca cea din multiacces, performanţele calculatorului scad la 50 % faţă de cele obţinute în OS/VS2.

3.3. SISTEMUL DE OPERARE MULTICS [Sol 74] [Jur 84] [Hel 75] [Sch 72]

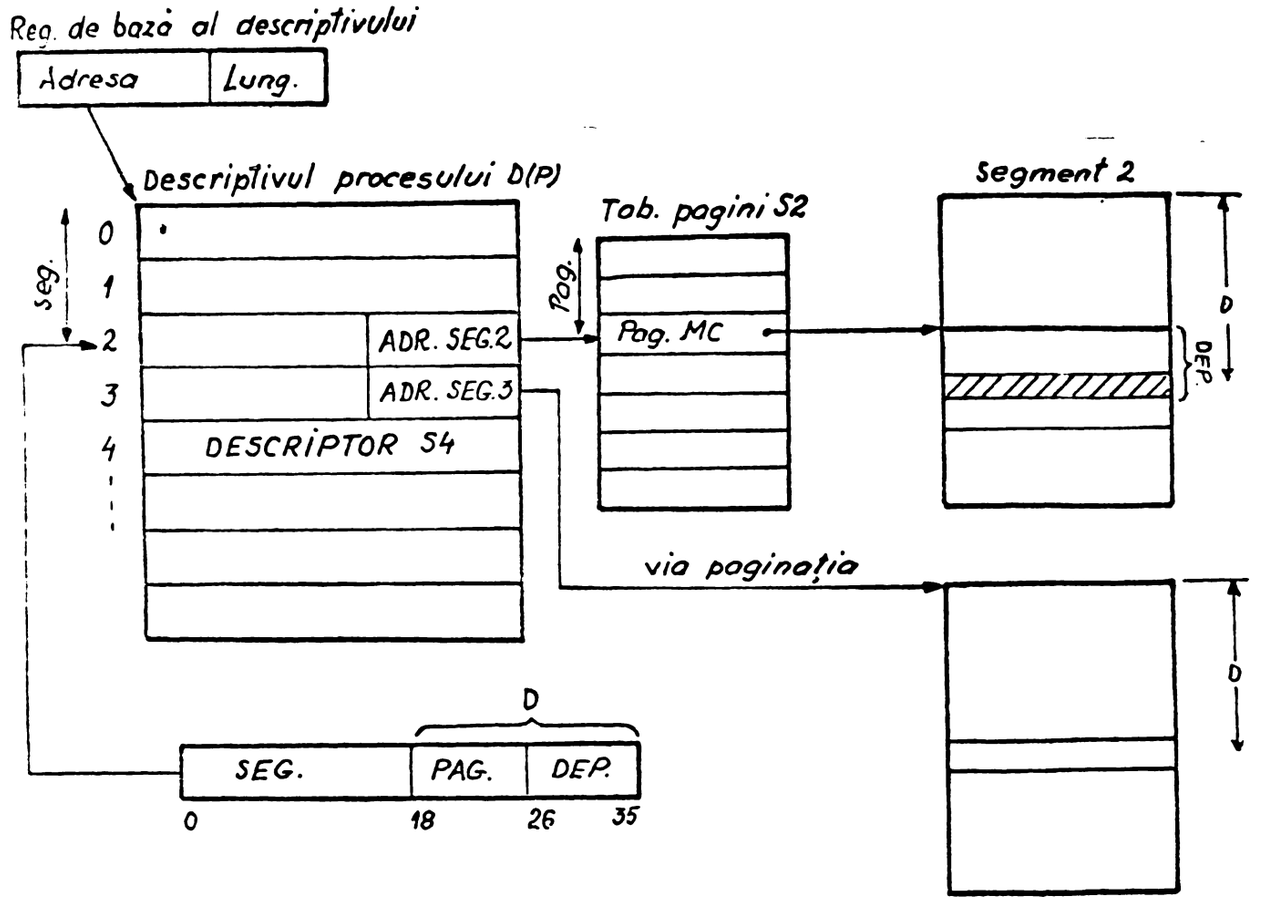
MULTICS (Multiplexed Information and Computing Service), este cel mai complex sistem de operare definit special pentru multiacces larg şi generalizat. Toţi utilizatorii se consideră că lucrează în mod conversaţional. Numărul lor este teoretic nelimitat. Calculatorul a fost adaptat ca hardware modelului software conceput. Obiectivul principal

urmărit este de a facilita utilizatorilor să partajeze toată informaţia aflată în sistem, cu măsuri corespunzătoare de securitate şi protecţie.

Sistemele de operare actuale IBM, pornesc de la principiile clasice de multiprogramare pentru compatibilitate pe care le dezvoltă în noul context al memoriilor virtuale şi le adaugă facilităţile sporite de multiacces prin utilizarea unui monitor de multiacces (TSO sau CMS). Sistemul MULTICS pleacă de la un concept nou, total conversaţional, care reuneşte toate funcţiile de multiacces prezentate în cap.1.2. Utilizatorii trebuie să aibă acces la toată informaţia aflată în sistem, indiferent pe ce nivel ierarhic de memorie e depusă. Dispare noţiunea de fişier, sistem de gestiune a fişierelor, operaţii de intrare ieşire, bază de date, periferice, supervizor de I/O. Toate suporturile de informaţie sînt considerate unitar ca o memorie virtuală foarte vastă, împărţită în segmente (256.000 codificate pe 18 biţi), fiecare segment putînd avea dimensiunea de 1Mo (256 pagini) şi fiecare pagină circa 4 Ko (1 K cuvinte de 36 biţi). Localizarea fizică a informaţiilor pe suporturi este cunoscută de sistem prin tabele de segmente şi pagini. Se asigură transferul automat între nivelele de memorie a segmentelor şi paginilor la cererea utilizatorilor. Prin program se poate referi orice informaţie, prin adresa virtuală (segment, pagină, cuvînt în pagină). Fiecare segment poate fi partajat şi se protejează individual prin atribute de acces.

Sistemul MULTICS foloseşte memorie virtuală de tip AS (n). Fiecare utilizator are spaţiul de memorie delimitat printr-o tabelă de segmente care face referiri la cîte o tabelă de pagini, ş.a.m.d. Modelul de adresare este de tip cu segmentare şi paginaţie (vezi 3.1.). Segmentele de program sînt reentrante şi procedurile recursive, iar editarea legăturilor este dinamică.

Segmentele pot fi de procedură sau de date. Utilizatorului i se asociază pe perioada unei sesiuni de lucru un proces, care execută segmente de procedură ce lucrează pe segmente de date. Spaţiul de adresare al procesului este dat de către segmentele utilizate. Descriptorii segmentelor unui proces (DS) sînt grupaţi într-un segment particular numit descriptiv proces D (P). Registrul de bază al descriptivului (fig.3.7.) indică la un moment dat procesul activ, prin referirea descriptivului lui.



D - deplasare în segment
DEP - deplasarea în pagină

Fig.3.7. Adresarea într-un proces MULTICS.

Descriptorul conține numărul virtual al segmentului, lungimea lui, adresa de lansare și drepturile de acces ale procesului asupra segmentului. La apelarea segmentului se încarcă prima pagină, care poate conține spațiu pentru generarea tabelei de pagini a segmentului, se încarcă pagina cu adresa de lansare și se lansează execuția. Celelalte pagini se vor încărca prin cerere de pagină. La adresare, numărul segmentului va indica intrarea în descriptivul procesului. Descriptivul

nu e accesibil procesului pe care îl descrie, deoarece nu există un descriptor pentru el şi nici pentru alte procese. In MULTICS nu există procese sistem . Monitorul este format dintr-un ansamblu de proceduri partajate, executate de procesele utilizator.

Pentru acelaşi segment pot exista simultan mai mulţi descriptori în procese diferite, cu drepturi de acces diferite care partajează segmentul (fig.3.8) [Rău 82]

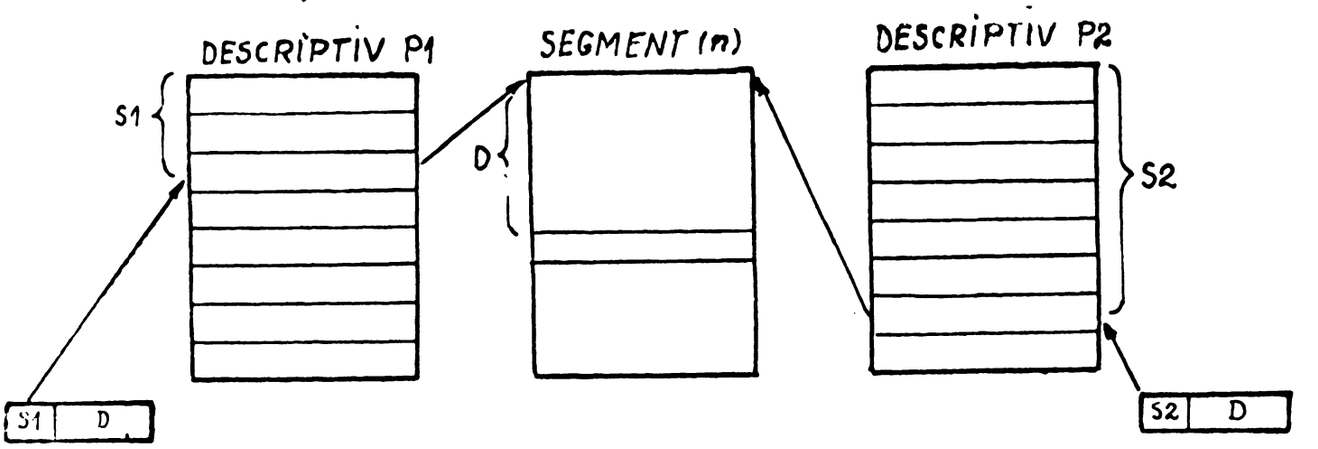


Fig.3.8. Partajarea segmentelor între procese.

Procesul P 1 va apela segmentul n cu numele S1, iar procesul P 2 cu numele S2. Adresa relativă a informaţiei în segment va fi aceeaşi.

Execuţia unui proces se face în segmente-proceduri.

Instrucţiile sînt de lungime fixă şi cuprind : codul operaţiei, registrul general utilizat (R0-R15), cîmp de adresă, indicator de indexare sau/şi indirectare, registru de bază (RB). Adresa calculată a operandului se calculează ca (RB) + Deplasamentul şi poate fi corectată prin adunarea registrului index (R4), dacă indicatorul de indexare e pe "1". Registrul R0 conţine prin convenţie adresa segmentului - procedură în curs de execuţie. Adresările locale în segment se fac în forma (R0, D) şi sînt salturi sau încărcări de constante (fig.3.9)

Obiectele (variabile) utilizate într-o procedură pot fi interne, externe sau parametri . Obiectele interne pot fi de tip etichetă (desemnează instrucţii şi constante utilizate în citire), locale şi globale.

Obiectele locale sînt create la fiecare apel al unui proces la procedură şi sînt distruse la terminarea execuţiei procedurii. Valorile lor sînt cele generate în faza de compilare.

ADRESARE IN PROCEDURI :

OBIECTE - INTERNE - etichete
- locale
- globale
- EXTERNE - cuvinte în seg.date
- porți de intr. în seg.proc.
- PARAMETRI

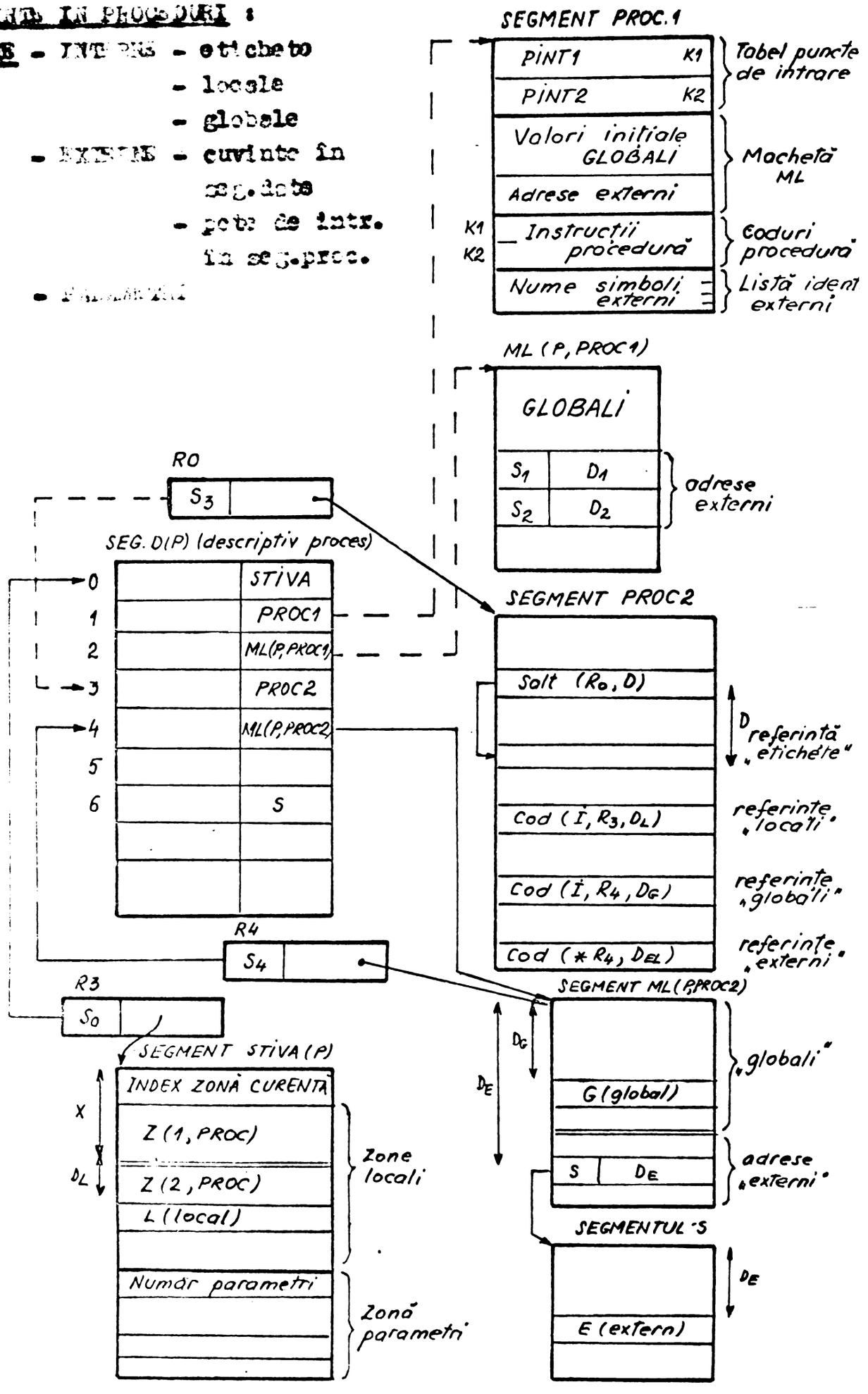


Fig. 3.9. Adresare obiecte interne și externe în procedura MULTICS.

Se asigură astfel <u>invocarea procedurii și reintrarea ei</u>. Pentru recursivitatea procedurilor zonele de locali se gestionează în stivă. Fiecărui proces i se atașează un segment special ce conține această stivă. Adresa vîrfului stivei pentru procedura ce se execută se găsește în B7 (fig.3.9) și în primul element al stivei. Adresa acestui segment se găsește în descriptivul procesului. În fig. 3.9 se arată modul de adresare a unui local L, dintr-o procedură P, la al doilea apel recursiv, într-un proces P, care are descriptivul D (P). Se folosește indexarea cu B7. În stiva de locali a procesului P, (care evoluează în joc) sînt create două noui de locali - (1, P, D) și 3 (2, P, D), corespunzătoare celor două apeluri recursive ale procedurii P, D.

<u>Obiectele globale</u> ale unei proceduri se creează la prima execuție a procedurii de către un proces și se distrug la terminarea procesului. La prima execuție a procedurii globalii au valoarea fixată la compilare, iar pe parcurs vor păstra modificările suferite în execuțiile anterioare ale procedurii. Valorile inițiale se găsesc în procedură. Valorile obținute în cursul execuțiilor procedurii se păstrează într-un segment modul de legătură ML (P, PROC), creat în momentul primei apelări a procedurii de către proces. Descriptivul segmentului ML (P, PROC) se introduce în descriptivul procesului. Acest segment nu e accesibil altor procese sau altor proceduri. Procesul va avea pentru fiecare procedură un segment de legătură ML (P_i, PROC). Referirea unui global G dintr-o procedură se face prin registrul 4 sub forma Cod (B4, D_G), ca în fig. 3.9, el găsindu-se în modulul de legătură al procedurii, la distanța D_G față de început.

<u>Accesul la obiectele externe</u> din alte segmente, care nu s-au creat în procesul în curs de execuție se face prin intermediul modulului de legătură. Externii sînt componente elementare ale unui segment de date (cuvinte), care pot fi citite sau modificate, și puncte de intrare într-un segment de procedură. Procedura utilizatoare nu cunoaște adresele externilor ei și face referințe prin nume simbolice. Prin editarea legăturilor în modulul de legătură ML (P, PROC) se vor plasa adresele segmentate ale externilor. Referirea lor se va face prin adresare indirectă referitoare la adresele lor din ML (P, PROC), sub forma Cod (* B4, D_{EL}), unde D_{EL} este deplasamentul în ML (P, PROC) a adresei segmentate a externului S, de forma (S, D_S). Prin S s-a notat numărul segmentului de date referit,

iar prin D_E deplasamentul cuvîntului E în segment (fig.3.9). Descriptorul segmentului S se găseşte în descriptivul D (P) al procesului.

Parametrii sînt obiecte transmise procedurii de alte proceduri, pentru utilizare pe timpul execuţiei, în momentul apelului. Parametrii formali din procedura apelată desemnează cuvinte, care în momentul execuţiei vor conţine adresa parametrului efectiv. Referirile la parametrii se vor face prin adresarea indirectă a acestor cuvinte. Parametrii efectivi se dau la apel şi pot fi interni (constante, locali, globali), externi sau parametrii ai procedurii apelante. Adresa unui parametru efectiv este calculată în mod dinamic de către procedura apelantă în funcţie de tipul parametrului. Numărul şi adresele calculate ale parametrilor efectivi sînt memorate în stiva procesului, într-o zonă special rezervată, în continuarea zonei locale a procedurii apelante. Pentru transmiterea unei constante (c) se plasează valoarea constantei ca local în stivă şi adresa lui segmentată (S, D + D_c), utilizînd registrul R3 (fig. 3.10). Pentru un global (G), care se găseşte în modulul de legătură ML (P,PROC), folosind registrul R4 se calculează adresa (ML, D_G). Adresa unui extern e preluată din ML (P,PROC) şi trecută zona rezervată parametrilor în stivă sub forma (S, D_E). Pentru transmiterea unui parametru al procedurii apelante furnizat de altă procedură, adresa se se copiază dintr-o listă în alta prin registrul R5.

Tabela adreselor parametrilor se pot transmite procedurii apelate prin registrul R3. Din cauză că protecţia în MULTICS se realizează pe 8 inele, fiecare proces are 8 stive (una pe inel) şi opt module de legătură, parametrii se adresează prin registrul R5 din procedura apelată cu indirectare. Prin mecanismele de acces prezentate se permite accesul la toate tipurile de obiecte, în diferite forme. La apelarea procedurii în R5 se încarcă adresa de început a zonei de parametrii creată de procedura apelantă în stiva procesului de pe inel i, în care se va executa procedura apelată (fig.3.11).

Conţinutul vechi al registrelor L4 şi R5, corespunzător procedurii apelante se memorează în stivă. Pentru procedura apelată se creează o zonă de locali în stivă, în faţa căreia

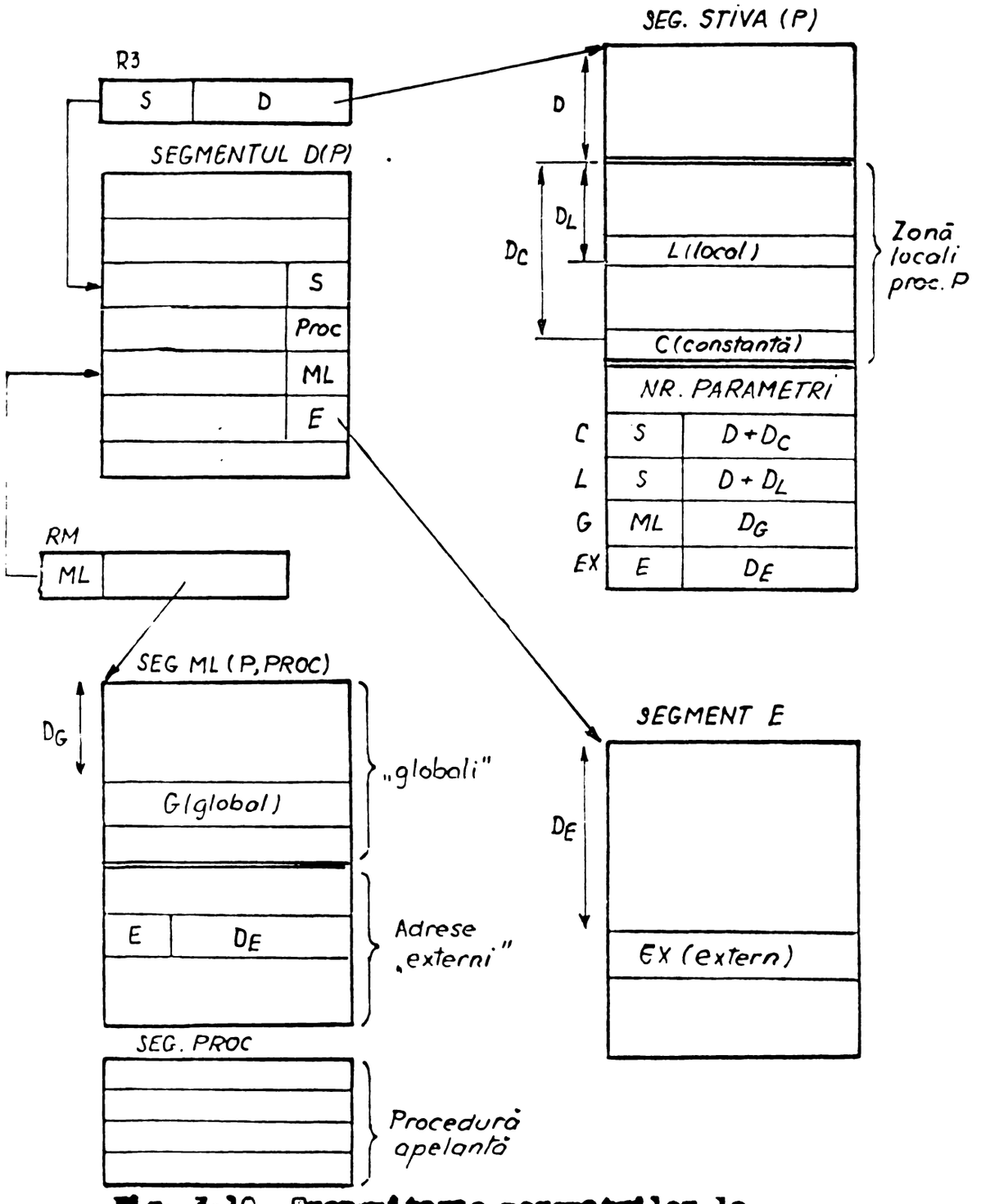
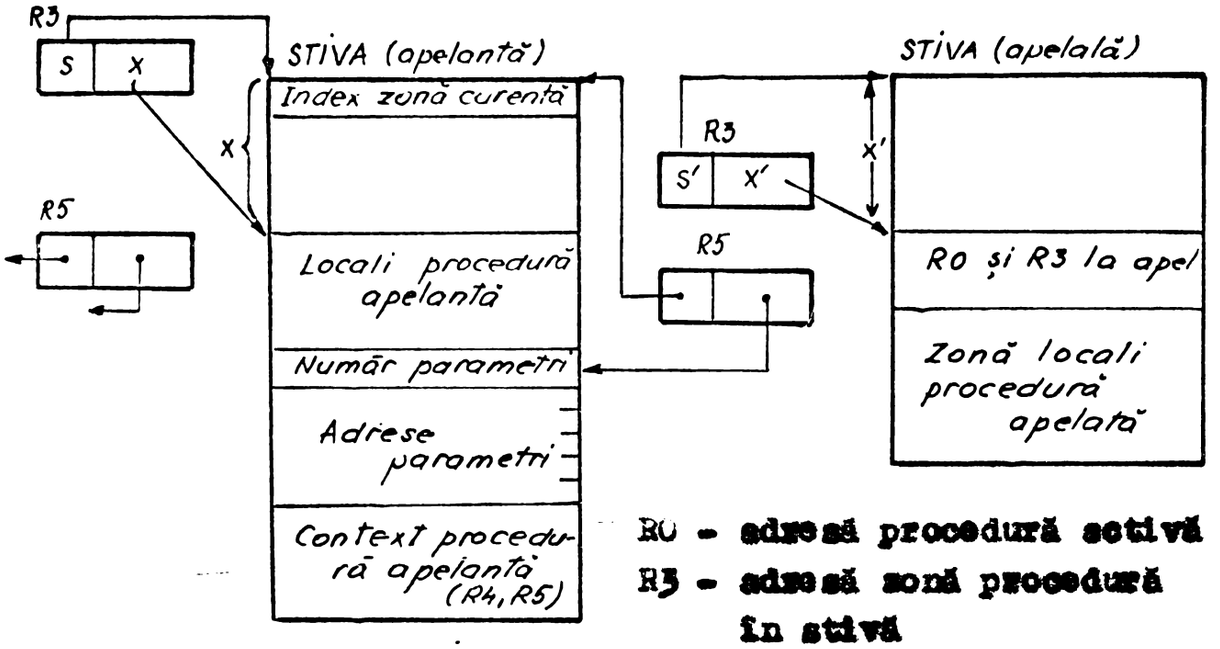


Fig. 3.10. Transmiterea parametrilor la apelul unei proceduri MULTICS.

se memorează vechile valori ale lui R0 şi R3. Se creează şi pentru această procedură un nodul de legătură, pe inelul i, care va conţine globalii şi adresele externilor utilizaţi. Pentru externii al căror adrese nu se cunosc, se lansează printr-o deviere editarea dinamică a legăturilor. Pe baza sumelor simbolice ale externilor (fig. 3.9) din procedura apelată se identifică adresele lor, folosind catalogul sistemului. Adresele se generează cu un număr de segment corespunzător poziţiei segmentului, care conţine externii în descriptivul procesului. Având R0, R3, R4 şi R5 încărcaţi corespunzător se lansează execuţia procedurii.



R0 - adresă procedură activă
R3 - adresă zonă procedură în stivă
R4 - adresă ML procedură în stivă
R5 - adr.tab.parametrii de apel.

Fig. 3.11. Apelarea unei proceduri în MULTICS.

La revenirea din procedură apelată se distrug localii creați, se refac R0 și R3, se reactivează procedura apelantă, care își poate reface singură R4 și R5.

Protecția informației în MULTICS se bazează pe conceptul de inele de protecție [Sch 72], [Sol 74]. Segmentele sînt unități elementare de partaj și protecție. Ele sînt accesibile proceselor în a căror descriptiv au fost introduse. Fiecărui segment i se asociază în spațiul numelor simbolice (Catalog) o listă de elemente de forma (P_i, D_{ij}), care indică pentru ce procese P_i segmentul este accesibil și ce drepturi de acces are. La prima cerere de acces a procesului P_i la un segment S_j se verifică dacă acesta e trecut în lista de acces a segmentului, care e introdus în descriptivul procesului. Varietatea drepturilor de acces D_{ij} ale procesului P_i asupra segmentului S_j pot varia funcție de contextul execuției, specificat printr-o variabilă numită inel de protecție. In MULTICS sînt 8 inele de protecție (0 - 7), posibilitățile de acces fiind invers proporționale cu numărul inelului. Se asigură astfel drepturi diferite unor anumite clase de procese. Inelele sînt distribuite astfel :

0 - funcţiile vitale ale sistemului, proceduri de I/E, de alocare resurse, validare acces, etc.
1 - traductoare, gestiune fişiere
2 - 3 proceduri de bibliotecă, subsisteme utilizator protejate
4 - 7 programe utilizator.

Procesele din inelele inferioare au toate drepturile inelelor superioare. Lista care specifică drepturile procesului cînd lucrează în diferite inele se numeşte <u>parantează de acces</u> şi se dă pentru fiecare operaţie posibilă : citire (C), scriere (S), execuţie (E).

Schimbarea contextului de execuţie al procesului poate amplifica sau diminua drepturile sale de acces. Un proces P_1 are acces la un segment de procedură numai prin anumite puncte de intrare, numite <u>porţi de acces</u>, care sînt o submulţime a punctelor de intrare. După ce s-a validat apelul printr-o poartă de acces, procesul poate evolua în alt inel, care îi oferă drepturi suplimentare. Pentru a putea apela o poartă de acces, trebuie să evolueze în anumite inele, specificate într-o listă asociată perechii poartă-acces numită <u>parantează de apel</u>.

Implementarea controlului informaţiilor prin inele de protecţie presupune existenţa unor facilităţi hardware de verificare a accesului în timpul execuţiei. Descriptorul segmentului din descriptivul procesului conţine şi drepturile procesului asupra segmentului. El conţine (fig. 3.12.) adresa fizică (AF) şi lungimea (L) a segmentului, numere de inele I1, I2, I3, care determină parantezele de acces la segment. Paranteza de apel este de obicei un inel, excepţînd procedurile partajate pentru care se specifică mai multe inele. Apelurile la procedură se validează numai prin porţile de acces definite de cîmpul AP. Operaţiile posibile pe segment sînt indicate de C=1 (citire), S=1 (scriere), E=1 (execuţie).

AF	L	I1	I2	I3	C	S	E	AP

PARANTEZE DE-SCRIERE [0,I1]
-CITIRE [0,I2]
-EXECUŢIE [I1, I2]
-APEL [I1 + 1, I3]
OPERAŢII POSIBILE : - citire (C = 1)
- scriere (S = 1)
- execuţie(E = 1)

Fig.3.12. Structura unui descriptor de segment

Verificarea drepturilor de acces se face folosind şi informaţiile din registrele generale şi speciale, care conţin şi numărul inelului curent în care evoluează procesul. Controlul se efectuează la execuţia fiecărei instrucţii, în diferite faze (extragere instrucţie, calcul adresă, execuţie). Pentru protecţia zonei de lucru a unui proces se rezervă câte o stivă a procesului în fiecare inel. Stiva dintr-un inel este accesibilă numai procedurilor din acel inel.

Majoritatea implementărilor de MULTICS se fac cu 3-5 inele de protecţie. Lucrul în cele două regiuni clasice supervizor/program se poate considera ca o protecţie cu două nivele. Pentru proiectarea unor sisteme de operare flexibile şi sigure sînt suficiente 4 inele de protecţie :

0 - proceduri de alocare resurse, proceduri de I/E, tratare întreruperi şi devieri, control acces, etc.

1 - sistem de gestiune fişiere, traductoare, bibliotecar, editor de legături.

2 - subsisteme specializate utilizator pentru multiacces, gestiune baze de date, gestiune timp real, etc.

3 - programe utilizator.

O astfel de soluţie se aplică în noua serie de calculatoare C11 - MB/64.

O metodă mai generală de organizare a accesului la informaţie şi protecţie, derivată din sistemul MULTICS este cea bazată pe noţiunea de capabilitate şi domeniu de protecţie [Fab74], [Jur84].

După 20 de ani de la definirea sa, la Institutul Tehnologic din Massachusetts şi implementarea pe calculatoare Honeywell 6000, sistemul de operare MULTICS rămîne cel mai general şi mai puternic sistem cu multiacces. In ultimii ani a fost implementat şi pe calculatoarele IRIS-80 şi C11-MB "level 64" produse în Franţa şi pe cele mai puternice calculatoare de tip CRAY.

3.4. SOLUŢII DE IMPLEMENTARE A UNOR SISTEME DE OPERARE CU MEMORIE VIRTUALĂ ŞI MULTIACCES PE CALCULATOARE DE CAPACITATE MEDIE

Memoria virtuală reprezintă o necesitate la calculatoarele de capacitate medie moderne, uşor de implementat avînd în vedere viteza mare a circuitelor de memorare existente la un preţ relativ redus. Raportul performanţe/cost justifică această implementare, care permite perfecţionarea considerabilă a sistemelor de operare în special în domeniul multiaccesului, aşa cum s-a văzut în subcapitolele anterioare. Unele metode aplicate la calculatoarele mari pot fi adoptate, iar pentru anumite probleme trebuie găsite soluţii noi acceptabile ca preţ.

La proiectarea unui sistem de calcul cu memorie virtuală trebuie avut în vedere contextul hardware-software care realizează virtualizarea memoriei şi compatibilitatea cît mai mare cu vechile sisteme de operare. Proiectarea completă a unui sistem de operare necesită un efort enorm de concepţie, evaluat la 5000 om x an. Din acest motiv se recomandă ca noile sisteme de operare să preia o parte cît mai mare din vechile componente, într-o arhitectură nouă, cu performanţe superioare. Componentele hardware trebuie şi ele completate pentru a obţine noi posibilităţi de dezvoltare software. Cu toate perfecţionările aduse sistemului de operare (descrise în 2.1), nu se va putea realiza un subsistem cu multiacces performant şi flexibil fără memorie virtuală.

In acest subcapitol se vor prezenta soluţii de modificare hardware şi software minime, pentru a realiza un sistem cu memorie virtuală, cu performanţe cît mai bune la un raport cost/performanţă redus. Ca metode de virtualizare a memoriei, pentru calculatoarele de capacitate medie, se poate lua în discuţie numai virtualizarea cu paginaţie şi virtualizarea cu segmentare (vezi 3.1). Virtualizarea cu segmentare şi paginare e cea mai perfecţionată, dar foarte scumpă implicînd circuite complexe de translatare a adreselor virtuale.

