Sisteme de operare. Mecanisme interne și principii de proiectare

1. Introducere

Cursul tratează conceptele și principiile fundamentale ale teoriei și practicii sistemelor de operare (SO). Sunt prezentate definițiile principale și unele clasificări, interfețele sistemelor de operare, modul de organizare a procesului de calcul, gestionarea memoriei și a dispozitivelor periferice, administrarea informației și gestiunea fișierelor. Lucrarea acoperă aspectele asociate bazelor contemporane ale metodelor şi mijloacelor de elaborare a resurselor program de sistem (inclusiv, operaţii asincrone, tratarea întreruperilor, compromisele dintre dispozitivele tehnice şi resursele program, interfeţele sistemelor de operare), având drept obiectiv final pregătirea cititorului pentru analiza şi proiectarea sistemelor de operare. Cursul include capitole teoretice și materiale ilustrative practice, chemate să familiarizeze cititorul nu atât cu modalitatea utilizării unor sisteme de operare concrete, ci în special cu metodele și algoritmii, care stau la baza mecanismelor interne ale sistemelor de operare și modul în care acestea sunt concepute, proiectate și implementate.

Un sistem de calcul constă din două tipuri de resurse: resursele fizice și resursele logice. Resursele fizice posedă caracteristici tehnice avansate şi pot fi utilizate în cele mai diverse scopuri. Însă aceste resurse fără componentele logice de sistem (software de sistem) întâmpină dificultăţi mari în relaţia cu mediul în care trebuie să funcţioneze. *Acesta este unul dintre* ***motivele principale*** *ale creării* ***sistemelor de operare****, destinaţia cărora este* ***administrarea*** *(gestiunea, controlul)* ***resurselor tehnice*** *principale şi* ***asigurarea unei interfeţe comode*** *(plăcute, prieteneşti) între utilizator şi calculator* (fig.1.1, [1]).

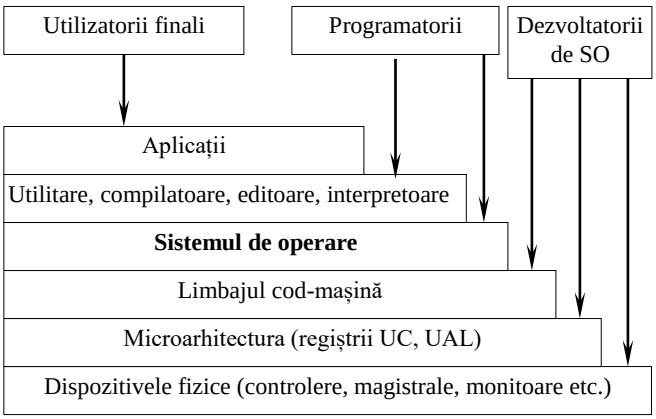


Fig.1.1. Locul sistemului de operare în cadrul unui sistem de calcul

Există mai multe motivaţii ale necesităţii studierii de către viitorii specialişti IT a sistemelor de operare, cele mai importante fiind următoarele:

- pentru utilizarea resurselor hardware în scopuri speciale poate fi necesar să fie creeat un sistem de operare propriu sau să se introducă modificări într-unul existent;
- de alegerea corectă a sistemului de operare şi a versiunii concrete poate depinde viabilitatea şi eficacitatea sistemului de calcul;
- este ideal ca utilizatorul să interacţioneze cu sistemul de operare cunoscând toate subtilităţile ultimului, deoarece sistemul de operare este un intermediar între calculator şi utilizator;
- multe metode şi concepte, utilizate în domeniul SO, pot fi implementate cu succes şi în alte domenii.

Prin noţiunea ***sistem de operare*** înţelegem în primul rând modulele program ale unui sistem de calcul, care administrează resursele tehnice (procesoare, memoria operativă şi secundară, dispozitive de intrare/ieşire, fişiere). Modulele în cauză soluţionează situaţiile de conflict, optimizează productivitatea sistemului, sporesc eficienţa utilizării lui. Ele sunt un fel de intermediar (interfaţă) între programele utilizatorului şi componentele tehnice ale calculatorului. Alte denumiri istorice: program de administrare, monitor, supervizor.

Modulele destinate unor domenii anume, cum ar fi translatoarele, depanoarele, bibliotecile, mediile integrate de dezvoltare etc., nu sunt incluse în definiţia unui SO, fiind considerate şi ele „utilizatori” ai sistemului de operare.

1.1. Noţiuni de bază şi clasificări

Un calculator constă dintr-un ansamblu de componente funcţionale fizice şi logice, care cooperează pentru a satisface cerinţele utilizatorilor privind introducerea, stocarea, prelucrarea, transmisia şi căutarea informaţiilor. Aceste componente funcţionale sunt structurate pe niveluri, care interacţionează prin interfeţe bine definite.

Suportul fizic *(resurse tehnice, hardware)* constituie nivelul inferior al sistemului de calcul construit pe baza unor componente electronice, magnetice, optice, mecanice etc., mai mult sau mai puţin sofisticate în funcţie de stadiul de dezvoltare a tehnologiilor respective.

1.1.1. Noţiuni şi termeni din domeniul resurselor tehnice

Pentru a trece la noţiunile principale, legate de hardware, vom face cunoştinţă mai întâi cu funcţiile de bază ale unui calculator. Pot fi evidenţiate cinci funcţii esenţiale [2]: *iniţializarea (bootup), introducerea datelor, procesarea datelor, stocarea datelor* şi *extragerea rezultatelor*:

- ***Iniţializarea*** implică testarea părţilor importante ale calculatorului, rularea fişierelor de pornire şi încărcarea altor fişiere necesare, cum ar fi driverele de dispozitive;
- ***Introducerea*** reprezintă transferul datelor dintr-o sursă externă în calculator. Surse externe pot fi dischetele, tastatura, mouse-ul etc.;
- ***Procesarea*** se referă la manipularea datelor introduse în scopul producerii unui rezultat (ieşirea);
- ***Stocarea*** constituie procesul salvării informaţiilor (date sau programe) într-un dispozitiv de păstrare, de exemplu discul fix, pentru recuperarea ulterioară.

Prin ***structura*** unui calculator vom înţelege componentele (dispozitivele) care formează calculatorul şi legăturile dintre ele. Componentele principale sunt: *procesorul, memoria, inclusiv unităţile de stocare pe termen lung, dispozitivele de intrare-ieşire* (tastatura, display-ul, mouse-ul etc.). Relaţiile (legăturile) dintre aceste componente pot fi foarte variate, o structură devenită apoi clasică, este **structura John von Neumann (**fig.1.2).

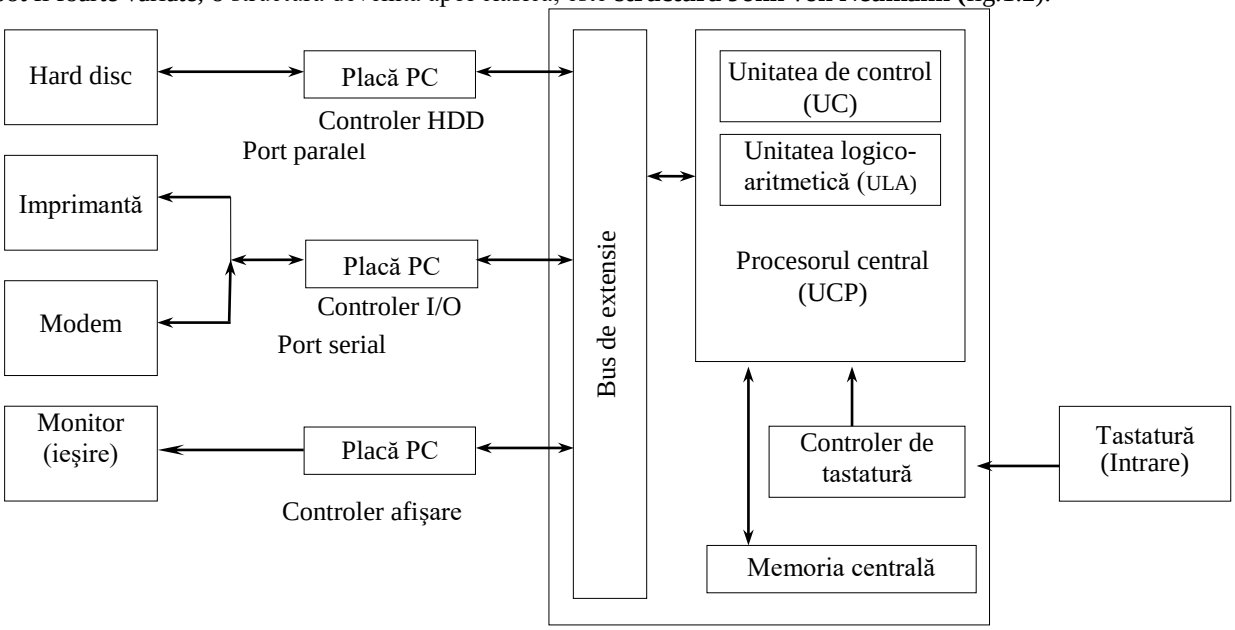


Fig. 1.2. Structura unui calculator

Componentele principale se conectează la placa de bază (motherboard) direct sau prin conectoare speciale, numite plăci de extensie (daughterboards). **Unitatea centrală de procesare – procesorul** (CPU) se găseşte într-un singur circuit integrat (cip) incorporând **unitatea de comandă (Control Unit, CU)** şi **unitatea logico-aritmetică (Arithmetic Logical Unit, ALU)**.

Unitatea de comandă controlează funcţionarea unităţii logico-aritmetice. **Memoria** este o zonă de lucru de mare viteză, unde sunt stocate datele şi programele pentru a fi accesate de CPU în mod rapid. Memoria poate fi organizată în mod ierarhic, caz în care există cel puţin două nivele de ierarhie – **memoria centrală** (operativă, internă) şi **memoria secundară** (externă, de lungă durată). Memoria operativă este electronică sub formă de cipuri, de obicei cu acces aleator (RAM – Random Access Memory) şi trebuie să fie alimentată cu tensiune pentru a păstra datele (memorie vie). Pentru salvarea datelor atunci când se întrerupe alimentarea sau pentru păstrare de lungă durată, datele sunt stocate în memoria secundară, care le reţine oricât de mult timp. Dispozitivele cele mai obişnuite de introducere a datelor sunt tastatura şi mouse-ul, iar dispozitivul de ieşire cel mai utilizat este monitorul. Procesorul şi memoria operativă formează **nucleul** calculatorului, toate celelalte dispozitive fiind cunoscute sub denumirea de **periferie** (dispozitive periferice).

Instrucţiunile care vor fi îndeplinite de calculator sunt stocate în memorie sub formă de programe. Unitatea de comandă ţine evidenţa şi interpretează instrucţiunile dintr-un program, fiind responsabilă cu transmiterea de sarcini

specifice diferitelor elemente ale calculatorului. Unitatea de control controlează în principal funcţiile de intrare-ieşire (I/O), de memorizare şi stocare, colaborează cu ALU, care răspunde de efectuarea operaţiilor de calcul.

Mai menţionăm noţiunile: mărimea magistralelor interne şi externe de date, mărimea adresei de memorie, frecvenţa ceasului.

Mărimea magistralei interne de date: procesoarele păstrează datele în locaţii speciale, numite ***registre***. Datele sunt transferate între registre, CU, ALU şi alte componente ale procesorului prin intermediul unei magistrale realizate în circuitele procesorului. Numărul de linii în această magistrală oferă o măsură a cantităţii de date pe care procesorul o poate transfera într-o singură operaţie. Valoarea ei poate varia între 8 şi 32 de biţi (în scopuri speciale 64, 128 sau chiar mai mult).

Mărimea magistralei externe de date măsoară câte date pot fi transferate între procesor şi dispozitivele periferice într-un tact de ceas. Magistrala este un sistem de conectări şi cablări ce distribuie datele prin calculator. Cu cât magistrala de date e mai mare, cu atât performanţele calculatorului se îmbunătăţesc. Numărul de linii şi aici variază între 8 şi 32 de biţi.

Mărimea adresei de memorie determină volumul de memorie care poate fi gestionat de calculator fără eforturi speciale. Prin eforturi speciale înţelegem resurse fizice sau logice, care permit gestionarea unei memorii cu capacitate mai mare decât volumul obţinut prin calcularea adresei reieşind din numărul de linii fizice ale memoriei.

Un **ceas** electronic (generator de sincronizare) asigură coordonarea corespunzătoare a componentelor calculatorului. Componentele calculatoarelor cu performanţe superioare pot opera la **frecvenţe de ceas** mai mari. Frecvenţa ceasului indică viteza de operare a CPU şi se măsoară în miliarde de impulsuri pe secundă, adică în *GigaHertzi* (GHz).

1.1.2. Noţiuni şi termeni din domeniul sistemelor de operare

Un **sistem de operare** este un ansamblu de programe de control şi de serviciu care ghidează un calculator în executarea sarcinilor sale şi asistă programele de aplicaţie şi utilizatorul prin intermediul anumitor funcţii. Natura funcţiilor şi modul în care acestea sunt realizate determină **atributele** care caracterizează un sistem de operare: *timpul de răspuns, simultaneitatea utilizării, eficienţa, partajarea şi protecţia, universabilitatea, flexibilitatea, extensibilitatea, fiabilitatea şi disponibilitatea, transparenţa şi vizibilitatea* [3].

Timpul de răspuns exprimă durata intervalului delimitat de lansarea unei cereri de serviciu şi achitarea acesteia de către sistem. Are două componente: timpul de aşteptare pentru ca cererea să fie luată în consideraţie şi timpul de execuţie a acestei cereri.

Simultaneitatea utilizării măsoară gradul în care un sistem poate să execute în acelaşi timp mai multe lucrări.

Eficienţa măsoară proprietatea unui sistem de a folosi în mod optim resursele de care dispune.

Partajarea şi **protecţia** caracterizează nivelul la care utilizatorii au posibilitatea să utilizeze în comun informaţia prezentă în sistem şi nivelul la care pot să comunice între ei, în deplină siguranţă (în sensul evitării accesului neautorizat şi/sau alterării intenţionate sau accidentale a informaţiei).

Unversalitatea, flexibilitatea, extensibilitatea măsoară gradul în care un sistem poate fi folositor (universalitate) şi adaptabil (flexibilitate) unui context specific, exprimat prin nivelul de limitare impus programelor utilizatorului, precum şi gradul în care se pot include în sistem noi componente hardware şi software fără eforturi de proiectare şi programare suplimentare (extensibilitate).

Fiabilitatea şi **disponibilitatea** exprimă proprietatea unui sistem de a cădea foarte rar în pană şi de a evita goluri în funcţionare din cauza defectării uneia sau mai multor componente ale sale.

Transparenţa şi **vizibilitatea** exprimă pe de o parte proprietatea unui sistem de a face invizibil utilizatorului ceea ce se află sub interfaţa de utilizare care i se oferă şi, pe de altă parte, capacitatea de a permite utilizatorilor săi să obţină anumite informaţii despre modul cum el lucrează, informaţii de care în mod teoretic ei nu au nevoie pentru a beneficia de o utilizare completă, însă care ar putea să-i ajute la obţinerea unei utilizări mai eficiente [3].