Pentru a reduce memorie rapidă necesară pentru translatare, vom considera pagini de 4 Ko şi segmente formate din număr întreg de pagini. In acest caz numărul de biţi necesari pentru adresele de segment, pagină virtuală şi pagină din MC

se reduce mult. Dăm mai jos calculul dimensiunii tabelelor de translatare şi structura lor pentru cele trei tipuri de virtualizare, în cazul unei memorii virtuale (MV) maxime de 16 M, cu segmente de maxim 64 K şi memorie centrală maximă de 2 Mo.

S-a luat MV maximă de 16 M pentru a se putea adresa cu 24 biţi şi a păstra compatibilitatea cu modul adresare în calculatoarele FELIX şi IBM/360/370 unde adresa calculată poate fi pe maxim 24 biţi (după bazare şi eventual indexare). S-au ales segmentele de maxim 64 K pentru a asigura compatibilitatea cu formatul instrucţiei, care are pentru partea de adresă (deplasament în segment) 16 biţi. Lungimea paginilor nu se poate menţine la 2 K deoarece s-ar dubla dimensiunea tabelelor şi a memoriei rapide de translaţie (MRT). Pagini de 4 K folosesc toate calculatoarele moderne, asigurându-se o utilizare suficient de bună a MC.

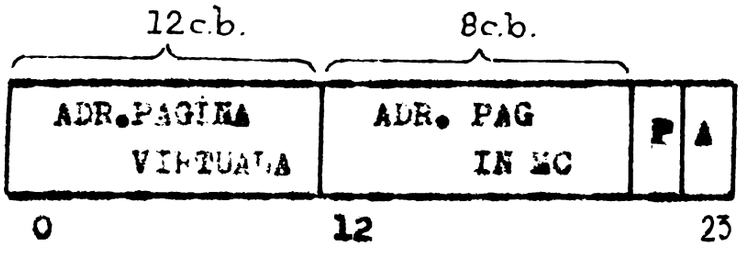
A. <u>Memorie virtuală cu paginaţie</u> şi translaţie cu memorie asociativă (MA)

MV = 16 Mo = 4 K pag. - necesar 12 biţi pentru Nr. pag. în MV.

MC = 1 Mo = 256 pag. - necesar 8 biţi pentru Nr. pag. în MC.

PAG= 4 Ko - necesar 12 biţi pentru adr.octet în pag.

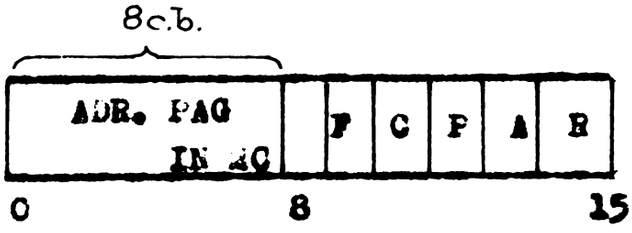
Un cuvînt din memoria asociativă va avea 3 octeţi (24 biţi) cu structura :



P = 1 pagină în MC.
A = 1 pagină activă

Tabela paginilor virtuale, din MC va avea 4 K x x 2 oct = 8 Ko.

Fiecare element al tabelei va avea 2 octeţi şi structura :



F = 1 pagină fixă
C = 1 pagină de procedură reentrantă
A = 1 pagină activă
P = 1 pagină prezentă în MC.
R = 1 pagină referită

Pentru o memorie asociativă de 32 registre, la o MC de 1 Mo, pot apare numeroase referiri la pagini care nu sînt în MA, mai ales la programe cu cod neoptimizat. Aceasta reduce viteza de lucru, mai ales cînd se lucrează în multiacces cu multe programe în execuţie paralelă. Mărirea MA duce la o creştere a timpului de acces. Utilizarea unei memorii rapide de 8 K pentru a conţine toată tabela de pagini nu se justifică ca preţ.

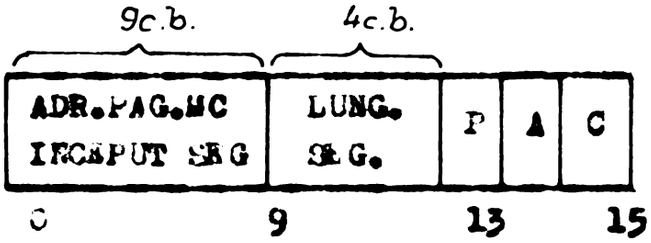
B. <u>Memorie virtuală cu segmentare simplă</u>

MV = 16 Mo = 256 seg. x 64 K - necesar 24 biţi pentru adresare, 8 biţi pentru Nr.seg.

MC = 2 Mo = 512 pag - necesar 9 biţi adr. pag.în MC.

SEG= 16 pag. x 4 K = 64 K - necesar 4 biţi adr. pag.în seg.;16 biţi adr.oct.în seg.

Un element din tabela de translaţie segmente va avea 2 octeţi şi va conţine adresa de început a segmentului în MC (prima pagină) şi lungimea segmentului (nr. de pagini).



P = 1 segment present în MC
A = 1 segment activ
C = 1 segment de procedură reentrantă sau nemodificat

La aceste convenţii de adresare, lungimea unui element din tabela de segmente se reduce la jumătate faţă de adresarea clasică (adresă segment în MC în Mo → 12 biţi + 16 biţi lungime). Pentru o MV de 16 Mo este necesară o memorie rapidă, pentru tabela de segmente :

256 seg x 2 oct = <u>512 oct</u>

Aceasta înseamnă foarte puţin dacă ţinem cont că pentru protecţia prin chei de protecţie de 4 biţi, a unei memorii centrale de 2 Mo (FELIX 8010) cu pagini de 2 Ko este necesară o memorie rapidă :

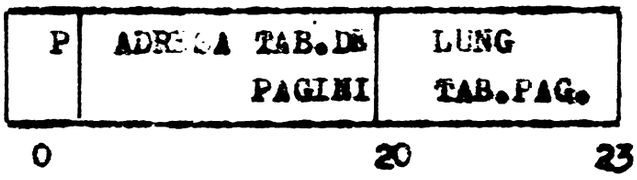
1024 pagini x 0,5 oct = 512 oct.

In plus memoria rapidă de translaţie asigură şi protecţia programelor din MC (prin bit de activitate A) şi dimensiunea ei nu depinde de dimensiunea MC, ea le calculează două tipuri de virtualizare. Deci cu aceeaşi memorie se realizează atît virtualizarea memoriei cît şi protecţia.

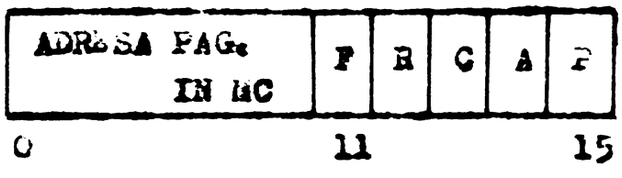
C. Memorie virtuală cu segmentare şi paginare

MV = 16 Mo = 256 seg x 64 K = 4 K pag x 4 K
MC = 2 M = 512 pag x 4 K - necesar 9biţi adr.pag.în MC
SEG= 16 pag x 4 K = 64 K - necesar 4 biţi pt.lung.tab. pag/seg.
PAG = 4 K - necesar 12 biţi adresă oct. în pag.

Dacă se foloseşte o memorie rapidă pentru tabela de segmente de 1 K şi una de 8 K pentru tabele de pagini/segment, elementele celor două tabele pot avea structura :



- element tabela de segmente



- element tabelă de pagină/seg.

Tabela de segmente ocupă :

256 seg x 3 oct = 768 oct.

Tabela de pagini la o încărcare 100 % a MV ocupă :

256 seg x 16 pag = 4 K pag
4 K pag x 2 oct = 8 Ko

- In realitate MV e încărcată mult sub 100 % şi memoria rapidă poate să fie de capacitate mai redusă. Dacă se depăşeşte, tabelele pot fi depuse în MC. La calculatoarele mari, memoria rapidă trebuie să asigure poziţionarea " on line " a biţilor de modificare (C) şi de referire (R). Deasemenea va compara automat lungimea tabelei de pagini cu numărul paginii din adresa virtuală. Din acest motiv aceste memorii vor avea o construcţie specială. Pentru mărirea vitezei de translaţie se prevede şi o memorie asociativă (vezi 3.13).

Proiectarea structurii "hardware" şi "software" a unui sistem cu memorie virtuală cu segmentare şi facilităţi largi de multiacces.

Din analiza celor trei soluţii de virtualizare a memoriei prezentate, am ales memoria virtuală cu segmentare simplă, deoarece are cele mai bune caracteristici performanţă/cost, pentru calculatoarele de capacitate medie. MV cu paginaţie are cel mai redus cost de implementare, necesitând doar o memorie asociativă (32 - 64 cuvinte), restul translaţiei se rezolvă prin tabele şi proceduri "software". Se nu ţine cont de structura segmentată a programelor. Asocierea unor anumite pagini virtuale consecutive unui segment de program se face printr-o tabelă de segmente, iar un program e plasat într-o partiţie virtuală. Paginaţia asigură o utilizare intensivă a MC, dar deselе pagini care lipsesc din memoria asociativă şi trebuie translatate prin software, reduc viteza de adresare. In sistemele cu multiacces se găsesc în lucru multe programe, pentru care transferurile între MC şi MV sînt frecvente. Efectuarea acestor transferuri pagină cu pagină nu e economică, implicînd multe adresări la disc (şi chiar poziţionări). In sistemele mari se adoptă metoda transferului pe blocuri de pagini, numită în sistemul IBM/370 OS/VS2 T30 "block-paging technique".

Vom prezenta în continuare elementele de bază ale unui sistem cu memorie virtuală cu segmentare simplă, care asigură prin cooperarea hardware - software, funcţii variate flexibilitate şi performanţe ridicate. S-a ţinut cont de compatibilitatea la nivel de program cu sistemele de operare clasice (SIRIS - 3, HELIOS). S-a ales o MV de 16 Mo, pentru a putea fi adresabilă cu 24 biţi, paginile de 4 K şi segmentele de 64 Ko, obţinute printr-o editare de legături obişnuită din module bine-translatabile generate de compilatoare.

Pentru o MC de 2 Mo, a rezultat din calculele făcute mai sus (pct. B) o memorie rapidă pentru tabela de segmente de 512 octeţi, cu 256 intrări şi structura din fig. 3.13. La un moment dat în MC există mai multe segmente, încărcate prin alocare dinamică, ce aparţin unor programe diferite. Adresele şi lungimea lor se găsesc în memoria rapidă de segmente virtuale şi au bitul P = 1 (segment prezent). Segmentele programului activ sînt indicate prin A = 1 (segment activ). Dintre ele unele pot să nu fie încărcate în MC.

Secvenţa de adresare prin hardware a memoriei, pornind de la adresa din instrucţie, într-un spaţiu virtual, este dată principial în fig. 3.13. S-a luat o instrucţie de format fix (ADLIX C256), care e generată după compilare şi editarea legăturilor cu adresa operandului relativă la începutul segmentului în care e definit (deplasament). Segmentul respectiv se găseşte plasat în MV şi are un anumit număr în tabela generală de segmente (Fig. 3.14). La intrarea în segment se încarcă în registrul de bază (R_B) indicat în instrucţie numărul segmentului pe octetul 2. Mecanismul de adresare calculează o adresă pe 24 biţi adunînd deplasamentul din instrucţie cu conţinutul lui R_B. Dacă adresarea e fără indexare aceasta va fi adresa virtuală a operandului la adresare directă.

La adresarea indexată (X = 1) la adresa calculată anterior se adaugă conţinutul registrului index R_X indicat de octetul 3. Dacă adresarea e cu bază, indexul va fi pe 2 octeţi. Dacă adresarea e fără bază, R_X va conţine o adresă virtuală (SEG, DSP) ce indică începutul unui tablou.

Adresa virtuală rezultată va adresa MC prin tabela de segmente, astfel :

- Numărul segmentului adresează o intrare în tabelă şi conţinutul ei se obţine în registrul de date al memoriei rapide (MR).
- Se verifică dacă segmentul e prezent în MC (P = 1), iar în caz contrar se lansează printr-o derută programul de încărcare segmente, care încarcă segmentul, actualizează tabela de segmente şi relansează adresarea.
- Se verifică dacă segmentul aparţine programului activ (A = 1). În caz contrar se lansează o deviere eroare de adresare (încălcare protecţie memorie).
- Se verifică dacă deplasamentul adresei virtuale este mai mic decît lungimea segmentului (D < L). În caz contrar, deviere eroare de adresare. Se folosesc primii 4 biţi din D, deoarece LS este în pagini.

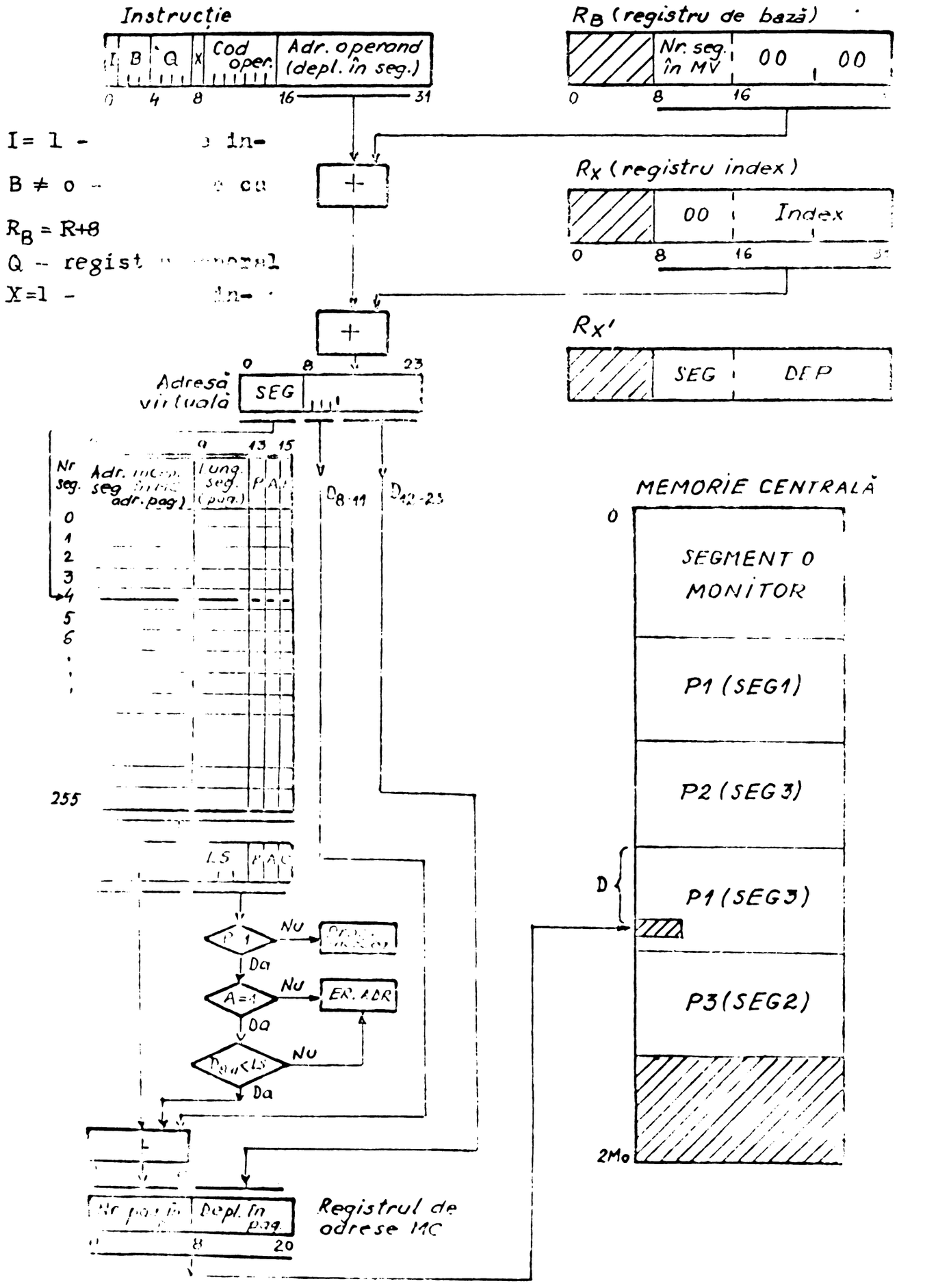


Fig. 3.13. Mecanismul de adresare a memoriei virtuale cu segmentare.

- Cînd toate condiţiile sînt îndeplinite, se adună adresa de început a segmentului cu primii 4 biţi ai deplasamentului şi rezultă numărul paginii MC. Adresa din MC se completează cu ultimii 12 biţi din deplasament (adr.rel. în pagină).

$$A_{pag} = (A_{MSG}) + (D_{8-11})$$

Adresa rezultată este adresa operandului din MC.

Întregul mecanism de translaţie a adreselor virtuale e format din memoria rapidă de 512 oct., un sumator pe 9 biţi şi cîteva circuite logice simple. Calculul adresei virtuale se realizează cu circuitele clasice existente. Un asemenea sistem se poate adăuga la calculatoare existente. Astfel s-a procedat la calculatoarele IBM/370 model 155 şi 165, care au fost lansate pe piaţă fără memorie virtuală şi s-a adăugat ulterior un mecanism de translaţie (DAT = dynamic address translation).

Toate celelalte operaţii de gestiune a memoriei virtuale se fac prin software, în modul în care se va arăta mai jos. Se folosesc de către modulul de gestiune a memoriei virtuale mai multe tabele plasate în zona rezidentă a monitorului. Principalele tabele sînt :

- tabela generală de evidenţă a segmentelor virtuale în lucru, are 256 intrări (cîte una pentru fiecare segment virtual posibil).
- tabela de ocupare a MC, utilizată la evidenţierea spaţiului liber din MC unde se pot înscrie segmente, are cîte un bit pentru fiecare pagină (512 biţi = = 64 octeţi).
- tabela de ocupare a MV, utilizată la gestiunea dinamică a segmentelor în MC, are cîte un bit pentru 4 pagini (4 K pag. : 4 : 8 = 128 octeţi).
- tabela de control a programelor, conţine adresa tabelei de segmente asociate programului, zonă de salvare registre şi alte informaţii de stare (cîte o intrare pentru fiecare program în lucru).
- tabela de segmente a programului (cîte una pe program) conţine numărul segmentului în memoria virtuală (intrare în tabela generală de segmente), drepturile de acces ale programului asupra segmentului (scriere, citire, execuţie) şi numărul punctelor de intrare admise (porţi de acces).

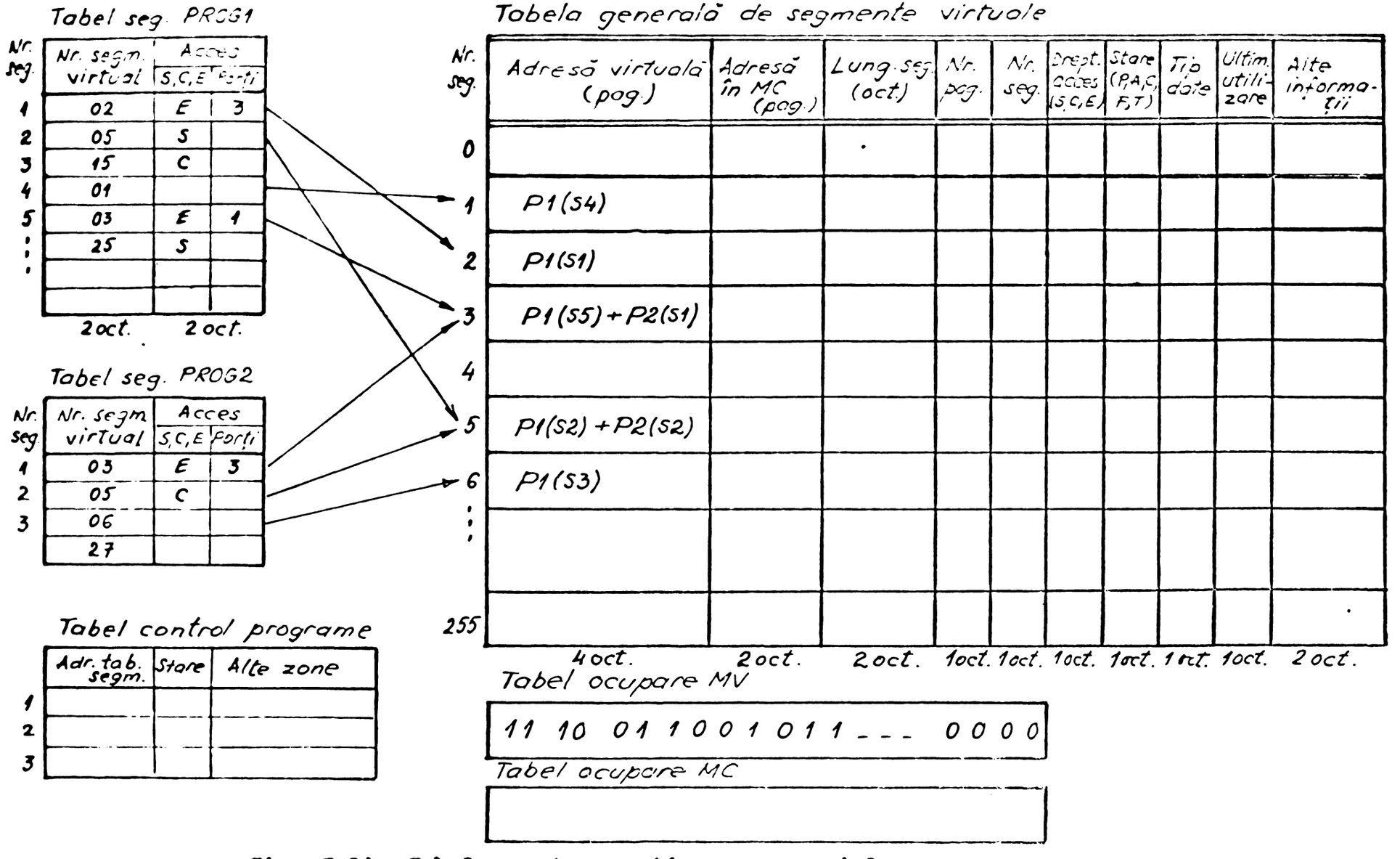


Fig. 3.14. Tabele pentru gestiunea segmentelor.

In fig. 3.14 se prezintă structura propusă pentru tabelele de segmente (TS) ale programelor, tabela generală de segmente (TGS) şi comunicarea între ele. In TGS se găseşte pentru fiecare segment virtual : adresa virtuală de început, adresa de implantare în MC (în nr. de pagini), lungimea (în octeţi), drepturi de acces (S = scriere, C = citire, A = execuţie), stare de segment (P = 1 prezent în MC, A = 1 activ, C = 1 reentrant, F = 1 fix, T = 1 transfer I/E în curs), tipul informaţiilor conţinute (program normal, reentrant partajabil, rezident sistem, nerezident sistem, date proprii, date comune, de interfaţă SGF), număr program, nr. segment în program, (pentru segmente proprii unui program), momentul ultimei utilizări. Cu această structură TGS ocupă 256 seg x 16 oct = 4 K. Ordinea segmentelor în MV e oarecare, ea nu respectă ordinea din TGS. Segmentele noi se plasează dinamic în zonele libere cu dimensiune suficientă de alocatorul MV. Unele segmente pot fi proprii unui program, iar altele pot fi partajabile (de date sau proceduri). Acestea din urmă sînt referite cu acelaşi număr din mai multe tabele de segmente program. Drepturile diferitelor programe pot fi diferite asupra aceluiaşi segment. Unele pot scrie într-un segment de date, altele pot numai citi. La segmentele de procedură partajabile se pot da porţi de apel. Plasînd adresa punctelor de intrare la începutul procedurii, ea poate fi apelată prin una sau mai multe porţi. Pentru 3 porţi specificate se permite accesul prin primele 3 puncte de intrare.

Cu această organizare TS a unui program este un descriptiv al procesului, care poate fi completat cu parametrii de acces, dacă există implementate prin hardware însă de protecţie. Se pot folosi astfel de mai mulţi utilizatori, în condiţii de securitate deplină, aceleaşi segmente de date sau procedură. Se reduce astfel swapingul în multiacces, creşte viteza de răspuns a sistemului, creşte eficienţa încărcării memoriei centrale şi virtuale, creşte numărul utilizatorilor ce pot fi deserviţi simultan.

Tabelele se gestionează dinamic la încărcarea unor noi programe în MV, la fiecare activare de program în UC, la saltul dintr-un segment în altul. Orice salt într-un segment se face prin macroinstrucţia CALL dacă e segment de procedură. Monitorul va verifica în acest caz dreptul de acces şi poarta de acces, va citi valoarea unui contor de timp (minut, secundă)

şi îl va memora în TSG la segmentul care a făcut apelul, ca moment al ultimei utilizări. Acesta va servi ca date algoritmului LRU de înlocuire a segmentelor în MC. Se îmbunătăţeşte astfel mult algoritmul LRU cu utilizarea bitului de referinţă care în medie este 50 % înlocuire aleatoare. Scade astfel mult traficul de pagini în multiacces, reducîndu-se pericolul de " trashing ".

La segmentele de date metoda nu e utilizabilă şi se menţine utilizarea numai a bitului de referinţă R, care trebuie poziţionat pe 1 în memoria rapidă la orice adresare în segment. Tot aici se poate adăuga un bit C de modificare, ce se poziţionează pe 1 la încărcarea segmentului în MC. Dacă se face o operaţie de memorare în segment, se forţează C = 0. La înlocuirea segmentului dacă C = 1 nu e necesară salvarea lui, fiind nemodificat de la ultima încărcare. Poziţionarea bitului C şi îl necesită circuite suplimentare. La înlocuire nu se vor salva niciodată segmentele reentrante sau explorare în citire, care au în TSG bitul C = 1.

Pentru o protecţie eficientă şi diferenţiată a informaţiei în sistem, se pot implementa inele de protecţie prin adăugarea unor biţi care să indice inelul curent în registrul de stare program şi inelul segmentului în memoria rapidă. Poziţionarea acestor biţi trebuie făcută în monitor. La fel se pot adăuga biţi de protecţie la scriere, citire, şi execuţie (R, C, E). Posibilitatea efectuării acestor controale în timpul execuţiei, prin tabela de translaţie a segmentelor, permite implementarea unor tehnici software moderne cum sînt: capabilităţi, inele de protecţie, domenii de protecţie. Aceste facilităţi nu pot fi implementate pe sistemele cu virtualizare prin paginaţie.

Cea mai grea problemă de rezolvat la sistemele cu memorie virtuală şi în special cînd se lucrează în multiacces, este cea a operaţiilor de intrare/ieşire. Ele se efectuează simaron prin canale, utilizînd programe de canal. Adresele zonelor tampon din programele de canal nu sînt translatabile. Operaţiile de I/E lansate nu pot fi întrerupte. Din acest motiv translatarea acestor adrese se face prin software la majoritatea sistemelor.

Pentru rezolvarea acestei probleme propunem următoarele două soluţii :

A. Blocurile de comandă, programele de canal, modulele de legătură SGF şi zonele tampon aferente vor fi segmente separate. Aceste segmente se încarcă dinamic în MC prima dată cu translaţia adreselor prin program. La următoarele încărcări se vor plasa în acelaşi loc în MC. Dacă spaţiul este ocupat de un segment de procedură sau date, acesta va fi realocat dinamic în MC, lăsînd locul liber. In momentul lansării unei operaţii de I/E, se va poziţiona starea segmentului T = 1 (transfer în curs), care va interzice înlocuirea segmentului pînă cînd revine T = 0.

B. Se încarcă segmentul de tip SGF în MC dinamic, utilizînd pentru protecţie în timpul transferului T = 1. Se păstrează în MV la începutul segmentului tabela de adrese absolute. După încărcarea în MC se translatează prin program aceste adrese. Metoda este mult mai flexibilă, nu introduce întîrzieri şi e aplicabilă pe segmente de program mixte (cu module SGF), cu condiţia ca zona tampon să fie în acelaşi segment. Tabela de adrese absolute se găseşte în programul IMP din bibliotecă.

Deoarece asupra transferurilor de date de I/E făcute prin programul de canal nu se face nici o verificare în timpul execuţiei, este necesar un control riguros în avans al adreselor folosite (să fie în program) şi interzicerea modificării lor dinamice. Dacă elementele SGF sînt într-un segment, se poate folosi acelaşi mecanism hardware cu memorie rapidă pentru translatarea adreselor ca la UC, care asigură şi protecţia informaţiei. La sistemele cu paginaţie acest lucru nu e posibil.

Folosind acest sistem de organizare şi control al informaţiilor se asigură posibilităţi largi de cooperare între procese (şi utilizatori), prin utilizarea în comun a unor segmente. Se înlocuiesc vechile metode de comunicare între partiţii prin zonă intermediară comună interpartiţii.

Din cele de mai sus rezultă cît de legate sînt performanţele unui subsistem cu multiacces de existenţa memoriei virtuale şi de rezolvarea virtualizării prin software într-o manieră flexibilă, cu posibilitatea de protecţie la nivel de segment şi cu posibilitatea partajării segmentelor. Se pot activa astfel pentru fiecare utilizator procese protejate individual, care să lanseze în execuţie însăşi compilatoarele, editorul de legături sau programe utilitare, care pot fi partajate la utilizare.

4. SUBSISTEM CONVERSAȚIONAL DE TELETRANSMISIE CU MULTIACCES (SCOT)

4.1. PRINCIPII DE REALIZARE A SUBSISTEMULUI

Execuția interactivă în multiacces a programelor, scrise în limbaje de programare de nivel înalt și traduse prin compilatoare, este o problemă nesoluționată în sistemele de operare cu multiprogramare. Subsistemele cu multiacces existente (vezi 1.2) rezolvă această problemă numai folosind limbaje conversaționale specializate bazate pe interpretoare, sau lansează execuția programelor în partiții cu prelucrare în loturi și recuperează ulterior rezultatele prin symbiont. Rezolvarea acestei probleme s-a făcut destul de complet în subsistemul conversațional cu multiacces propus de autor și implementat pe calculatoarele FELIX C/256/512/1024 sub sistemul de operare ARIES-3 V15 și V16. In acest capitol se vor prezenta principiile ce au stat la baza subsistemului și modul lui de organizare și funcționare.

Folosirea limbajelor conversaționale cu interpretor mărește inadmisibil timpul de execuție a programelor lungi și cu număr mare de cicluri (vezi 1.2.1), față de timpul lor de execuție dacă s-ar folosi compilatoare la traducerea programului sursă. Editarea conversațională a programelor sursă, compilarea și execuția lor într-o partiție serială (vezi 1.2.3), lipsește pe utilizator de posibilitatea de a urmări interactiv execuția programelor și de a putea interveni pentru a schimba cursul execuției, pe baza rezultatelor intermediare.

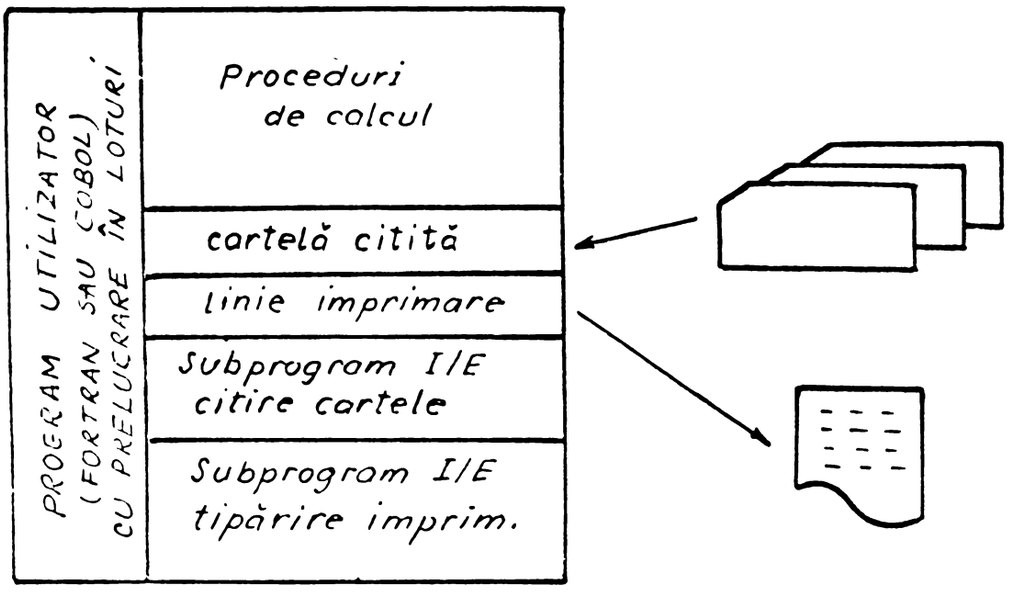
Subsistemul SCOT execută interactiv în multiacces, programe din biblioteci IMT (imagini memorie translatabilă), rezultate din programe sursă scrise în FORTRAN sau COBOL, compilate și cu legăturile editate în partiții serie. Pregătirea programelor IMT se poate face în prelucrarea în loturi sau folosind facilitățile subsistemului ARIES în regim semi-conversațional (vezi 1.2.3.).