Resursele program reprezintă seturi de programe şi date utilizate pentru soluţionarea anumitor probleme. **Programul** este transcrierea într-un limbaj de programare a unui algoritm, altfel – programul este o secvenţă de instrucţiuni sau simplu **cod**. **Utilizatorul** este oricare doritor să îndeplinească anumite lucrări la calculator. Prin **lucrare (sarcină, task)** vom înţelege un set de acţiuni, necesare pentru îndeplinirea unui lucru anume. Sarcina poate conţine mai mulţi **paşi**. **Paşii de sarcină** sunt unităţi de lucru, care vor fi îndeplinite consecutiv, de exemplu trei paşi – compilare, încărcare şi executare. Primind o lucrare de la utilizator, sistemul de operare poate creea câteva **procese**, prin proces înţelegând la moment, **calcule** care pot fi efectuate paralel cu alte calcule.

Procesul mai poate fi definit drept traiectoria procesorului, atunci când ultimul îndeplineşte un set oarecare de programe. Ansamblul programelor şi datelor accesate în timpul procesului, formează **spaţiul de adrese**. Una din destinaţiile sistemului de operare este de a asigura **proiectarea** spaţiului de adrese a unui proces în memoria fizică. Pentru rezolvarea acestei probleme sunt utilizate atât resurse tehnice (sisteme cu organizarea memoriei pe segmente sau pe pagini), cât şi resurse program speciale.

Multiprogramarea este un termen utilizat în cazul unui sistem în care pot exista simultan câteva procese în etapa de execuţie. Un proces se consideră în **execuţie**, dacă calculele au început, dar la momentul considerat nu au fost terminate sau întrerupte (terminare din cauza unei erori sau din alte motive). Nu este obligatoriu ca un proces care se află în **execuţie** să fie şi executat de procesor la un moment dat.

Resursele hardware de **protecţie** sunt utilizate cel mai des pentru controlul accesării memoriei. **Întreruperea** este un mecanism care impune procesorul să observe anumite evenimente. Pot exista mecanisme care permit să nu se acorde atenţie unei anume întreruperi – **întrerupere mascată**. Resursele hardware de **întrerupere** permit sistemului de operare să coordoneze operaţiile simultane; pot fi utilizate şi pentru a schimba ordinea de execuţie a programelor.

1.1.3. Tipuri și exemple de sisteme de operare

Valorile concrete ale atributelor sistemelor de operare şi combinaţii ale acestora determină diverse tipuri de sisteme şi restricţii de implementare. Conform acestor atribute pot fi evidenţiate sisteme de operare [3]:

- secvenţiale
- cu multiprogramare
- cu prelucrare multiplă
- în timp real, etc.

Majoritatea sistemelor de operare recunosc **programul** ca **cea mai mică unitate de prelucrare**, căreia i se atribuie o identitate şi pe care un utilizator o poate prezenta spre execuţie. Unele sisteme permit ca un program să fie considerat ansamblu de sarcini ale căror execuţii (inclusiv în paralel) contribuie la atingerea obiectivului urmărit de acest program.

Un sistem ***secvenţial*** (tratare pe loturi, batch processing en., traitement par lots fr.) execută la un moment dat un singur program, care trebuie terminat înainte de a lansa un alt program în execuţie.

Sistemele cu ***multiprogramare*** acceptă la un moment de timp dat mai multe programe în memoria centrală, acestea aflându-se în diverse stadii de execuţie.

Un sistem de calcul cu ***prelucrare multiplă*** dispune de mai multe procesoare, care pot să execute simultan unul sau mai multe programe. Utilizarea efectivă a prelucrării multiple necesită atributul de multiprogramare. Execuţia simultană a unui singur program de către mai multe unităţi presupune existenţa posibilităţii de a descompune acest program în mai multe *procese* sau *fire de execuţie*.

Sistemele în ***timp real*** sunt dedicate, de obicei, funcţionării în cadrul unor sisteme de comandă şi este necesar ca valorile anumitor atribute să se încadreze în anumite limite, dictate de dinamica proceselor comandate.

Clasificarea pe tipuri de sisteme de operare enumerate mai sus nu este nici disjunctă şi nici exhaustivă. Majoritatea sistemelor existente pot fi încadrate în mai multe clase, atunci când se face o analiză prin prisma obiectivelor pe care le urmăresc. La capitolul obiective vom aminti în primul rând **maximizarea eficienţei** sistemului de calcul şi a **satisfacţiei** utilizatorilor. Tot la obiective poate fi trecută şi cererea de **minimizare a posibilităţii de apariţie a erorilor** şi de **maximizare a transparenţei** sistemului de operare, **garantarea securităţii** datelor, **optimizarea controlului comunicaţiilor** în cazul SO de reţea.

Un obiectiv foarte important este **minimizarea** efortului **concepţie-realizare** a sistemului, ultim în enumerare, dar foarte important pentru specialişti.

Toate aceste obiective sunt consecinţe ale dezideratului principal: un sistem de operare ***este destinat să administreze*** resursele sistemului de calcul şi anume memoria, procesorul (procesoarele), dispozitivele şi informaţia.

Un sistem de operare este obligat:

- să păstreze informaţia despre starea fiecărei resurse
- să ia decizia cărui proces să i se aloce resursa, în ce cantitate şi când
- să aloce resursa şi
- la momentul oportun să o retragă.

Exemplele care urmează [4] vor ilustra diversitatea funcţiilor îndeplinite de către un sistem de operare, fără pretenţii de exhaustivitate. Pentru fiecare exemplu vom indica funcţiile puse în şarja sistemului de operare şi caracteristicile principale ale acestuia.

1.1.3.1. Cazul calculatoarelor personale

Configuraţia cea mai simplă a unui calculator personal (PC) include o unitate centrală, o memorie principală, un display, o tastatură şi un mouse. Această configuraţie, de obicei, este completată de o memorie secundară şi o imprimantă (fig.1.3).

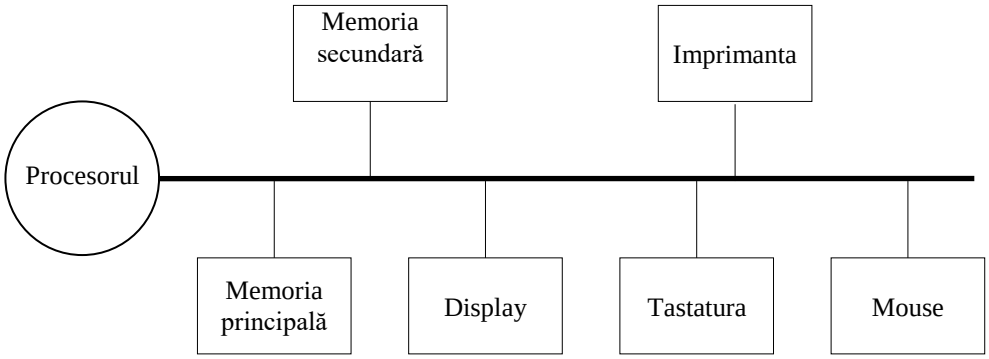


Fig. 1.3. Structura unui calculator personal

Utilizatorul unui atare sistem va cere minimum următoarele două tipuri de servicii:

- executarea unor programe existente în PC sau introduse sub formă de fişiere; introducerea datelor necesare pentru executarea programului (de la tastatură, din fişier sau din alte surse); listarea rezultatelor la display, imprimantă sau copierea lor într-un fişier;
- identificarea şi crearea unor fişiere sau mulţimi structurate de informaţii; stocarea acestor fişiere în memoria secundară; transferarea informaţiilor între fişiere şi dispozitivele de intrare/ieşire.

Sistemul de operare poate acorda aceste servicii prin intermediul unui limbaj special, numit **limbaj de comandă**, introducându-se de la tastatură instrucţiuni de forma <acţiune> <parametri>, sau utilizând mouse-ul şi o interfaţă grafică a utilizatorului (GUI - graphical user interface), acţionările mouse-lui fiind imediat interpretate de sistem.

Iată două exemple de secvenţe tipice de activităţi în cazul unui PC:

- elaborarea unui program;
- introducerea programului cu ajutorul tastaturii şi a unui editor de texte;
- executarea programului introducându-se datele necesare de la tastatură şi extrăgând rezultatele la display sau imprimantă;
- modificarea programului, dacă rezultatele nu sunt satisfăcătoare şi repetarea execuţiei;
- perfectarea versiunii finale a programului, inclusiv documentarea la necesitate a acestuia;
- exploatarea unui program;
- cererea de executare a unui program deja existent. Vor fi pregătite în prealabil date de intrare sau acestea vor fi introduse in mod interactiv la cerere cu ajutorul tastaturii;
- afişarea rezultatelor pe ecran, listarea la imprimantă sau copierea lor într-un fişier pentru o utilizare ulterioară.

Într-un atare sistem funcţia *partajare a resurselor* poate fi lipsă, or PC-ul este folosit de un singur utilizator care are controlul total asupra acestuia. Alocarea resurselor este legată de gestionarea memoriei şi administrarea fişierelor. Funcţiile principale vizibile ale sistemului de operare constau în administrarea fişierelor, realizarea operaţiilor de intrare/ieşire şi interpretarea comenzilor provenite de la interfaţa utilizator-sistem de operare.

Pentru acest tip de sisteme cele mai importante caracteristici sunt:

- fiabilitatea;
- eficacitatea;
- simplitatea utilizării;
- facilitatea extensibilităţii prin adăugarea unor utilite noi sau adaptarea la periferice noi.

Ultimele două aspecte pun în evidenţă importanţa interfeţelor oferite de sistem (limbajul de comandă sau GUI).

1.1.3.2. Comanda unor procese industriale

La o uzină chimică sunt utilizate doua materii prime *A* şi *B* pentru sinteza produsului *C* conform fig. .4. Procesul de producere este comandat de un calculator care îndeplineşte următoarele funcţii:

- *Reglare.* Pentru o derulare bună a procesului de fabricaţie parametrii de funcţionare (temperatura, presiunea, concentraţia, etc.) trebuie să se afle într-o plajă de valori predefinite. Pentru aceasta va fi acţionat debitul de intrare a materiilor prime *A* sau *B.* Parametrii de funcţionare sunt măsuraţi cu ajutorul unor captoare.

Calculatorul preia aceste măsurări şi, în dependenţă de algoritmul de comandă, acţionează robinetele de intrare.

- *Înregistrare.* Rezultatele măsurărilor sunt periodic înregistrate; valorile lor sunt afişate pe un tablou de bord şi recopiate într-un fişier ("jurnal de bord") în scopul unor prelucrări ulterioare (date statistice).
- *Securitate.* În cazul în care unul dintre parametrii măsuraţi depăşeşte o valoare critică predefinită reactorul trebuie oprit imediat.

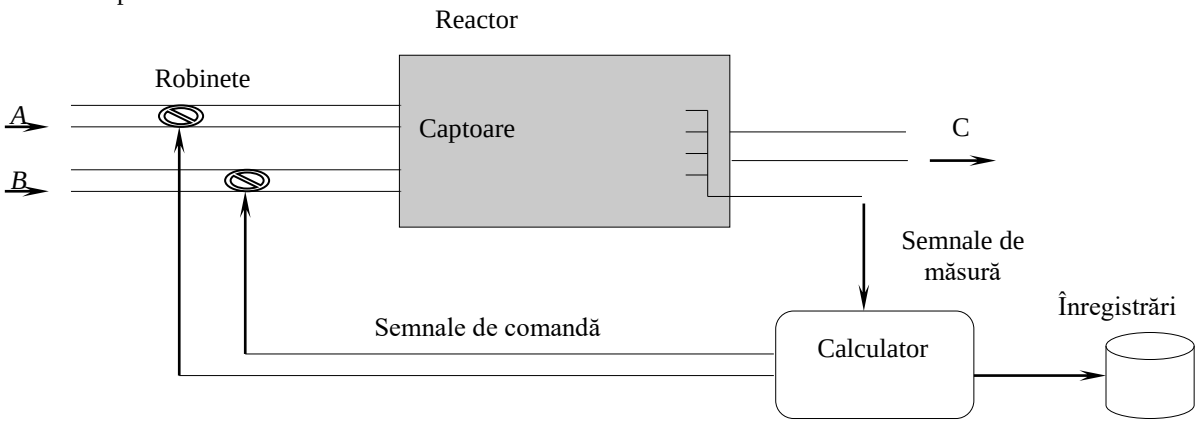


Fig. 1.4. Schema unui proces chimic

Acest mod de funcţionare introduce unele restricţii:

1. Măsurările se produc periodic; fie *T* valoarea intervalului de timp dintre două măsurări consecutive (perioada de eşantionare), iar *t* - timpul total de prelucrare a datelor de către calculator (măsurarea propriu-zisă a semnalelor observate, înregistrarea, calcularea semnalelor de comandă şi acţionarea robinetelor). Sistemul va funcţiona doar în cazul respectării relaţiei $t \le T$.
2. Securitatea sistemului are prioritate maximă. Depăşirea unor valori critice trebuie să fie detectată în orice moment şi tratarea acestor accidente va întrerupe toate operaţiile în curs de execuţie.

Funcţiile principale ale sistemului de operare sunt:

- acţionarea organelor externe (citirea semnalelor captoarelor, comanda robinetelor);
- evidenţa timpului real (declanşarea periodică a ciclului de calculare a semnalelor de comandă);
- reacţia la evenimentele exterioare (oprire de urgenţă);
- gestiunea informaţiilor (păstrarea şi întreţinerea jurnalului de bord).

Existenţa unor restricţii stricte privind durata de prelucrare a informaţiilor, noţiunea de tratare prioritară, conectarea la dispozitive exterioare de măsurare şi acţionare sunt caracteristice aplicaţiilor informatice *în timp real*. Pot fi menţionate şi alte domenii cu comandă în timp real: centralele telefonice, comanda aparatelor de zbor, robotica, monitoringul medical, etc.

În cazul acestor sisteme **caracteristica principală** este **fiabilitatea**, or rezultatele unei funcţionări neadecvate pot fi catastrofale. Sistemul trebuie să garanteze un serviciu minim în cazul unor căderi în pană a dispozitivelor tehnice, unor evenimente accidentale sau erori umane.

1.1.3.3. Sisteme tranzacţionale

Caracteristicile principale ale sistemelor tranzacţionale sunt următoarele:

- sistemul gestionează un set de informaţii sau baze de date, care pot atinge volume importante;
- asupra acestor informaţii pot fi executate anumite operaţii predefinite, sau tranzacţii, adesea interactive;
- sistemul este dotat cu un mare număr de puncte de acces şi un mare număr de tranzacţii pot derula simultan.

Ca exemplu pot fi menţionate sistemele de rezervare a biletelor, de gestiune a conturilor bancare, de arhivare şi consultare a documentelor.

Restricţiile sunt în primul rând legate de integritatea şi coerenţa internă a informaţiilor, care formează bazele de date. Aceste restricţii depind, evident de aplicaţie. De exemplu, numărul de locuri rezervate într-un avion nu poate depăşi numărul locurilor disponibile, un loc distinct poate fi atribuit unei singure persoane, etc.

Calităţile obligatorii ale unui sistem tranzacţional sunt disponibilitatea şi fiabilitatea; pentru unele sisteme poate fi importantă şi toleranţa la defecţiuni. O caracteristică importantă ale sistemelor tranzacţionale este multitudinea activităţilor paralele, iar în multe cazuri şi repartizarea geografică a componentelor.

1.1.3.4. Sisteme cu partajarea timpului

Destinaţia principală a unor astfel de sisteme este furnizarea serviciilor necesare unei mulţimi de utilizatori, fiecare dintre ei beneficiind de servicii:

- echivalente serviciilor unui calculator individual;
- legate de existenţa unei comunităţi de utilizatori: partajarea informaţiilor, comunicaţii între utilizatori.

Problemele care apar datorită conceptului de partajare a timpului sunt o combinaţie a problemelor existente în cazul unui calculator individual cu cele din sistemele tranzacţionale şi pot fi clasificate după cum urmează:

- definirea maşinii virtuale, oferite fiecărui utilizator;
- partajarea şi alocarea resurselor fizice comune: procesoare, memorii, organe de comunicaţie;
- gestionarea informaţiilor partajate şi a comunicaţiilor.

Caracteristicile obligatorii unui atare sistem combină în egală măsură calităţile unui sistem de operare al unui calculator individual şi al unui sistem tranzacţional: disponibilitatea, fiabilitatea, securitatea, exploatarea optimă a resurselor fizice, calitatea interfeţei şi serviciilor utilizatorului, simplitatea adaptării şi extensibilităţii.

1.2. Puncte de vedere asupra sistemelor de operare

1.2.1. Sistemul de operare şi procesele

Noţiunea de **proces**, introdusă mai sus, este asociată conceptului de lucrare (pentru a lua în considerare aspectele dinamice) şi poate fi definită altfel ca o **suită temporală de execuţii de instrucţiuni,** fiind o entitate de bază în descrierea sau analiza funcţionării unui sistem de operare. Evoluţia în timp a unui proces presupune un consum de resurse, dictat de natura şi complexitatea instrucţiunilor de executat. Orice utilizare a unei resurse este asociată, la un moment dat, unui proces şi procesul respectiv îşi asumă răspunderea utilizării acestei resurse. În particular, rezultă că ori de câte ori se execută procedurile de sistem, resursele pe care le utilizează sistemul intră în administrarea procesului (fie el şi al utilizatorului), care a cerut serviciul. Mulțimea resurselor (procesorul, memoria centrală, informaţia, dispozitivele) alocate unui proces variază în timp (dinamica procesului).

Anterior un sistem de operare a fost definit ca un set de programe destinat să administreze resursele. Care sunt relaţiile dintre programele sistemului de operare? Atunci când un proces este creat, care este ordinea de utilizare a unui anume program al SO? Pentru a răspunde la aceste întrebări (şi la altele) vom face cunoştinţă cu ciclul de viaţă a unui proces. În fig. 1.5 sunt prezentate trei procese (trei sarcini ale utilizatorilor, lansate în execuție) existente într-un sistem cu multiprogramare.

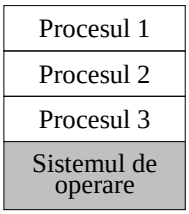


Fig. 1.5. Trei procese într-un sistem cu multiprogramare

Ciclul de viaţă a unui proces poate fi reprezentat printr-un set de stări ale procesului şi tranziţiile de la o stare la alta. Vom evidenţia trei stări **elementare** ale unui proces: proces ***ales*** (sau *exe*) – *procesului i s-a alocat un procesor, este în curs de execuţie,* proces ***blocat*** (*wait*)– procesul aşteaptă să se producă un anumit eveniment a cărui apariţie este indispensabilă, proces ***eligibil*** (*ready*)– procesul are la dispoziţie toate resursele necesare lipsă fiind doar procesorul, adică este pregătit să fie execute din momentul alocării unităţii centrale (fig. 1.6).

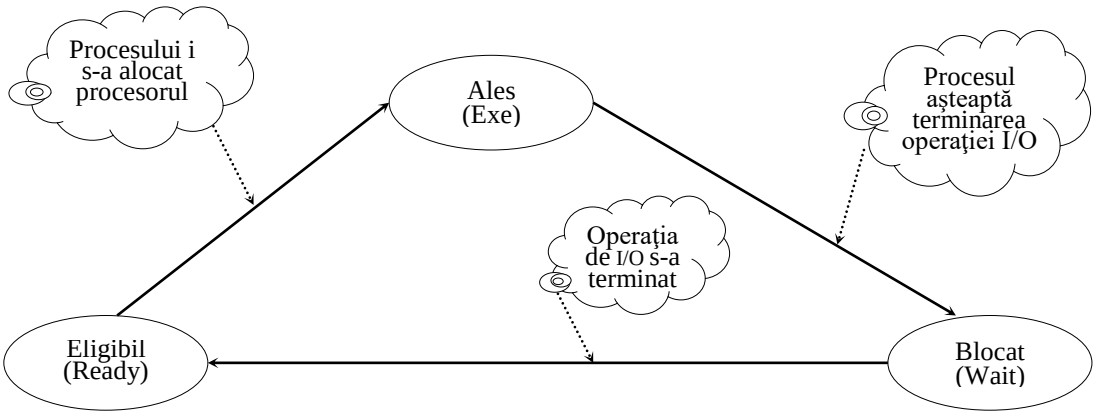


Fig. 1.6. Stările elementare din ciclul de viaţă a unui proces

Modelul prezentat în figura 1.6 este unul puternic simplificat, în realitate existând abateri substanţiale în direcţia complexităţii (fig. 1.7). Sistemul de operare este acea componentă, care va ghida procesul prin toate aceste stări.

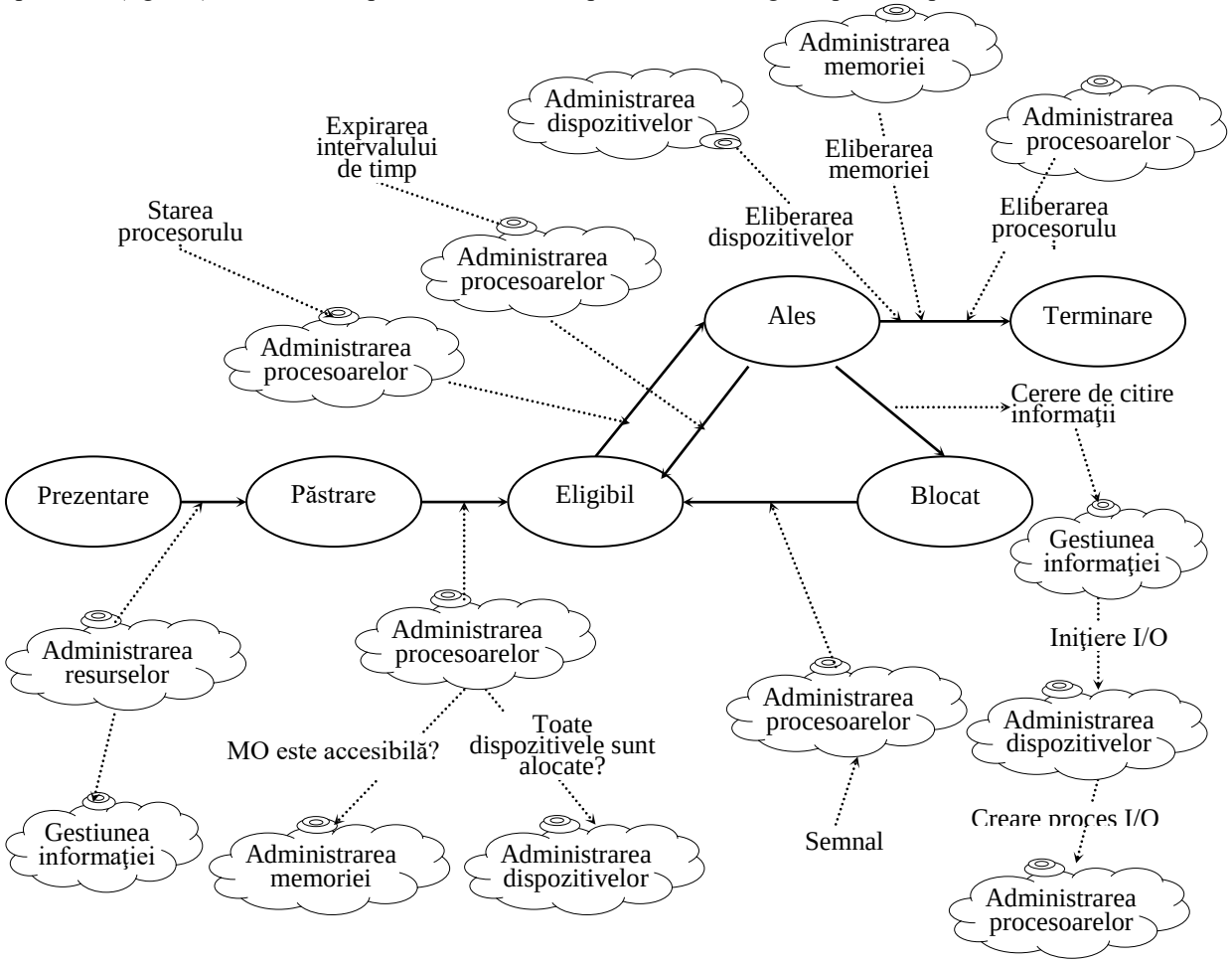


Fig. 1.7. Modelul stărilor unui proces şi componentele SO responsabile de tranziţii

1.2.2. Maşină ierarhică şi maşină extinsă

Astăzi este greu de închipuit că doar cu câteva decenii în urmă un utilizator era nevoit să programeze în zerouri şi unităţi, utilizând o "maşina goală". Chiar şi specialiştii nu prea iubesc să scrie programe într-un limbaj de asamblare sau (şi mai evident) utilizând doar instrucţiuni din setul, garantat de resursele fizice. Un program elaborat de un specialist poate fi de forma [1]:

1. Transferă C, B	Stabileşte C=B
2. Găseşte zona 80, X	Să se găsească 80 de octeţi de memorie liberi şi să se plaseze adresa zonei în X
3. Introdu date în X	Să se citească datele indicate în zona X
4. Compară X(2), '/'*	Coincide conţinutul primilor 2 octeţi ai zonei X cu '/*'?
*5. Dacă **da,** STOP*	Dacă coincid, salt la STOP

Instrucţiunile 1, 4 şi 5 sunt instrucţiuni standard în multe calculatoare contemporane. Însă execuţia corectă şi eficientă a instrucţiunilor 2 şi 3 poate solicita îndeplinirea a zeci, sute sau chiar mii de comenzi din setul standard, deoarece aceste instrucţiuni cer interacţiunea cu unele resurse cheie, cum ar fi memoria operativă şi dispozitivele de intrare-ieşire. Sistemul de operare asigură instrucţiuni pentru realizarea unor astfel de funcţii de administrare a resurselor: instrucţiunile 2 şi 3 sunt instrucţiuni ale **maşinii extinse** ele neavând echivalenţi în setul de instrucţiuni hardware. Sistemul de operare completează setul standard, realizat hardware, cu instrucţiuni de acest gen.

Setul de instrucţiuni realizat hardware împreună cu instrucţiunile suplimentare ale sistemului de operare formează **sistemul de comenzi al maşinii extinse**. Grafic conceptul de maşină extinsă poate fi reprezentat conform fig. 1.8. Sistemul de operare este executat pe maşina "goală", iar programele utilizatorului – pe maşina extinsă.

Să facem cunoştinţă acum cu modul în care sunt organizate într-un tot întreg componentele sistemelor de operare.

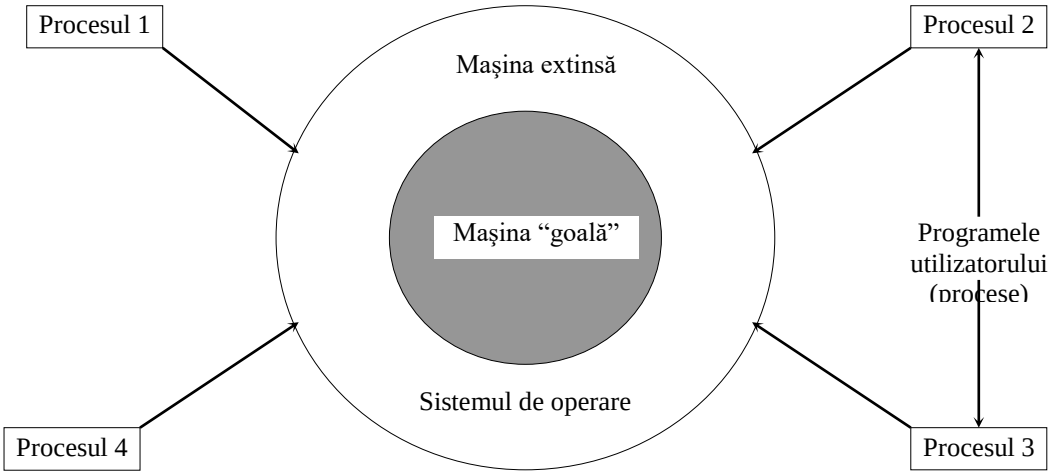


Fig. 1.8. Maşina extinsă

Primele sisteme de operare erau formate dintr-un singur program mare. Dar, odată cu sporirea complexităţii sistemelor, această abordare liniară conducea la dificultăţi serioase şi s-a propus să se utilizeze şi în acest domeniu conceptul de *maşină extinsă.* Acest concept, în cazul sistemelor de operare, poate fi utilizat în două nivele (fig. 1.9) şi conduce la noţiunea de ***maşină ierarhică*** [4]: primul nivel - funcţiile cheie, utilizate de majoritatea modulelor de sistem, pot fi realizate în cadrul unei *maşini extinse interne* şi nivelul doi - unele module pot fi executate în cadrul unei *maşini extinse externe*, analogic proceselor utilizatorului.