Programele în format IMT conțin subprograme speciale, care asigură operațiile de transfer între UC și periferice. Aceste subprograme sînt atașate programului în ultima fază a editării

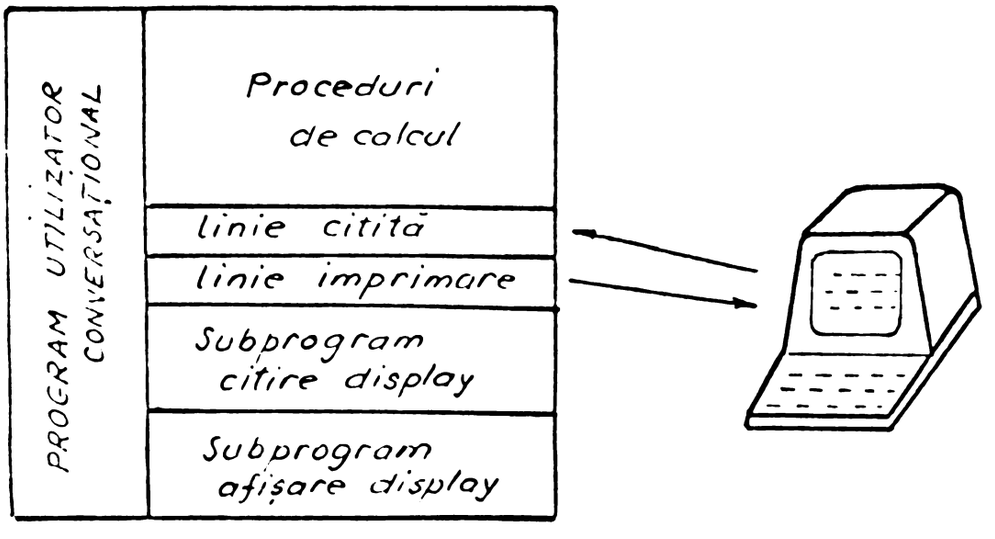
legăturilor. Programul comunică subprogramului de I/E adresa şi lungimea mesajului transmis (fig. 4.1.a.). Lăsînd nemodificat programul şi schimbînd modulele de I/E se obţin funcţii de transfer pentru diferite tipuri de periferice avînd caracteristici diferite. Posibilitatea schimbării dinamice a modulelor de I/E este prevăzută în sistemul de operare [illegible]S şi se realizează cu macroinstrucţii speciale [Hel 81] .

Schimbînd modulele de I/E pentru fişierul sistem de intrare (# 1) şi pentru fişierul sistem de ieşire (# 2), cu module de recepţie şi respectiv de afişare pe un dispozitiv de afişare pe tub catodic (display), orice citire de cartelă va avea ca efect introducerea unui rînd de la terminal, iar orice tipărire la imprimantă va afişa un rînd pe ecranul terminalului. Am transformat astfel programul scris pentru prelucrarea în loturi într-un program conversaţional. Fiecare compilator are de regulă propriile lui module pentru citire şi scriere pe fişierele # 1 şi # 2. Sistemele de operare cu limbaj de comandă conversaţional pentru mini şi microcalculatoare au posibilitatea de a comuta aceste fişiere la consola centrală. Un asemenea sistem este conversaţional cu un singur utilizator ("mono-user"), permiţînd introducerea programului şi datelor de la terminal şi afişarea resultatelor pe acelaşi terminal (în mod normal numai comenzile se dau dela terminal şi se afişează mesajele sistem). Într-un sistem minicalculator cu multiprogramare, fiecare partiţie are un terminal consolă centrală de la care se poate lucra conversaţional. Un astfel de sistem de operare deserveşte mai mulţi utilizatori, fiecare lucrînd într-o anumită partiţie("multi-user").

La sistemele de calcul de capacitate medie,lucrînd în multiprogramare, nu este admis ca o partiţie să fie utilizată conversaţional de un singur utilizator. Numărul partiţiilor fiind redus (12 - 14), fiecărei partiţii îi revine un spaţiu mare de memorie şi un număr de periferice exploatate neraţional. Viteza de introducere la terminal nu poate depăşi în cel mai bun caz 1-2 caractere pe secundă, iar viteza de extragere pe terminal atinge 30 - 120 car/secundă. Ultima limitare e impusă de caracteristicile liniei de teletransmisie, a modemurilor şi terminalelor folosite şi de necesitatea citirii resultatelor afişate sau de tipărirea lor prin recopierea ecranului.



a). [illegible]



b). [illegible]

Fig. 4.1. Prelucrare în loturi și conversațională.

Aceste viteze sînt total nesatisfăcătoare dacă se ţine cont că în sistemele cu multiprogramare cititoarele de cartele (1600 car/sec) şi imprimantele rapide (2000 car/sec) sînt cele mai lente din sistem şi limitează viteza de lucru. Se utilizează transferul anticipat al cartelelor pe suporturi magnetice şi ieşirea listelor pe fişiere, cu listare ulterioară (SPOOL). Aceste operaţii sînt controlate automat prin programul SYMBIONT, care lucrează într-o partiţie paralelă cu mai multe cititoare de cartele şi imprimante.

Utilizarea terminalelor de teletransmisie conversaţionale pe sisteme de calcul de capacitate medie, poate fi rentabilă numai prin utilizarea unor subsisteme lucrînd în multiacces într-o partiţie paralelă în regim de multiprogramare complexă (vezi 1.1.6.). Resursele alocate partiţiei vor fi partajate între utilizatori prin subsistem, care asigură alocarea resurselor şi protecţie lor (vezi 1.2.). Încărcarea completă a sistemului se realizează prin lucrul în loturi în alte partiţii.

Subsistemul SCOT asigură execuţia interactivă a mai multor programe, încărcate din biblioteci IBM, care utilizează în comun aceeaşi partiţie. La un moment dat un singur program utilizator este încărcat în memorie, iar celelalte sînt depuse pe un fişier de manevră pe disc prin tehnica "swaping" (fig. 4.2.). Dimensiunea subsistemului (20 K) şi a programelor utilizator (60 - 100 K) nu permit în general încărcarea a două programe simultan în MC. Chiar dacă ar fi o partiţie de dimensiuni corespunzătoare, la încărcarea a două programe în MC nu s-ar mai putea asigura protecţia lor reciprocă, deoarece cheia de protecţie e pe partiţie. Ar apare în plus probleme dificile de relocare dinamică generate de adresele absolute din programele de canal, generate dinamic în program de modulele de acces SGF. Ţinînd cont de spaţiul redus ocupat de subsistemul SCOT, se recomandă în acest caz utilizarea a două partiţii care să lucreze în multiacces, fiecare cu un anumit număr de utilizatori. Se obţine astfel reducerea timpului mediu de acces, fiind două programe în lucru în MC la un moment dat, protejate între ele. Cînd unul efectuează operaţii de swaping, celălalt este activ şi se încarcă mai bine UC.

Interfaţa dintre subsistem şi programele utilizator este realizată printr-un modul special care înlocuieşte modulele de I/E pentru fişierul = 1 şi = 2. Orice operaţie de citire de cartelă

ve cere subsistemului să solicite utilizatorului să introducă o linie de date. Orice tipărire la imprimantă va transmite subsistemului o linie, ce va fi afişată pe terminalul utilizatorului asociat programului activ. Deoarece transferurile de date cu terminalul se fac lent, pe timpul transferului programul activ este pus în aşteptare şi depus în fişierul de salvare pe disc, iar în locul lui este încărcat în MC un alt program care este pregătit şi are prioritate maximă. In timp ce un program lucrează pe UC alte programe fac transferuri de date cu terminalele în paralel. Aceste transferuri sînt posibile chiar dacă programele sînt depuse pe disc, deoarece toate terminalele sînt gestionate în regim de multitasking de către un modul de dialog al subsistemului şi fiecărui program (utilizator) i se rezervă un spaţiu în subsistem egal cu lungimea unei linii de text. La fiecare mesaj primit subsistemul identifică utilizatorul căruia îi aparţine şi îl transmite programului asociat. La fiecare cerere de transfer din partea programului, se identifică programul activ şi utilizatorul (terminalul) care îi corespunde.

Programele în lucru se păstrează complet pe fişierul de manevră, deoarece ele conţin date modificabile, care pot diferi de la o activare la alta. Acelaşi program folosit de mai mulţi utilizatori trebuie păstrat în mai multe copii, fiecare fiind într-o anumită fază de execuţie şi cu anumite date de intrare. Programele generate de compilatoare nu sînt reentrante.

Subsistemul e conceput modular, pentru a putea fi uşor dezvoltat sau modificat şi pentru a fi segmentat. La concepţia sa s-au utilizat la maxim posibilităţile sistemului de operare DOS-3, fără a se opera modificări în monitor. Se asigură astfel compatibilitatea între versiunile sistemului de operare şi proiectele existente în sistem. Programele utilizator pot utiliza fără restricţii fişierele acceptate de DOS, existînd în acest sens facilităţi superioare altor subsisteme (CP/M-80). Subsistemul DOOT cuprinde următoarele module orientate pe funcţiile realizate :

- modulul de iniţializare,
- modulul de dialog cu utilizatorii în multitasking,
- modulul de încărcare a programelor din biblioteca IMT,

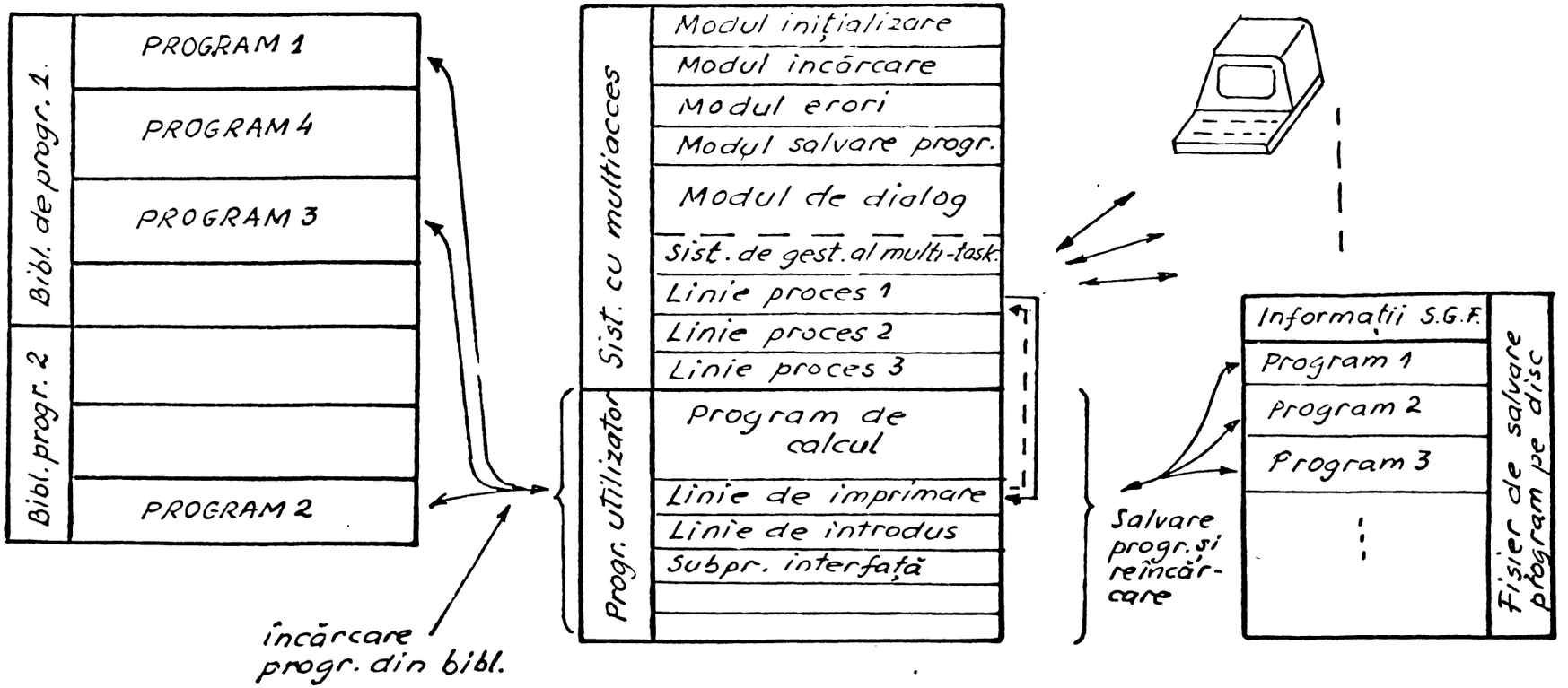


Fig. 4.2. Structura și funcționarea subsistemului SCOT.

- modulul de salvare şi reîncărcare programe (swaping),
- modulul de tratare a erorilor şi devierilor.

Iniţializarea subsistemului se face de la consola centrală, indicîndu-se fişierul de manevră folosit şi adresa terminalelor activate. La încărcarea în partiţie subsistemul iniţializează multitaskingul şi consideră programul utilizator activ ca un segment al său, declarat fictiv în programul sursă. Utilizatorul poate începe dialogul apăsînd tasta RETURN la terminal. Prin dialog foarte explicit şi flexibil, utilizatorul indică programul dorit şi biblioteca din care acesta face parte.

Pe baza acestor date modulul de selectare încarcă rădăcina programului solicitat, ca un segment independent din biblioteca indicată. El se plasează în partiţie după segmentele subsistemului şi se lansează imediat în execuţie. Citirile de cartele şi scrierile la imprimantă sînt redirectate, cum s-a arătat prin modulul de interfaţă spre terminalul care l-a solicitat. Modulul de interfaţă dintre programul utilizator şi subsistem se adaugă la editarea legăturilor (specificat printr-o cartelă TASKHd).

Dacă între timp un alt utilizator solicită subsistemul, el va fi deservit similar. Programul anterior lansat se salvează pe disc şi în MC se încarcă noul program din bibliotecă. Cînd mai multe programe sînt în execuţie, ele vor fi deservite pe rînd cînd sînt pregătite. Punctele de discontinuitate în program sînt transferurile la terminal. Dacă există un alt program pregătit în acestă perioadă, el va fi activat prin "swaping". La reactivare programul va continua din punctul în care a fost întrerupt. Starea programului întrerupt este memorată într-o zonă proprie fiecărui task în zona subsistem. Se asigură astfel o divizare naturală a puterii de calcul între utilizatori indiferent de numărul lor.

Dacă un program lucrează cu fişiere, în zona rezervată pe disc se va salva şi zona de comunicare SGF corespunzătoare (ZONA). Acestea va fi încărcată în partiţie numai la deschiderea fişierelor, fiind necesare informaţiile de ASSIGN şi LABEL, care se furnizează dinamic.

La realizarea subsistemului s-a avut în vedere păstrarea controlului în cazul apariţiei oricăror tipuri de erori (erori de programare, de BBZ, de teletransmisie, în cartelele de date, etc). Ele se tratează unitar, utilizatorul este avertizat asupra incidentului produs şi se permite în anumite cazuri corectarea erorii, fără a influenţa lucrul celorlalţi utilizatori.

Un program poate fi întrerupt pe parcurs prin BRK sau la fiecare umplere de ecran. El poate fi reluat cu alte date, sau poate fi continuat pe altă ramură dacă s-au prevăzut în program parametrii de ramificaţie. La terminarea lucrului terminalul poate fi deconectat de la reţea şi reluarea se poate face la reconectare apăsând tasta BRK. Nu este necesar ca la lansarea subsistemului terminalul să fie conectat.

Folosind direct sau indirect funcţiile existente în sistemul de operare, întregul subsistem, nesegmentat, cu facilităţi de utilizare a fişierelor a programelor segmentate, nu depăşeşte pentru 16 utilizatori 20 K. Pentru fiecare utilizator suplimentar se adaugă cca. 800 octeţi necesari pentru tabele de descriere a liniei şi zona proprie utilizatorului.

4.2. Modulul de dialog cu utilizatorii

Modulul de dialog constituie rădăcina procedurii reentrante utilizate pentru comunicaţie în regim de multiacces cu terminalele gestionate de subsistem. Se foloseşte în acest scop funcţiile sistemului de gestiune a teletransmisiilor, care asigură un multitasking restrâns. Fiecărei linii de teletransmisie, declarată printr-o directivă LCB (Line control block), i se asociază un task. Un LCB este zona de control şi salvare a task-ului. Fiecare linie are conectat la capătul ei, local sau la distanţă un terminal conversaţional la care poate lucra un utilizator.

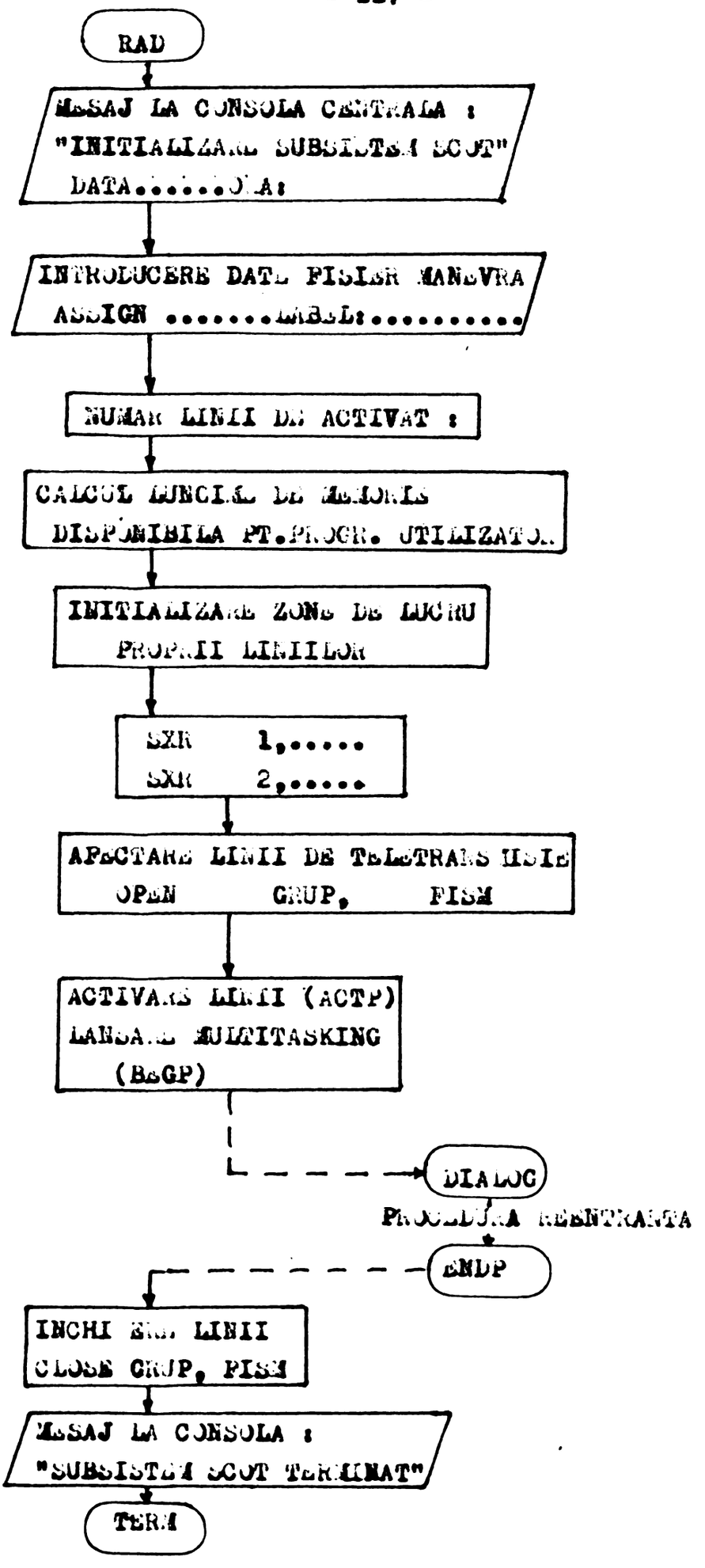


Fig. 4.3. Iniţializarea subsistemului.

Numărul de linii de teletransmisie ce pot fi utilizate este dat la generarea subsistemului. Fiecărei linii îi este rezervată o zonă de date proprie, exploatată printr-o secţiune fictivă, care îi descrie structura (fig.4.4.). Deoarece la activarea unui task, SGF-ul dă în R13 adresa LCB-ului corespunzător, aceste s-a plasat la începutul zonei proprii. In acest fel utilizînd R13 ca bază a secţiunii fictive, se încarcă automat baza la activarea task-ului.

Rădăcina subsistemului cuprinde (fig.4.3) la început un dialog de iniţializare de la consola centrală, prin care se dau numărul terminalelor ce se activează,se cer datele fişierului de manevră utilizat pentru salvarea programelor. Se iniţializează apoi zonele de lucru, se activează subprogramele de tratare a devierilor prin macroinstrucţii SXR, se afectează liniile de teletransmisie partiţiei, se activează liniile şi se lansează multitasking-ul (BEGP). Procedura de multitasking utilizată de toate liniile este DIALOG. Prin ea se apelează celelalte module ale subsistemului şi se asigură conversaţia cu utilizatorii.

Inceperea dialogului cu subsistemul o face utilizatorul la apăsarea tastei BRK a terminalului. Pe display apare mesajul (fig.4.5) care indică începutul sesiunii de lucru, data, ora, şi lungimea programelor ce pot fi utilizate. Dacă nu se cunoaşte modul de lucru se prezintă pe display instrucţiuni de utilizare. Se cere apoi numele şi contul utilizatorului,datele bibliotecii şi programelor utilizate. Datele furnizate se verifică şi se depun în zona proprie utilizatorului, unde se pot memora în avans numele a 5 programe din aceeaşi bibliotecă.

Se pregăteşte încărcarea primului program transmiţînd datele de identificare modulului de încărcare (SELECT) şi se anunţă utilizatorul la terminal că programul s-a lansat. Inainte de a trece la încărcarea programului se verifică dacă în MC există deja un program încărcat. Dacă există se va salva pe disc cu subprogramul DEPUN. Incărcarea programului din biblioteca IMT specificată se face prin subprogramul SELECT,care se va prezenta în subcapitolul următor. Programul încărcat va fi lansat în execuţie de la adresa memorată de editorul de legături în bibliotecă.

Programul lansat va reda comanda subsistemului în următoarele cazuri :

- dacă programul cere tipărirea unor date la imprimantă,acestea vor fi transmise subsistemului prin modulul de

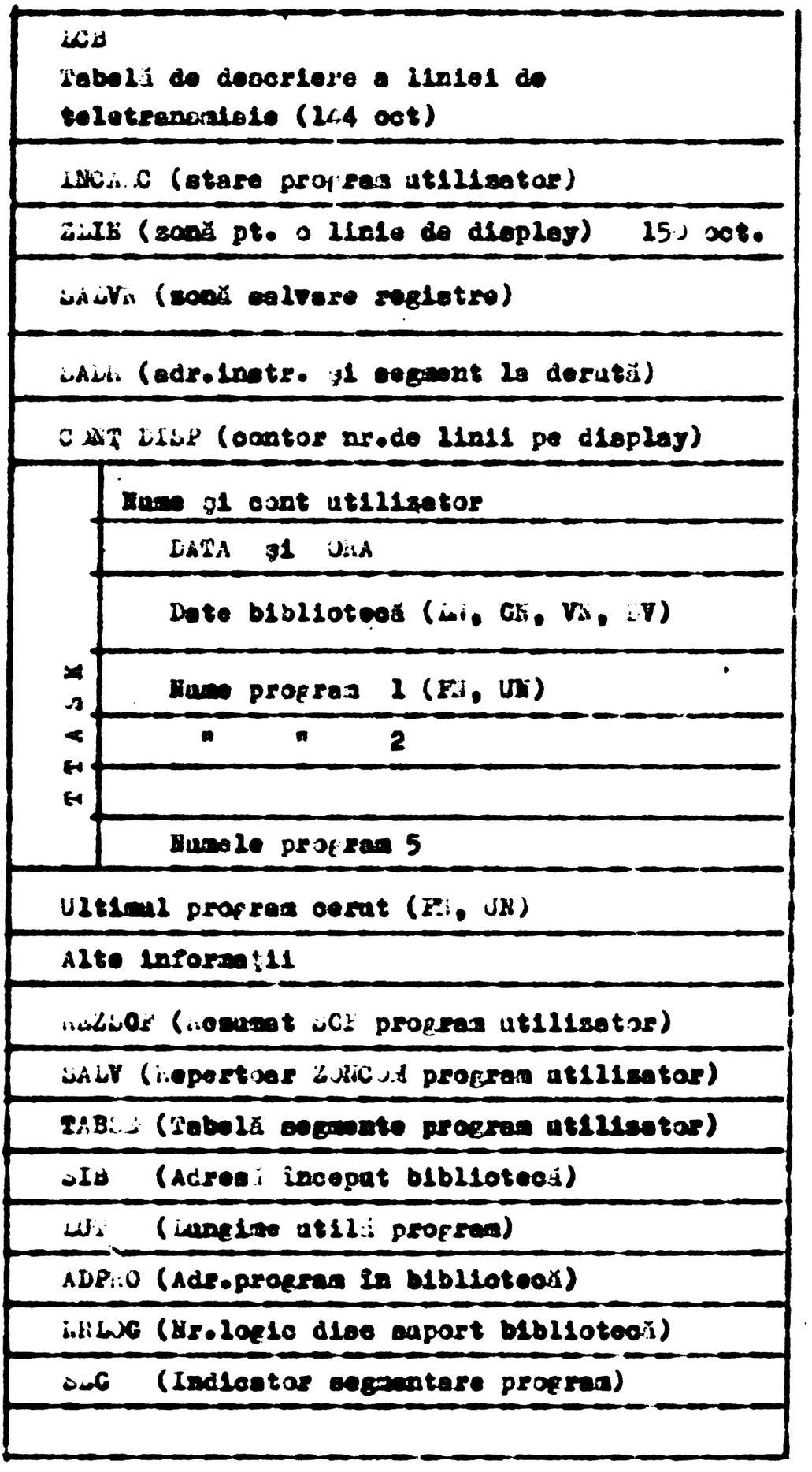


Fig.4.4. Structura zonei proprii unui utilizator.

interfaţă în zona [illegible], de unde vor fi afişate pe display ;

- dacă programul cere citirea unor date de pe cartele, modulul de interfaţă va anunţa subsistemul să solicite utilizatorului un rînd (şir de caractere), care va fi memorat în zona proprie ;

- dacă programul are erori care produc devieri ele vor fi transmise subsistemului prin subprogramele speciale şi vor fi afişate la terminal;

- dacă programul s-a terminat normal, se afişează mesajul de terminare şi se lansează programul următor.

La operaţiile de I/E pe terminal se numără rîndurile şi la umplerea ecranului se cere confirmarea continuării (apăsare tastă C sau NL). Se permite astfel utilizatorului să analizeze rezultatele sau să le copieze. Dacă rezultatele intermediare nu corespund,se poate opri execuţia apăsînd în acest moment tasta E.

Modulele de interfaţă din programul utilizator, introduse la editarea legăturilor, comunică cu subsistemul prin două adrese fixe din tabela de segmente, care conţin adresele de revenire. Pentru aceasta s-au declarat două segmente fictive în subsistem. Pe perioada dialogului un alt program e activat. Reluarea programului pus în aşteptare se face din punctul unde s-a întrerupt.

Erorile de completare a datelor de intrare prin nerespectarea formatului, nu duc la terminarea programului, se permite reintroducerea rîndului ("cartelei") eronat. Erorile apărute duc la terminarea programului şi reînceperea unui nou dialog de lansare.

La terminarea unui program utilizatorul alege modul de continuare (R,C,S). Poate relansa acelaşi program cu alte date (R), poate continua cu următorul program memorat în şirul de aşteptare (C) sau continuă cu alte programe. Dacă utilizatorul termină sesiunea la terminal se trece la o nouă aşteptare de BRK. Aceasta permite deconectarea terminalului şi reconectarea lui după un anumit timp cînd se reîncepe cu apăsarea tastei BRK.

Pentru a asigura verificarea completă şi diferenţiată a erorilor de teletransmisie, care pot fi foarte variate, toate operaţiile de transmisie spre terminal (SSKD) şi de recepţie de la terminal (RECDIVS) se fac într-un subprogram special numit SSRREC.

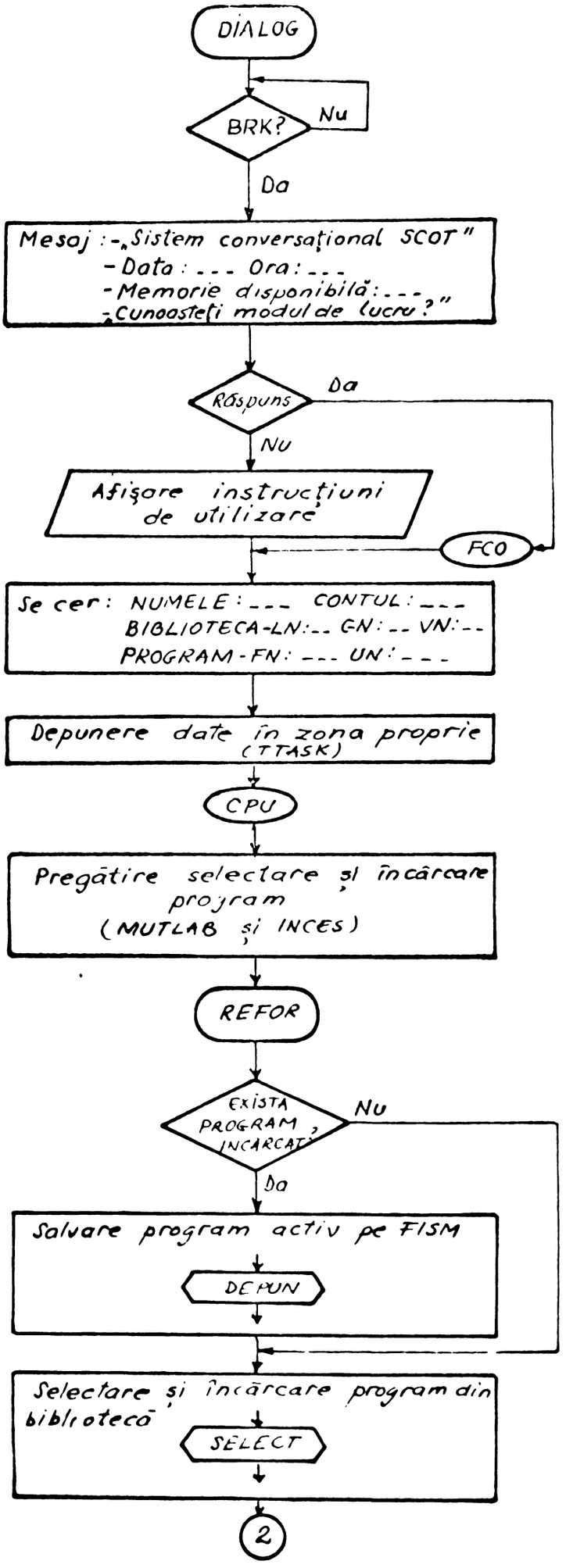
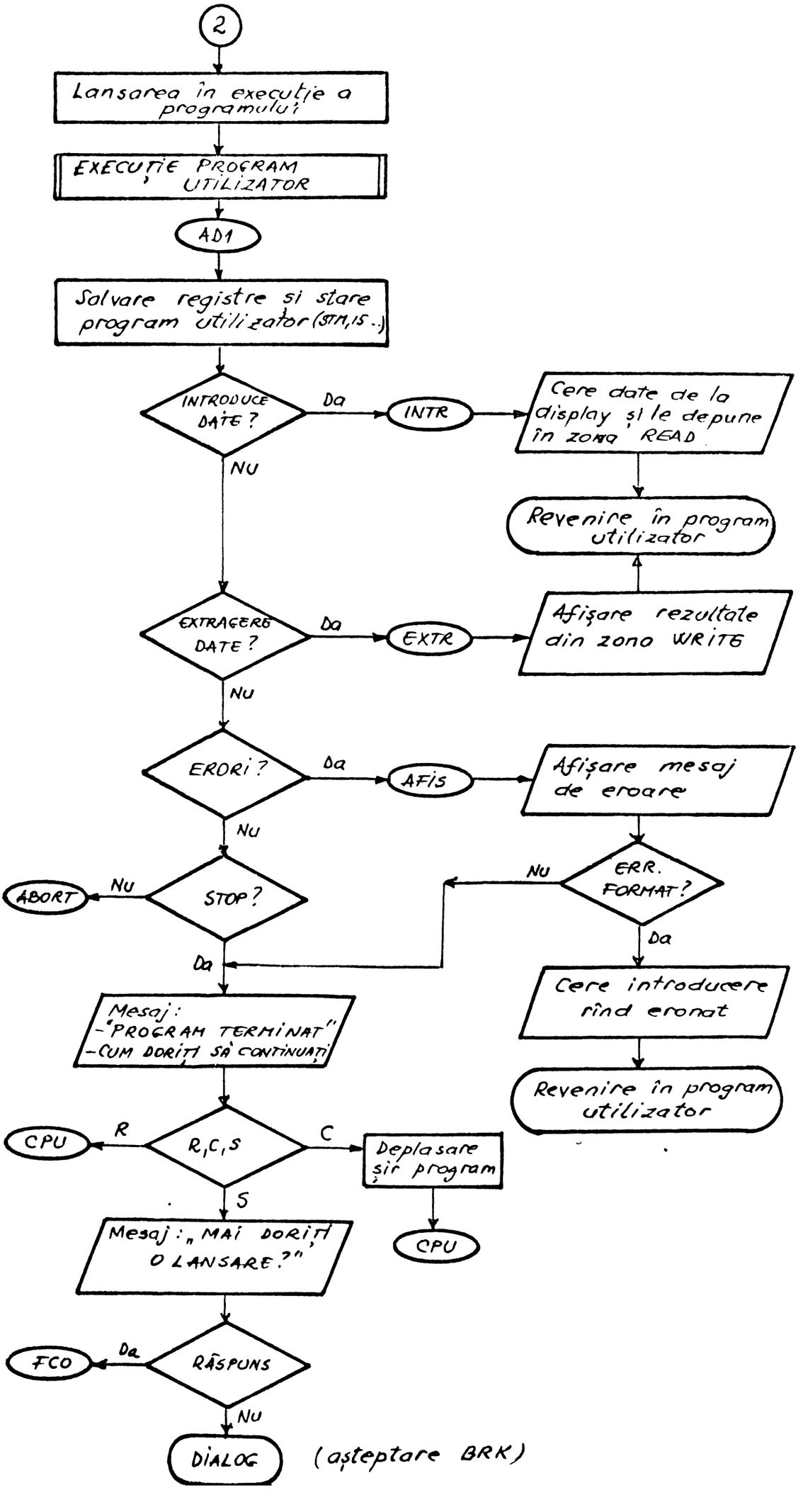


Fig.4.5. Graful root al principiului de lucru al dialo...


2
Lansarea în execuție a programului
EXECUȚIE PROGRAM UTILIZATOR
AD1
Salvare registre și stare program utilizator (STM, IS..)
INTRODUCE DATE ?
Da
INTR
Cere date de la display și le depune în zona READ
NU
Revenire în program utilizator
EXTRAGERE DATE ?
Da
EXTR
Afișare rezultate din zona WRITE
NU
ERORI ?
Da
AFIS
Afișare mesaj de eroare
NU
ERR. FORMAT ?
NU
STOP ?
ABORT
NU
Da
Da
Cere introducere rînd eronat
Revenire în program utilizator
Mesaj:
-"PROGRAM TERMINAT"
-CUM DORIȚI SĂ CONTINUAȚI
R,C,S
R
CPU
C
Deplasare șir program
CPU
S
Mesaj: „MAI DORIȚI O LANSARE?"
RĂSPUNS
Da
FCO
NU
DIALOG
(așteptare BRK)

4.3. MODULUL DE IEŞIRE/INTRARE A PROGRAMELOR

4.3.1. Structura programelor în format I&T.

Subsistemul SCOT este prevăzut cu un încărcător de programe din biblioteci în format I&T, care foloseşte datele de identificare a bibliotecii (LN, CA, VE, DV) şi programului (PR, DE), furnizate de utilizator la terminal. S-a adoptat această soluţie pentru a asigura flexibilitate în exploatare. Alte subsisteme (STRATOS) utilizează programe utilizator (proceduri). Utilizarea macroinstrucţiei EXECUTE cere obligatoriu ca programul lansat să fie în biblioteca utilizator standard a sistemului.

Programele utilizator pot fi catalogate în biblioteci diferite. Catalogarea lor în bibliotecă se face de utilizator după testarea într-o partiţie serie. La editarea legăturilor, printr-o cartelă 'LIB ', se cere înlocuirea subprogramelor de I/E pentru citiri de cartele şi tipăriri la imprimantă cu module de interfaţă cu subsistemul SCOT. Utilizatorul nu trebuie să modifice nimic în program pentru a putea fi exploatat conversaţional, dar trebuie să pună înainte de LIBR toate cartelele de comandă necesare pentru fişiere.

Bibliotecile sînt zone partajate pe disc, iar programele sînt fişiere membre ale zonei pentru care alocarea spaţiului se face dinamic prin zone elementare înlănţuite. Se asigură astfel refolosirea spaţiului eliberat de ştergerea unor programe. În fig. 4.6 se prezintă structura unui program într-o bibliotecă în format I&T, considerînd blocurile plasate succesiv. Fiecare bloc are 1024 octeţi. Primul bloc conţine informaţiile de identificare a programului (nume, număr de punere la zi, date de creare, adresa de lansare, lungime) repertoarul zonei de comunicaţie SGF (ZONCOM) şi tabela de segmente (fig. 4.7).

Un bloc ZONCOM conţine informaţiile de ASSIGN, LABEL şi FILE, date prin cartele de comandă înainte de editarea legăturilor pentru fişierele exploatate prin program. Pentru fiecare tabelă de descriere a fişierelor (TDF) din program există un grup de aceeaşi informaţii numit FILCOM care ocupă 256 oct. Dacă se utilizează mai multe fişiere, există mai multe blocuri ZONCOM fiecare conţinînd 4 FILCOM-uri (fig.4.8).

Segmentele de program sînt formate din mai multe blocuri de text, care reprezintă programul propriu-zis. În faţa fiecărui segment se găseşte un bloc de control al segmentului a cărui

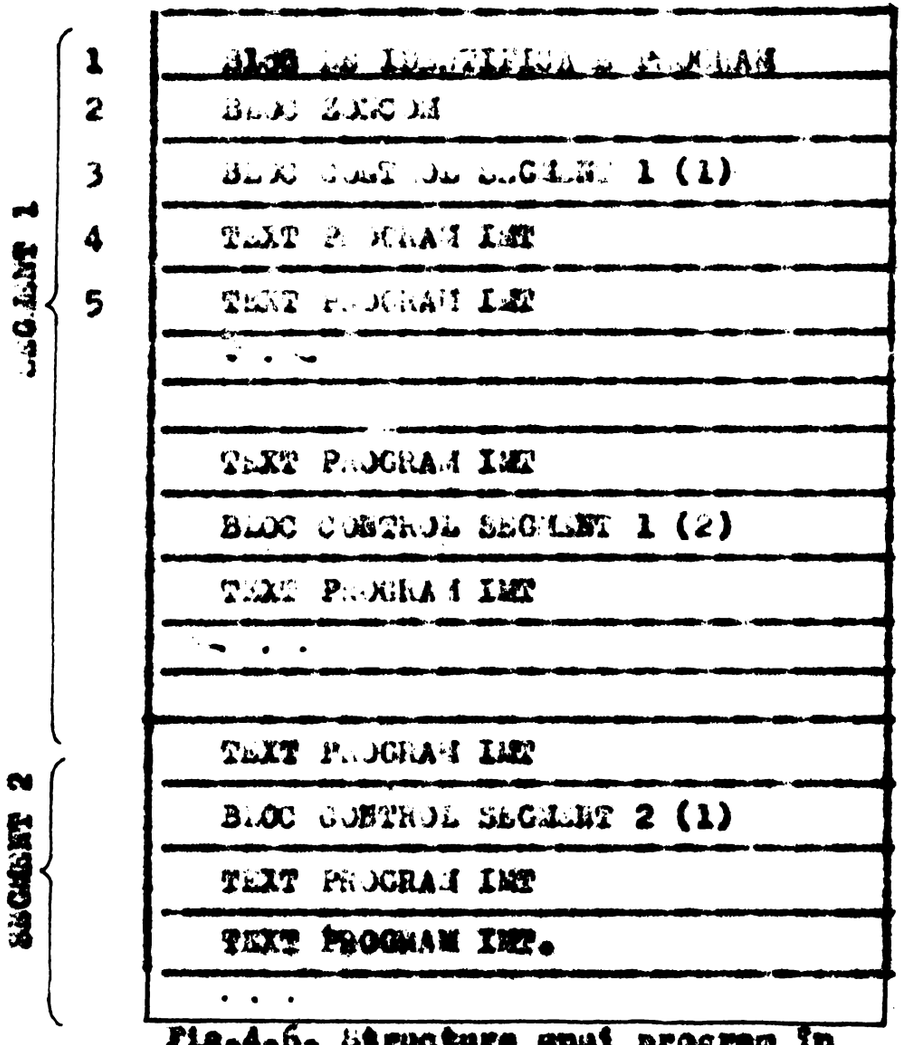


Fig.4.6. Structura unui program în format IMT în bibliotecă.

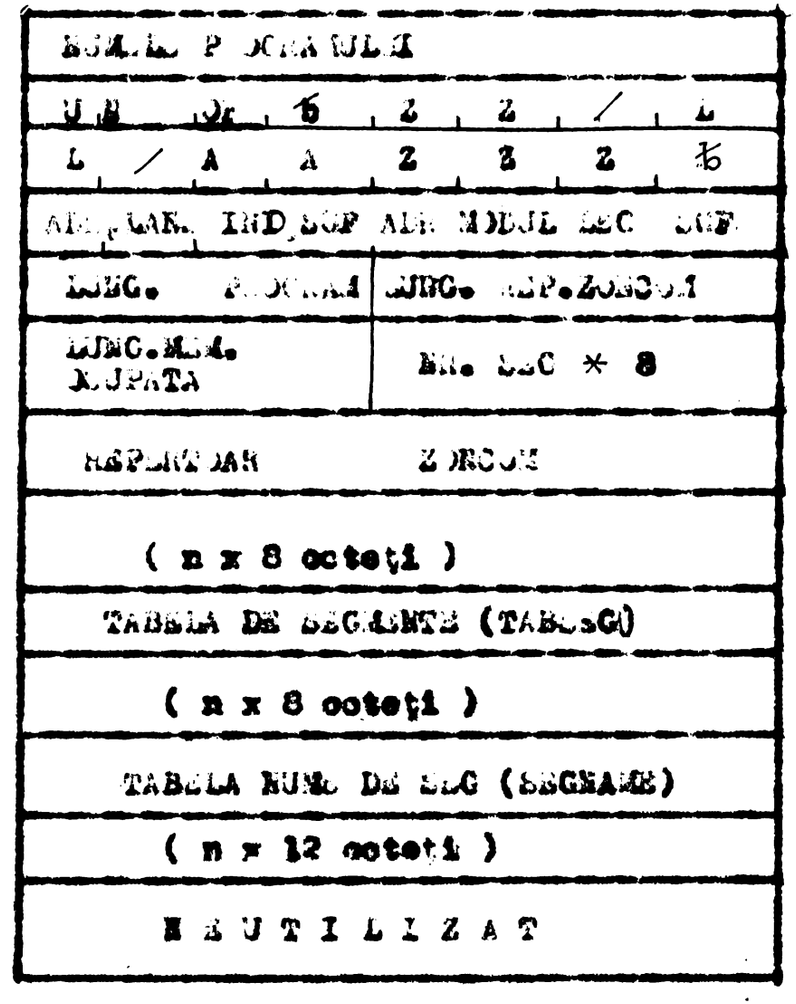


Fig.4.7. Structura blocului de identificare program.

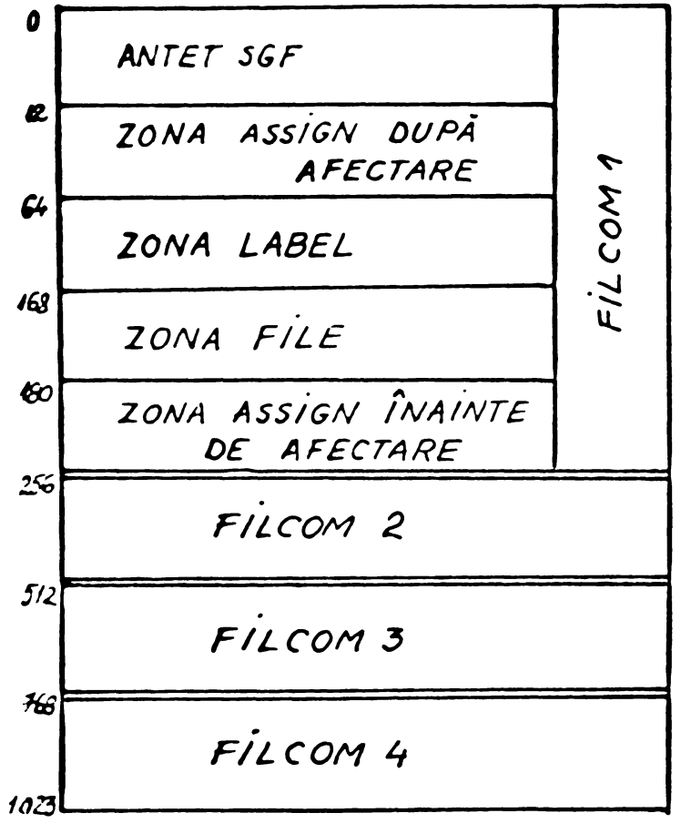


Fig.4.8. Structura unui bloc ZONCOM

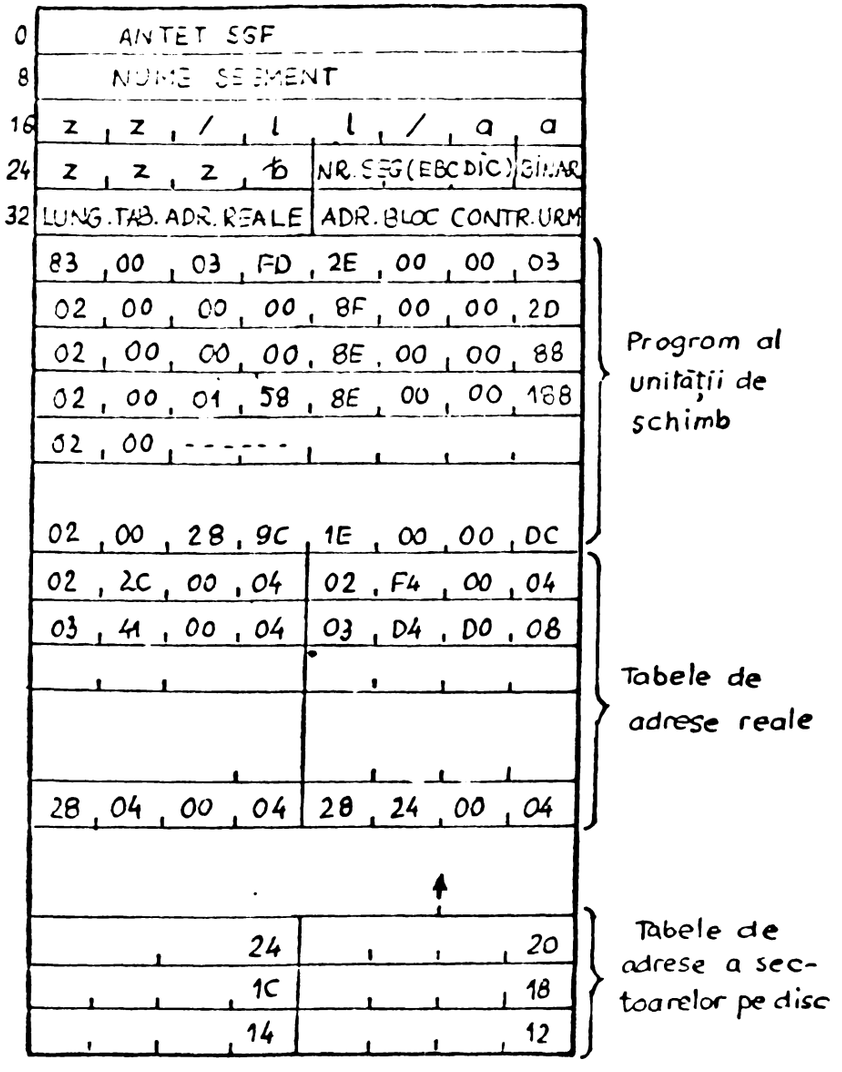


Fig.4.9. Structura unui bloc de control segment.

structură este dată în fig.4.9. El cuprinde în principal un program al unității de schimb (PUS) pentru încărcarea blocurilor de text următoare, o tabelă a adreselor relocatabile (absolute) și o tabelă a adreselor sectoarelor pe care se vor face poziționări în PUS.

Pentru o încărcare rapidă a programului la editarea legăturilor se crează un PUS pentru grupuri de blocuri de text. Acesta conține adrese de sector relative la începutul bibliotecii și adrese de memorie centrală relative la adresa de început a programului. Din acest motiv la încărcarea programului în MC se citește acest bloc, se corectează adresele din PUS și se poate lansa acesta prin EXUP. Blocurile unui segment nu sînt pe o zonă de disc continuă ci sînt plasate aleator și înlănțuite între ele. Adresele acestor zone se dau în tabela de sectoare în ordinea plasării lor pe disc și nu în ordinea în care apar ele în MC. Blocurile se încarcă în memorie în această ordine, pentru ca timpul de poziționare să fie minim (baleere sectoare). Zonele de program rezervate, care nu conțin informații nu se găsesc memorate în bibliotecă. Pentru ele se rezervă spațiu în MC prin comenzi de înlănțuiri de date în PUS.

Pentru segmentele de dimensiuni mai mari, care conțin multe zone rezervate (comenzi de înlănțuire), cu zonele de text dispersate pe disc, PUS crește ca lungime și este necesar să fie fragmentat și plasat în mai multe blocuri de control segment înlănțuite între ele. Modulele de acces BGF și modulul de legătură sînt plasate ca segmente separate, cu blocuri de control proprii. Blocurile de text program au în față un antet de 32 octeți de identificare, ce nu se încarcă în MC (salt în PUS).

Comenzile utilizate în PUS sînt :

- 83 poziționarea capetelor de citire pe un sector a cărui adresă este în tabela de sectoare ;
- 02 citirea unui bloc de text ;
- 80 salt în PUS la un dublu cuvînt de comandă.

Transferurile specificate în PUS se fac în moduri diferite, specificate prin masca din dublul cuvînt de comandă care are următoarele semnificații :

- 2x - înlănțuire de comenzi, întrerupere pe sfîrșit anormal, lungime incorectă neglijată, oprire pe eroare de transfer ;

- SF - înlănţuire de date şi citire fără transfer în MC, utilizat pentru saltul antetului de bloc de text;
- SS - înlănţuire de date, pentru a plasa date din acelaşi bloc la adrese de memorie diferite, dacă între ele există o zonă rezervată (hole);
- LE - întrerupere pe sfîrşit de transfer, care indică sfîrşitul unui FUS (ultimul dublu cuvînt de comandă).

Fiecare bloc de control conţine în octetul 36 un indicator, care are normal valoarea X '00', şi pe următorii 3 octeţi se găseşte adresa sector a blocului de control următor (relativ la începutul bibliotecii). La ultimul bloc de control indicatorul are valoarea X '80'.

4.3.2. Încărcarea programelor.

Ţinînd cont de structura programelor în format LMC, s-a realizat modulul de selectare şi încărcare a programelor a cărui ordinogramă de principiu este dată în fig.4.11. Încărcarea programului se poate face la orice adresă din MC.

Modulul DIALOG furnizează modulului SELECT informaţiile de identificare a bibliotecii şi programului cerut, care sînt utilizate pentru ASSIGN şi LABEL dinamic, efectuat pentru o tabelă de descriere a unei zone partajate (PSL) şi respectiv o tabelă de descriere a unui fişier membru al unei zone partajate (programul). Se forţează fişierul program închis în TDP + 15 şi în rezumatul fişierelor, iar apoi se deschide în citire cu modul de acces LOCATED. Prin această procedură selectarea fişierului program se face prin controale normale de SCP-monitor, fără a utiliza proceduri nestandard complicate.

După deschidere se comandă citirea unei înregistrări (fişier neblocat) prin macroinstrucţia GET care este blocul de identificare a programului (PROGID). Se memorează din PROGID adresa de lansare a programului, lungimea de memorie ocupată şi se verifică dacă încape în spaţiul din partiţie disponibil. Dacă spaţiul e insuficient se abandonează încărcarea şi se anunţă utilizatorul asupra naturii incidentului. Se memorează apoi repertoarul ZONEI şi tabela de segmente (TABSEG). Se

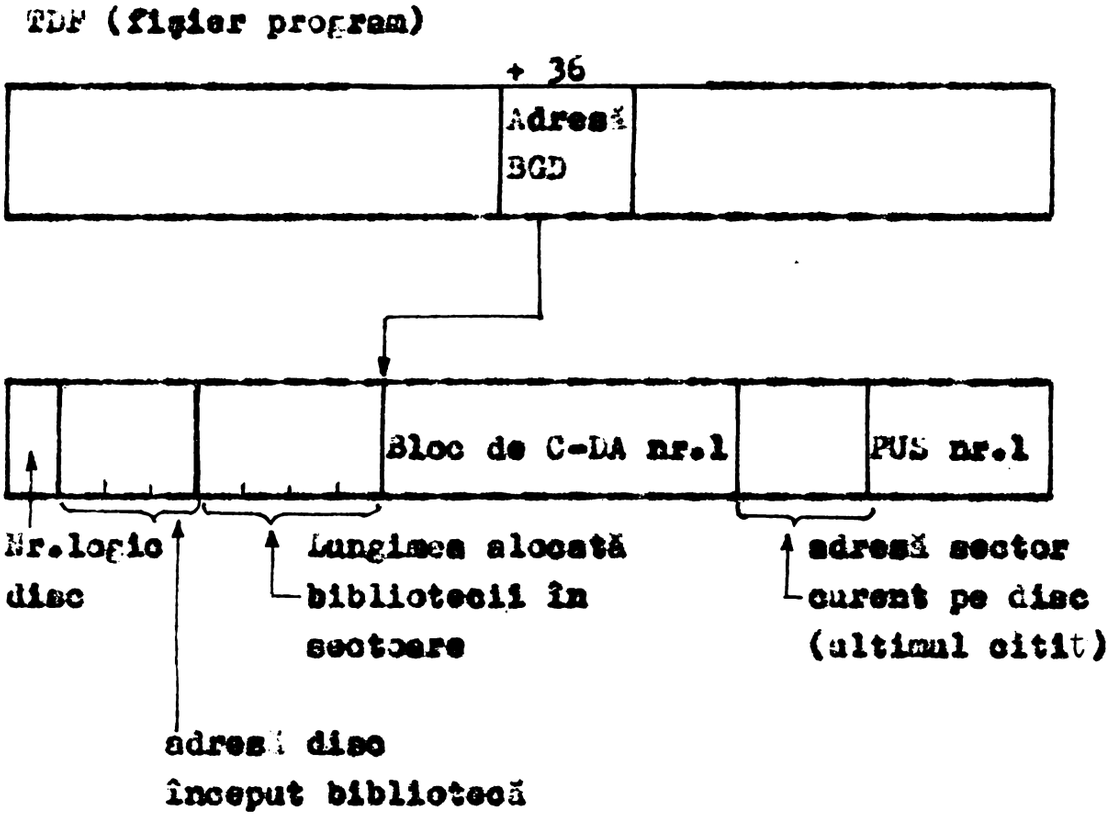


Fig.4.10. Identificarea adresei pe disc a bibliotecii.

trece apoi la citirea următorului bloc, care este blocul ZONCOM (fig. 4.6.). Dacă programul conţine fişiere, altele decît * 1 şi * 2, informaţiile din ZONCOM se memorează în fişierul de manevră, de unde se vor utiliza la execuţia programului (vezi 4.3.3.). Următorul bloc citit va fi blocul de control al segmentului 1.

In continuare nu se vor mai utiliza funcţiile SGF şi se va lucra la nivel de macroinstrucţii EXUP.

Programul unităţii de schimb va fi corectat astfel :

- adreselor referitoare la zonele de MC unde se va plasa textul program li se va adăuga adresa de început a programului.

- adreselor referitoare la tabela de sectoare, din comenzile de poziţionare (83) li se va adăuga adresa de plasare a blocului în MC.

- adreselor de sectoare li se va aduna adresa de început a bibliotecii pe disc.

Adresa unde va fi plasat programul în MC se calculează dinamic. Pentru aceasta s-a definit la sfîrşitul subsistemului, un segment fictiv (PUC), pentru care se rezervă o adresă în tabela de segmente a partiţiei. Adresa de implantare a acestui segment va fi adresa de început a programului. Diferenţa dintre sfîrşitul partiţiei şi această adresă dă dimensiunea maximă a programului admis.

Adresa de început a bibliotecii se va lua dintr-un cuvînt plasat în faţa primului bloc de comandă, asociat fişierului program şi calculat din TDP436 (fig. 4.10). Această adresă este calculată şi depusă de OPEN.

După ce PUS din blocul de control segment a fost corectat, se lansează în execuţie prin EXUP. Se vor încărca astfel toate blocurile (pagini) de text program asociate. Dacă citirea cu înlănţuire de comenzi depăşeşte limita unui cilindru se va semnala eroare de violare protecţie disc. Ea va fi depăşită prin relansarea PUS din locul incidentului, acceptată de supervizor după verificarea noului cilindru.

Pe baza tabelei de adrese relocatabile acestea se corectează prin adunarea adresei de început a programului. Adresele din tabelă sînt date relativ la începutul segmentului, iar pe primul octet conţin numărul segmentului din care adresa face parte înmulţit cu 4. Se pot face astfel referiri de corecţie adrese în alte segmente.

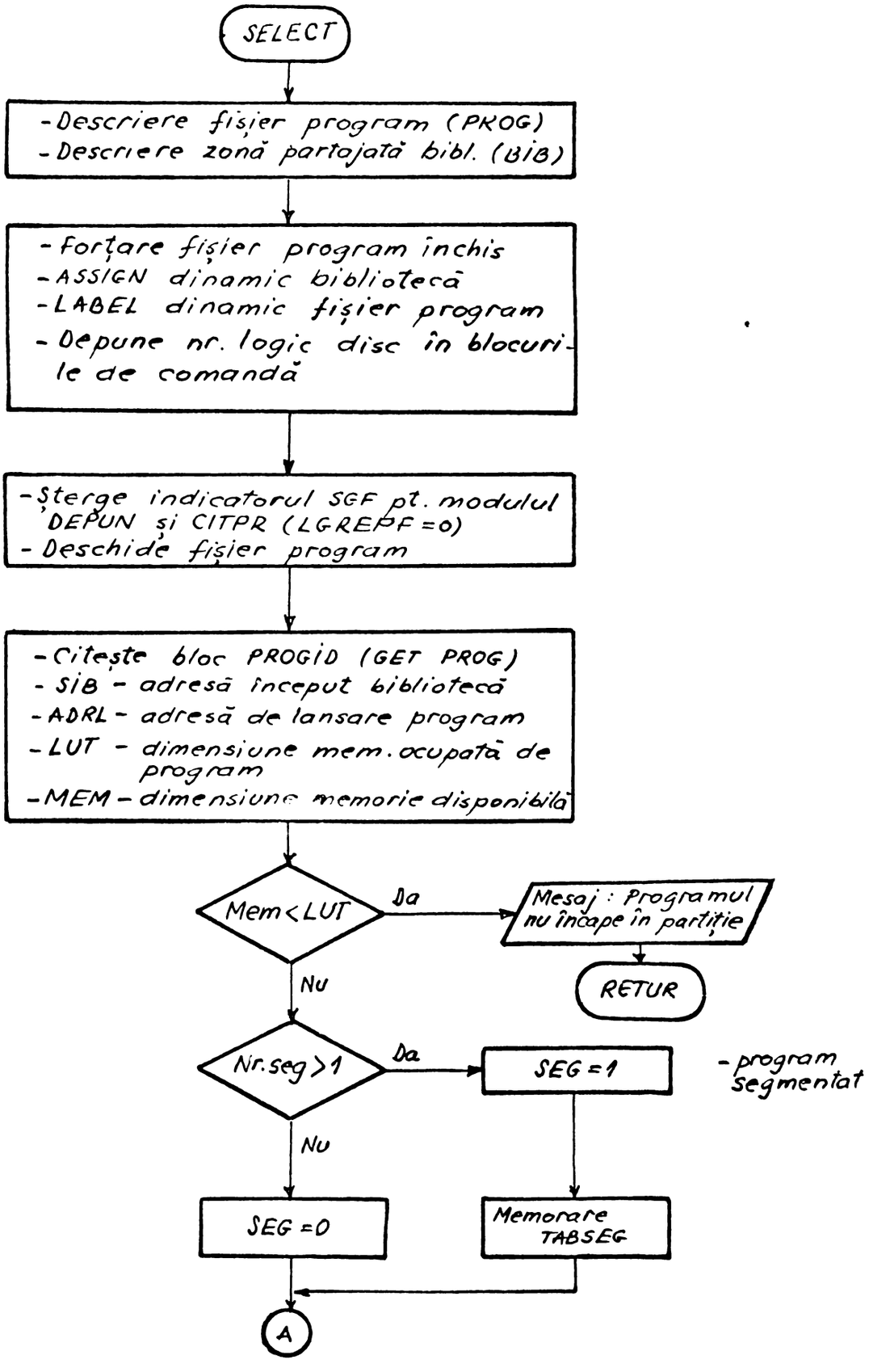
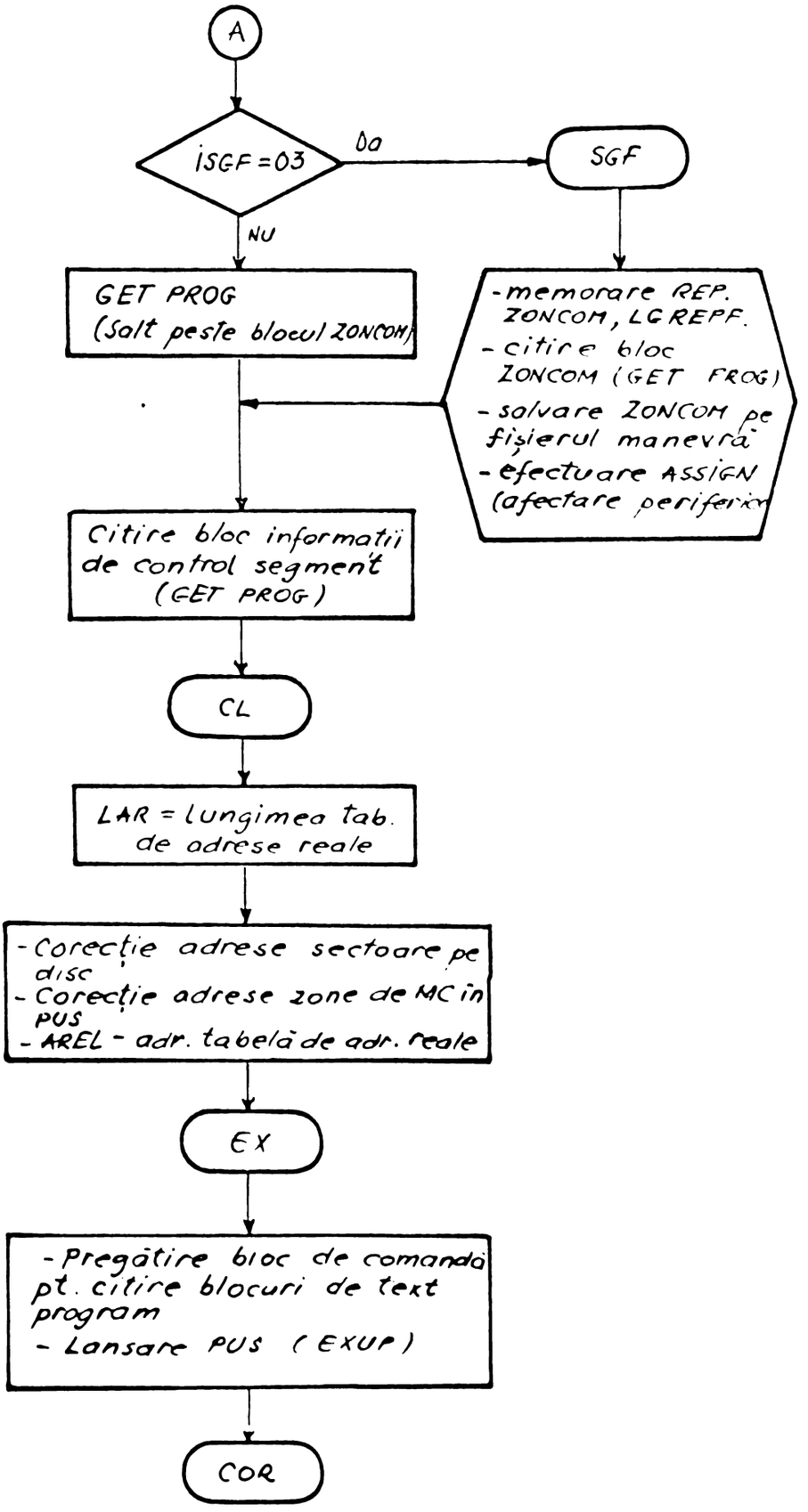
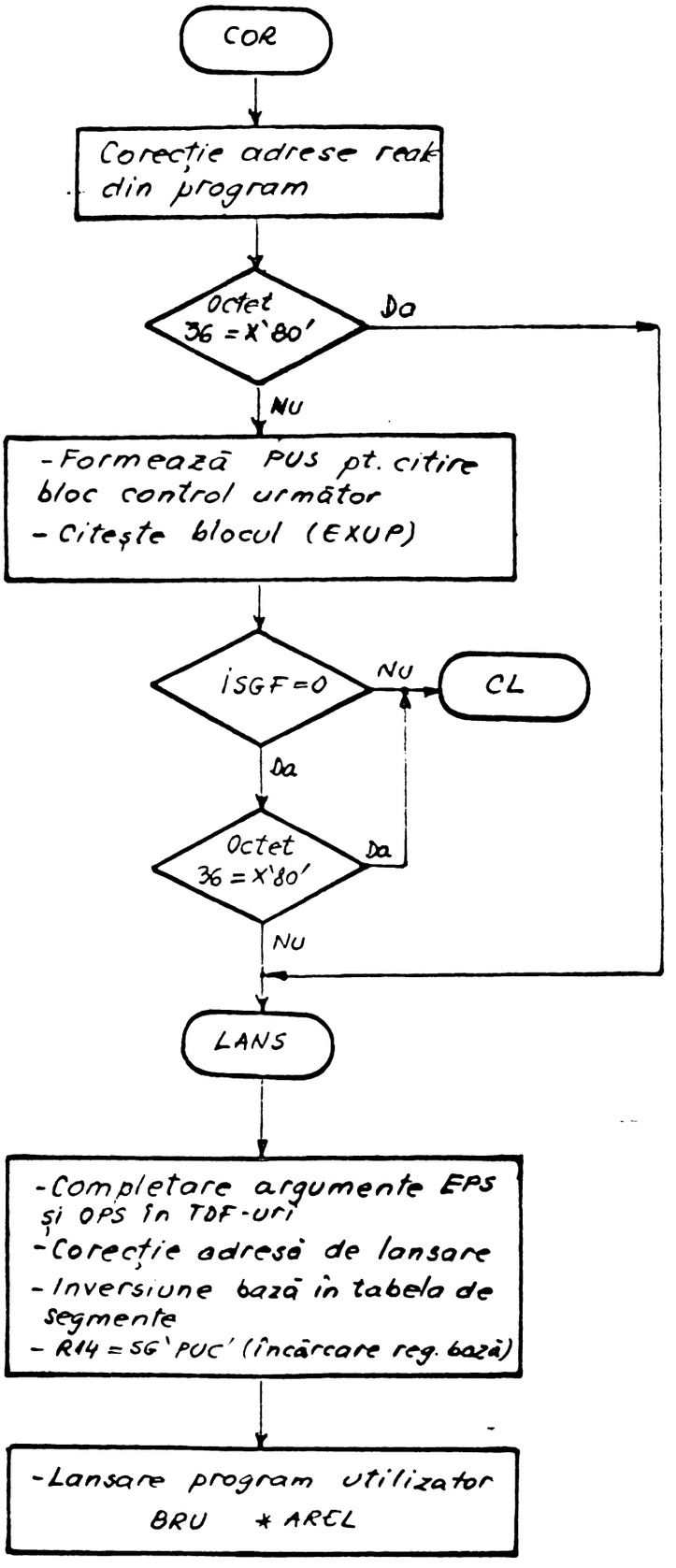


Fig.4.11. Ordinograma modulului de selectare și încărcare programe (SELECT)




COR
Corecție adrese reale din program
Octet 36 = X'80'
Da
NU
- Formează PUS pt. citire bloc control următor
- Citește blocul (EXUP)
ISGF=0
NU
CL
Da
Octet 36 = X'80'
Da
NU
LANS
- Completare argumente EPS și OPS în TOF-uri
- Corecție adresă de lansare
- Inversiune bază în tabela de segmente
- R14 = SG 'PUC' (încărcare reg. bază)
- Lansare program utilizator
BRU * AREL

Dacă octetul 36 din blocul de control segment este X'00', atunci mai există un bloc de control cu blocuri de text asociate, a cărui adresă este pe următorii 3 octeţi (37-39). Cu un bloc de comandă şi un PUT special se citeşte acest bloc şi se repetă operaţiile descrise anterior. După terminarea operaţiilor pentru ultimul bloc de control segment, se corectează în toate TDF-urile argumentele EPS (error program sequence) şi OPS (open program sequence) cu adresa procedurii de tratare erori SCF şi respectiv procedura de modificare dinamică a parametrilor de ASSIGN şi BASE în momentul OPEN-ului de fişier (vezi 3.3.). Dacă programul nu conţine fişiere,ultimele pagini de text program nu se încarcă fiindcă ele conţin modulele de acces *1, *2 şi modulul de legătură SCF, care nu sînt utilizate.