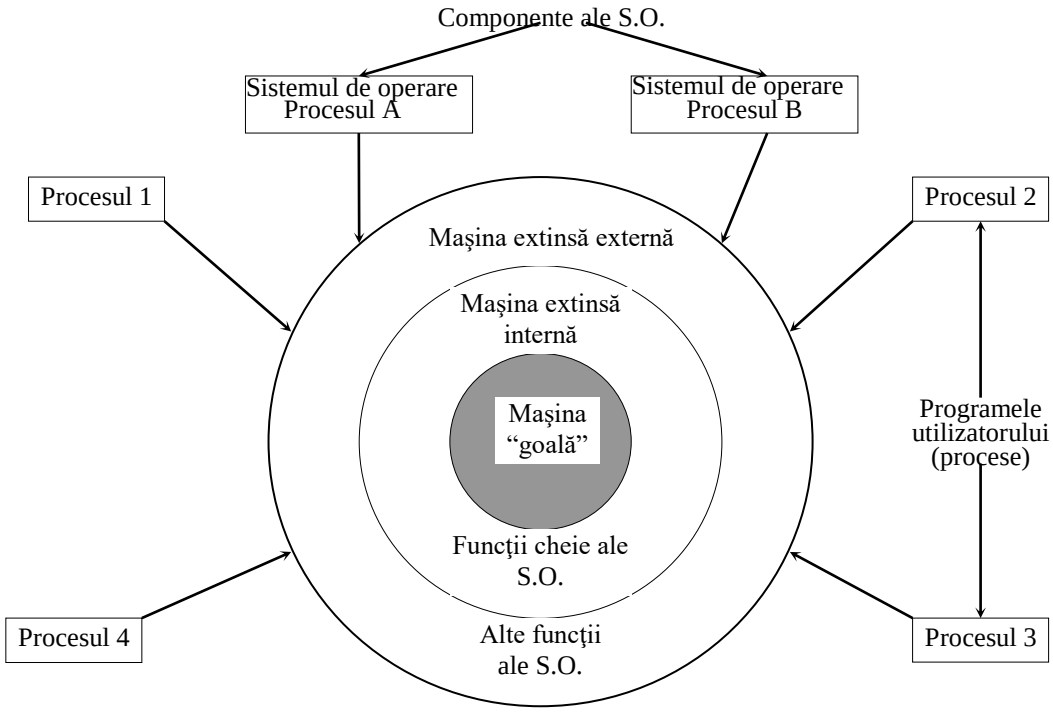


Fig. 1.9. Ilustrarea conceptului de maşină ierarhică

În această abordare ierarhică apare imediat problema alegerii corecte a nivelului de ierarhie pentru fiecare modul al sistemului de operare (în maşina extinsă internă, externă sau va fi prezent în calitate de proces extern). Pot fi evidenţiate subnivele ale celor două maşini extinse, iar interacţiunea proceselor sistemului de operare ne conduce la necesitatea introducerii mai multor *straturi ale proceselor*. Modulele sistemului, plasate în cadrul maşinii extinse, spre deosebire de modulele care aparţin straturilor proceselor, formează **nucleul** sistemului de operare. Concepţia maşinii ierarhice este pe larg utilizată în proiectarea programelor mari utilizând metodele, cunoscute sub denumirea *programare modulară* sau *programare structurală.* Nu există reguli stricte în privinţa numărului de nivele, amplasării modulelor pe nivele, componenţa nucleului. De obicei, nucleul conţine doar cele mai necesare şi evidente funcţii, celelalte, atunci când este posibil, vor fi prezente ca procese de sistem separate. O detalizare a conceptului de

maşină ierarhică este adusă în fig. 1.10. Procesele (incluse în dreptunghiuri) se adresează către funcţiile nucleului şi utilizează împreună sursele sistemului. Unele procese generează sau comandă alte procese (graniţa dintre ele este prezentată de liniile în zig-zag, care separă diferite straturi ale proceselor). Într-o realizare strict ierarhică modulele, situate într-un nivel oarecare, pot accesa (se pot adresa) numai resursele nivelelor inferioare.

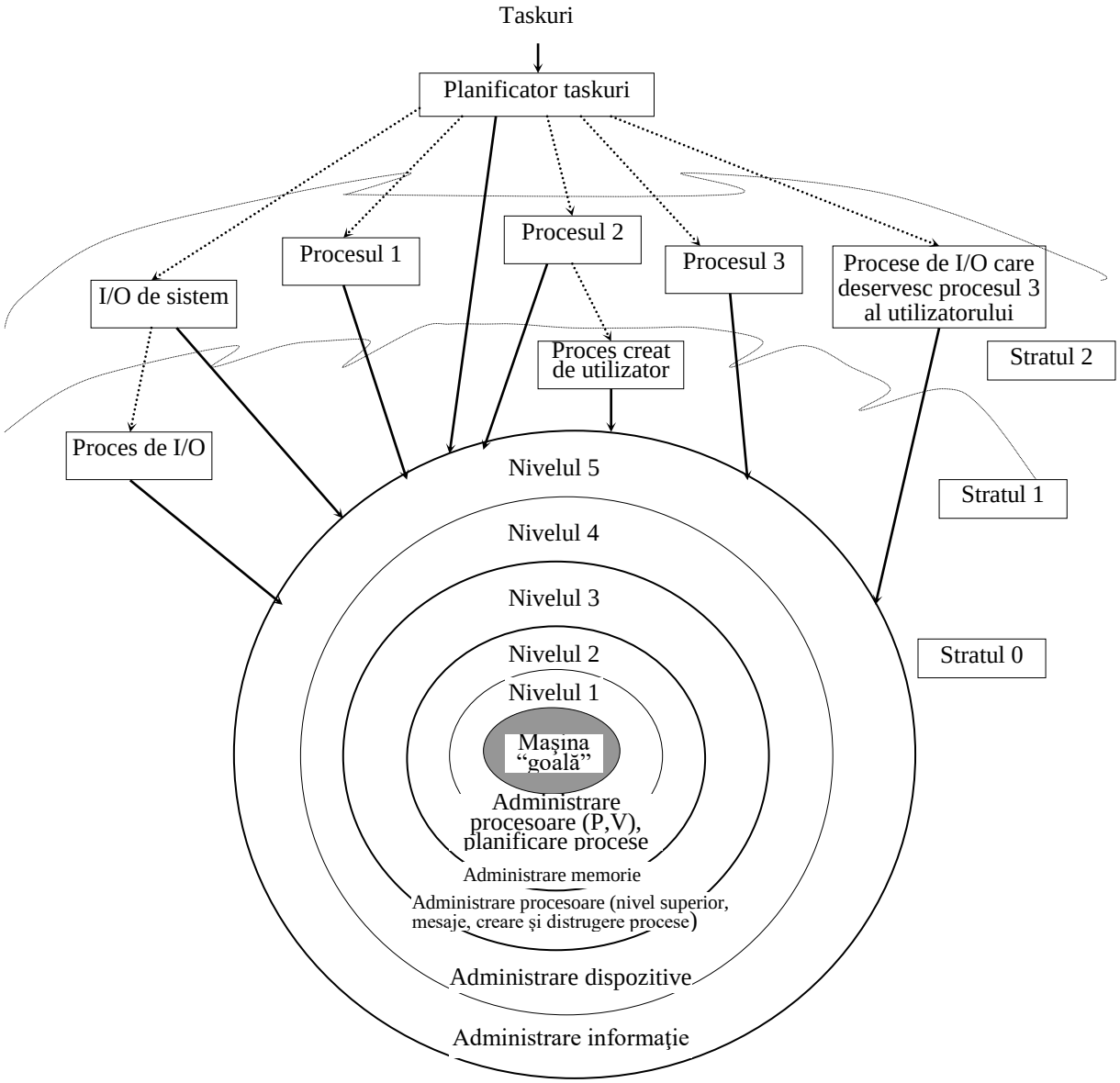


Fig. 1.10. Structura ierarhică a sistemului de operare

La nivelul 1 (nivelul cel mai inferior) sunt situate funcţiile de care au nevoie toate componentele administrării resurselor. Una dintre acestea este funcţia care urmăreşte repartizarea resurselor, funcţie, care la rândul său, solicită anumite mijloace de sincronizare. Aceste operaţii elementare sunt numite *P-operator* (ocuparea resursei sau cererea de acces) şi *V-operator* (eliberarea resursei). Sincronizarea se face printr-o tehnică de programare, numită *semafor*. Fiecărei resurse îi este ataşat un semafor. Atunci când în sistem apare o cerere pentru o resursă oarecare, pentru testarea semaforului respectiv este utilizat *P-operatorul*; dacă semaforul este "pe verde" (resursa este liberă), *P-operatorul* îl trece "pe roşu" (îl închide) şi returnează controlul. În caz contrar, procesul care a generat cererea este trecut în stare de aşteptare, pentru ca mai târziu, atunci când *V-operatorul* va elibera resursa şi va trece semaforul "pe verde", să acceseze resursa.

Amplasarea funcţiilor elementare pe nivele poate fi făcută în felul următor:

Nivelul 1. Administrarea procesoarelor (nivelul inferior)

- P - operatorii de sincronizare
- V - operatorii de sincronizare

- planificarea proceselor (aparatul de multiprogramare)

Nivelul 2. Administrarea memoriei
- alocarea memoriei
- eliberarea memoriei

Nivelul 3. Administrarea procesoarelor (nivelul superior)
- crearea şi distrugerea unui proces
- transmiterea şi recepţionarea mesajelor între procese
- lansarea unui proces
- oprirea unui proces

Nivelul 4. Administrarea dispozitivelor
- urmărirea stărilor tuturor dispozitivelor periferice
- planificarea intrărilor/ieşirilor
- iniţierea operaţiilor de intrare/ieşire

Nivelul 5. Administrarea informaţiei
- crearea şi distrugerea unui fişier
- deschiderea şi închiderea unui fişier
- citirea şi înscrierea unui fişier.

Nucleul sistemului de operare este format de subprogramele, care asistă execuţia proceselor. Pentru a decide care funcţii pot fi realizate în formă de procese separate este necesar să se stabilească funcţiile care pot fi executate independent şi în mod paralel cu procesele utilizatorului (nu se va câştiga nimic evidenţiind în procese separate funcţii care trebuie să fie îndeplinite secvenţial). Sistemele avansate permit crearea oricărui număr de procese, ceea ce este foarte comod pentru organizarea calculelor paralele sau a regimurilor de timp real. În cadrul tehnologiilor noi noţiunea de proces a fost substanţial modificată, introducându-se alte concepte (fire, thread en.), care exploatează într-un mod mai eficient ideile multitasking-ului şi multiprogramării.

1.2.3. Alte puncte de vedere

În compartimentele precedente au fost tratate sistemele de operare din diferite puncte de vedere, cum ar fi SO şi procesele, SO şi maşina extinsă sau SO şi maşina ierarhică. Există şi alte puncte de vedere asupra sistemelor de operare pe care un specialist ar trebui să le cunoască.

Pentru un utilizator obişnuit, convins că un calculator este doar un instrument care îl ajută în rezolvarea unor probleme din domeniul său de activitate, noţiuni cum ar fi administrarea memoriei cu paginaţie sau driverele dispozitivelor nu semnifică prea multe. Destinaţia principală a unui sistem de operare pentru această categorie de utilizatori este punerea la dispoziţie a unui set de programe care ar ajuta în formularea şi rezolvarea problemelor concrete. Abordarea sistemelor de operare din acest punct de vedere (abordare funcţională) poate conduce la confundarea lor cu unele programe, utile şi foarte importante cum sunt translatoare, bibliotecile, mediile integrate, etc. Pentru a evita posibilitatea apariţiei unei astfel de probleme aceste programe, de obicei, nu sunt considerate componente ale sistemului de operare.

Interfaţa sistemului de operare cu utilizatorul prezintă un interes aparte. Progresul în acest domeniu este spectaculos, dacă vom lua în consideraţie că în primele sisteme de operare utilizatorul era obligat să indice în mod explicit şi manual fiecare pas, oricât de nesemnificativ ar fi părut. Formularea paşilor cu ajutorul limbajului de control al lucrărilor (Job Control Language, JCL) nu a schimbat substanţial situaţia. Acest limbaj, nefiind agreat de utilizatorii simpli, care l-au denumit *limbaj păsăresc,* aşa şi nu a fost acceptat de către aceştia.

Conform JCL utilizatorul trebuie să încorporeze programul propriu într-un set de instrucţiuni, care indicau începutul, sfârşitul programului şi al datelor de intrare, paşii şi conţinutul concret al paşilor. JCL în principiu era un metalimbaj de programare (programare la nivel macro). Pentru mijloacele tehnice de la acea perioadă JCL a sporit substanţial eficienţa sistemelor de calcul, deşi au existat multe inconveniente, principalul fiind lipsa posibilităţii lucrului interactiv.

Microprocesoarele şi memoriile anilor 1970 au pus problema lansării pe piaţă a calculatoarelor personale (PC) cu toate consecinţele asociate. Una din consecinţe este şi interfaţa utilizator-calculator, sistemul de operare devenind până la urmă responsabil de aceasta. Interfaţa grafica a utilizatorului (Graphical User Interface - GUI) a apărut mai întâi ca un complement al sistemului de operare (pentru MS DOS - Windows 1, Windows 2 sau chiar Windows 3, de exemplu), pentru ca mai apoi să fie integrată în cadrul sistemului (Windows 95, Windows NT, etc.). Un sistem de operare nu este, în principiu, obligat să posede o interfaţă sofisticată, totul este determinat de baza tehnică utilizată şi

de necesităţile concrete. Oricum, un specialist trebuie să distingă aceste două noţiuni – sistemul de operare şi interfaţa utilizatorului.

1.3. Evoluţia sistemelor de operare

O analiză cronologică a dezvoltării sistemelor de operare este greu de realizat, deoarece multe din principiile foarte importante au fost realizate pentru prima dată cu mult înainte de a deveni unanim acceptate. De exemplu, conceptele de memorie paginată şi memorie virtuală au fost realizate pentru prima dată în 1959 în cadrul sistemului "Atlas" [5], fiind utilizate la mijlocul anilor 1960 în unele sisteme cu destinaţie specială, pentru ca în 1972 să fie preluate de firma IBM în cadrul familiei de calculatoare mari.

Primele sisteme erau caracterizate prin **prelucrarea secvenţială** a sarcinilor. Timpul de execuţie a programelor era relativ mare, instrumentele de depanare – primitive, fiecare programator îşi încărca în mod individual programul (pachetul de cartele perforate), apăsa butoane, controla conţinutul locaţiunilor de memorie, etc. (1950 – 1956).

Sporirea vitezei de calcul, dar şi a preţului calculatoarelor cerea o utilizare mai eficientă a timpului de calculator. Nu putea fi tolerată situaţia ca un calculator să "nu facă nimic", atunci când utilizatorul îşi încarcă în mod manual programul. Au fost propuse programe de monitorizare (monitoare), care treceau de la o lucrare la alta în mod automat, utilizatorul fiind responsabil de organizarea corectă a programelor în cadrul unui pachet – primele încercări de **prelucrare pe loturi** (1956 – 1959).

Odată cu creşterea complexităţii calculatoarelor, îndeosebi în ceea ce consta administrarea dispozitivelor periferice, au fost propuse **sisteme** supervizoare (**executive**), care se aflau în memoria calculatorului în mod constant şi acordau utilizatorilor servicii în gestiunea operaţiilor de intrare/ieşire (1959 – 1963). În aceste sisteme de operare erau realizate şi o serie de facilităţi noi, cum ar fi controlul unor posibile încercări din partea programului de a încălca restricţiile existente în sistem, culegerea informaţiilor de evidenţă, etc.

Au urmat apoi sistemele cu **multiprogramare** menite la început să rezolve problema armonizării vitezei de calcul a unităţii centrale şi a perifericelor. Drept consecinţă, au apărut o mulţime de limbaje de control a lucrărilor, a fost realizată o standardizare substanţială a operaţiilor de intrare-ieşire.