Lansarea directă a programului nu e posibilă, deoarece toate apelurile de subprograme utilizează instrucţia LD4,14 [illegible], care a fost generată de editorul de legături prin LD4,14 15.4. Se încarcă astfel baza segmentului 1 din partiţia care este subsistemul. Din acest motiv înainte de lansare se inversează în tabela de segmente, adresa segmentului 1 cu adresa segmentului fictiv (program utilizator). Se încarcă în R14 baza acestui segment şi se face salt la adresa de lansare a programului. Următoarele reveniri din programul utilizator se vor face în modulul DIALOG la adresa ADI plasată în tabela de segmente la sfîrşit (fig.4.5).

Pentru economie de spaţiu zona tampon pentru citirea blocurilor de control segment s-a definit peste secvenţa de iniţializare, care nu se mai utilizează.

4.3.3. Tratarea informaţiilor referitoare la fişiere.

In subsistemele cu acces multiplu, utilizarea fişierelor în programele utilizator este admisă cu restricţii severe. Restricţiile constau în numărul, tipul şi modul de exploatare a fişierelor.

Subsistemul SIMPLOG (100 Ko memorie ocupată) permite numai utilizarea fişierelor cu acces direct şi numai unul deschis la un moment dat. Programele utilizator trebuie să existe în subsistem la editarea legăturilor şi li se mai aplică o post-link-editare specială. Subsistemul RPS destinat calculatoarelor

IBM/360/370, admite exploatarea restrictivă a unei zone de disc cu o organizare proprie.

Restricţiile care apar la utilizarea fişierelor în programele utilizator în subsistemele cu multiacces nu sînt întîmplătoare. Ele se datoresc modului de organizare a sistemului de gestiune a fişierelor (SGF), care este conceput modular, cu mare flexibilitate, eficienţă şi siguranţă în funcţionare la prelucrarea în loturi. Aceste facilităţi, diversitatea funcţiilor realizate şi interfeţele multiple cu restul sistemului de operare, au dus la o mare complexitate, care nu permite o adaptare uşoară pentru noile condiţii. Descrierea unor module specializate pentru acces la fişiere nu se justifică şi facilităţile oferite sînt reduse faţă de SGF.

Ţinînd cont de particularităţile de organizare şi funcţionare SGF, în subsistemul SCOT s-a rezolvat problema acceptării fişierelor în programele utilizator, fără a introduce restricţii de utilizare. In figura 4.12. se prezintă legăturile care există între program, SGF şi sistemul de operare SIRIS-3. Fişierele utilizate într-un program se declară prin tabele de descriere (TDF), generate de compilatoare pe baza descrierii făcute de programator prin directivele din programul sursă. Ele sînt completate de editorul de legături, încît în execuţie TDF-ul conţine majoritatea informaţiilor referitoare la organizarea fişierului, plasarea modulelor de acces SGF, a FUS asociate, a zonelor tampon utilizate. Modulele de acces SGF şi FUS sînt plasate la sfîrşitul programului şi sînt urmate de modulul de legătură SGF. Aceste asigură interfaţa şi zonele tampon pentru modulele SGF-monitor (OPEN/CLOSE) şi e folosit în comun de toate fişierele. Adresa modulului de legătură este plasată în primul cuvînt al zonei de informaţii generale, urmat de adresa repertoarului de fişiere (REPFIS). In REPFIS se găseşte la început numărul de fişiere utilizate în program, adresa fişierului DCZ urmat de cîte 8 octeţi pentru fiecare fişier. Aceştia conţin indexul fişierului, adresa FILCOM în DCZ, indicatorul de deschidere (X'01'- deschis, X'00'-închis) şi adresa TDF-ului în program.

In urma încărcării unui program, în partiţia afectată din MC, el are toate adresele relocatabile corectate corespunzător implantării. Lipsesc informaţiile referitoare la adresa periferiecelor folosite, locul de plasare a informaţiilor pe suportul magnetic, zonele tampon utilizate, lungimea informaţiilor de transferat. Ele au fost date de programator prin TDF, cartele de comandă ASSIGN şi LABEL, dar sînt păstrate, pentru a putea fi modificate, pînă în

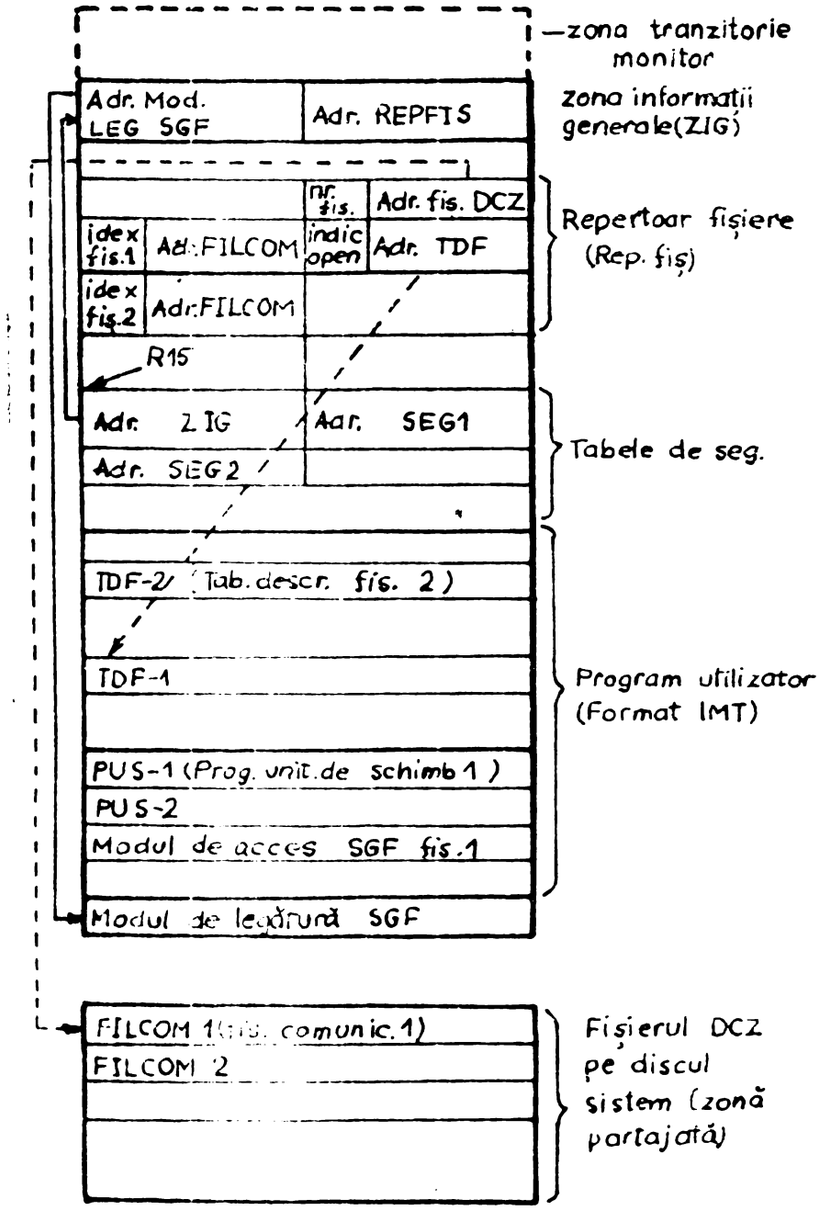


Fig.4.12. Structura informaţiilor SGF dintr-un program activ.

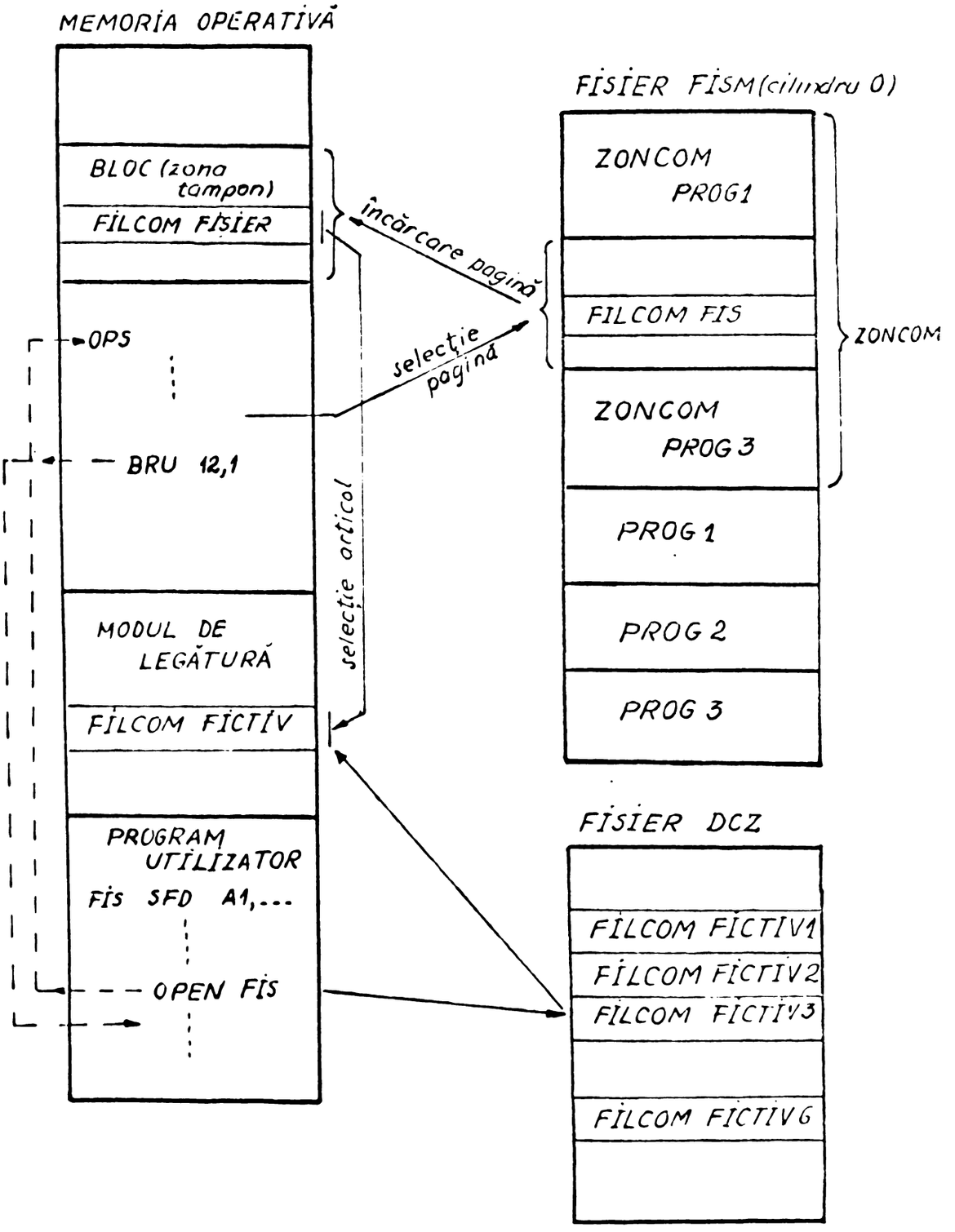


Fig. 4.13. Utilizarea dinamică a informațiilor FILCOM.

momentul efectuării instrucţiilor OPEN. Pentru economie de memorie informaţiile de ASSIGN, LABEL şi FILE (FILCOM) sînt păstrate pe un fişier DCZ (disc communication zone) de pe discul sistem unde sînt copiate la încărcare din zona ZONCOM a programului IMT. La încărcare programului se efectuează macro ASSIGN pe zona ASSIGN înainte de afectare (fig.4.8), care conţine lista aparatelor utilizate şi completează zona ASSIGN după efectare. Prin aceasta se afectează perifericele partiţiei. La încărcare se rezervă atîtea FILCOM-uri în DCZ cîte TDF-uri sînt în program. Informaţiile din DCZ sînt utilizate la OPEN, cînd se validează accesul la fişier şi se identifică poziţia lui pe disc. Generarea FUB şi blocului de comandă asociat pentru efectuarea transferului de pe fişier se face dinamic la execuţia macroinstrucţiilor de I/E de către modulul de acces SCF înglobat în program. Pentru economie de memorie modulele OPEN/CLOSE se încarcă dinamic în momentul execuţiei într-o zonă tranzitorie a monitorului.

Pentru a utiliza fişierele în programele utilizator din subsistemul cu multiacces SCAT, s-au declarat în segmentul fictiv (FUC) 6 fişiere fictive, pentru ca la încărcarea subsistemului să se rezerve spaţiu în RSPFIS şi în DCZ. Acesta va fi numărul maxim de fişiere admis în fiecare din programele în lucru. La încărcarea programului, se citeşte din bibliotecă zona ZONCOM, se execută ASSIGN-urile şi se depun toate FILCOM-urile pe fişierul de manevră. Nu se depun în DCZ fiindcă ar necesita un spaţiu mare şi ar apare confuzii de index, dacă se iau în considerare toate programele active. La lansarea în execuţie a programului nu se utilizează zona FILCOM de pe disc. Se completează însă RSPFIS-ul subsistemului cu RSPFIS-ul programului utilizator, peste zona fişierelor fictive. Instrucţia OPEN va citi de pe DCZ FILCOM-ul fişierului fictiv în modulul de legătură. Conţinutul lui nu contează fiindcă va fi înlocuit cu FILCOM-ul fişierului din program, luat de pe fişierul de manevră (fig.4.13). Înlocuirea este făcută de un subprogram activat de argumentul OPN, completat la încărcarea programului. Deschiderea fişierului se va face corect şi programul va rula în continuare efectuînd transferuri cu fişierul.

Cu această soluţie, la fiecare salvare a programului pe disc nu vor trebui salvate şi FILCOM-urile fişierelor. Restul informaţiilor referitoare la SCF, ca module de acces, TDF-uri, blocuri de comandă FUB, zone tampon, DL sînt salvate şi reîncărcate odată cu programul. La încărcarea programelor informaţiile referitoare la fişierele *1 şi *2 sînt neglijate şi eliminate.

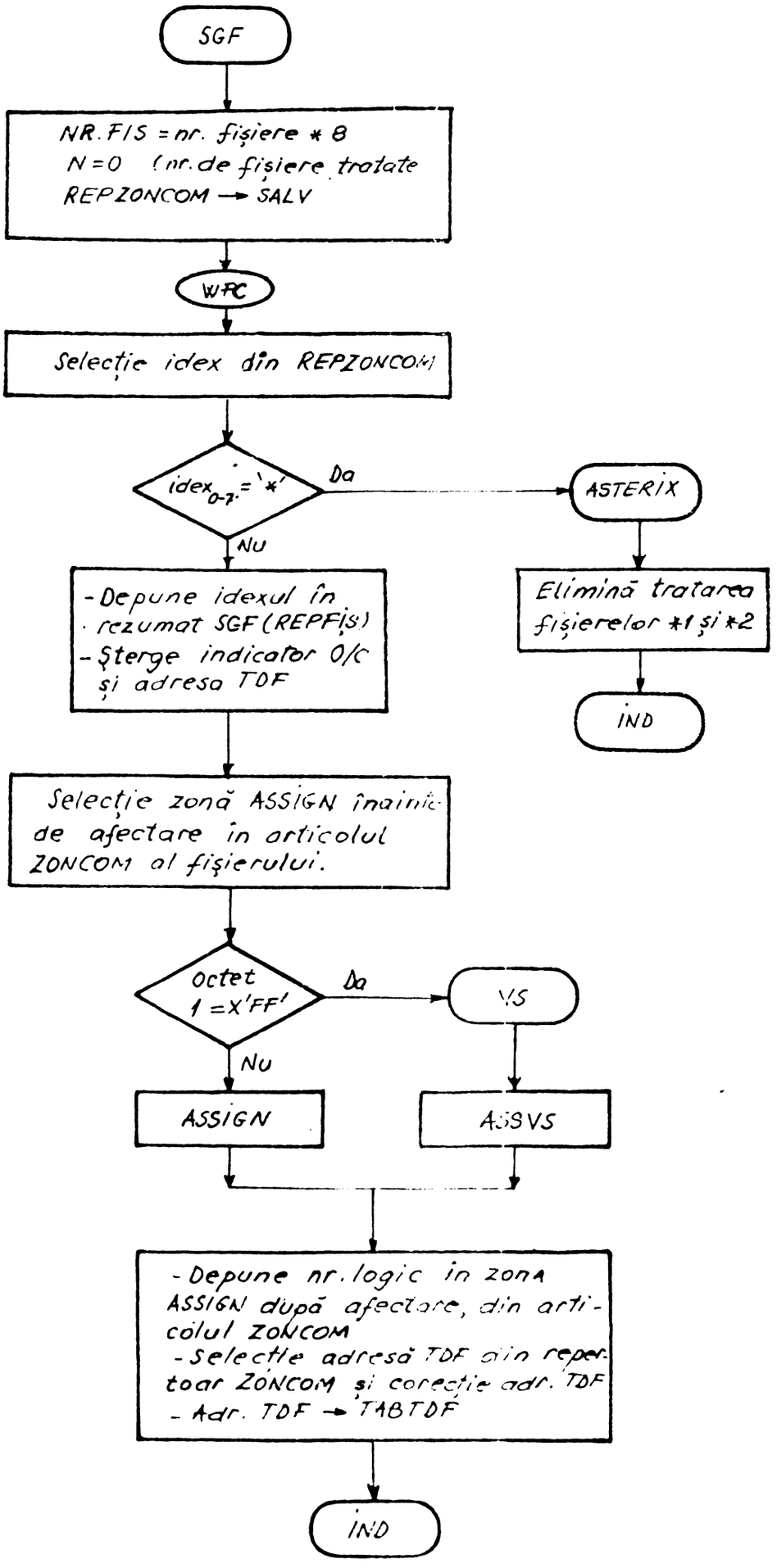
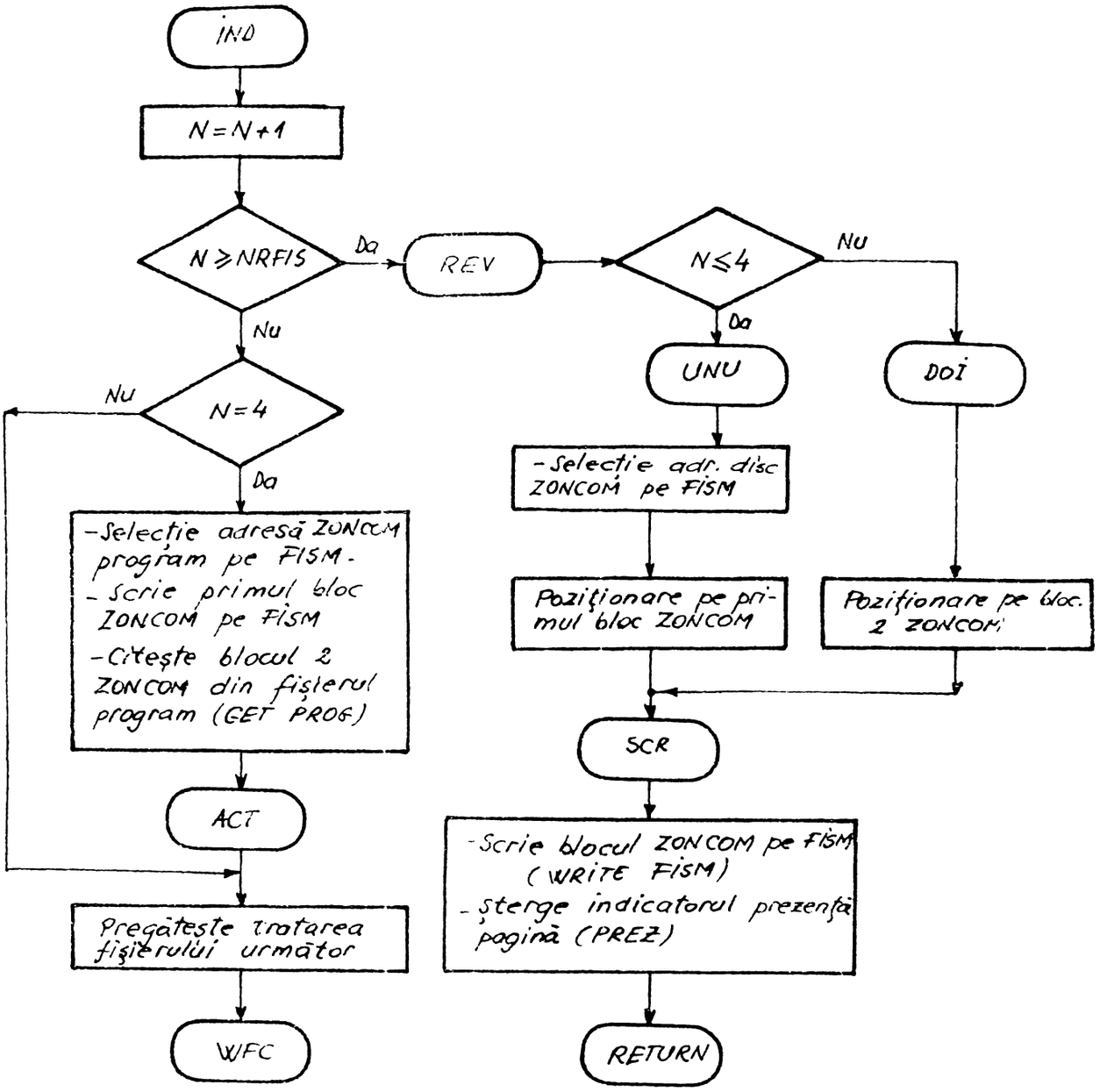
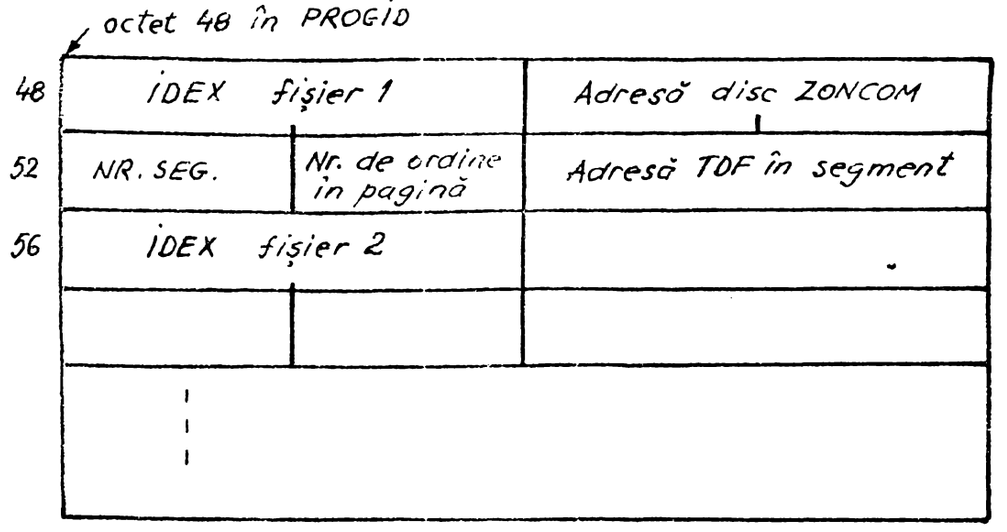


Fig. 4.14. Ordinograma secvenței de tratare SGF la încărcarea programului.





Structură repertoar ZONCOM.

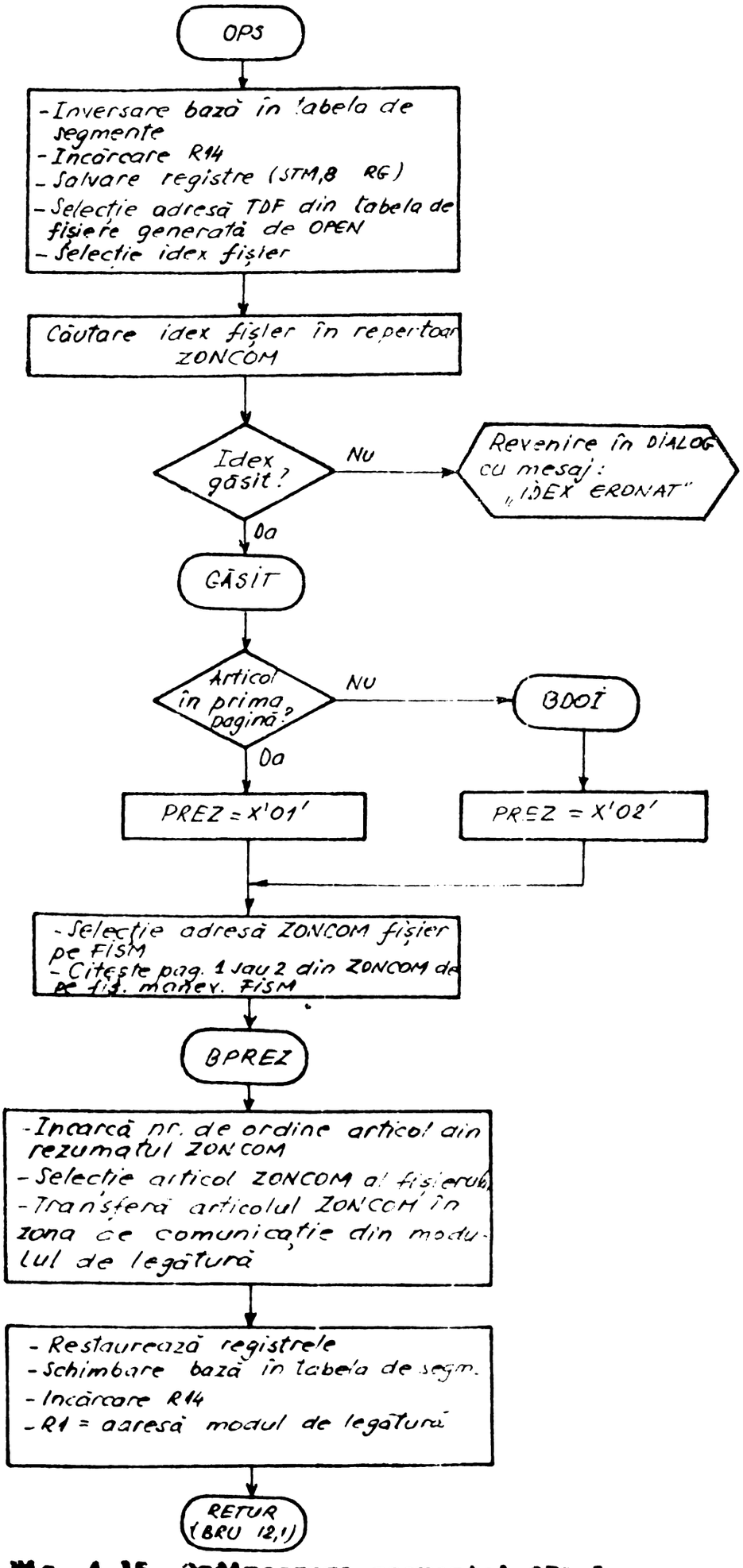


Fig. 4.15. Ordinograma secvenței OPS la deschiderea unui fișier utilizator.

La fişierele din programele utilizator nu se impun restricţii de idex, tip de organizare sau mod de exploatare. Se admit chiar şi fişiere secvenţiale. Nu se admite utilizarea idexului U,V şi A folosite pentru fişierele bibliotecă, program şi manevră în subsistem. Toate erorile posibile de SGF sînt preluate şi tratate de modulul de erori prin secvenţa ERS. Din punct de vedere al utilizării fişierelor, subsistemul SCOT are cele mai bune performanţe dintre referinţele cunoscute în literatura de specialitate.

Se asigură o compatibilitate completă la nivel de SGF cu prelucrarea în loturi a programelor.

4.3.4. Tratarea programelor segmentate

In subsistemul cu multiacces nu se admite în general utilizarea unor programe segmentate. Restricţia apare datorită modului de tratare a segmentelor de către sistemul de operare şi felului în care se realizează comunicarea între segmente. Intr-un program segmentat adresele de implantare a segmentelor în program se calculează de către editorul de legături pe baza structurii de arbore definită prin cartela de comandă TREE. Aceste adrese se memorează în tabela de segmente (TABSEG) din blocul PROGID al programului IMT din bibliotecă (Fig.4.7). Pentru fiecare segment de program există cîte 8 octeţi în TABSEG care cuprinde poziţia segmentului în arbore (numărul segmentului anterior), adresa de implantare în program, şi adresa primei pagini a segmentului pe disc (primul bloc de control segment).

Cu aceste informaţii, folosind modulul de încărcare a programelor IMC, s-ar putea încărca oricare segment în MC. Metoda nu poate fi aplicată, deoarece apelul segmentelor în program se face prin macroinstrucţia CALL. Se generează o instrucţie inexistentă, în care segmentul este reperat prin numărul său de ordine în tabela de segmente. Monitorul central verifică dacă segmentul apelat e încărcat în MC. In acest caz identifică adresa segmentului şi adresa punctului de intrare specificat. Tabela punctelor de intrare se găseşte la începutul segmentului. Saltul la punctul de intrare se face din monitor cu refacerea indicatorilor de condiţie, pentru ca funcţional saltul prin CALL să fie echivalent cu saltul prin BRU.

Dacă segmentul chemat nu e prezent în MC, monitorul apelează subprogramul de încărcare a segmentului de pe disc. Se folosesc informaţiile din TABSEG din blocul PROGID, care indică adresa pe disc şi în MC a segmentului. După încărcare se face saltul la punctul de intrare specificat. Asemănător se întîmplă la încărcarea segmentelor de către

programator prin macroinstrucţia LDSG. In ambele cazuri se încarcă şi toate segmentele care preced segmentul în arborele dat.

Acest mod de lucru nu permite într-un subsistem cu multi-acces încărcarea dinamică a segmentelor programului utilizator prin folosirea unui subprogram de încărcare segmente. Apelul de încărcare segment (LDSG sau CALL) se face prin apel monitor, care face ca subsistemul să piardă controlul. Referirile se fac la tabela de segmente, care este actualizată la lansarea programului. Dacă însă segmentul nu este încărcat, sistemul de operare nu are cunoştinţă de existenţa în lucru a programului utilizator. Pentru el programul activ este subsistemul. Pentru încărcarea segmentului cerut el va căuta segmentul X din subsistem, deoarece în tabela de control a partiţiei (TCT = table de controle du travail), aflată în monitor, se găseşte adresa disc a subsistemului în biblioteca din care face parte.

Ca soluţie optimă de încărcare a segmentelor program în subsistemul SCAT s-a ales schimbarea adresei pe disc a programului şi bibliotecii în TCT, la activarea programului. In acest fel încărcarea segmentelor este lăsată pe seama sistemului de operare. Pentru ca încărcarea şi chemarea segmentelor să se poată face în bune condiţii, la încărcarea programului se completează o tabelă de segmente proprie programului, pe baza informaţiilor din TABSEG. Această tabelă va înlocui tabela de segmente a subsistemului (partiţiei), la fiecare activare a programului. La revenirea în subsistem se reface tabela iniţială. Pentru a avea loc suficient în tabela de segmente a partiţiei pentru programe puternic segmentate, s-au declarat în subsistem 30 segmente fictive, care vor da numărul maxim de segmente într-un program. Pentru a modifica TCT-ul partiţiei la lansarea unui program segmentat, se trece pentru scurt timp în regim supervizor. La terminarea programului sau la punerea lui în aşteptare, se reface conţinutul iniţial al zonelor TCT modificate. Aceasta este singura operaţie care încalcă protecţiile sistemului de operare. Ea simplifică însă considerabil rezolvarea problemei utilizării programelor segmentate, care altfel pare fără soluţie acceptabilă.

Introducerea facilităţii de utilizare a programelor segmentate introduce 150 octeţi suplimentari în zona proprie a utilizatorului, din care 120 octeţi sînt ocupaţi de tabela de segmente. Creşterea performanţelor subsistemului este apreciabilă, numărul programelor segmentate este mare în aplicaţiile de proiectare asistată de calculator. Se reduce astfel dimensiunea partiţiei necesare rulării programelor complexe.

4.3.5. Salvarea programelor în aşteptare.

Subsistemul deserveşte mai mulţi utilizatori aflaţi la terminale diferite, fiecare lucrînd cu un anumit program pe care l-a solicitat. La un anumit moment dat în memorie se află un singur program care execută calcule în UC sau utilizează fişiere. Incărcarea simultană a mai multor programe nu se recomandă din considerente de utilizare intensivă a UC şi din lipsa de protecţie individuală a programelor. Operaţiile de intrare/ieşire pe terminale sînt foarte lente. La un debit de transfer de 120 car/sec (1200 baud) pentru transferul unei linii de tipărit (afişat) de 130 caractere se foloseşte mai mult decît 1 sec. Pentru introducerea unui rînd de 40 caractere de la terminal la o viteză de tastare de 2 caractere/sec, se aşteaptă 20 sec. Acest timp este considerabil şi se foloseşte pentru activarea altui program,care se va încărca de pe fişierul de manevră FISM pe care a fost salvat. Timpul de salvare a programului activ pe FISM şi încărcare a programului activat este redus. La o dimensiune medie a programelor de 60K, dacă se utilizează discuri de 50 Mo,operaţia durează cca. 0,3 sec. Pe această perioadă UC nu este utilizată şi e folosită de programele care lucrează în partiţii cu prelucrare în loturi. Transferurile cu terminalele se execută permanent pentru toţi utilizatorii în regim de multitasking prin modulul DIALOG.

După activarea programului,el se pune în aşteptare la orice operaţie de transfer cu terminalul (citiri, afişări). Cererea de transfer se adresează modulului DIALOG, care o realizează şi comanda se dă altui proces pregătit. Activarea programului corespunzător noului proces activ se face prin utilizarea subprogramului de salvare a programului în aşteptare (DEPUN) şi a subprogramului de reîncărcare a programului activat (CITPR) de pe FISM.

Fiecărui utilizator îi este rezervat pe fişierul de manevră FISM un cilindru (120 Ko). Adresele acestor zone se ţin într-un tabel ZOND adresat prin indexare cu numărul procesului asociat. Se pot utiliza programe care ocupă maxim 120 K memorie. Pentru păstrarea informaţiilor SGF aferente fiecărui program (ZONCOM), încărcate din fişierul program din bibliotecă se rezervă primul cilindru din FISM. Fiecare proces va avea rezervat în ordine (fig. 4.16) cîte 2 sectoare de disc (2 x 1024 oct). Această zonă va fi exploatată la deschiderea fişierelor de secvenţa OPS prezentată anterior.

Pentru controlul procesului de salvare/reîncărcare de programe se folosesc următorii indicatori :

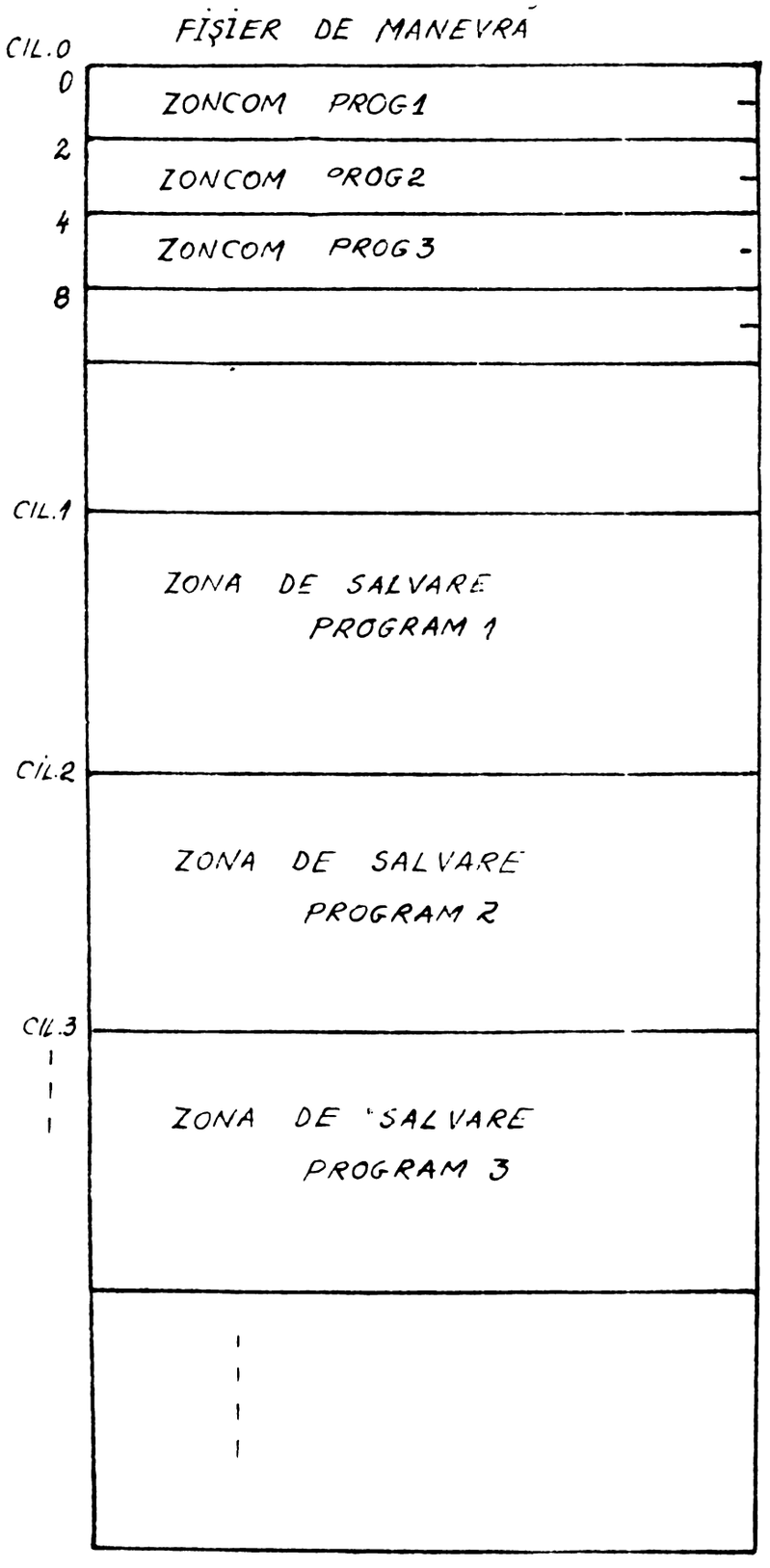


Fig.4.16. Structura fişierului de manevră (FIL)

a) Indicatorul INCARC se păstrează în zona proprie liniei şi specifică starea programului asociat.

INCARC = 0 program neselectat şi neîncărcat (iniţial);

INCARC = 1 programul în aşteptare pe fişierul FISM;

INCARC = 2 programul este încărcat în memorie.

Indicatorul se poziţionează pe valoarea 2 la încărcarea programului din bibliotecă (modulul SELECT) şi la reîncărcarea de pe fişierul FISM (modulul CITPR). Poziţionarea pe 1 se face la salvarea programului pe fişierul FISM (DEPUN).

b). Indicatorul PROGI specifică numărul liniei a cărui program este calcularea adresei zonei din FISM pe care se va salva la punerea în aşteptare.

c). Indicatorul DELANS specifică numărul liniei active. El se calculează pornind de la R13 în care SCT-ul dă adresa LCB-ului liniei active. Se foloseşte pentru aceasta un tabel al adreselor zonelor proprii fiecărei linii (ADZONE). Indicatorul DELANS serveşte la calcularea adresei zonei din fişierul FISM, care conţine programul ce trebuie activat.

La activarea unei linii se verifică dacă programul asociat este încărcat (INCARC = 2). Dacă nu este, se verifică dacă un alt program e încărcat (PROGI $\neq$ 0), se salvează acest program pe FISM şi se încarcă programul activ de pe FISM. La salvarea programului se va depune repertoarul de fişiere REPFIS şi tabela de segmente care îi aparţine, în zona proprie liniei (subprogramul DEPUN). Scrierea programului pe FISM se face cu lungimi de transfer de un cilindru, pentru a avea cît mai puţine operaţii de scriere.

La încărcarea programului de pe FISM, prin subprogramul CITPR se va reface tabela de segmente, repertoarul de fişiere şi se va modifica tabela TCT dacă programul e segmentat (adresa programului în bibliotecă).

4.4. MODULUL DE TRATARE A ERORILOR.

Erorile care pot apare în partiţie în timpul lucrului în multiacces sînt diverse şi pot avea cauze diferite:

- Erori datorate funcţionării defectuoase a liniilor, şi echipamentelor de teletransmisie (cuplatoare, modemuri, terminale);
- erori de programare a operaţiilor de teletransmisie;
- erori de SCF în programul utilizator;
- utilizarea macroinstrucţiei TERM sau ABORT, în programul utilizator;

- erori de programare care produc derute în programul utilizator;
- erori în cartele de date FORTRAN (erori FORMAT, radical sau logaritm din număr negativ, etc);
- erori în subsistem;
- întreruperi de la consola centrală.

Toate aceste tipuri de erori se tratează unitar de către subsistem, care nu trebuie să piardă comanda în nici un caz.

Erorile se detectează în zona în care se produc şi se transmite modulului de erori un cod. Pe baza acestui cod se analizează tipul erorii, se identifică cauza şi se afişează un mesaj la terminal sau la consola centrală.

Toate operaţiile de teletransmisie (SEND, RECEIVE) sînt grupate într-o secvenţă comună, urmate de analiza indicatorilor de stare Z, S şi a conţinutului lui R12 care dă codificat tipul erorii. Se transmit apoi terminal mesajele corespunzătoare de eroare şi se încearcă corectarea erorii prin închiderea şi redeschiderea liniei. Dacă eroarea persistă se suprimă linia şi se anunţă operatorul central.

Erorile de SGF în programul utilizator se detectează printr-o secvenţă specială de analiză, a cărei adresă se trece în argumentul ERR din TDF-urile tuturor fişierelor la încărcarea programului. Mesajul de eroare dat de SGF în acest caz se recuperează din modulul de legături şi se afişează la terminal. Utilizatorul are posibilitatea să relanseze programul dacă eroarea s-a datorat unor argumente furnizate greşit. Argumentele de identificare a programului şi bibliotecii se detectează în dialogul iniţial şi se cere corectarea.

La iniţializarea subsistemului se specifică adresa unei secvenţe de SAR, care va prelua toate derutele apărute în partiţie. Se analizează originea derutei (subsistem sau program), tipul ei (adresă inexistentă, depăşire flotantă sau binară, etc) şi se va afişa la terminal un mesaj corespunzător indicînd adresa relativă în program unde ea a apărut. Programul respectiv va fi terminat fără a perturba ceilalţi utilizatori.

O altă categorie de erori proprii programelor FORTRAN sînt cele detectate la citirea cartelor de date care nu respectă FORMAT-ul, ("ERREUR DANS LA CHAINE D'ENTREE"). In acest caz se afişează mesajul şi se permite utilizatorului să reintroducă ultima linie de date. Erorile detectate în diferite proceduri FORTRAN (SQRT sau ALOG din număr negativ, etc) sînt preluate prin modificarea subprogramului ISAR1, se afişează la

terminal mesajul de eroare şi se termină programul.

Erorile din subsistem se datoresc în general unor erori de I/E apărute pe suporturi magnetice, sau erori de sistem şi sînt foarte rare. Ele impun relansarea subsistemului.

Macroinstrucţiile TERM sau ABORT întîlnite accidental în programele utilizator, datorate unor salturi sau indexări greşite sînt preluate utilizînd SXR M. Se asigură astfel o mare siguranţă în exploatarea subsistemului, încît erorile în programele utilizator sau în sistemul de teletransmisie sînt semnalate la terminal, dar nu deranjează pe ceilalţi utilizatori.

4.5. DIVIZAREA TIMPULUI INTRE UTILIZATORI

4.5.1. Modelarea unui sistem cu multiacces [Hel75], [Dod81], [Jur79], [Pri76]

Performanţele unui subsistem cu multiacces se măsoară ţinînd cont de numărul utilizatorilor deserviţi în sistem la un moment dat şi timpul mediu de răspuns obţinut la terminale. Aceste performanţe depind de mai mulţi factori din care amintim caracteristicile sistemului de exploatare, structura subsistemului, viteza şi capacitatea memoriei centrale, viteza unităţii centrale, viteza de transfer şi poziţionare a discurilor folosite pentru fişierele de lucru şi de manevră, etc.

Pentru a analiza matematic comportarea subsistemului la deservirea a n utilizatori vom folosi teoria firelor de aşteptare. Notăm : - $P_j(t)$ probabilitatea ca la momentul t sistemul să se găsească în starea j. Starea j înseamnă că j utilizatori se găsesc în firul de aşteptare al sistemului, în care se include cel în curs de servire. Fiindcă există o probabilitate egală ca sistemul să se găsească în orice stare $\sum_{j=0}^{n} P_j(t) = 1$.

- h un interval de timp scurt
- $a_j h$ probabilitatea ca în starea j un utilizator să intre în fir în intervalul (t, t + h), adică cerere de servire
- $w_j h$ probabilitatea ca în starea j un utilizator să părăsească firul în intervalul (t, t + h), adică o servire terminată.

Considerăm că a_j şi w_j nu depind de j. Dacă intervalul h este suficient de mic, este posibil ca la un timp t în fir să intre sau să iasă un utilizator sau niciunul. Considerînd starea j, la ea

se poate ajunge-din starea j-1 dacă intră un utilizator şi nu iese nici unul;
- din starea j dacă nu intră şi nici nu iese din fir utilizatori;
- din starea j dacă un utilizator intră şi altul iese;
- din starea j + 1 dacă iese un utilizator şi nu intră nici unul.

Probabilitatea stării j la momentul t + h se poate scrie faţă de probabilităţile stărilor j-1, j şi j+1 la momentul t :

$$P_j(t+h) = a_{j-1}hP_{j-1}(t)+(1-(a_j+w_j)h)P_j(t)+w_{j+1}hP_{j+1}(t)+h^2f \quad (1)$$

Lăsând în membrul stâng şi drept $P_j(t)$ şi împărţind cu h obţinem pentru $h\to 0$ derivata lui $P_j(t)$ în raport cu timpul :

$$P_j'(t)=a_{j-1}P_{j-1}(t) -(a_j+w_j)P_j(t) + w_{j+1}P_{j+1}(t) \quad (2)$$

Pentru j = 0 când firul de aşteptare nu conţine cereri de utilizare $a_{j-1}=a_{-1}=0$ şi $w_0=0$ fiindcă nu pot exista ieşiri dintr-un fir vid. Atunci ecuaţia (2) devine :

$$P_0'(t) = -a_0P_0(t) + w_1P_1(t) \quad (3)$$

Ecuaţiile (2) şi (3) descriu comportarea sistemului în timp. După un regim tranzitoriu iniţial sistemele de deservire descrise prin ecuaţiile de mai sus devin stabile şi $P_1'(t) = 0$. Din ecuaţia (3) rezultă condiţia de stabilitate pentru orice j şi t.

$$P_1 = \frac{a_0}{w_1}P_0 \quad (4)$$

Rezolvând în regim de stabilitate ecuaţia (2) pentru P_{j+1} obţinem

$$P_{j+1} = \frac{a_j + w_j}{w_{j+1}}P_j - \frac{a_{j-1}}{w_{j+1}}P_{j-1} \quad (5)$$

Făcând substituţii succesive pentru valoarea lui j din (4) şi (5) obţinem :

$$P_j = \frac{a_0\,a_1\,a_2\cdots a_{j-1}}{w_1\,w_2\cdots w_j}P_0 = \frac{a_0}{w_j}P_0\prod_{i=1}^{j-1}\frac{a_i}{w_i} \quad (6)$$

Însumând valorile probabilităţii pentru toate valorile lui j din ecuaţia (6) trebuie să obţinem 1. Rezolvând ecuaţia faţă de P_0 rezultă :

$$P_0 = \frac{1}{1+\frac{a_0}{w_1}+\sum_{j=2}^{n}\frac{a_0}{w_1}\left(\prod_{i=1}^{j-1}\frac{a_i}{w_i}\right)} \quad (7)$$

Ecuaţiile (6) şi (7) sînt de bază în descrierea firelor de aşteptare stabile. Din (6) şi (7) se obţine lungimea medie a firului

$$\overline{j} = \sum_{j=0}^{n} j P_j = \left(\frac{a_0}{w_1} + \frac{2a_0a_1}{w_1 w_2} + \frac{3a_0 a_1 a_2}{w_1 w_2 w_3} + \ldots \right) P_0 \quad (8)$$

Dacă în ecuaţiile (6) şi (8) luăm a_j şi w_j cu valori identice a şi w, vom defini intensitatea de trafic cu unitatea de măsură erlangs ca fiind raportul:

$$r = \frac{a}{w} = \frac{\text{media timpului de servire al unui utilizator}}{\text{media timpului între două cereri de servire succesive}}$$

Cu această notaţie ec. (6) devine :

$$P_j = r^j P_0 \quad (9)$$

Deoarece $\sum_{j=0}^{n} P_j = 1$ substituind P_j din ec. (9) obţinem:

$$P_0 = \frac{1 - r}{1 - r^{n+1}} \quad (10)$$

Substituind valoarea lui P_0 din (10) în (9) rezultă:

$$P_j = \frac{r^j (1 - r)}{1 - r^{n+1}} \quad (11)$$

Lungimea medie a firului de aşteptare (inclusiv utilizatorul în curs de servire) se obţine substituind (11) în (8):

$$\overline{j} = \frac{r}{1 - r^{n+1}} \cdot \frac{1 + r^n (nr - n - 1)}{1 - r} \quad (12)$$

pentru : $r \ll 1$ rezultă $\overline{j} = r + r^2$

$r \to 1$ " $\overline{j} = \frac{n}{2} + \frac{n(n+2)(r-1)}{12}$

$r \gg 1$ " $\overline{j} = n - \frac{1}{r}$

Aplicăm rezultatele obţinute mai sus la modelarea unui subsistem cu multiacces, care deserveşte n utilizatori. In timpul funcţionării subsistemului utilizatorii pot fi în starea de aşteptare a rezultatelor sau în starea de introducere a unor noi date, care o vom numi stare de gîndire. Orice introducere de date de la terminal se poate considera ca o întrebare, sau o cerere de serviciu, care este preluată de subsistem în firul de aşteptare în momentul terminării ei. Introducerea de date se face în general linie cu linie. O sesiune la terminal este o succesiune de întrebări şi răspunsuri. In timpul aşteptării răspunsului nu se pot introduce noi date. Aşteptarea cuprinde timpul de prelucrare efectivă, timpul cît cererea este în firul de aşteptare (sînt deservite cereri mai prioritare) şi timpul de pregătire a lucrării (refacere registre şi tabele şi timpul de efectuare a operaţiilor de " swaping "). Vom nota:

- T - timpul mediu de gîndire a unui utilizator,
- a = 1/T frecvenţa medie de cerere de serviciu a unui utilizator, cînd se găseşte în stare de gîndire,
- $a\,h$ probabilitatea ca un utilizator singur (oarecare) să ceară serviciu într-un interval de timp scurt h,
- a_j frecvenţa medie de cerere pentru un utilizator oarecare cînd sistemul e în starea j (j cereri în firul de aşteptare şi n-j utilizatori în starea de gîndire),
- c timpul mediu de servire pentru un utilizator, care include timpul de swaping (salvare program existent în MC şi încărcarec programului utilizatorului servit),
- W = 1/C frecvenţa medie de servire, cu care se obţin răspunsuri din sistem,
- $a_j\,h$ probabilitatea de cerere în starea j pentru un utilizator într-un interval de timp h,
- $W_j\,h$ probabilitatea ca un utilizator să fie scos din şir (complet servit) în intervalul de timp h.

Deoarece în starea j a subsistemului numai utilizatorii în aşteptare (n-j) pot să ceară serviciu avem probabilitatea de cerere:

$$a_j\,h = \frac{(n-j)\,h}{T} \quad \text{de unde} \quad a_j = \frac{n-j}{T} = (n-j)\,a \qquad (13)$$

Probabilitatea ca un utilizator să iasă din aşteptare (complet servit) în starea j este:

$$W_j\,h = \frac{h}{C} \quad \text{de unde} \quad W_j = W \qquad (14)$$

Înlocuind valorile lui a_j şi W din (13) şi (14) în ecuaţia (6) obţinem probabilitatea stării j cînd subsistemul a ajuns într-un regim de deservire stabil.

$$P_j = P_0 \cdot \frac{na}{W} \cdot \frac{(n-1)\,a}{W} \cdot \frac{(n-2)a}{W} \cdots \frac{(n-j+1)a}{W}$$

Introducem intensitatea de trafic $r = \frac{a}{W} = \frac{C}{T}$:

$$P_j = P_0\,\frac{n!\,r^j}{(n-j)!} \qquad (15)$$

Deoarece suma probabilităţilor P_j este 1 obţinem valoarea lui P_0 utilizînd ecuaţia (15): (16)

$$P_0 = \frac{1}{\sum_{j=0}^{n} \frac{n!\,r^j}{(n-j)!}}$$

Substituim valoarea lui P_0 din (16) în ecuaţia (15):

$$P_j = \frac{n! \, r^j}{(n-j)!} \cdot \frac{1}{\sum_{j=0}^{n} \frac{n! \, r^j}{(n-j)!}} \qquad (17)$$

Lungimea medie a firului de aşteptare în regim de lucru stabil este :

$$\bar{j} = \sum_{j=0}^{n} j \cdot P_j \qquad (18)$$

Ecuaţia care exprimă relaţia dintre lungimea medie a firului $\bar{j}$ şi timpul mediu de aşteptare în fir $\bar{t}_q$ în starea de lucru stabil este:

$$\bar{a} \cdot \bar{t}_q = \bar{j} \qquad (19)$$

unde $\bar{a}$ este frecvenţa medie de cereri de serviciu în sistem luînd în considerare toţi utilizatorii.

$$\bar{a} = \sum_{j=0}^{n} a_j \cdot P_j = \sum_{j=0}^{n} \frac{n-j}{T} \cdot P_j \qquad (20)$$

Ţinînd cont că o sumă de P_j este 1 şi de ecuaţia (18) obţinem :

$$\bar{a} = \frac{n}{T} \sum_{j=0}^{n} P_j - \frac{1}{T} \sum_{j=1}^{n} j \cdot P_j = \frac{n-\bar{j}}{T} \qquad (21)$$

Înlocuind (21) în ecuaţia (19) obţinem timpul mediu de aşteptare în şir $\bar{t}_q$:

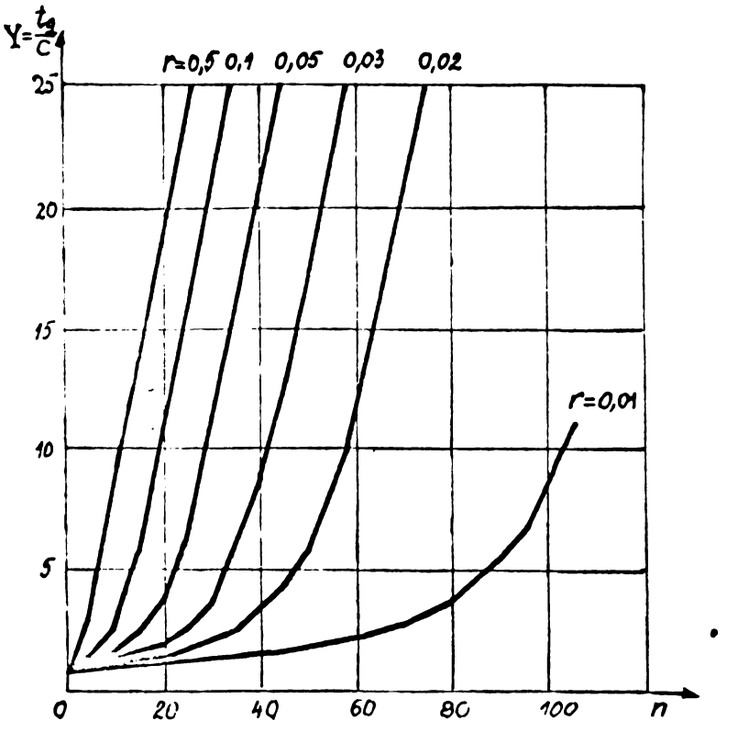
$$\bar{t}_q = \frac{\bar{j} \cdot T}{n - \bar{j}} \qquad (22)$$

Pentru a ţine cont de timpul mediu de prelucrare C, care include performanţele sistemului de calcul în [Hel 75] se demonstrează că :

$$\frac{\bar{t}_q}{C} = \frac{n}{1-P_0} - \frac{1}{r} \qquad (23)$$

Înlocuind în (23) din ecuaţia (16) se obţine

$$\frac{\bar{t}_q}{C} = \frac{n}{1 - \frac{1}{\sum_{j=0}^{n} \frac{n! \, r^j}{(n-j)!}}} - \frac{1}{r} \qquad (24)$$



C - timpul mediu de prelucrare + swaping

T - timp mediu de gîndire

$r = \frac{C}{T}$

n - nr. de utilizatori activi

t_q - timp mediu de aşteptare.

Fig.4.17. Timpul mediu de aşteptare normalizat într-un subsistem cu acces multiplu

Formula (24) poate fi direct utilizată pentru tabelare şi reprezentare grafică funcţie de numărul de utilizatori n, luînd ca parametru r. Notînd cu Y (n) raportul t_q/C se poate obţine o formulă recursivă pentru calcul

$$Y(n+1) = (n+1) - \frac{n}{1+rY(n)} \qquad (25)$$

unde Y (1) = 1

Folosind formula (25) se obţine graficul funcţiei Y (n) cu parametru r reprezentat în fig. 4.17.

Aceste rezultate teoretice permit o bună estimare a performanţelor unui subsistem cu multiacces şi indică elementele care influenţează pozitiv şi pot fi avute în vedere la proiectare. In continuare se vor folosi rezultatele de mai sus pentru estimarea performanţelor subsistemului SCOT. Se vor scoate în evidenţă alţi factori obiectivi care limitează performanţele.

4.5.2. Evaluarea performanţelor subsistemului SCOT

Se vor analiza performanţele subsistemului SCOT, ţinînd cont de caracteristicile echipamentelor folosite în exploatarea sa pe calculatoarele FELIX C256/512. Vom considera o divizare naturală a timpului între utilizatori, în care un program este pus în aşteptare la fiecare operaţie de I/E cu terminalul şi se activează alt program pregătit prin operaţii de swaping. In aceste condiţii timpul mediu de prelucrare notat anterior cu C cuprinde :

- timpul de calcul în UC (t_c),
- timpul de refacere tabele şi registre la reactivarea programului (t_a),
- timpul de depunere (salvare) a programului activ (t_d),
- timpul de încărcare a programului activat (t_i),

$$C = t_c + t_a + t_d + t_i$$

Considerăm un timp mediu de calcul t_c = 350 ms în care pentru o viteză a UC de 300.000 operaţii pe secundă se pot efectua 100.000 de instrucţii. Număr suficient de mare pentru prelucrarea unei linii de date introduse (instrucţii efectuate între două I/E la terminal).

Timpul auxiliar t_a de refacere şi salvare a stării programului la activare este în SCOT foarte redus. Tabela de segmente şi repertoarul de fişiere se păstrează în MC şi mutarea lor se face cu cca. 150 instrucţii (5 msec). Deoarece tabelele ZONCOM ale fişierelor se păstrează pe disc, ele se scriu la încărcarea programului şi se încarcă în MC doar la deschiderea şi la închiderea fişierelor. Prin această soluţie se reduce cu cca. 40 % timpul de pregătire faţă de cazul cînd s-ar actualiza fişierul DCZ la fiecare activare a programului. Se elimină o operaţie de citire din FISM şi cel puţin una de scriere în DCZ a informaţiilor ZONCOM.

Fiind vorba de acelaşi set de programe depuse şi încărcate de pe disc $t_d = t_i$. Suma lor reprezintă timpul de swaping

$$t_s = t_d + t_i.$$

Timpul de depunere este compus din timpul de poziţionare pe sectorul de început a zonei program (t_p) şi timpul efectiv de transfer (t_t).

Pentru discurile de tip MD50, cu o capacitate de 50 Mo, şi 400 de cilindri , timpul de poziţionare pe cilindru variază între 25 şi 80 msec, corespunzător deplasării de la o pistă la alta, respectiv peste 400 piste. Timpul mediu de acces se calculează probabilistic pentru numărul de cilindri împărţit la 3 (Nc/3) şi este de cca. 40 ms. La aceasta se adaugă timpul de poziţionare pe sector, care în medie corespunde cu timpul de rotaţie (25 msec) împărţit la 2 (12,5 msec). Dacă fişierul de manevră pentru swaping se găseşte pe un disc folosit partajat în multiprogramare atunci timpul de poziţionare este

$$t_p = 40 + 12{,}5 = 52{,}5 \text{ msec.}$$

Deoarece frecvenţa poziţionărilor pe FISM este mai mare decît pe bibliotecă sau alte fişiere acest timp se reduce şi 50 msec este acoperitor.

Timpul de transfer efectiv pentru o lungime medie a programelor de 50 Ko şi un debit disc = 315 K/sec este:

$$t_t = \frac{50\ K}{315\ K/sec} = 160 \text{ ms}$$

Rezultă timpul de swaping:

$$t_s = 2\ t_d = 2\ (t_p + t_t) = 2\ (160 + 50) = 420 \text{ msec.}$$

Timpul de prelucrare mediu va fi :

$$C = t_c + t_a + t_s = 350 + 5 + 420 = 775 \text{ msec.}$$

Se observă ponderea redusă a timpului auxiliar t_a şi importanţa mare a timpului de swaping, care se reduce la utilizarea unor discuri rapide şi creşte la cele lente. La utilizarea unor discuri cu debit de 120 K (DIAM) t_t = 415 msec

$$t_s = 2\ (415 + 50) = 930 \text{ msec}$$

$$C = 350 + 5 + 930 = 1285 \text{ msec}$$

Pentru un timp de gîndire mediu al utilizatorilor de T=10 sec (tipic considerat în literatura) rezultă parametrul r din diagrama din fig. 4.17.

$$r = \frac{C}{T} = \frac{0{,}775}{10} = 0{,}08 \text{ pentru discuri MD50 (D=315 K/sec)}$$

$$r = \frac{C}{T} = \frac{1{,}285}{10} = 0{,}13 \text{ pentru discuri DIAM (D=120 K/sec)}$$

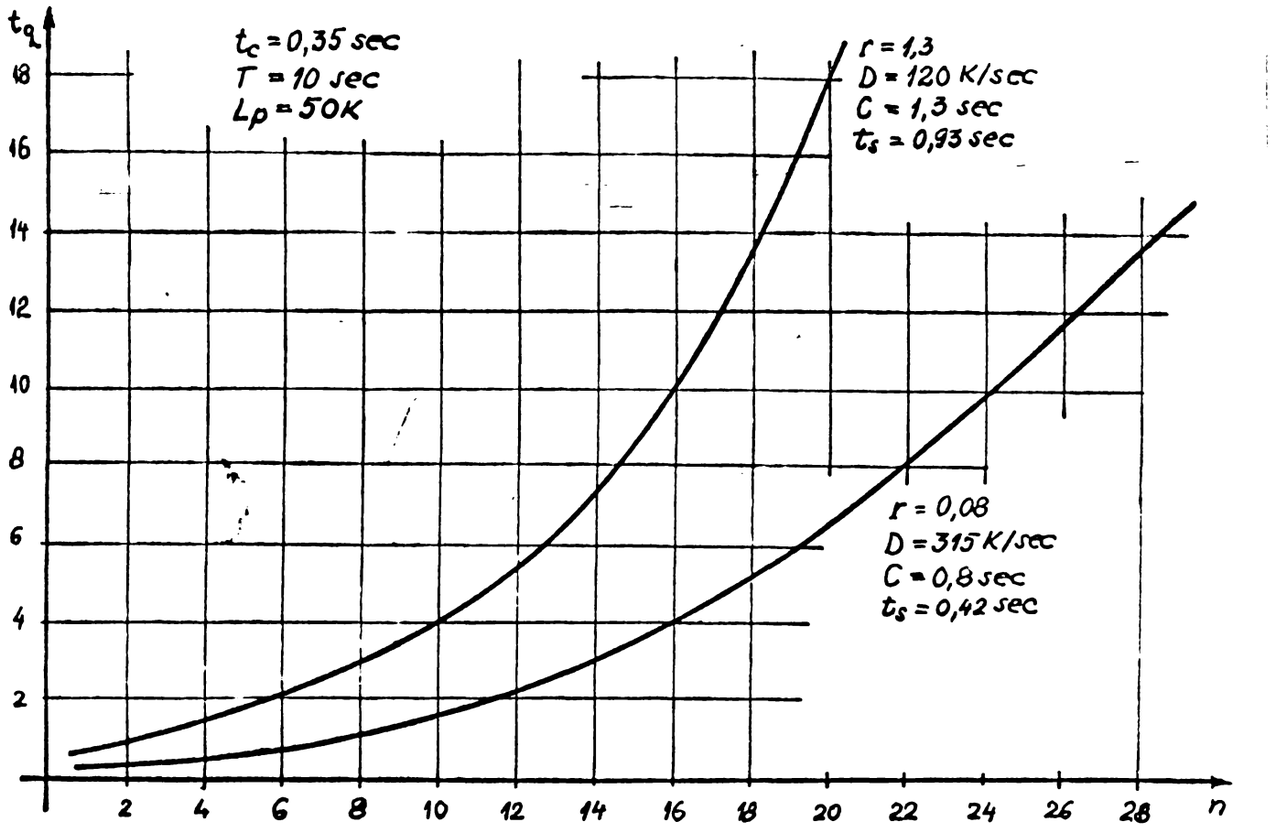


Fig. 4.18. Variaţia timpului de răspuns t_q funcţie de numărul de utilizatori n.

Din diagrama din fig. 4.17 rezultă pentru 20 utilizatori conectaţi (n = 20) un timp mediu de răspuns (aşteptare).

$t_q = Y.C = 7 \cdot 0,8 = 5,6$ sec. pentru D = 315 K/sec.

$t_q = Y.C = 12 \cdot 1,3 = 15,6$ sec. pentru D = 120 K/sec.

In fig. 4.18 s-a trasat variaţia lui t_q funcţie de numărul de utilizatori n.