După 1965 au apărut primele sisteme cu **partajare a timpului** (time sharing), au fost propuse sisteme sofisticate de administrare a informaţiei (sisteme de gestiune a datelor sau sisteme de fişiere, File Systems). Principiul **time sharing** oferea posibilitatea lucrului interactiv a mai multor utilizatori pe un singur calculator, fiecărui utilizator în mod ciclic acordându-i-se un interval anume de timp (cuantă de timp) şi, datorită vitezei mari de calcul a unităţii centrale, creându-i-se impresia posesiei tuturor resurselor calculatorului.

Memoria virtuală şi **maşinile virtuale** sunt nişte principii care nici până astăzi nu au fost exploatate până la capăt. Progresele ultimilor ani în domeniul resurselor tehnice au permis implementarea acestor principii nu numai în cadrul sistemelor de calcul mari, ci şi pentru calculatoarele personale. Specificaţiile sistemelor de operare au fost în mare măsură standardizate, diversitatea SO devine tot mai mică, mulţi specialişti exprimându-şi îngrijorarea de o posibilă monopolizare a domeniului într-un viitor apropiat. Evident, aceasta nu poate să sugereze nici într-un caz ideea că studierea principiilor de bază (mai vechi şi mai noi) ale sistemelor de operare ar fi de prisos, ca şi familiarizarea sau chiar cercetarea minuţioasă a unor sisteme existente, nicidecum nu poate însemna, în special pentru un specialist, pierderea interesului faţă de analiza şi concepţia sistemelor de operare.

O prezentare succintă a evoluţiei sistemelor de operare facilitează înţelegerea caracteristicilor actuale ale acestora şi a termenilor deja introduşi.

1.3.1. De la "poartă deschisă " la tratarea pe loturi

Primele calculatoare nu dispuneau de sisteme de operare. Fiecărui utilizator i se rezerva pentru un timp determinat calculatorul cu toate resursele acestuia. Interacţiunea era directă, programul şi datele fiind introduse în mod manual sub formă de zerouri şi unităţi. Utilitarele care au apărut aveau destinaţia de a asista elaborarea programelor (asambloare, compilatoare, etc.) sau de a facilitata operaţiile de intrare-ieşire.

Acest mod de exploatare, numit "poartă deschisă", era de o eficacitate minimă, dispozitive foarte costisitoare fiind utilizate ineficient. Din această cauză la sfârşitul anilor 1950 au apărut primele "**monitoare de înlănţuire**" - programe care permiteau executarea secvenţială a unui set de lucrări, pregătite anticipat, trecerea de la o lucrare la alta fiind realizată în mod automat.

Funcţia principală a unui atare sistem era gestiunea resurselor: memoria, procesorul, dispozitivele perifierice. Automatismul acestei gestionări implică o funcţie de protecţie a setului de lucrări contra unor riscuri perturbatorii in caz de eroare:

- limitarea timpului de ocupare a procesorului pentru a evita blocarea sistemului atunci când un program conţine o buclă infinită;

- administrarea corectă a intrărilor-ieşirilor pentru a evita blocajele în utilizarea perifericelor;
- protecţia zonei de memorie rezervate monitorului pentru a împiedica modificarea accidentală a acestuia.

Deşi utilizarea monitoarelor de înlănţuire a ameliorat notabil randamentul utilizării procesorului, acest randament rămânea foarte scăzut din cauza că procesorul nu era folosit în timpul operaţiilor de intrare-ieşire. O soluţionare ar fi constat în utilizarea a două calculatoare - unul (principal) pentru executarea programelor şi altul (auxiliar) pentru operaţiile de intrare-ieşire. O planificare adecvată a executării lucrărilor permitea utilizarea celor două calculatoare în paralel, dând posibilitatea sporirii la maximum a randamentului calculatorului principal.

1.3.2. Multiprogramarea

Progresul tehnologic şi conceptual al anilor 1960 - 1970 a permis excluderea unor limitări caracteristice sistemelor de tratare pe loturi. Astfel, au fost introduse procesoare specializate pentru operaţiile de intrare-ieşire (unităţile de schimb sau canalele), care permiteau eliberarea procesorului central de gestionarea dispozitivelor de intrare-ieşire. Introducerea principiului multiprogramării cu partajarea memoriei între mai mulţi utilizatori au permis o utilizare şi mai bună a procesorului central.

1.3.2.1. Organizarea intrărilor - ieşirilor în zone de memorie tampon

Un canal este un procesor specializat în executarea autonomă a operaţiilor de intrare-ieşire, paralel cu procesul de prelucrare a informaţiilor. Viteza de lucru a organelor periferice este relativ mică din cauza unor dispozitive mecanice care intră în componenţa acestora. Pentru excluderea influenţei perifericelor asupra vitezei de lucru a sistemului de calcul s-a propus să se păstreze în memorie în anumite zone tampon datele de intrare (care vor fi utilizate la necesitate) şi rezultatele (care vor fi imprimate în mod autonom) mai multor lucrări. Această situaţie este prezentată în fig.1.11.

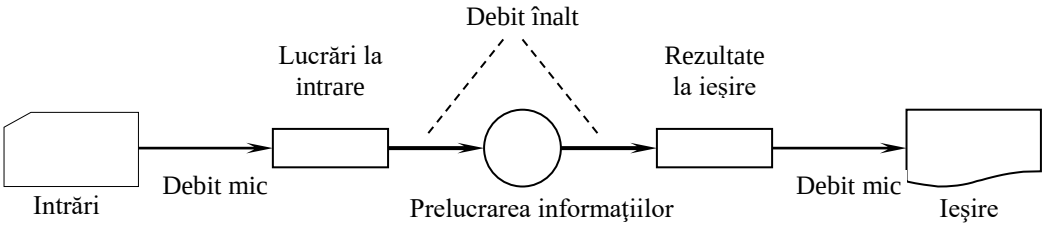


Fig.1.11. Buferizarea intrărilor-ieşirilor

Pentru a exclude utilizarea ineficientă a memoriei operative prin formarea zonelor-tampon, acestea erau organizate în memoria secundară de capacitate înaltă, transferurile între aceste două nivele ale memorie fiind de asemenea comandate de un canal.

Deşi utilizarea memoriilor tampon prezintă o serie de avantaje privind sporirea randamentului dispozitivelor calculatorului, totuşi două momente negative pot fi menţionate:

- atunci când lucrarea în curs de execuţie are nevoie de date unitatea centrală rămâne inactivă pe toată perioada citirii acestora;
- o lucrare de scurtă durată, sosită în timpul execuţiei unei lucrări "lungi", trebuie să aştepte terminarea acesteia din urmă.

1.3.2.2. Multiprogramarea

Ultimele două momente au condus la ideea utilizării unui mod de funcţionare a SO în care:

- o lucrare aflată în starea eligibil să poată folosi unitatea centrală, eliberată de o lucrare care trece în aşteptarea datelor,
- unitatea centrală să poată modifica timpul său de alocare înainte de terminarea unei lucrări în scopul satisfacerii cerinţelor legate de timpul de răspuns.

În acest caz este necesar ca valoarea timpului de comutare a unităţii centrale să fie mai mică în raport cu durata unui transfer între nivelele memoriei. Aceasta implică prezenţa simultană în memoria operativă a mai multor programe sau părţi de programe. Acest mod de funcţionare este numit **multiprogramare** [6] şi este prezentat în figura 1.12.

Schema din figura 1.12 evidenţiază două momente importante: rolul principal al memoriei operative (în SO anterioare acest rol era deţinut de către unitatea centrală) şi fluxul informaţional între memoria operativă şi cea secundară. Drept consecinţă, volumul memoriei operative şi viteza de transfer între cele două nivele ale memoriei devin caracteristici determinante ale performanţei sistemului.

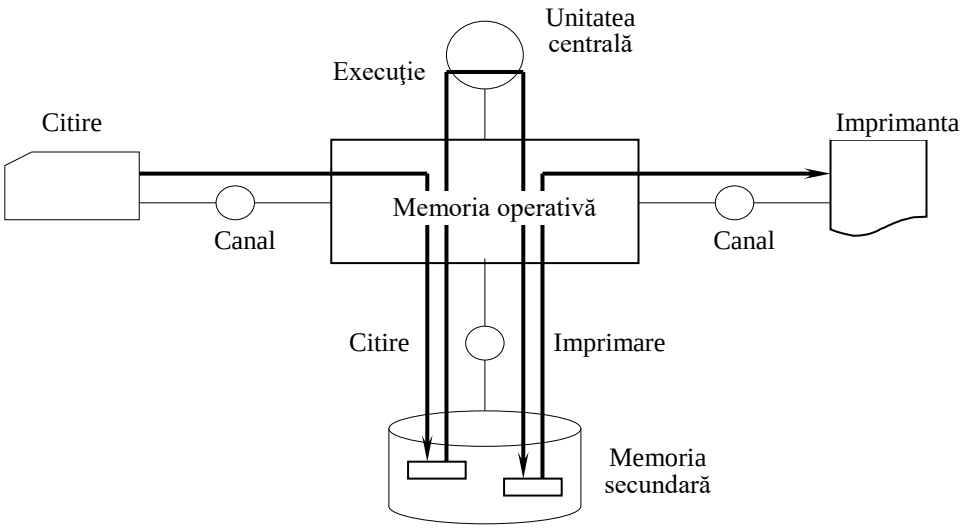


Fig. 1.12. Fluxul informaţional într-un sistem cu multiprogramare

Principalele avantaje şi restricţii ale multiprogramării pot fi prezentate în rezumat după cum urmează:

- un sistem cu multiprogramare este mai complicat de conceput și implementat pentru că el trebuie să asigure partajarea memoriei şi protecţia reciprocă a programelor;
- multiprogramarea solicită dispozitive speciale pentru relansarea programelor şi protecţia memoriei;
- un sistem cu multiprogramare asigură o utilizare mai uniformă a resurselor: unitatea centrală, memoria, organele de intrare-ieşire;
- multiprogramarea permite reducerea timpului de răspuns în cazul lucrărilor de durată mică într-un sistem cu prelucrare secvenţială.

1.3.2.3. Partajatarea timpului

Un sistem de operare cu partajarea timpului trebuie să garanteze fiecărui utilizator un timp acceptabil de răspuns. Acest rezultat este obţinut prin alocarea succesivă a procesorului pentru tranşe de timp (*cuante*) relativ mici programelor utilizatorilor. Viabilitatea unei asemenea tehnici este legată de caracteristicile lucrului interactiv. Activitatea unui utilizator conţine două componente: *timpul de reflecţie (gândire)*, în care utilizatorul elaborează, propune subiecte de lucru, introducând în calculator informaţii şi *timpul de aşteptare,* când aşteaptă prestarea serviciului solicitat. Prima componentă este de o durată medie mult mai mare decât a doua şi sistemul poate să servească simultan mai mulţi utilizatori, folosind timpul mort, datorat perioadei de reflecţie.

Ca exemplu, fie un sistem cu partajarea timpului, care deserveşte 1000 utilizatori cu un comportament mediu identic. Admitem că durata timpului de gândire este în medie de 9 ori mai mare decât a timpului de aşteptare, acesta din urmă reprezentând 10% din timpul total. Avem în medie 100 utilizatori activi (care se află în aşteptarea unui răspuns la un serviciu cerut). Presupunem că valoarea cuantei este de 5 ms. Dacă executarea unei cereri durează o cuantă, timpul de răspuns va fi de ordinul unei jumătăţi de secundă.

Am admis în acest exemplu că toate programele utilizatorilor activi se află în memoria principală: timpul de comutare între programele a doi utilizatori se reduce la timpul de realocare a unităţii centrale, care poate fi neglijat în comparaţie cu valoarea cuantei. Multiprogramarea devine obligatorie, dacă ţinem cont de raportul dintre acest timp şi timpul necesar pentru încărcarea unui program în memoria secundară. Situaţia este de fapt şi mai complicată: volumul memoriei principale nu permite aflarea aici a tuturor programelor utilizatorilor activi. Este posibil ca o informaţie să fie lipsă în memoria operativă atunci când unitatea centrală are nevoie de ea. Rolul sistemului de operare este de a minimiza pe cât este posibil probabilitatea apariţiei unui atare eveniment.

Dezvoltarea sistemelor de operare cu partajarea timpului a pus în evidenţă importanţa interacţiunii om-calculator. Acest aspect a fost mult timp neglijat, dar cei care au procedat altfel au avut doar de câştigat.

Apariţia microprocesoarelor cu posibilităţi mari de calcul şi preţuri tot mai mici a permis pe de o parte utilizarea la scară largă a calculatoarelor personale, iar pe de altă parte, satisfacerea cerinţelor utilizatorilor sistemelor informatice de partajare a resurselor fizice şi logice distribuite geografic. Au apărut noi tipuri de sisteme informatice: calculatoarele personale, reţelele locale şi reţelele globale.

Conceptele şi tehnicile elaborate pentru sistemele centralizate au rămas valabile şi în cazul serverelor sau sistemelor de operare pentru calculatoarele personale. Repartizarea distribuită a informaţiei, precum şi transmiterea

ei la distanţe mai mari sau mai mici a ridicat probleme noi, cum ar fi coordonarea activităţilor la distanţă sau menţinerea coerenţei informaţiilor distribuite. Toate acestea au trebuit să fie luate în consideraţie la elaborarea sistemelor de operare de rețea.

1.3.3. Windows, Unix şi alte sisteme

Paralel cu evoluţia tehnică şi funcţională a sistemelor de operare a avut loc şi o importantă evoluţie conceptuală, care a permis o mai bună înţelegere a funcţionării SO şi a condus la elaborarea unor metode proprii de concepere.

Anul 1964 poate fi considerat de debut pentru cercetările ştiinţifice în domeniul sistemelor de operare, cercetări care au fost generate de rezultatele obţinute în sfera tehnică - primele sisteme cu partajare a timpului (Thor, CTSS), anunţarea seriei IBM 360 şi introducerea canalelor de intrare-ieşire, prima memorie paginată (Atlas), etc.

În perioada 1965-1968 au fost fundamentate concepte teoretice importante, necesare pentru conştientizarea gestionării activităţilor fizice sau logice paralele: proces secvenţial, excluderea mutuală, sincronizarea, semaforul. Acestea au fost aplicate cu succes la elaborarea sistemelor de operare: sistemul THE (1967) utilizează semafoarele, sistemul Multics (1964-1970) conţine un nucleu de gestionare a proceselor. Metode şi utilitare destinate sistemelor distribuite sunt propuse la sfârşitul anilor 1970. Utilitarele de sincronizare încep să fie introduse în limbajele de programare. Sunt propuse primele metode de specificare şi control al validităţii sincronizărilor.

Problema definirii informaţiei a fost pusă iniţial în seama limbajelor de programare: fiecare limbaj definea universul său de obiecte accesibile şi mijloacele efective de accesare. Sistemul de operare trebuia doar să implementeze obiectele definite de limbajul de programare în memoria fizică, direct adresabilă.

Restricţiile de capacitate şi preţ a memoriei principale a condus foarte repede la modificarea modului de utilizare a memoriei secundare şi introducerea unor mecanisme de schimb de informaţii între aceste două nivele ale memoriei. Prima memorie paginată apare în 1962 (Atlas), tot atunci compilatoarele limbajului Algol 60 folosesc paginaţia virtuală pentru gestionarea programelor în curs de execuţie. În 1965 este fundamentată noţiunea de segmentare, deşi încă în 1959 sistemul Burroughs B5000 utiliza un mecanism de adresare logică prin unităţi de volum variabil - segmente. În 1966 este propus un model de arhitectură pentru calculatoarele cu partajare a timpului, implementat în IBM 360/370. În acest calculator segmentele sunt doar simulate printr-o dublă paginaţie, din care cauză noţiunea de segmentare va rămâne pentru un timp puţin înţeleasă de comunitatea informatică. Problemele partajării şi protecţiei informaţiei, alocării resurselor conduc la noţiunea de memorie virtuală, care permite izolarea mecanismelor de gestionare a ansamblului “memorie principală - memorie secundară”, lăsând utilizatorilor simpli impresia unui spaţiu de adrese uniform (contiguu). Tot atunci au fost introduse noţiunile de modularitate şi structurare ierarhică, obiect, etc.

Astăzi sistemele de operare în sensul tradiţional de concepere sunt mai mult obiectul unor elaborări industriale, decât de cercetare. Aceasta se datorează atât existenţei unei rezerve extraordinare de metode şi algoritmi, cât şi standardizării stricte a funcţiilor şi interfeţelor sistemelor de operare.

1.3.3.1. SO UNIX şi sistemele deschise

Sistemul de operare UNIX, primul sistem “mobil” care asigură un mediu fiabil de dezvoltare şi utilizare a softului de aplicaţie, este fundamentul practic de elaborare a sistemelor fizico-logice deschise. Implementarea largă a SO UNIX a permis trecerea de la declararea conceptului sistemelor deschise la dezvoltarea în practică a acestui concept. Este greu de supraestimat şi influenţa activităţilor de standardizare a interfeţelor SO UNIX asupra dezvoltării domeniului sistemelor deschise. Cu toate acestea, pot fi evidenţiate câteva versiuni ale SO UNIX, care diferă atât prin realizare, cât şi prin interfeţe şi semantică, chiar dacă, odată cu dezvoltarea procesului de standardizare, aceste diferenţe devin tot mai nesemnificative.

Modulele sursă ale SO UNIX au fost scrise de către colaboratorii companiei AT&T şi timp îndelungat drepturile de autor, ca şi drepturile de licenţiere, au aparţinut acestei companii. Mai târziu, din mai multe motive (complexitate tehnică în elaborarea şi mentenanţa acestui produs program complicat, probleme juridice, etc.), compania AT&T a creat entitatea USL (UNIX System Laboratories), responsabilă exclusiv de dezvoltarea şi susţinerea SO UNIX.

Compania USL a propus versiunea UNIX System V 4.0, care a devenit standardul de-facto şi baza mai multor versiuni UNIX, create de producătorii de staţii de lucru şi servere. În ultimul succes al USL în calitate de filială a firmei AT&T - versiunea SVR 4.2 - pentru prima oară în istoria SO UNIX a fost realizat mecanismul proceselor “lejere” (fire, en. thread), care erau executate în baza unei memorii virtuale comune şi permitea exploatarea conceptului “arhitectură multiprocesorală simetrică” în care mai multe procesoare au drepturi egale de accesare a memoriei operative comune.

În anul 1993 USL a fost absorbită de compania Novell, devenind astăzi departament al acesteia, marca înregistrată UNIX fiind cedată consorţiumului X/Open. La începutul anului 1995 compania Novell a anunţat o variantă nouă a SO UixWare 2.0 bazată pe System V 4.2. Era un SO cu un sistem de fişiere foarte fiabil, fiind admis

accesul la fişierele păstrate pe serverele NetWare, administratorul avea la dispoziţie o interfaţă grafică bine pusă la punct, etc.

Pe parcursul a mai multor ani sistemul de operare de bază al companiei Sun a fost UNIX BSD. Însă, începând cu SunOS 4.0, s-a trecut la System V 4.0, firma Sun introducând o serie de modificări şi extensii în această versiune. Ca exemplu, Sun a implementat paralelizarea programelor pentru sistemele multiprocesorale simetrice. Solaris este o interfaţă a lui SunOS cu mijloace suplimentare GUI şi resurse de nivel înalt pentru organizarea lucrului în reţea (de exemplu, apelarea procedurilor la distanţă – RPC). Şi la baza SO HP/UX, DG/UX şi AIX se află SVR 4.x din care cauză setul de bază al funcţiilor de sistem şi biblioteci este acelaşi. Variantele SO UNIX, propuse de compania SCO şi destinate exclusiv platformelor Intel, sunt bazate pe modulele iniţiale ale System V 3.2, fiind compatibile cu toate standardele de bază

Open Software Foundation (OSF) a fost prima companie comercială, care a încercat elaborarea SO UNIX în baza micronucleului Mach. A fost creat sistemul de operare OSF-1, care nu era „curat" (în sens de licenţiere), deoarece folosea o parte a modulelor iniţiale din SVR 4.0.

Variantele SO UNIX, propuse de Universitatea din California, sunt o alternativă reală pentru UNIX AT&T. De exemplu, UNIX BSD 4.2 era pus la dispoziţie în module sursă şi folosită chiar în fosta URSS pe calculatoarele de tip CM-3, CM-4. Grupul BSD a influenţat enorm dezvoltarea sistemelor de operare UNIX, printre realizări amintim mult controversatul SO UNIX BSD 4.4, construit în baza principiilor „micronucleare", sistemul FreeBSD şi altele.

Nu putem să nu amintim aici şi de realizarea originală a SO UNIX pentru platformele Intel, propusă de Torvald Linus – LINUX. Este un sistem de operare foarte popular în mediul studenţesc, care are şi una din cele mai populare teleconferinţe în Internet.

1.3.3.2. Standarde UNIX

Odată cu ieşirea SO UNIX pe piaţă şi creşterea substanţială nu numai a numărului de utilizatori, ci şi a numărului de specialişti din toate colţurile lumii, care propun modificări şi chiar variante proprii, a apărut necesitatea elaborării unor standarde, care ar asigura compatibilitatea.

Unul dintre primele standarde de-facto a fost cel publicat de USL pentru versiunea SO UNIX System V Release 4 - System V Interface Definition (SVID). Majoritatea variantelor comerciale respectau standardul SVID. Evident, SVID fiind un document de firmă, publicat fără discuţii publice nu putea fi adoptat ca și standard oficial.

Paralel exista direcţia BSD (Berkeley Standard Distribution), susţinută de comunitatea universitară. Deşi majoritatea realizărilor comerciale se bazau pe Sistem V, UNIX BSD era foarte popular în universităţi din care cauză a fost elaborat un standard, care a unit practic AT&T cu BSD. Acest lucru a fost început de asociaţia programatorilor profesionişti din cadrul UniForum (Sistemele Deschise) şi continuat de grupurile de lucru POSIX (Portable Operating System Interface). Cel mai popular standard, adoptat de ISO la recomandarea IEEE, POSIX 1003.1 defineşte cerinţele minime pentru componentele unui sistem de operare.

Organizaţia internaţională X/Open, care activează în domeniul elaborării şi propagării ideilor sistemelor deschise, culege şi sistematizează standardele de-jure şi de-facto de importanţă industrială în aşa numitul X/Open Common Application Environment (CAE). Specificaţiile interfeţelor componentelor, care formează CAE, sunt publicate în X/Open Portability Guide (XPG).

Pentru lumea UNIX este foarte important şi standardul limbajului de programare C, adoptat mai întâi de ANSI şi apoi de ISO. În acest standard sunt specificate, în afara limbajului C, bibliotecile obligatorii într-o realizare standard. Deoarece chiar de la apariţie limbajul C, sistemele de programare erau strâns legate de UNIX, componentele bibliotecilor standard corespundeau exact mediului standard al SO UNIX.

Mai menţionăm standardul de-facto SPARC Complience Definition, propus de organizaţia SPARC International, propunerea organizaţiei 88/Open pentru procesoarele RISC Motorola, standardul sistemului de ferestre, susţinut de X Consorţium (Institutul de Tehnologie din Massachussets) şi OSF/Motif, elaborat de Open Software Foundation.

1.3.3.3. Sisteme de operare cu micronucleu

Micronucleul este partea minimă principală a unui sistem de operare, folosită pentru asigurarea modularităţii şi transportabilităţii. Noţiunea de micronucleu a fost introdusă de compania Next prin sistemul de operare cu micronucleul Mach. Nucleul acestui sistem de operare, de dimensiuni mici, în jurul căruia se situau subsistemele executate în regim user, trebuia să asigure o flexibilitate şi modularitate foarte înaltă. Dar în realitate acestea au fost umbrite de prezenţa serverului monolit, care realiza sistemul de operare UNIX BSD 4.3, ales de compania Next în calitate de nivel superior pentru micronucleul Mach. Totuşi, utilizarea micronucleului Mach a permis introducerea administrării mesajelor şi a unei serii de funcţii de serviciu orientate pe obiecte, în baza cărora a fost creată o interfaţă grafică elegantă a utilizatorului cu mijloace simple de configurare, administrare şi dezvoltare program.

Următorul SO cu micronucleu a fost MS Windows NT, în care momentul principal declarat era, în afara modularităţii, transportabilitatea. Acest sistem de operare poate fi utilizat în sistemele mono- şi miltiprocesor, bazate pe procesoarele Intel, Mips, şi Alpha. Mai mult, deoarece NT trebuia să execute şi programele scrise pentru DOS, Windows, OS/2 şi SO, compatibile cu standardele Posix, compania Microsoft a folosit modularitatea abordării „micronucleare" pentru crearea unei structuri generalizate, care nu repetă sistemele de operare existente, fiecare SO fiind emulat printr-un modul separat sau printr-un subsistem.

Au aderat la tehnologia „micronucleară" şi companiile Novell/USL, Open Software Foundation (OSF), IBM, Apple şi altele. Unul din concurenţii principali ai lui NT în domeniul SO cu micronucleu sunt Mach 3.0, creat în Universitatea Carnegy-Mellon, şi Chorus 3.0 al companiei Chorus Systems.

Un exemplu practic

Scopul exerciţiilor de mai jos [7] este de a pune în evidenţă influenţa evoluţiei istorice a sistemelor de operare asupra unor caracteristici ale acestora.

Vom considera un calculator periferia căruia este un dispozitiv de introducere a cartelelor perforate (1 000 cartele pe minut) şi o imprimantă (1 000 linii pe minut). O *lucrare medie* este definită astfel:

- citeşte 300 de cartele,
- utilizează procesorul 1 minut,
- tipăreşte 500 de linii.

Se presupune, că toate lucrările au caracteristici identice cu ale *lucrării medii.*

Definim două caracteristici de performanţă a sistemului:

- debitul mediu D - numărul de lucrări executate într-o oră,
- randamentul unităţii centrale ŋ - partea din timpul total de utilizare a unităţii centrale în care aceasta execută lucru util, altul decât gestionarea perifericelor.

Exerciţiul 1. **Presupunem că periferia este gestionată de către unitatea centrală**. Să se calculeze *D* şi *ŋ* în următoarele ipoteze de funcţionare:

- sistemul este exploatat în regim "poartă deschisă"; durata unei sesiuni de exploatare este de 15 minute,
- sistemul este exploatat utilizând un monitor cu înlănţuire secvenţială a lucrărilor.

Exerciţiul 2. **Presupunem că periferia este gestionată de un calculator separat**, care conţine o bandă magnetică în care sunt introduse datele la intrare şi o bandă magnetică pentru ieşire (tipărirea se va efectua de pe această bandă). Datele la intrare sunt introduse în calculatorul principal de pe banda magnetică, acesta producând datele la ieşire pe bandă. Timpul necesar citirii de pe şi scrierii pe benzi nu se va lua în consideraţie. Timpul de transfer a benzilor de pe un calculator pe altul este de 5 minute în fiecare sens; se presupune că o bandă va grupa un lot de 50 de lucrări (v. fig. 1.13).

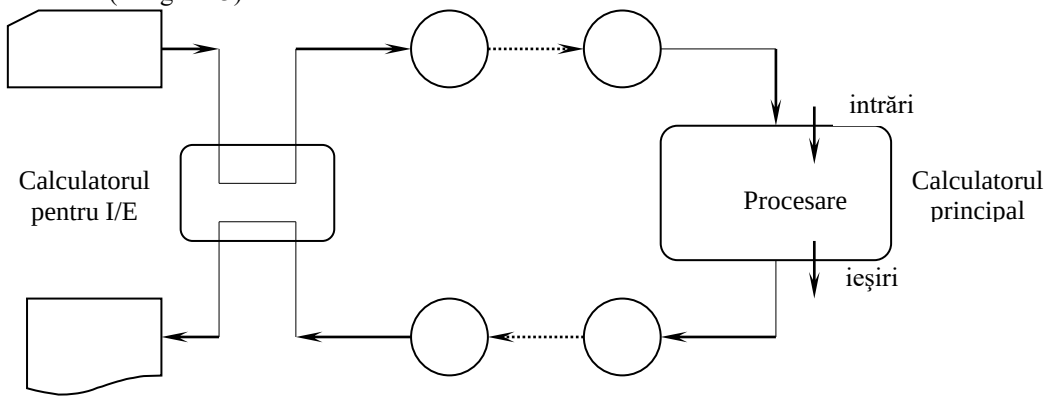


Fig. 1.13. Exemplificarea cazului 2.

Presupunem că intensitatea sosirii lucrărilor este suficientă pentru a ocupa tot timpul calculatorului central. Calculaţi *D* şi *ŋ*.

Stabiliţi modul de planificare a seturilor de lucrări şi calculaţi timpul mediu de aşteptare a unui utilizator (timpul din momentul predării lucrării şi până la recepţionarea rezultatelor). Vom admite, că lucrările sosesc într-un ritm regulat, că timpul necesar formării unui program (pregătirea unui complet de cartele perforate corespunzătoare unui program) este 10 minute şi timpul distribuirii rezultatelor unui lot (decuparea şi trierea listelor utilizatorilor) la fel este 10 minute.

Exercițiul 3. **Periferia este administrată de un canal de intrare-ieşire**. Sistemul este monoprogram, iar monitorul de înlănţuire permite unităţii centrale să prelucreze o lucrare paralel cu citirea următoarei şi tipărirea precedentei lucrări. Calculaţi în aceste condiţii D şi η. Aceeaşi întrebare dacă lucrarea medie presupune citirea a 1200 de cartele şi imprimarea a 1500 linii într-un minut de utilizare a unităţii centrale.

Exercițiul 4. **Operaţiile de intrare-ieşire sunt gestionate cu ajutorul unei memorii-tampon** (bufer) pe discul fix (spooling pentru citire şi imprimare). Lucrarea medie este cea definită în punctul 3.

Presupunem că o cartelă conţine 80 octeţi, iar o linie de imprimare 120. Care este capacitatea minimă a memoriilor-tampon pentru citire şi imprimare necesare pentru ca unitatea centrală să fie folosită la randamentul maxim? Care va fi în acest caz debitul lucrărilor?