Din cele de mai sus rezultă că în condiţiile enunţate subsistemul SCOT asigură un timp de răspuns mediu bun (5,6 sec) pentru 20 de utilizatori activi, dar el este prea mare pentru discuri cu debit 120 K/sec.

Pentru 25, 30 şi 40 de utilizatori activi timpul mediu de răspuns este

$t_q = Y \cdot C = 12 \cdot 0,8 = 9,6$ sec. pt. n = 25

$t_q = Y \cdot C = 18 \cdot 0,8 = 14$ sec. pt. n = 30

$t_q = Y \cdot C = 25 \cdot 0,8 = 20$ sec. pt. n = 40

Timpul de răspuns este deja nesatisfăcător pentru n=30.

Imbunătăţirea timpului de răspuns se poate obţine prin utilizarea unor unităţi centrale mai rapide. Totuşi din cazul analizat rezultă că timpul de calcul reprezintă sub 50 % din timpul de prelucrare pentru D = 315 K/sec şi abia 25 % la utilizarea unor discuri cu debit D = 120 K/sec. Timpul de poziţionare a discului reprezintă 25 % din timpul de swaping şi el nu poate fi simţitor îmbunătăţit.

Raportul luat între timpul de calcul t_c şi timpul de swaping t_s asigură o utilizare a UC în multiacces de cca. 50 %. Restul de 50 % poate fi utilizat în partiţiile ce lucrează în loturi sau într-o altă partiţie cu multiacces. Rezultă că pentru a utiliza simultan în multiacces mai mult de 25 de terminale se recomandă folosirea a două partiţii care lucrează în multiacces, între care să se împartă în mod egal terminalele active. In acest fel se pot deservi cu un timp de răspuns sub 10 sec maxim 50 utilizatori.

Dacă faţă de cazul considerat se introduc mai multe date de către anumiţi utilizatori, creşte timpul mediu de gîndire T peste 10 sec şi timpul de răspuns mediu se reduce. Acelaşi rezultat se obţine dacă prelucrarea dintre două I/E este mai simplă şi necesită un număr mediu de instrucţii executate mai redus decît 100.000.

4.5.3. Soluţii pentru divizarea timpului între utilizatori.

In cele prezentate mai sus s-a analizat comportarea subsistemului în regim stabil, considerînd timpul mediu de calcul utilizat între două operaţii de intrare/ieşire t_c = 350 msec, timpul mediu de gîndire T = 10 sec şi lungimea medie a programelor de 50 K. Se asigură astfel deservirea în condiţii bune a 25 utilizatori într-o partiţie sau 50 dacă se folosesc două partiţii în care rulează subsistemul.

Dacă anumite programe efectuează calcule complexe iterative, timpul de calcul poate depăşi frecvent, cu mult valoarea medie avută în vedere. In acest caz dacă subsistemul este încărcat spre limită cu terminale active, se perturbă grav funcţionarea stabilă şi creşte inadmisibil timpul de răspuns la celelalte terminale. Pentru a preîntîmpina aceste efecte se trece la divizarea forţată a timpului pentru programele care depăşesc timpul mediu de calcul t_u. Programul activ este pus într-un şir de aşteptare şi se activează numai după ce au fost deservite restul terminalelor sau dacă nici un program nu este pregătit.

Utilizarea algoritmului "Carusel multiplu" reduce mult timpul de swaping, cînd se fac întreruperi de divizare a timpului,

reducînd-se astfel timpul de răspuns. După fiecare întrerupere se micşorează prioritatea programului şi se dublează cuanta de timp (vezi 1.1.3.). Programul va fi lansat după două cicluri de prelucrare cu o cuantă de timp $2t_u$, apoi după 3 cicluri cu o cuantă $4t_u$, ş.a.m.d. Pentru implementarea acestui algoritm se poate folosi un fir de aşteptare unic de tip FIFO (first input first output) în care pentru fiecare program se va specifica prioritatea, lungimea cuantei de timp folosit şi un contor de prioritate, care se decrementează la fiecare parcurgere a şirului. Vor fi lansate programele pentru care contorul de prioritate după decrementare este zero. Dacă după expirarea noii cuante, programul nu a ajuns la o operaţie de I/E (sfîrşit de cerere), i se va mări cu 1 prioritatea, valoarea se copiază în contorul de prioritate şi se dublează cuanta. Prin această divizare a timpului se măreşte numai timpul de răspuns a programelor care efectuează calcule complexe. Această mărire este însă mult redusă faţă de cazul diviziării simple a timpului (Carusel simplu).

In perioadele în care numărul de terminale active este mai redus se recomandă modificarea dinamică (prin comandă de la consola centrală) a cuantei de timp utilizată, funcţie de numărul de terminale active şi de timpul mediu de gîndire estimat. Acest ultim parametru variază funcţie de volumul de date prelucrat şi de îndemînarea utilizatorilor. Se va ţine seama şi de cota minimă de timp de UC rezervată partiţiilor ce lucrează în serie în multiprogramare cu subsistemul. In calculele anterioare această cotă minimă a fost:

$$X = 1 - \frac{tc + ta}{tc + ta + ts} = 1 - \frac{tc + ta}{C} = 1 - \frac{355}{775} \approx 0,5$$

La o încărcare maximă, la lucrul în multiacces în două partiţii, cu 50 de terminale, x tinde spre zero.

Gradul de încărcare a unităţii de legătură a discurilor magnetice, datorată numai operaţiilor de swaping este:

$$g = \frac{ts}{C} = \frac{420}{775} \approx 0.5$$

Aceasta impune o limitare inferioară a cuantei de timp tu la 300 - 400 msec. Altfel se ajunge la o încărcare inadmisibilă a unităţii de legătură a discurilor, care împiedică utilizarea UC în perioadele de swaping de către programele ce se execută în partiţii serie. Se reduc astfel performanţele globale ale sistemului de calcul.

Calculul cuantei optime de timp o facem plecînd de la următorul raţionament. Dacă avem un timp mediu de gîndire T, atunci pe această perioadă este probabil ca din cei n utilizatori, n/2 să fie

în faza de gîndire şi n/2 în aşteptare. Timpul T de UC se va împărţi între cei n/2 utilizatori în aşteptare. În mod egal şi programele din alte partiţii în timpul operaţiilor de swaping (t_s). Fiind n/2 programe care trebuie activate, în timpul T vor avea loc n/2 operaţii de swaping, restul timpului se împarte între utilizatori. Cuanta optimă de timp va fi :

$$t_u = \frac{T - (n/2)\, t_s}{(n/2)} = \frac{2T}{n} - t_s$$

Formula de calcul este foarte simplă şi e verificată de calculele de probabilitate făcute anterior. Pentru 25 de terminale active şi T = 10 sec obţinem

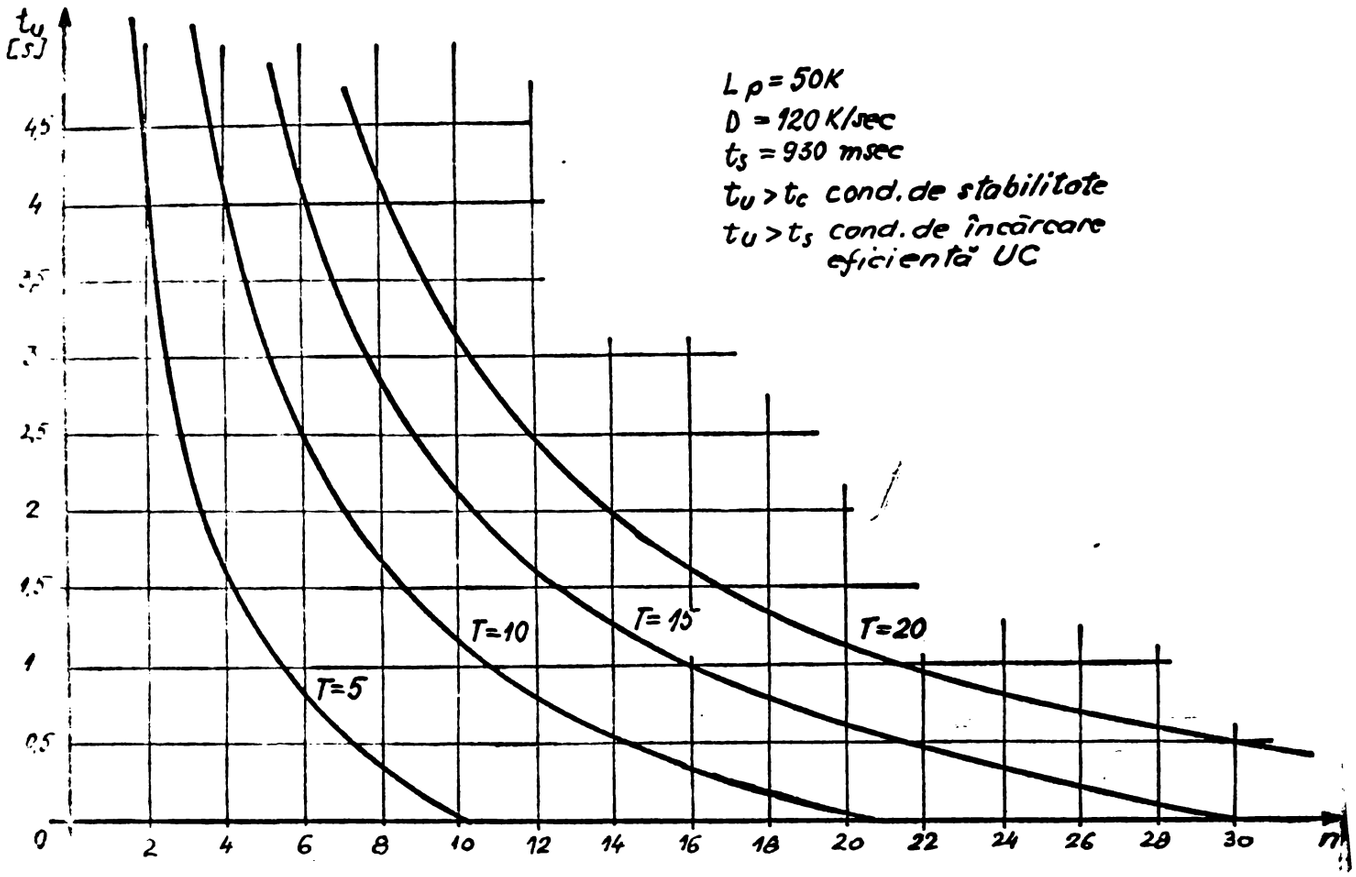
$$t_u = \frac{2 \cdot 10}{25} - 0{,}42 = 0{,}8 - 0{,}42 = 0{,}38 = 380 \text{ msec.}$$

La n = 25 a fost limita încărcării maxime în regim stabil a sistemului, pentru t_c = 350 şi timpul de răspuns t_q = 8 sec < T, conform diagramei din fig. 4.17. Prin acest calcul se elimină începerea divizării timpului în perioade în care numărul de utilizatori este redus şi o creştere medie a timpului de calcul peste valoarea prevăzută iniţial (350 msec), nu periclitează funcţionarea stabilă a subsistemului. In fig. 4.19 se prezintă diagrama variaţiei cuantei optime de timp funcţie de numărul utilizatorilor activi în sistem, pentru programe de lungime medie L_p = 50 Ko şi pentru utilizarea discurilor cu debit D = 315 Ko/sec şi D = 120 Ko/sec. Sistemul îşi menţine stabilitatea pentru un timp mediu de calcul t_c mai mic ca t_u. Utilizarea eficientă a sistemului se obţine pentru t_u mai mare decît timpul mediu de swaping t_s, cînd UC e folosită în proporţie de peste 50 % şi unitatea de legătură a discurilor ocupată sub 50 pentru swaping. Aceasta se poate exprima şi faţă de timpul mediu de prelucrare C prin $t_u/C > 0{,}5$. Diagramele exprimă clar importanţa debitului unităţii de discuri şi a timpului mediu de gîndire a utilizatorilor în limitarea numărului n de utilizatori deserviţi de sistem.

Pentru N = 15 şi T = 10 obţinem

$$t_u = \frac{2T}{n} - t_s = \frac{20}{15} - 0.42 = 1{,}3 - 0{,}42 = 0{,}88 \text{ sec.}$$

In acest caz limita începerii divizării timpului s-a dublat. Dacă s-ar fi păstrat cuanta t_u = 350 msec şi toţi cei 10 utilizatori ar fi avut timpul de calcul t_c = 800 msec între două operaţii de I/E, ar fi avut loc două operaţii inutile de swaping la divizarea



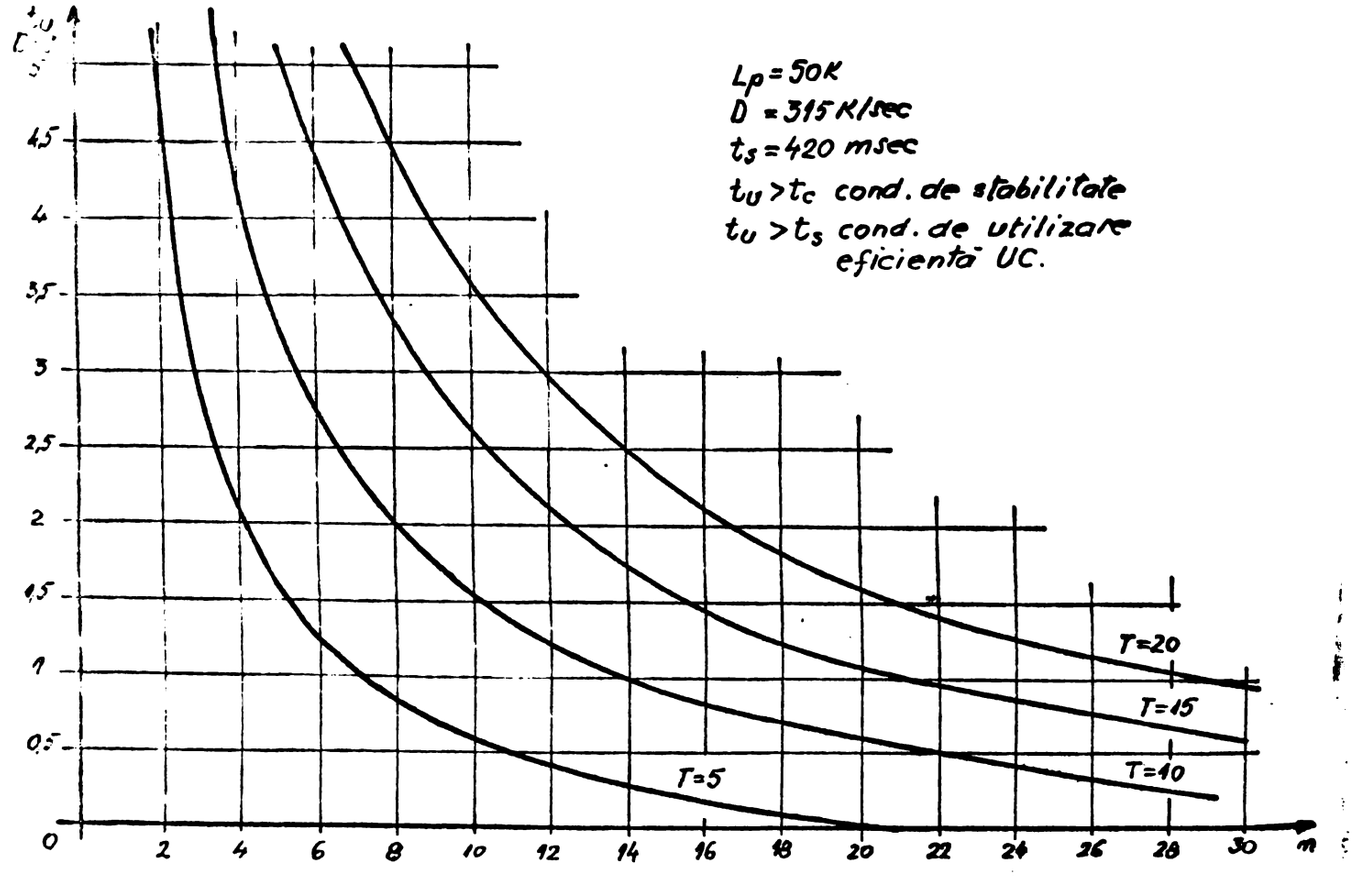


Fig. 4.19. Diagrama variației cuantei optime de timp t_u funcție de numărul n al utilizatorilor.

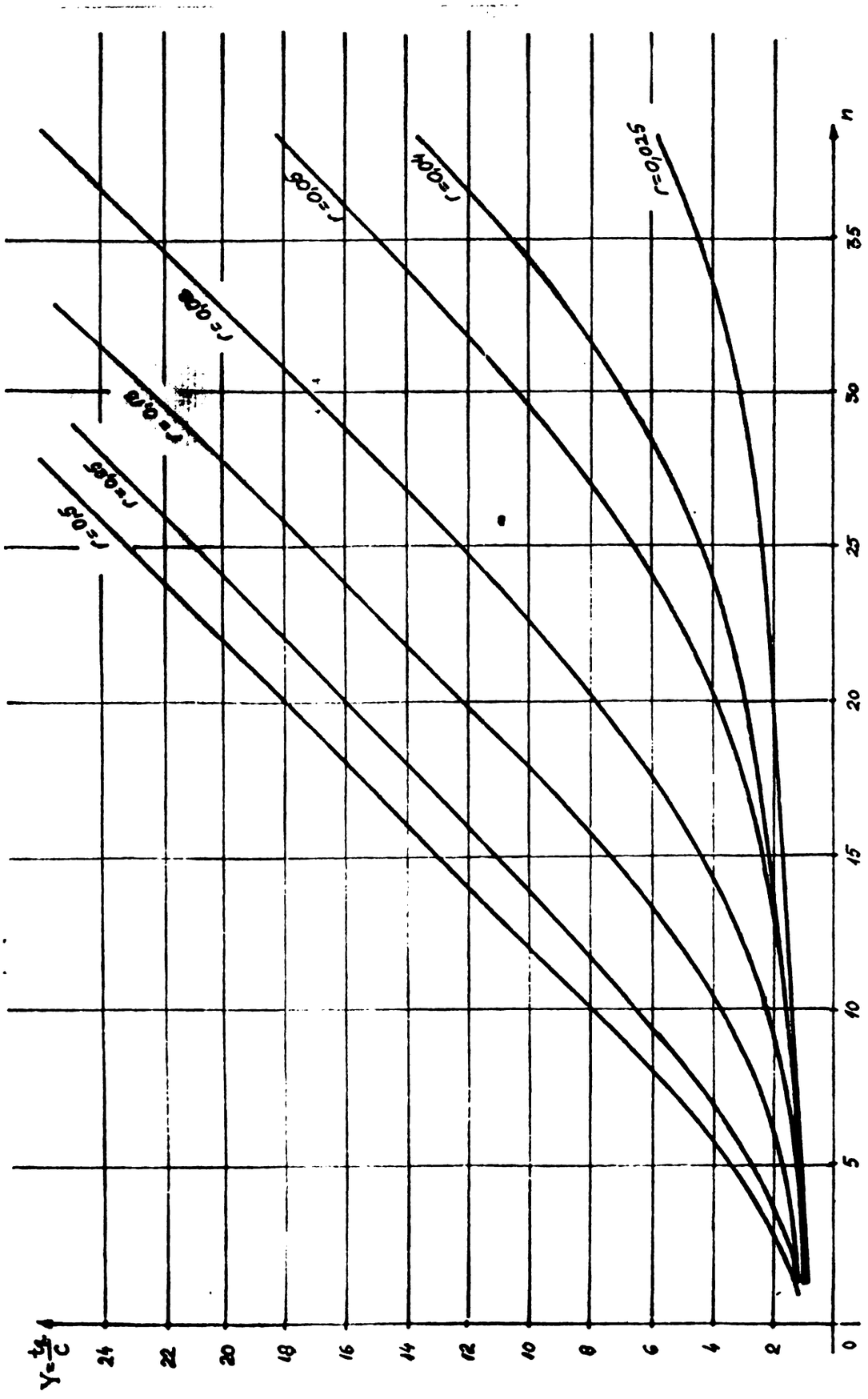


Fig.4.20. Diagrama timpului de răspuns normalizat

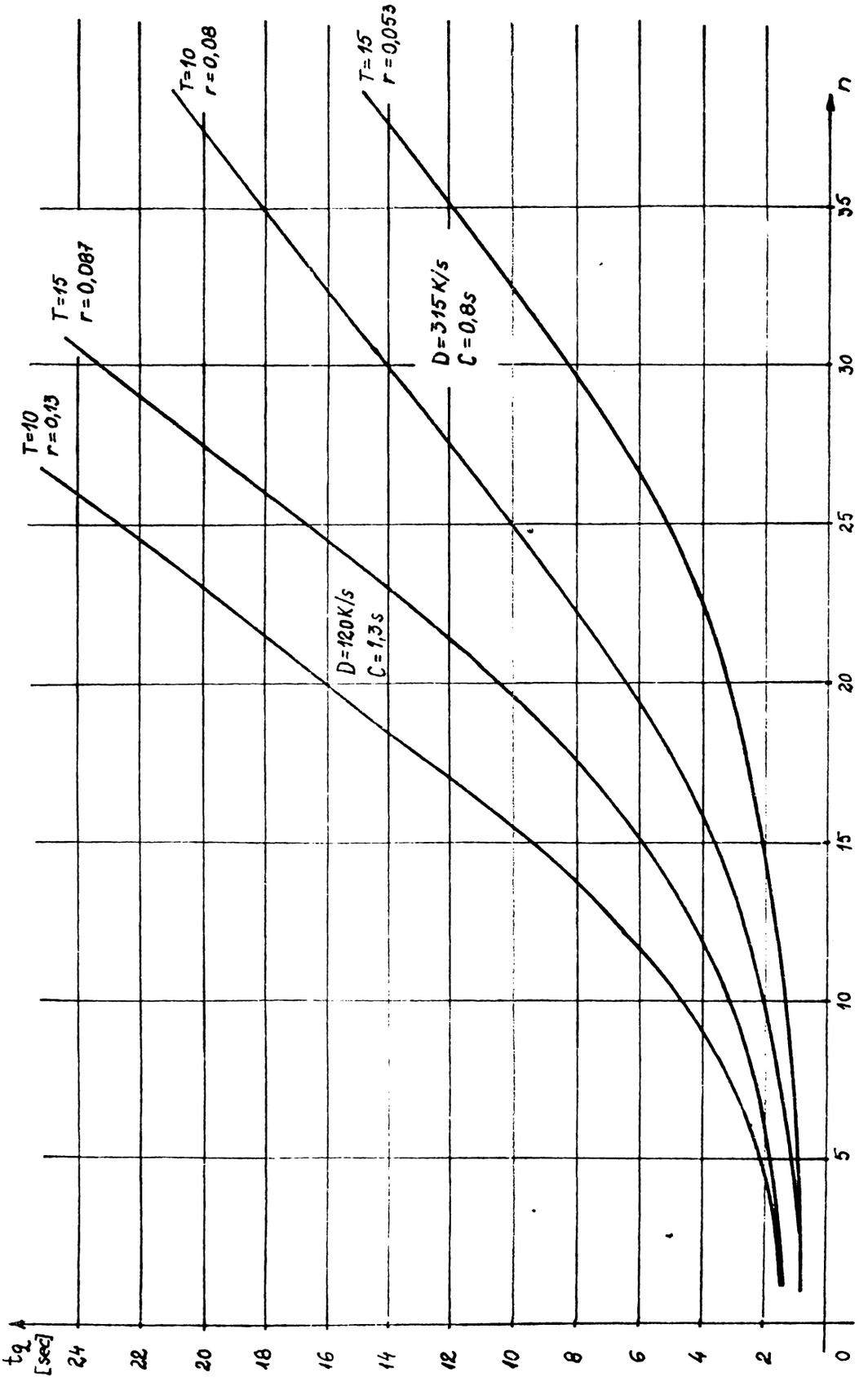


Fig.4.21. Diagrama timpului de răspuns

prin algoritmul "Carusel simplu" şi una la algoritmul "Carusel multiplu". Timpul mediu de prelucrare va fi:

$C = t_c + 3\,t_s = 0{,}80 + 3.0{,}42 = 2{,}06$ sec (Carusel simplu)
$C = t_c + 2\,t_s = 0{,}80 + 2.0{,}42 = 1{,}54$ sec (Carusel multiplu)
$C = t_c + t_s = 0{,}8 + 0{,}42 = 1{,}22$ sec (cuantă variabilă)

Deoarece variabila C s-a dublat la o cuantă t_u = 350 msec şi algoritm carusel simplu, apare o instabilitate timpurie în funcţionarea subsistemului pentru valori mici a lui N, care aparent nu se justifică. Calculul dinamic al unei cuante de timp optime este esenţial dacă se aplică divizarea timpului între utilizatori, pentru o utilizare flexibilă a subsistemului. Practic se permite dublarea timpului mediu de calcul t_c, dacă numărul maxim de utilizatori se reduc la jumătate. Cuanta calculată poate fi comparată cu timpul mediu de calcule estimat, pentru limitarea numărului de utilizatori.

La întreruperea unui program pentru care a expirat cuanta de timp afectată, dacă se utilizează fişiere, nu se permite depunerea sa pe disc pînă nu s-au terminat toate transferurile lansate. In caz contrar zonele tampon rămîn incomplete, iar anumite blocuri scrise pe fişiere pot fi greşite. Aceasta presupune verificarea tuturor blocurilor de comandă asociate fişierelor. Cînd toate sînt libere, operaţiile cu fişierele sînt terminate. In SCOT aceasta se face pornind de la repertoarul de fişiere (MAPFIS), unde se caută adresele TDF-urilor fişierelor deschise. Pentru acestea în TDF+36 se găseşte adresa primului bloc de comandă utilizat. Dacă programul utilizează fişiere nedefinite prin subprograme ASSIGN, blocurile de comandă trebuie evidenţiate separat.

4.6. INTERFAŢA INTRE SUBSISTEM SI PROGRAMELE UTILIZATOR.

Subsistemul SCOT admite pentru execuţie interactivă în multiacces programe complexe, de mari dimensiuni, care pot utiliza fişiere de orice tip de organizare şi pot fi segmentate. Testarea unor asemenea programe este posibilă în bune condiţii numai în prelucrarea în loturi, care permite utilizarea facilităţilor de depanare puse la dispoziţie de sistemul de operare (liste de erori, lista modulelor obiect, lista tabelelor de variabile locale, referinţe externe şi încrucişate, vidaje de memorie, etc). Timpul de rulare în timpul testării este scurt şi detectarea erorilor o face utilizatorul pe baza listingului primit. Corectarea erorilor se face direct în pachetul de cartele sursă sau folosind serviciile programului bibliotecar.

Programul poate fi scris modular în diferite limbaje de programare FORTRAN, COBOL cu subprograme ASSIRIS. Pentru programele mai simple se poate folosi pentru testare conversaţională subsistemul ARIEL. Crearea programelor sursă complexe conversaţional şi testarea lor în acest mod este laborioasă şi costisitoare. După testarea programelor şi catalogarea lor în biblioteci de format IMT, ele pot fi exploatate interactiv de mai mulţi utilizatori simultan prin subsistemul SCOT.

Pentru a putea lucra sub comanda subsistemului este necesar să existe o interfaţă între program şi acesta. La concepţia acestei interfeţe s-a plecat de la următoarele principii :

- utilizatorul nu trebuie să cunoască modul de lucru al unui sistem de teletransmisie;
- utilizatorul cunoaşte numai limbajele de nivel înalt uzual FORTRAN sau COBOL;
- utilizatorul nu trebuie să modifice programul sursă pentru a putea să-l exploateze în multiacces interactiv;
- nu se vor introduce instrucţii noi în limbajele de programare;
- dialogul dintre subsistem şi utilizator ca şi lansarea subsistemului vor fi foarte simple.

Pentru realizarea acestor deziderate s-a adoptat următoarea soluţie. Toate operaţiile de citire de cartele şi de tipărire la imprimantă au fost redirectate spre subsistem. Aceasta s-a făcut prin rescrierea sau completarea unor subprograme utilizate de compilatoarele FORTRAN şi COBOL. Pentru limbajul FORTRAN s-au modificat în principal următoarele module I%DATA, I%READ, I%PRINT, I%ERR1, I%STOP şi I% INIT. Modul de comunicare cu subsistemul a subprogramelor I%READ şi I%PRINT este la nivel de cartelă respectiv linie, aşa cum s-a descris în lucrare (vezi 4.1). Modulul I%ERR1 transmite subsistemului mesajele de eroare pentru erorile detectate în subprogramele proprii FORTRAN (ALOG, SQRT, FMMAT, etc).

Subprogramul I% STOP va anunţa subsistemul de terminarea programului, permiţînd astfel continuarea dialogului.

S-a dat subprogramelor modificate aceleaşi nume cu cele originale, pentru a putea fi direct referite de compilatorul FORTRAN. Zona de comunicare cu subsistemul este reprezentată de două cuvinte la sfîrşitul tabelei de segmente unde se memorează adresele de revenire. Adresele zonelor de date transmise se dau prin registre. Subprogramele modificate sînt catalogate într-o bibliotecă de subprograme de format RSL, pe cînd cele originale sînt în biblioteca

standard de subprograme (BSS) a sistemului.

Cînd se lucrează în prelucrarea în loturi, la editarea legăturilor vor fi încărcate toate subprogramele referite de compilator din BSS şi se va genera un program IMT pentru prelucrare normală. La terminarea testării programului, pentru catalogarea lui în format IMT pentru lucru conversaţional, se va plasa înainte de LINK o cartelă LIB care va referi biblioteca RSL cu module modificate. Conform regulii, se vor căuta de către editorul de legături subprogramele referite întîi în această bibliotecă şi vor fi adăugate programului. Subprogramele negăsite în această bibliotecă sînt cele nemodificate şi vor fi căutate şi încărcate ulterior din BSS. Programul astfel modificat va fi asamblat de editorul de legături şi depus în biblioteca IMT. El va avea interfaţa cu subsistemul SCOT şi va putea fi încărcat şi lansat de către acesta pentru a lucra în multiacces. Trebuie remarcat că subsistemul SCOT poate încărca orice program IMT catalogat şi îl lansează în execuţie. Ne avînd interfaţă de comunicaţie nu poate să îi asigure date. La catalogare se vor prevedea cartele de comandă pentru toate fişierele utilizate, plasate înainte de LINK.

Se remarcă simplitatea operaţiei de catalogare modificată a programului de către utilizator. El trebuie să pună o cartelă LIB înainte de cartela LINK. Toate instrucţiile de tip READ (105,...)vor cere de la utilizator prin subsistem o linie de date. Pentru a fi sesizat utilizatorul de această cerere se afişează pe terminal cuvîntul " DATE ". Utilizatorul ştie în ce ordine şi în ce format se vor introduce datele în programul său (imagine cartele).

La întîlnirea unor instrucţii de tipul WRITE (108,...) prin subsistem se vor afişa pe terminal liniile de date ce trebuiau scrise la imprimantă, ca o imagine a listingului. Pentru a face dialogul mai inteligibil, utilizatorul poate introduce în program tipărirea unor linii de comentarii. Acestea vor fi afişate pe terminal dînd indicaţii asupra datelor ce vor trebui introduse, sau va indica faza în care a ajuns programul la un moment dat. Utilizatorul poate prevedea în program prin READ cererea unor parametri , pe baza cărora să modifice dinamic mersul programului (alegerea metodei de proiectare, a variantei etc). La fel se pot afişa prin WRITE rezultate intermediare, care să indice evoluţia calculului. Utilizatorul poate interveni pentru a opri execuţia prin apăsarea tastei BRK sau la sfîrşitul afişării

unui ecran cînd subsistemul întreabă " CONTINUATI ? ". Dacă se răspunde NU, programul se termină şi se permite relansarea lui cu alte date (vezi 4.2.). Erorile de FORMAT se pot corecta imediat prin reintroducerea ultimei linii. Funcţiile celorlalte instrucţii READ şi WRITE, pentru alte numere de periferice se menţin nemodificate. Programele conversaţionale sînt compatibile la nivel sursă cu prelucrarea în loturi.

4.7. EXPLOATAREA SUBSISTEMULUI

Comunicarea între utilizator şi subsistem este foarte simplă şi se învaţă în mai puţin de o oră în faţa terminalului. Activarea terminalului, după punerea sub tensiune şi pregătirea sa, se face apăsînd tasta BRK. Apare mesajul de prezentare al subsistemului. Se poate cere afişarea unor instrucţiuni concise de utilizare, dacă nu se cunoaşte modul de lucru. Se cer apoi datele utilizatorului, numele bibliotecii şi a programelor utilizate. Programul indicat este lansat în execuţie, se cer date de la terminal şi se afişează pe rînd rezultatele obţinute. La terminarea programului se poate lansa altul ş.a.m.d. pînă la sfîrşitul şedinţei. Dacă se dispune de imprimantă de recopiere a ecranului, anumite rezultate se pot tipări. In continuare se dă un exemplu de dialog la terminal pentru execuţia conversaţională a unui program de proiectare a unor aparate de măsură magnetoelectrice.

```
***  SISTEM CONVERSATIONAL ECOT V01  ***

PUTETI UTILIZA 128 K MEMORIE
CUNOASTETI MODUL DE LUCRU ? (Y,N) Y
DATA=21/09/84   ORA=14H 23M 57S
VA ROGAM SA NE INDICATI
CONTUL:COSY   NUMELE:MIKE
VA ROG DATELE ASUPRA BIBLIOTECII
LN:BIBRUTI   CH:0001   VN:01   DV:RD3
DATE CORECTE ?(Y,N)  Y
PUTETI CERE IN AVANS MAXIMUM 5 PROGRAME
VA ROG NUMELE PROGRAMELOR UTILIZATE
PN:PROGRAM1  UN:02
```

```
MAI DORITI PROGRAME DIN BIBLIOTECA ?(Y,N) N
PROGRAM PROGRAMI STARTED

PROGRAM DE PROIECTARE APARATE MAGNETOELECTRICE
CARE TIP DE APARAT DORITI SA-L PROIECTATI ?
DATE: M10
VARIANTA (AMPERI)
DATE: 0.0001,
INDUCTIA IN INTREFIER SOLICITATA
DATE: 0.168,
RAZA PIVOTULUI
DATE: 0.016,
RAZA LAGARULUI
DATE: 0.08,

ERROR :FORMAT (ADRESSE:0262B8) CARACTER ILLEGAL DANS
LA CHAINE D'ENTREE
DATE: 0.05,
    APARAT TIP M10
VARIANTA STAS 4640/1974
CURENT CADRU MOBIL 0.00010 AMPERI
DIAM.CONDUCTOR BOBINARE 0.02000 MM
NUMAR SPIRE 874
GREUTATE ECHIP. MOBIL 0.57800 GR
CUPLU SPECIFIC 24.89300 MEXCM
CADERE TENSIUNE 0.47900 V
FACTOR DE CALITATE 0.7175
COEFICIENT TOTAL DE AMORTIZARE 0.16008-5 NE/RAD
TIP DE STABILIZARE 03.26 SNC
CLASA DE PRECIZIE 1.45000

PROGRAM PROGRAMI TERMINATED
CUM DORITI SA CONTINUATI ? (R,C,S) S
MAI DORITI O LUCRARE ?(Y,N) N
    *** SCOT Vol - STEP TERMINATED ***
        LA REVEDERE
```

5. C O N C L U Z I I

Domeniul sistemelor de calcul cu multiacces este de mare actualitate, ţinînd cont de dezvoltarea teleprelucrării datelor în întreaga lume. Problemele care apar sînt de mare complexitate şi se pot rezolva prin soluţii combinate aplicate în proiectarea structurii calculatoarelor şi a sistemelor de operare. Soluţiile propuse pentru realizarea sistemelor cu multiacces sînt foarte variate şi ele privesc implementarea unor subsisteme de operare specializate pe calculatoarele concepute pentru lucrul în multiprogramare, sau proiectarea unor sisteme de calcul noi cu posibilităţi largi de multiprogramare şi multiacces. Acestea din urmă se bazează pe conceptul de memorie virtuală şi gestionarea memoriei şi folosesc metode perfecţionate de protecţie a informaţiilor din sistem.