Intensitatea sosirii lucrărilor şi capacitatea memoriei-tampon pentru citire sunt cele din a), iar memorie-tampon pentru tipărire este de 2 Mo. Care este randamentul unităţii centrale?

2. TEHNICI DE EXECUŢIE ŞI COMUNICAŢIE

Mecanismele, care dau posibilitatea unui calculator să interacţioneze cu mediul exterior, pe de o parte, şi să manipuleze mai multe sarcini, pe de alta, formează baza oricărui sistem de operare. Cunoaşterea detaliată a structurii şi modului de funcţionare a acestor mecanisme este obligatorie pentru un specialist în domeniu.

2.1. Modul secvenţial de execuţie a unui program

Nucleul unui calculator este format dintr-o memorie adresabilă, în care sunt păstrate instrucţiunile şi datele programului, şi procesorul, care interpretează instrucţiunile. Instrucţiunile şi datele se află în segmente distincte, prin segment înţelegând o mulţime de informaţii, desemnate şi manipulate ca un tot întreg, care ocupă în memorie o suită contiguă de amplasamente. Vom presupune că segmentele care conţin programul nu pot fi modificate în timpul execuţiei acestuia.

Execuţia instrucţiunilor, care formează un program secvenţial, conduce la evoluţia stării calculatorului. Evoluţia stării calculatorului poartă un caracter discret, starea este observabilă doar în momente de timp numite puncte de observare, care corespund, de obicei, începutului şi sfârşitului unei instrucţiuni. Unele instrucţiuni mai complexe pot conţine puncte observabile intermediare.

Starea calculatorului este definită de starea procesorului şi starea memoriei. Starea memoriei este determinată de conţinutul locaţiunilor segmentelor încărcate în memorie. Starea procesorului este definită de conţinutul unui set de registre (programabile şi interne), ataşate procesorului.

2.1.1. Noţiuni fundamentale

Orice program secvenţial constă dintr-o mulţime de proceduri, care se pot apela reciproc. Fiecărei proceduri îi este asociat un segment distinct de procedură. Datele sunt reprezentate de asemenea prin segmente, care pot fi proprii unei proceduri sau partajate între mai multe proceduri.

Numim **activitate** *fenomenul care rezultă din execuţia neîntreruptă a unei proceduri unice*. În acest fel, execuţia unui program secvenţial constă dintr-un lanţ de activităţi.

Numim **context** al unei activităţi ***mulţimea informaţiilor accesibile procesorului în cursul acestei activităţi***. Contextul activităţii este compus din contextul procesorului (registrele programabile şi interne) şi contextul memoriei (segmentul procedurii şi segmentul datelor). Trecerea de la o activitate la alta este realizată de instrucţiuni speciale: **apelarea** şi **returul** din procedură, care realizează *comutarea contextului*.

Exemplul 2.1. O procedură p (apelantă) provoacă execuţia unei proceduri q (apelate) cu ajutorul unei secvenţe de apel care conţine următoarele etape:

- Pregătirea parametrilor, transmişi de către procedura p procedurii q,
- Salvarea parţială a contextului lui p, necesar la retur,
- Înlocuirea contextului procedurii p prin contextul lui q.

La revenire din procedura q avem o schemă practic simetrică, doar contextul lui q fiind pierdut:

- Pregătirea rezultatelor, transmise de către procedura q procedurii p,
- Restabilirea contextului procedurii p, salvat înainte de apel.◄

Exemplul 2.2. În cazul funcţionării în **corutine** (proceduri simetrice, recursie mutuală – două proceduri se pot apela reciproc), procedurile apelantă şi apelată joacă un rol simetric: secvenţa de retur este identică cu cea de apel. Activitatea care rezultă din apelul unei proceduri p ia în calitate de context iniţial contextul salvat la ultimul apel executat de p. Primul apel al unei proceduri trebuie să specifice valoarea iniţială a contextului său.

Secvenţa de schimbare a contextului (reluarea) conţine următoarele etape (p este corutina părăsită, q este cea reluată):

- Pregătirea parametrilor, transmişi de către procedura p procedurii q,
- Salvarea parţială a contextului lui p, care trebuie să fie regăsit la reluare,
- Restabilirea contextului salvat la ultimul abandon al lui q sau formarea contextului iniţial în cazul primului apel. ◄

2.1.1.1. Mecanisme de execuţie secvenţială

Vom descrie un mecanism general pentru execuţia programelor secvenţiale, formate dintr-o suită de activităţi, rezultate din execuţia unei proceduri sau corutine. Acest mecanism realizează funcţiile următoare:

- Salvarea şi restabilirea contextului la apelare şi retur,

- Transmiterea parametrilor între procedurile apelantă şi apelată,
- Administrarea unei zone de lucru proprii fiecărei proceduri, cu permisiunea apelurilor recursive.

***Cazul procedurilor*.**

Structura de date utilizată în acest caz este o **stivă de execuţie**. Sunt utilizate diverse variante, care diferă în mod esenţial prin specificarea detaliată a contextului şi prin modul de comutare a acestuia la apelare sau retur. Schema execuţiei poate fi programată direct (de exemplu, în limbajul de asamblare) sau determinată de structura mecanismului de execuţie, definit de un limbaj de programare. În ultimul caz prin context înţelegem mulţimea variabilelor accesibile conform regulilor de vizibilitate definite în limbaj (structura blocurilor, modulele, etc.). Descriem o organizare foarte apropiată schemei de execuţie din limbajul de programare C, presupunând respectarea următoarelor ipoteze:

- Parametrii unei proceduri sunt transmişi prin valori; la retur un singur rezultat este returnat,
- Procedurile pot fi apelate recursiv (direct sau indirect).

La fiecare apelare a procedurii o structură de date numită **regiune a mediului** (sau simplu ***mediul*** *procedurii*) este creată în vârful stivei de execuţie, care va dispare la returul respectiv. O procedură apelată recursiv are în stivă atâtea medii câte execuţii neterminate există pentru această procedură. La un moment concret de timp mediul din top-ul stivei corespunde procedurii *q* în curs de execuţie (procedura activă), mediul imediat următor este al procedurii *p*, care a apelat procedura *q*, etc. Stiva este administrată prin intermediul a doi pointeri (fig.2.1.):

baza pointerul care indică baza mediului procedurii active,

top pointerul care indică primul amplasament liber pentru a crea un nou mediu.

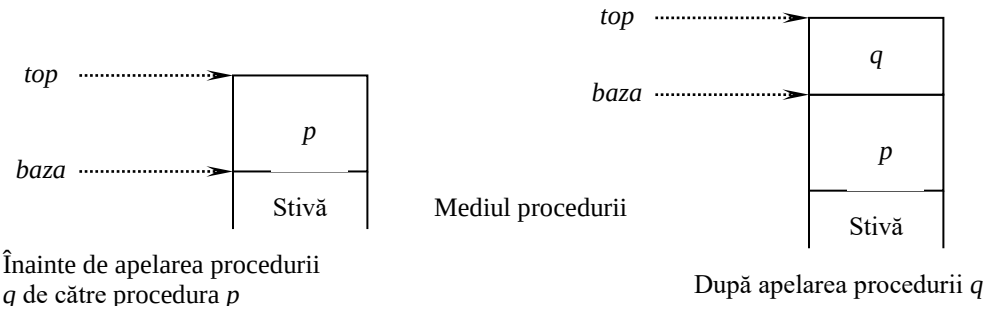


Fig.2.1. Stiva de execuţie

Mediul conţine următoarele informaţii:

a) informaţii de salvare şi legăturile de retur
   - un pointer la baza contextului procedurii apelante
   - adresa returului
   - un amplasament pentru rezultat

b) parametri
   - *n+1* amplasamente, primul conţinând numărul de parametri, iar următoarele - valorile acestora
   - variabilele locale şi spaţiul de lucru al procedurii.

Aceste informaţii, cu excepţia spaţiului de lucru, sunt păstrate în amplasamente pentru care deplasarea în raport cu originea mediului este cunoscută şi fixă, fiind posibilă adresarea relativă a lor (faţă de baza mediului). Spaţiul de lucru este adresat din vârful stivei.

Operaţiile executate la apelarea şi returul procedurii sunt următoarele:

Apelare

1) alocarea unei zone în stiva de execuţie pentru mediul procedurii apelate (dimensiunea acestei zone, cu excepţia spaţiului de lucru, este cunoscută anticipat)
   temp:=baza
   baza:=top
   top:=top+dimensiunea mediului
2) salvarea informaţiilor de retur

baza_veche:=temp
memorizarea adresei de retur

3) ordonarea parametrilor

4) executarea unei ramificaţii la procedura apelată.

Retur

1) salvarea rezultatului într-un amplasament stabilit

2) restabilirea informaţiilor de retur şi eliberarea mediului
temp:=adresa de retur
top:=baza
baza:=baza_veche

3) returul
*ramificare *temp* *ramificare indirectă*

O eventuală salvare şi restabilire a registrelor sunt lăsate în seama procedurii apelante.

Cazul corutinelor.

Schema precedentă este funcţională doar pentru cazuri simple de execuţie a două corutine pentru care mediul salvat se reduce la adresa de retur. Pentru cazuri mai generale, îndeosebi când o corutină este ea însăşi formată din mai multe proceduri, schema stivei unice nu poate fi aplicată. Fie ca exemplu următorul caz:

```
corutina p                          corutina q
    begin                               procedura q1
      …                                   …
    reluare q;                            reluare p;
      …                                   …
    reluare q                             retur
      …                                   end q1;
    end p                               begin
                                          …
                                          q1;
                                          …
                                        end q
```

În momentul în care *q* execută reluarea lui *p* stiva conţine mediile *p*, *q* şi q_1. Mediul care trebuie reluat este cel al lui *p*, dar mediile lui *q* şi q_1 trebuiesc păstrate. Soluţia este de a avea câte o stivă distinctă pentru fiecare corutină (fig.2.2).

Fig.2.2. Schema de execuţie pentru corutine

Pentru un moment concret de timp stiva curentă este stiva corutinei în curs de execuţie. O operaţie *schimbă_stiva* permite schimbarea stivei curente, salvând adresa de reluare în top-ul fostei stive curente:

```
schimbă_stiva (p):
        stiva_curentă.top:=adresa de reluare;
        stivă_curentă     :=stiva(p)
```

Secvenţa de reluare va arăta astfel

```
reluare (p):
        schimbă_stiva (p);
        if prima apelare then
            apelare (p)        ---- secvenţa de apel primar
```

```
else
    temp:=stiva_curentă.top;
    ordonează parametrii în stiva_curentă;
    ramificare *temp
endif
```

Rezumat: **contextul memoriei** unei activităţi la un moment curent de timp **conţine**, în afara segmentului procedurii în curs de execuţie şi segmentele de date, **zona mediului curent din stiva de execuţie.**

2.1.2. Starea şi contextul procesorului

Starea procesorului este determinată de conţinutul registrelor acestuia. Registrele pot fi:

- adresabile, numite **registre generale**, manipulate de programe,
- registre speciale, de obicei grupate sub forma unor informaţii sintetice, numite **cuvânt de stare program** sau **cuvânt de stare a procesorului** (în engleză, **program status word, PSW**).

Cuvântul de stare a procesorului conţine informaţii, care pot fi clasificate astfel:

1) Informaţii referitoare la starea procesorului:

   (a) **Starea de execuţie**. Procesorul se poate afla în starea **activ** în care îndeplineşte instrucţiuni, sau în starea de **aşteptare**, când execuţia este suspendată. Tranziţia *activ* → *aşteptare* se realizează prin încărcarea cuvântului de stare sau prin executarea unei instrucţiuni speciale; trecerea inversă *aşteptare* → *activ* este în mod obligator rezultatul unei ***întreruperi***.

   (b) **Modul de funcţionare**. Din considerente de protecţie este util să se permită doar programelor sistemului de operare executarea unor instrucţiuni anumite (speciale), programele utilizatorilor neavând acces la aceste instrucţiuni. Aceasta poate fi realizat definind două moduri de funcţionare a calculatorului, care se deosebesc printr-un bit indicator în cuvântul de stare: modul **stăpân** (**master**, eng., **maître**, fr.) sau **supervizor** şi modul **sclav** (**slave,** eng., **esclave**, fr.) sau **program**. Setul de instrucţiuni care pot fi executate în **modul program** este o submulţime a setului corespunzător modului **supervizor**. Astfel instrucţiunile rezervate modului supervizor pot fi numite **privilegiate**, aici pot fi găsite instrucţiunile de intrare-ieşire şi cele legate de întreruperi şi protecţie. Pot exista şi alte moduri speciale.

   (c) **Masca întreruperilor**.

2) Informaţii referitoare la contextul accesibil al memoriei şi drepturile de acces asociate: tabele de segmente, indicatori de protecţie a memoriei, etc.

3) Informaţii referitoare la derularea activităţii curente: codul condiţiei, contorul operaţiei următoare (contor ordinal). Pentru unele calculatoare contorul, care conţine adresa operaţiei următoare, poate fi un registru distinct al cuvântului de stare, pentru altele el este un câmp al acestui cuvânt.

2.2. Întreruperi, devieri, apelarea supervizorului

2.2.1. Activităţi asincrone

Modelul descris mai sus poate fi folosit doar pentru un singur program, reprezentat printr-o secvenţă de activităţi, executate pe un singur procesor. Unicele mecanisme necesare pentru organizarea comunicărilor între activităţi sunt operaţiile de apelare şi retur pentru proceduri sau reluare pentru corutine.

Pentru cazuri mai generale sunt necesare mecanisme suplimentare, cum ar fi conceptele de **asincronism** sau de **protecţie reciprocă între activităţi**.

Prin **asincronism** *înţelegem efectul care îl pot avea asupra derulării unei activităţi anumite evenimente exterioare.*

Numim **protecţie reciprocă între activităţi** *o modificare mai profundă a* ***contextului****, atunci când se trece de la o activitate la alta, în comparaţie cu ceea ce are loc în cazul unei simple apelări de procedură.*

Un caz tipic de asincronism este executarea intrărilor-ieşirilor simultan cu execuţia unui program. Trebuie să fie asigurată posibilitatea informării programului despre terminarea unui transfer de informaţii. O primă posibilitate, care nu necesită utilizarea unor dispozitive speciale, este cea a **aşteptării active**, prezentată mai jos.

Arhitectura calculatoarelor CDC Cyber 170 conţinea un procesor central *PC* şi 10 procesoare periferice identice *PP*. Programele sistemului de operare erau executate pe *PP*: unul dintre acestea, numit *PS*, avea "grijă" de înlănţuirea lucrărilor şi alocarea resurselor, celelalte, notate *PES(i),* erau specializate în operaţii de I/E. *PC* şi *PP* comunicau prin intermediul unei memorii comune. Schematic, o operaţie de I/E se derula în felul următor:

1) Presupunem că un program al utilizatorului, executat pe *PC*, cere o operaţie de I/E. Procedura utilizată pentru aceasta iniţializează un cuplu de indicatori *D* (cerere) şi *F* (terminare): *D:=1, F:=0*. Specificaţia precisă a transferului necesar este plasată într-un descriptor, adresa căruia este cunoscută de către *PS*.