Ca rezultat al cercetărilor şi experimentărilor efectuate de autor în ultimii 8 ani, în prezenta lucrare se aduc contribuţii teoretice privind sistemele de operare cu multiacces în general şi contribuţii concrete în concepţia şi implementarea unui subsistem cu multiacces pentru execuţia interactivă a programelor catalogate în biblioteci în format LLF.

A. Contribuţiile teoretice ale autorului sînt următoarele :

1. Analiza metodelor utilizate în sistemele de operare cu multiprogramare pentru gestiunea memoriei centrale, a unităţii centrale, a dispozitivelor periferice, a şirurilor de lucrări şi a fişierelor sistem şi evidenţierea posibilităţilor oferite şi a limitărilor impuse de aceste metode de lucru în implementarea unor facilităţi de lucru în multiacces.

2. Se propun soluţii de dezvoltare a limbajului de comandă, prin subcomenzi noi de comenzi specializate pentru lucrul în multiacces, implementate în subsistemul de operare, care lucrează în contextul multiprogramării complexe şi realizează partajarea resurselor unei partiţii între mai mulţi utilizatori ce folosesc terminale conversaţionale pentru dialogul cu calculatorul.

3. Se realizează o clasificare sistematică a subsistemelor de operare cu multiacces existente pe baza scopului urmărit, funcţiilor realizate, metodelor aplicate, performanţelor atinse şi limitărilor impuse. Se prezintă pe scurt caracteristicile fiecărei clase, cauzele limitărilor existente.

4. Se prezintă o metodă originală de programare concurentă în limbaj de asamblare, folosită în scrierea subsistemelor cu multiacces.

5. Se propun facilităţi pe care să le ofere sistemele de operare, pentru a se putea implementa subsisteme cu multiacces cu performanţe superioare :

- compilatoare şi alte programe de aplicaţii reentrante apelabile prin macroinstrucţii din programele utilizator (bibliotecar, editor de legături, etc).
- macroinstrucţie pentru încărcarea unor programe din biblioteci IMF specificate prin nume simbolic.
- macroinstrucţie de scriere articole în fişierul de contabilitate al sistemului.
- set extins de macroinstrucţii de multitasking.
- generarea dinamică a unor interfeţe BCF (zonă de informaţii generale, rezumat BCF, modul de legătură BCF, zonă de reentranţă BCF).
- macroinstrucţii pentru generarea dinamică a tabelelor de descriere a fişierelor, a articolelor de FIDCM şi încărcarea dinamică a modulelor de acces la fişiere.
- macroinstrucţii mai flexibile de comunicare între partiţii.
- macroinstrucţii de sincronizare a proceselor concurente prin semafoare.

6. Se analizează posibilităţile de utilizare a limbajelor concurente în proiectarea subsistemelor cu multiacces şi limitările ce apar în performanţele produselor realizate.

7. Se demonstrează necesitatea gestiunii memoriei centrale prin conceptul de memorie virtuală şi necesitatea unor noi tehnici de protecţie a informaţiei la nivel de segment de program, pentru realizarea unor sisteme cu multiacces puternice şi flexibile, capabile să asigure protecţia individuală a programelor utilizator.

8. Se analizează comparativ cele trei metode de virtualizare a memoriei (paginare, segmentare, segmentare şi paginare) soluţiile de implementare din punctul de vedere performanţe/cost şi se recomandă pentru calculatoarele de capacitate medie metoda segmentării simple.

9. Se proiectează o structură de memorie virtuală cu segmentare, care necesită un volum redus de circuite, printr-o organizare originală a informaţiilor de control.

10. Se propune un algoritm simplu şi eficient de înlocuire a segmentelor în memoria centrală, superior algoritmilor folosiţi curent care reduce traficul de segmente între memoria centrală şi cea externă, folosind timpul de dezactivare a segmentelor.

11. S-a proiectat o structură de tabele "software" originală care asigură o protecţie individuală a programelor la nivel de segment, controlează drepturile de acces ale proceselor la segmente, permite partajarea unor segmente de program între procese, gestionează dinamic memoria centrală şi memoria virtuală.

12. Se propune o metodă originală de implementare a inelelor de protecţie şi de control a drepturilor de acces prin "hardware" în timpul execuţiei utilizând cîmpuri speciale în memoria rapidă de translaţie a adreselor virtuale.

13. Se analizează translatarea adreselor virtuale utilizate de programele de canal, care se realizează în general prin "software" în condiţii foarte restrictive. Se propune o soluţie de translatare a acestor adrese prin acelaşi mecanism ca pentru adresele din programele UC, asigurîndu-se astfel o mare flexibilitate şi un control complet al accesului la UC.

B. Prin proiectarea şi realizarea unui subsistem pentru execuţia interactivă a programelor, se rezolvă într-un mod original multe probleme incomplet soluţionate, privind implementarea unor astfel de subsisteme pe calculatoarele de capacitate medie lucrînd în multiprogramare. Contribuţiile concrete experimentale în acest domeniu sînt următoarele :

1. Concepţia originală globală a subsistemului SCOT.
2. Modul de realizare a dialogului în multiacces cu utilizatorii.
3. Soluţia de încărcare dinamică a programelor utilizator.
4. Metoda de tratare a informaţiilor SGF necesare programelor.
5. Transmiterea informaţiilor SGF spre programele utilizator numai în momentul utilizării lor în modulele OPSK/CLOSE.
6. Posibilitatea de folosire în programele utilizator a tuturor tipurilor de fişiere acceptate de SGF şi compatibilitatea lor cu fişierele create în prelucrarea în loturi.
7. Respectarea tuturor protecţiilor sistemului de operare, care oferă o mare siguranţă în exploatarea fişierelor şi programelor.
8. Utilizarea eficientă a spaţiului de memorie centrală, prin reducerea la maxim a dimensiunii subsistemului (20 K) faţă de altele similare (60 - 150 K).

9. Realizarea întregului subsistem fără nici o modificare în sistemul de operare şi folosirea în exclusivitate a posibilităţilor normale de programare, conform manualelor de utilizare ; se asigură astfel compatibilitatea subsistemului cu versiunile ulterioare ale sistemului de operare.
10. Soluţionare simplă şi eficace a încărcării şi utilizării fără restricţii a programelor segmentate.
11. Modul de salvare şi încărcare a programelor şi a informaţiilor aferente, la punerea lor în aşteptare şi la reactivare.
12. Tratarea unitară a erorilor de diferite tipuri, asigurarea posibilităţii sistemului şi a protecţiei între programele utilizator care se execută în acelaşi timp.
13. Soluţia de comunicare între programele utilizator şi subsistem.
14. Asigurarea unui dialog simplu între subsistem şi utilizatori.
15. Posibilitatea de modificare rapidă a programelor testate în prelucrarea în loturi, pentru a fi lansate în execuţie în multiacces.
16. Compatibilitatea completă la nivel de program sursă (scris în limbajul- FORTRAN sau COBOL) între prelucrarea în loturi şi cea conversaţională. Se asigură astfel posibilitatea unor programatori neprofesionişti să scrie şi să testeze singuri programe, care vor putea fi executate interactiv.
17. Reducerea considerabilă a timpului de testare a programelor conversaţionale prin testarea în prelucrarea în loturi. Se reduce corespunzător preţul de cost al proiectelor, faţă de cazul testării conversaţionale sau a utilizării unor simulatoare de regim conversaţional.
18. Asigurarea independenţei programelor utilizator faţă de subsistem conferă flexibilitate în exploatare. Ele pot fi modificate oricînd în biblioteca din care fac parte, fără a afecta funcţionarea subsistemului.
19. Se propun soluţii eficiente de divizare echitabilă a timpului între utilizatori, care să asigure un timp de răspuns cît mai scurt, o utilizare intensivă a UC şi să reducă transferurile de programe între MC şi discul magnetic. S-a experimentat un algoritm de tipul "carusel multiplu", cu priorităţi şi cuante de timp variabile, implementat pe un şir de aşteptare unic, care măreşte stabilitatea subsistemului.

20. Se analizează performanţele subsistemului SCOT, implementat pe calculatorul FELIX C/256/512, folosind un model matematic bazat pe utilizarea teoriei firelor de aşteptare, adaptat pentru multiacces. Pe baza acestui model se calculează timpul de răspuns al subsistemului funcţie de numărul de utilizatori, de timpul mediu de gîndire, lungimea medie a programelor şi performanţele discurilor magnetice utilizate.
21. Se propune la divizarea timpului utilizarea unei cuante variabile de timp calculată dinamic, care să elimine apariţia instabilităţii subsistemului şi traficul inutil de programe cînd lucrează un număr redus de utilizatori, care utilizează timpi de calcul mai lungi. Se dă o formulă de calcul simplă a acestei cuante optime.
22. Se evidenţiază factorii care influenţează timpul de răspuns şi se dau criterii de determinare a limitei de funcţionare stabilă a sistemului. Condiţia de stabilitate cere ca timpul mediu de calcul să fie mai mic decît cuanta optimă de timp calculată.
23. Se propune ca limită a încărcării eficiente a sistemului momentul în care UC este încărcată sub 50 % şi unitatea de legătură a discurilor peste 50. Aceasta se exprimă prin condiţia ca timpul mediu de swaping să fie mai mic decît timpul mediu de calcul.
24. S-a realizat un sistem de teleprelucrare a datelor prin care se leagă Centrul de Calcul al institutului cu laboratoarele din facultăţile de Electrotehnică, Mecanică şi Construcţii.

Rezultatele teoretice şi experimentale prezentate în lucrare au condus la realizarea, în cadrul unor contracte de cercetare, a mai multor versiuni ale subsistemului cu multiacces SCOT, care funcţionează de mai mulţi ani la Centrul de Calcul Electronic al Institutului Politehnic "Traian Vuia" din Timişoara şi în alte centre de calcul. Aceste cercetări se încadrează în programul naţional de elaborare a unor noi produse program, pentru extinderea teleprelucrării datelor, introducerea largă a informaticii în întreprinderi, şi creşterea eficienţei economice pe această bază.

BIBLIOGRAFIE

- ANDREWS, G.R. Syncronising Resources, TOPLAS, vol.3, nr. 4, oct. 1981.

- ANDREWS, G.R. The Distributed Programming Language SR - Mechanisms, Design and Implementation, SPE, vol. 12, 1982.

- ADIBA, M. ş.a. Les sistèmes de gestion de bases de donnèe. Min. de coop. Franţa 1978.

- ATKINSON, M. Systems for large data Base. Ed. North - Holand, 1977

- xx ARIEL. Manual de utilizare, ICI, 1980.

- ASENJO, J.P., SRINI, V.P. Analysis of cray - 1S Architecture IEEE Computer Architecture, vol.11, nr.3, iunie 1983.

- AUDOUX, M.S. IMSIS - le projet communautaire de système intégré d'information interinstitutionnel, Bulletin INRIA, nr. 78, 1982

- BALTAC, V. Realizări şi perspective în electrotehnică, electronică, tehnica de calcul, ASC nr.41. Ed. tehnică, Bucureşti 1964

- BARTHET, C. Introduction aux utilisantions de la télé-informatique, Bulletin INRIA, nr.21, dec. 1975.

- BALTAC, V. ş.a. FELIX C-256. Structura şi programarea calculatorului, ed.tehn., Bucureşti, 1974.

- BALTAC, V. Optimizarea sistemelor de operare ale calculatoarelor numerice, Ed.Facla, Timişoara, 1974.

- BELADY, L.A. A Study of Replacement Algorithms for a Virtual - Storage Computer, SJ, vol.5, nr.2 1966.

- BRINCH HANSEN, P. The Nucleus of a Multiprogramming System, CACM, vol.13, nr.4, apr. 1970.

- BRINCH HANSEN, P. Operating System Principles, Ed.Prentice-Hall, Englewood Cliffs, 1973

- BRINCH HANSEN, P. The Programming Language Concurrent Pascal, TSE, vol.1, nr.2, iunie 1975.

- BENSOUSSAN, A. ş.a. The Multics Virtual Memory: Concepts and Design, CACM, vol.15, nr.5, mai 1972.

- BJARI, M, MATALI, A. An Approach to the Implementation of Distributed Processes, A 4-a Conferinţă internaţională de Automatică şi calculatoare, Bucureşti, 1981.

- BLATNY, I. ş.a., On the Optimization of Performance of Time-Sharing Systems by Simulation, CACM, vol.15, nr.6, iunie, 1972.

- BULGAKOV ,A.,ş.a. Unitate microprogramată implementează nivelele 1,2 ale protocolului X.25, a 4-a. CIAC, Bucureşti 1981.

- CIOCÎRLIE, H. ş.a. Limbajele de programare PASCAL, Ed. Facla, Timişoara, 1984.

- COX, G.W, CORMIE, W.M, LAI, K., POLLAK, F.J. Interprocess Communication and Processor Dispatching on the Intel 432, JCS, vol.1, nr. 1, febr.1983.

- COFFMAN, E.G, DENNING, P.J. Operating System Theory, Ed. Prentice - Hall, Englewood Cliffs, 1973.

- COSSERAT, D.C. A Data Model Based on the Capability Protection Mechanism, RAIRO, sept. 1975.

- CRISTEA, V. A Tool for Proving Correctness of Parallel Process Systems, A 4-a. C.I.A.C. Bucureşti, 1981.

- CIOCÎRLIE, H., ş.a. Limbajul Pascal concurent..., A 4-a. C.I.A.C., Bucureşti, 1981.

- CHAILLET, B., CABANEL, J.P. - Les réseaux locaux en 1983. Classification et perspectives, Bulletin INRIA, nr.89, 1983.

- DODESCU, Gh., ş.a. Limbajul Basic şi aplicaţii, Ed.did.şi ped. Bucureşti 1978.

- DODESCU, Gh. ş.a. Sisteme electronice de calcul şi teleprelucrare, Ed. did. şi ped., Bucureşti 1981.

- DODESCU, Gh. ş.a. Minicalculatoare, Aplicaţii, ed. tehnică, Bucureşti, 1977.

- DUGAN, R.J. System/370. A program view of the channel Subsyst IBM Computer Architecture, vol.11, nr.3,1983.

- DIACONESCU, S., Memory Management in HELIOS operating Syist a 4-a. CIAC, vol.4, Bucureşti 1981.

- DODESCU Gh., ş.a., Calculatoare electronice şi sisteme de ope rare, Ed.did.şi ped., Bucureşti 1974.

- DODESCU Gh., ş.a., Memoria virtuală în sistemele de calcul, AMC, vol.23, Bucureşti, 1978.

- DODESCU Gh. Modelarea sistemelor de operare, Ed.şt. şi enciclopedică, Bucureşti 1981.

- DAVIES, D.W., ş.a. Distributed Systems - Arhitecture and Implementation, ed. Springer, Berlin, 1981.

- DALEY, R.C, DENNIS, J.B. Virtual Memory, Processes and Sharing in MULTICS, CACM, vol.11, nr.5,mai 1968

- DIJKSTRA, E.W. The Structure of the Multiprogramming System, CACM, nr.5, vol. 11, mai 1968.

- DAVIES, D.W., BARBER, D.L. Reţele de interconectarea calculatoarelor, ed. tehnică, Bucureşti 1976.

- DRAGANESCU M. A doua revoluţie industrială, Microelectronica, automatica, informatica. Ed.tehnică,Bucureşti 1

- DIACONESCU, S. ş.a. Componente reentrante în sistemul de operare HELIOS, A 4-a. CIAC, Bucureşti 1981.

- DUMITRESCU, S, VASI, A. Transmisii de date de viteză mare folosind ca suport de transmisie a informaţiei fibrele optice, a 4-a. CIAC, Bucureşti, 1981.

- IRIAZ, M. ş.a. Rébus, un system distribué pour la conduite en temps réel des procédés industriels, Bulletin INRIA, nr. 89, 1983.

- x x Exploatarea tehnicii de calcul în regim de teleprelucrare, ICI, 1982.

- ENGLAND, D.M., Capability Concept Mechanism and Structure in System 250, RAIRO, septembrie 1975.

- FABRY, R.S. Capability - based Addressing, CACM, vol.17, nr.7, iulie 1974.

- FULLER, S.H. Minimal - Total Processing Time Drum and Disk Scheduling Disciplines, CACM, vol.17, nr.7, 1974.

- GEORGESCU, H., PREDESCU, P. Introducere în sistemul de operare SIRIS, Ed. Albatros, Bucureşti, 1978.

- GEORGESCU I. Sisteme de operare pentru calculatoarele numerice, Ed.tehn., Bucureşti, 1974.

- GOLD, M.M., Time-Sharing and Batch-Processing, CACM, vol.12, nr.5, mai 1969

- GORAN, M. ş.a. Software Structure for Computer Networcks, a 4-a. CIAC, Bucureşti, 1981.

- GIOVANNETTI, G., TUCCI, S. Evolution of Recovery Procedures for a Distributed Data System, a 4-a. CIAC, Buc. 1981

- GOODMAN, J.... Using cache memory to reduce processor-memory trafic, IEEE Computer Architecture, vol.11, nr.3 iunie 1983.

- GILOI, W.K., BAER, P. A design principle for advanced multicomputer architectures, IEEE Computer Architectures, vol.11, nr.3, iunie 1983.

- GIBA, M. Presentation generale du projet-pilote SOL, Bulletin INRIA, nr.77, 1982.

- HELLERMAN, H. Some Principles of Time-Sharing Scheduler Strategies IBM Systems Journal, vol.8, nr.2, 1969.

- HELLERMAN H. CONROY T.P., Computer Systems Performance, Mc Graw-Hill Kogakusha, 1975.

- x x HELIOS operating system, Macroinstrucţiuni utilizator, ITC, 1981.

- HOLT, R.C. Concurent Euclid, the Unix System and Tunis, Ed. Addison - Wesley, Reading (SUA), 1983.

- HARRISON, M.A., RUZZO, W.L., ULLMAN, J.D. Protection in operating Systems, CACM, vol.19, nr.8, august 1976.

- HILL, F. J., PETERSON, G.R. Calculatoare numerice - Hardware - structură şi proiectare, Ed.tehn.Bucureşti, 1980.

- HUITEMA, C. Les protocoles pour les satellites de telecommunication, Bulletin INRIA, nr. 76, 1982.

- HARRUS, G. Modelisation des reseaux locaux, Bulletin IRIA, nr. 93, 1984.

- xx Introduction to the IAPX 432 Architecture, Intel Corp, Santa Clara, Calif. 1981.

- xx IMAGE - 100. Data base management systems, Hew-lett- Packard, 1977.

- IVERSON, K.E. A Programming Language, Ed.John Wiley&Sons, New-Iork, 1963.

- JUICA I. Simularea sistemelor continue şi discrete, lit.I.P.Timişoara, 1979.

- JUICA I. Sisteme de operare, curs, Lit.Inst.Polit. Timişoara, 1984.

- JUICA I. A Multiprocessor System with Multitasking Facilities, teză doctorat, TH Delft Olanda,1977.

- JUICA I., PETRIU D., CRETU V., A Study on concurent Programing Concepts Implemented in High Level Language, a 4-a. CIAC., Bucureşti, 1981.

- JIAN I. Proiectarea şi utilizarea baselor de date, curs, Lit. I.P. Timişoara, 1982.

- JIAN I. Sisteme de programe pentru calculatoare numerice, curs, Lit. I.P. Timişoara 1981.

- JIAN I. Subsistem conversaţional de teletransmisie cu mult acces, referat nr.2 I.P. Timişoara, 1981.

-JIAN I. Exploatarea echipamentelor în centrele teritoriale de calcul electronic, a 2-a. Reuniune a utilizatorilor de calculatoare,CHARTRES, mai 1973.

- JIAN I. Metode multiprogramare pentru sistemele de calcul FELIX C-256, sesiunea de comunicări ştiinţifice I.P. Timişoara, noembrie 1973.

- JIAN I. Metodă de exploatare eficientă a sistemului FELIX C-256, Sesiunea I.P. Timişoara, mai 1974.

-JIAN I. ş.a. Subsistem conversaţional de proiectare, simpozionul de informatică, Cluj-Napoca, 1977.

- JIAN I. ş.a. Subsistem de proiectare optimală asistată de calculator cu terminale conversaţionale, Simp.mat. de Informatică, Cluj-Napoca, 1976.

- JIAN I. Subsistem cu multiacces, Sesiunea ştiinţifică I.P. Timişoara, mai 1977.

- JIAN I. ş.a. Interacţiunea între limbaje de programare şi sisteme cu multiacces, Sesiunea şt.I.P.Tmş.1977.

- JIAN I. ş.a. Aspecte ale utilizării "Time-sharing-ului" în sistemele conversaţionale, Ses.şt.I.P.Tmş.mai 1977.

- JIAN I. ş.a. Utilizarea fişierelor în sistemele cu multiacces Sesiunea şt.I.P. Timişoara, mai 1977.

- JIAN I. ş.a. Probleme ale divizării timpului în sistemele cu multiacces, Sesiunea şt.I.P.Tşoara, mai 1977.

- JIAN I. ş.a. Programarea şi utilizarea calculatoarelor,curs lit.I.P. Timişoara, 1980.

- JIAN I. ş.a. Subsistem conversaţional cu multiacces pentru sistemele FELIX C-256, a 4-a. CIAC.Bucureşti,1981.

- JONES, A.K. s.a. Software Management of Cmm - A Distributed Multiprocessor, AFIPS, 1977.

- KNUTH, D.E., Tratat de programarea calculatoarelor, vol.1-2, Ed.tehn.,Bucureşti, 1974, 1975.

- KUPLEN, I., JUNCA I., s.a. Programarea în limbajul ADA, ed. Facla, Timişoara, 1982.

- KAHN, K.C. s.a. IMAX : A Multiprocessor Operating System for an Object - Based Computer, Proc. 8-th Sosp, dec.1981.

- KAMPEN, W.J. Sisteme de calculatoare cu divizarea timpului, Ed. tehnică, Bucureşti, 1976.

- KAMOLOP, K. Execution control and memory management of a data flow signal processor, IEEE Computer, Aschit.vol.11, nr.3, iunie 1983.

- KISHI, M. s.a. DDDP. : A. distributed data driven processor,IEEE Computer Architecture, vol.11, nr.3, iunie 1983.

- KUMAR, M. s.a. Switching strategies in a class of packet switching networks, IEEE Computer Architecture, vol.11,nr.3, iunie 1983.

- LAUER, H.C. Observations on the Development of an Operating System, Proc.8-th SOSP, dec. 1981.

- LORIE, R., DEITEL, H.M. Operating System, ed.Addison, Wesley, Reading, Massachusetts, 1981.

- LISKOV, B.H., The Design of the Venus Operating System CACM, vol. 15, nr.3, martie 1978.

- LYNCH, H.W., PAGE, J.B., The OS/VS2 Release 2 System Resources Manager, IBM Systems Journal, vol.13, 1984.

- LAMPSON, B.W., STURGIS, H.E., Reflections on an Operating System Design, CACM, vol. 19, nr.5, mai 1976.

- LE BIHAN, J. Le projet pilote SIRIUS, Bulletin INRIA, nr.61,1981.

- LORFANT, J. Quelle architecture pour les gigaflops? Bulletin INRIA, nr. 94, 1984.

- MICHELSEKY, A. Le CRAY-1 et son evolution,Bulletin INRIA, nr.94,1984

- MASCHLOCK, C. s.a. Arhitectura sistemului de operare HELIOS, a 4-a. CIAC, vol.4, Bucureşti, 1981.

- MYERS, G. Advances in Computer Architecture, ed. John Wiley & Son New-York, 1982.

- MADNICK, S.E., DONOVAN, J.J. Operating Systems, ed. McGraw-Hill, New-York , 1974.

- MARUSTER, S. Elemente ale sistemului de operare SIRIS-3, ed. Facla, Timişoara, 1980.

- MIHOC G. Utilizarea calculatoarelor în prelucrarea datelor, Ed. Dacia Cluj-Napoca, 1974.

- x x MANITOUR SIRIS-3. Manual d'utilisation et d'operations. CII Honeywell Bull, 1976.

- MARTINS, R.C.O, ILANI, K.B. On the Modeling of Simultaneity of Events in Concurrent Systems, a 4-a. C.I.A.C. Bucureşti 1981.

- MARTIN,J. Introduction to Teleprocessing, ed. Prentice-Hall, 1974.

- MARTIN,J. Computer data base organisation, Ed.Prentice-Hall, 1975.

- MARTIN,M, ş.a. Les réseaux locaux Danube, Bulletin INRIA, nr.70, 1981.

- NEEDHAM, R.M., WILKES, M.V. Domains of Protection and the Management of Processes, Computer Journal, vol. 17, nr. 2, 1974.

- x x Normes de programation sous SIRIS-2, CII, 1970

- NEGREANU, D. Echipamente de transmitere de date pentru conducerea centralizată a proceselor industriale distribuite, A 4-a. CIAC, Bucureşti 1981.

- NAFFAH, N. Le projet - pilote KAYAK, Bulletin INRIA,nr.69, 1981.

- OZIARD, P. Le projet RHIN, Bulletin INRIA,nr.61, 1980.

- PAUNESCU,F. Analiza şi concepţia sistemelor de operare, ed.ş. şi enciclopedică,Bucureşti, 1982.

- POP, V. Structura sistemelor de prelucrare a datelor numerice, curs vol.I,II, lit.I.P.Tmş. 1981.

- x x The Programming Language ADA. Reference Manual, ed. Springer, Berlin 1981.

- PRITCHARD, J.A. Quantitative methods in on-linesystems, Ed. NCC, Manchester, 1976.

- PRICE, W, L, ş.a. Teleinformatica. Reţele de calculatoare şi protocoalele lor. Ed.tehn.Bucureşti, 1983.

- POPEK, G.J., GOLDBERG, R.P. Formal Requirements for Virtualizable Third Generation Architectures, CACM, vol.17, nr.7, iulie 1974.

- PESCARU, V. ş.a. Iniţiere în teleprelucrarea datelor Ed.Tehn.Buc.197

- PESCARU, V. ş.a. Fişiere, Baze şi Bănci de date, Ed.tehn.Buc.197

- PETRESCU A. Microprogramare. Principii şi aplicaţii, Ed. Tehnică, Bucureşti, 1975.

- PETRESCU, M. The Optimization of Expression in the Context of Relational Model for data Bases, a 4-e. CIAC. Bucureşti, 1981.

- PHILIPSON, L.ş.a. A communication structure for a multiprocessor Computer with distributed global memory, IEEE Computer Architecture, vol.11,nr.3,iunie,1983.

- PUJOLLE, G. Réseaux locaux-état de l'art et perspectives, Bulletin INRIA, nr.89, 1983.

- x x Programarea calculatoarelor Ed. Facla, Tmş.198

- QUINT, V. L'édition interactiv des formules mathématique Bulletin INRIA, nr. 90, 1984.

- ROGOJAN,A. Calculatoare numerice, vol.1-2, Lit.I.P.Tmş.19

- x x RSX-11M. Utilities Procedures Manual,DEC,Maynard, Mass.

- xx RSX-11M. Operator's Procedures Manual, DEC,Maynard, Mass, 1977.

- RITCHIE, D.M. , THOMPSON, K. The UNIX Time Sharing System,CACM vol.17, nr.7, iulie 1974.

- RABINOVICI, S. ş.a. Metodă de elaborare şi implementare a sistemului de operare HELIOS, A 4-a. C.I.A.C., Bucureşti, 1981.

- RUSAN,M. ş.a. Transmisii de date pe canale Tc de banda largă, A 4-a. C.I.A.C., Bucureşti, 1981.

- STRINGA, L. EMMA. An industrial experience on large multiprocessing architectures, IEEE Comp. Arch., vol.11, nr.3, iunie 1983.

- SIRBU,M., NICA I., Data Teleprocessing Systems Performance Evoluation, a 4-a. C.I.A.C.,Bucureşti, 1981.

- x x Sistemul de gestiune al figierelor, MICM, 1972.

- SALTZER,H. Protection and Control of Information Sharing in MULTICS, CACM, vol.17, nr.7, iulie 1974.

- SHOCH,J.F.ş.a. Evolution of the Ethernet Local Computer Network, Computer, aug. 1982.

- SHAW,A.C. The Logical Design of Operating Systems, Ed. Prentice - Hall, Englewood Cliffs, 1974.

- SPECTOR,A.Z. Performing Remote Operations Efficiently on a Local Computer Network, CACM. vol.25,nr.4 apr. 1982.

- SCHROEDER,M.D. SALTZER,J.H. A Hardware Architecture for Implementing Protection Rings, CACM, vol. 15, nr.3, martie 1972.

- STRUGARU,C. Echipamente periferice şi transmiterea datelor, curs, Lit. I.P. Timişoara, 1979.

- x x SIRIS, Editorul de legături. Manual de utilizare MICM - ITC, 1972.

- x x SIT. Manuel d'utilisation, CII, Franţa, 1975.

- x x SFDX-18. Manual de utilizare, ICE, Bucureşti, 1980.

- x x SOCRATE. Manuel d'utilisation, CII, Franţa 1975.

- x x SOCRATE. Manuel d'operation, CII, Franţa, 1975.

- x x SISTÈME de gestion de transmission, CII, Franţa,1973.

- x x STRATEGE. Manuel d'utilisation, CII, Franţa, 1976.

- TEODORESCU,A., CATONA,I., POPESCU,C. Sistemul FELIX C-256. Limbajul ASSIRIS, Ed. Academiei R.S.R.,Buc.1974.

- x x TOTAL data base management System, CDC, 1977.

- WALKER,B.J., KEMMERER,R.A., POPEK,G.J. Specification and Verification of the UCLA Unix Security Kernel, CACM, vol.23, nr.2., febr. 1980.

- WULF,W. ş.a. HYDRA - The Kernel of a Multiprocessor Operating System, CACM, vol.17,nr.4, iunie 1974.

- WIEDERHOLD,G. Data base design. Ed. Mc.Grav-Hill, New-York,1977.

- WAH, B.W. A comparative study of distributed resource sharing on multiprocessors., IEEE Computer Architecture, vol. 11, nr.3, iunie 1983.

- WILKES,M.V. Sisteme de calcul cu acces multiplu, Ed.tehnică, Bucureşti, 1974.

- x x X-25 Recomandation, CCITT, Geneva, 1980
- ZIMMERMANN, H. Les réseaux informatiques, Bulletin INRIA, nr. 61, 1981.

Prescurtări utilizate în referințele bibliografice

AFIPS - American Federation of Information Processing Societies

AMC - Automatică, management, calculatoare, Ed.tehnică

CACM - Communications of the Associations for Computing Machinery

CIAC - Conferința Internațională de Automatică și calculatoare

ICS - ACM Journal on Computer Systems

RAIRO - Revue Française d'Automatique, Informatique et Récherche Opérationelle

SJ - IBM Systems Journal

SOSP - Symposium on Operating Systems Principles

TSE - IEEE (Institute of Electrical and Electronics Engineers) Transactions of Software Engineering.

Listă de prescurtări utilizate

I/E - intrare/ieşire
U C - unitate centrală
M C - memorie centrală
USM - unitate de schimburi multiple
1K - 1024 octeţi
MV - memorie virtuală
MA - memorie asociativă
IMT - imagine memorie translatabilă (program)
BT - binar translatabil (program)
MMT - memorie rapidă de translaţie adrese virtuale
SGF - sistem de gestiune a fişierelor
SGT - sistem de gestiune a teletransmisiilor
OS - operating systems (sisteme de operare IBM)
OS/MFT- multiprogramming with a fixed number of tasks
OS/MVT- multiprogramming with a variabil number of tasks
OS/VS - virtual storage
OS/VM - virtual machine
OS/TSS- time sharing system
TSO - time sharing option
CMS - conversational system (cu VM/370)
Cr67 - control programme (sist.de operare cu TS pe IBM/360-67)
CTSS - Compatible time sharing system (MIT)
SRM - System resources manager (cu OS/VS2)
SPOOL - Simultaneous peripheral operation on line
SMF - System management facility
TMP - terminal monitoring program (în TSO)
MULTICS- Multiplexed information an Computing system
SCOT - sisteme conversaţional de teletransmisie
IPL - interactive programme language