2) Un program de supraveghere, executat de *PS*, controlează periodic indicatorul *D*. În momentul în care îl găseşte pus în *1*, acest program alege unul din *PES* liber, fie *PES(i)*, şi-i cere să execute operaţia de I/E. Pentru aceasta sunt utilizaţi doi indicatori *DEX(i)* şi *FEX(i)*, analogici cu *D* şi *F*, transmiţând lui *PES(i)* adresa descriptorului cu informaţiile necesare operaţiei de I/E.

3) Programul de I/E, care este executat pe *PES(i)*, testează periodic indicatorul său *DEX(i)*. Atunci când acesta este găsit pus în *1*, programul execută transferul informaţiilor cerute, utilizând datele găsite în descriptor. La terminarea transferului acest eveniment este semnalizat de către program prin punerea în *1* a indicatorului *FEX(i)*.

4) Supervizorul scrutează periodic *FEX(i)*. Dacă acesta este găsit în *1*, supervizorul face acelaşi lucru cu indicatorul *F*, care este periodic consultat de către programul utilizatorului pentru a afla când se termină transferul.

Algoritmic putem reprezenta toate acestea astfel:

programul utilizatorului

```
<cerere de I/E>
    D:=1;
    F:=0;
    ……

<test terminare I/E>
    if F=1 then
……
```

programul supervizor

```
ciclu
  if D=1 then
    caută PES(i);
    DEX(i):=1;
    FEX(i):=0;
  endif
      ……
  <test terminare execuţie>
  if FEX(i)=1 then
    F:=1;
  endif
endciclu
```

programul pentru I/E

```
ciclu
  if DEX(i)=1 then
      execută I/E;
      <sincron>
      FEX(i):=1;
  endif
endciclu
```

Pot fi făcute următoarele comentarii:

- relaţiile între *PC* şi *PS*, pe de o parte, şi *PS* şi *PES*, pe de alta, au aceeaşi schemă. Fiecare indicator poate fi pus în *1* sau consultat de *un singur procesor*.
- procesorul central este în totalitate eliberat de operaţiile de I/E. El poate fi încărcat cu alte lucrări în timpul acestor operaţii. În acelaşi timp, programul, care cere transferuri de informaţii verifică în mod explicit terminarea operaţiilor de I/E.
- operaţiile de I/E sunt ele însăşi separate în două părţi: supervizarea (alegerea procesorului de intrare-ieşire în funcţie de starea lui *PES* şi urgenţa operaţiei) şi transferul propriu-zis al informaţiilor, care poate fi realizat într-un mod mai eficient de un procesor eliberat de funcţiile precedente.

Exemplele care urmează sunt chemate să ilustreze limitele metodei de aşteptare activă pentru organizarea comunicărilor între activităţile asincrone.

Exemplul 2.3. Sistem multiprocesoral. Fie un calculator, arhitectura căruia presupune o memorie comună la care sunt asociate mai multe procesoare identice (sistem multiprocesoral cu memorie comună), aceste procesoare nefiind specializate cum a fost în cazul precedent, ci banalizate, adică o activitate poate fi executată pe oricare procesor.

Comunicarea între activităţile executate pe diferite procesoare poate fi realizată prin modificarea şi testarea periodică a indicatorilor memoriei comune. Însă aşteptarea activă conduce în acest caz la eficienţă scăzută, deoarece consultarea periodică imobilizează procesorul. ◄

Exemplul 2.4. Măsurarea timpului. În cadrul derulării unui program adesea este necesar să se ia în consideraţie timpul real (cazul proceselor tehnologice, raţionamente de securitate - oprirea unui program care abuzează de procesor din cauza unei bucle infinite, etc.). Deoarece ceasul calculatorului este un

dispozitiv extern procesorului, el trebuie periodic consultat de către program, soluţie costisitoare şi adesea imprecisă. Este necesar un mecanism care ar permite ceasului la momente fixe să acţioneze asupra procesorului. ◄

Exemplul 2.5. Intervenţii externe. Din mai multe considerente (forţă majoră, securitate, etc.) poate fi necesar să existe posibilitatea întreruperii derulării unui program pentru a impune procesorul să execute o acţiune anume. Această întrerupere poate fi declanşată de către un operator uman, un dispozitiv automat, etc. Este o situaţie analogică cu cea din exemplul precedent; din aceleaşi considerente nu este binevenită consultarea periodică a unui indicator. Avem nevoie de un mecanism de acţionare directă în acest caz. ◄

2.2.2. Mecanisme de comutare a contextului

Comutarea contextului unui procesor permite executarea într-o manieră indivizibilă a următoarelor două operaţii:

- trecerea cuvântului de stare într-un amplasament specificat al memoriei,
- încărcarea în cuvântul de stare a conţinutului unui alt amplasament specificat al memoriei.

Aceasta se poate întâmpla doar dacă procesorul se află într-o stare observabilă (de exemplu, între execuţia a două instrucţiuni) sau de aşteptare. Din aceste considerente noţiunea de "punct de întrerupere" este sinonim al noţiunii "punct observabil".

Comutarea contextului poate fi necesară din mai multe cauze distincte. Presupunem că fiecărei cauze i-a fost asociat un număr de ordine. Pot fi întâlnite două scheme de comutare a contextului.

1) **Salvare în amplasamente fixe.** Fiecărei cauze (numărul unei cauze fiind *i*) îi sunt ataşate în mod constant o pereche de amplasamente în memorie adresele cărora le vom nota prin *csp_vechi[i]* şi *csp_nou[i]* (*csp* semnificând cuvânt de stare program). Comutarea se produce după cum urmează:

   Mp[csp_vechi[i]]:=<cuvânt de stare program>
   <cuvânt de stare program>:= Mp[csp_nou[i]]

În setul de instrucţiuni există o instrucţiune privilegiată ***schimbă_csp(adr)*** cu următorul efect:
   <cuvânt de stare program>:= Mp[adr].

2) **Salvare într-o stivă.** Aici avem acelaşi procedeu în ceea ce constă cuvântul de stare nou. Însă cuvântul de stare vechi nu este salvat într-un amplasament fix, ci într-o stivă specială:

```
                                ptr      :=ptr+1
                          stiva[ptr]     :=<cuvânt de stare program>
<cuvânt de stare program>            := Mp[csp_nou[i]]
```

Instrucţiunea ***schimbă_csp(adr)*** este identică celei din 1); există însă şi o instrucţiune ***restabileşte_csp***, care restabileşte cuvântul de stare a procesorului folosind pentru acesta top-ul stivei:

```
<cuvânt de stare program>  :=stiva[ptr]
                           ptr      :=ptr-1
```

Comutarea contextului, descrisă mai sus, este declanşată sub influenţa stării unor indicatori din cuvântul de stare program pe care procesorul îi consultă la interpretarea fiecărei instrucţiuni. Conform semnificaţiei acestor indicatori şi maniera în care ei sunt modificaţi, pot fi evidenţiate trei mecanisme de comutare, descrise în tabelul de mai jos:

Tabelul 2.1. Mecanisme de comutare a contextului

Denumirea mecanismului	Cauza	Utilizarea
Întrerupere	Exterioară derulării instrucţiunii curente	Reacţionare la un eveniment asincron extern
Deviere (excepţie)	Legată de derularea instrucţiunii curente	Tratarea unei erori sau situaţii excepţionale
Apelarea supervizorului	Comutare explicită	Apelarea unei funcţii a S.O.

Adesea toate aceste trei denumiri sunt înlocuite cu una singură - **întrerupere**.

2.2.3. Întreruperi

O **întrerupere** este comutarea contextului procesorului declanşată de o cauză externă derulării instrucţiunii curente. Fizic, întreruperea înseamnă trimiterea unui semnal procesorului, acest semnal provocând schimbarea stării unuia dintre indicatorii, consultaţi în cursul executării fiecărei instrucţiuni. Semnalul poate proveni de la alt procesor, de la un organ de I/E, de la un dispozitiv extern, şi în genere, de la orice proces fizic, extern procesorului întrerupt. O întrerupere permite să cerem procesorului să suspende executarea programului curent, din primul punct întreruptibil, şi să treacă la execuţia unui program predefinit. Acesta din urmă este numit **program de tratare a întreruperii** (*interrupt handler*, eng., *traitant de l'interruption*, fr.). Programul de tratare a întreruperii este executat într-un context diferit de cel al programului întrerupt, diferenţa fiind legată de modul de tratare, protecţie, informaţiile accesibile, etc.

Nivele de prioritate. Cauzele care duc la necesitatea întreruperii unui procesor sunt diferite, drept rezultat şi programele predefinite, care tratează întreruperile diferă. Mecanismul întreruperilor trebuie să poată deosebi aceste cauze. Două scheme de bază sunt utilizate în acest scop:

- Există un indicator unic pentru toate întreruperile şi acestui indicator îi este asociat un program unic de tratare a întreruperilor. O informaţie suplimentară (codul întreruperii), care se conţine în cuvântul de stare sau într-un amplasament de memorie, dă posibilitatea de a deosebi cauzele. Prima obligaţie a programului de tratare a întreruperilor este de a consulta codul întreruperii pentru a stabili originea acesteia, doar apoi se va trece la tratarea propriu-zisă.
- Fiecărei cauze este ataşat un indicator, se mai spune că fiecare cauză este ataşată unui nivel de întrerupere. Fiecărui nivel îi corespunde un program distinct de tratare a întreruperii, activat în mod automat de către mecanismul de comutare a contextului.

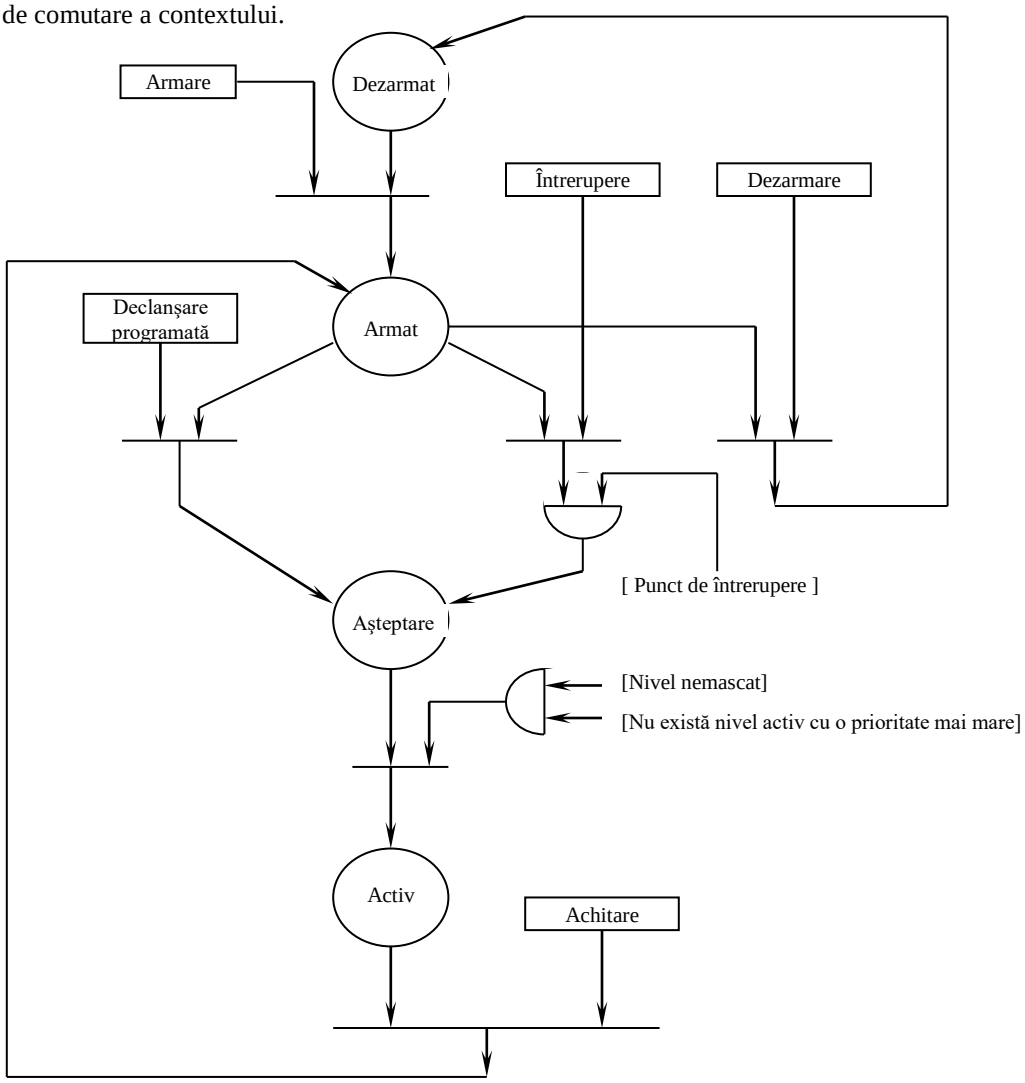


Fig.2.3. Stările unui nivel de întrerupere

Priorităţi şi mascarea întreruperilor. Atunci când există mai multe nivele de întrerupere este posibilă necesitatea modificării simultane a doi indicatori, legaţi de două cauze diferite. Conflictul poate fi reglat stabilind o ordine de prioritate în cadrul nivelelor de întrerupere. Această ordine poate fi fixată odată pentru totdeauna sau modificabilă prin microprogramare.

Adesea este utilă protejarea, contra unor întreruperi anume, a execuţiei unei suite de instrucţiuni (de exemplu, poate fi interzisă întreruperea unui program de tratare a întreruperii). Aceasta ar însemna întârzierea comutării contextului, comutare, provocată de sosirea unui semnal asociat unui nivel oarecare de întrerupere. Se zice atunci, că nivelul este **mascat** sau **inhibat** (**întrerupere mascată**). Ridicarea inhibării (*"demascarea"*) va autoriza schimbarea ulterioară a contextului. Informaţiile asociate mascării întreruperilor se află în cuvântul de stare a procesorului.

Pentru unele nivele de întrerupere efectul sosirii semnalului de întrerupere poate fi suprimat (nu doar întârziat). Totul se va petrece în acest caz ca şi cum însăşi cauza întreruperii a fost suprimată: nivelul întreruperii este **dezarmat.** Un nivel dezarmat poate fi **rearmat**, adică repus în serviciu.

Declanşare programată. Pentru unele calculatoare, îndeosebi cele utilizate în controlul proceselor, există o instrucţiune de declanşare programată care permite modificarea indicatorului, asociat unui nivel de întrerupere, din interiorul unui program. În acest fel este posibilă simularea apariţiei unei cauze externe de întrerupere.

Atunci când toate condiţiile necesare pentru ca o întrerupere să fie luată în consideraţie sunt prezente, nivelul se numeşte **activ**. Această stare corespunde executării programului de tratare a întreruperii. Notăm, că această execuţie poate fi suspendată pentru a permite tratarea unei întreruperi cu o prioritate mai înaltă. Ieşirea din starea activă se face prin **achitarea** întreruperii. Achitarea este realizată folosind instrucţiunea *schimbă_csp*, care termină tratarea schimbând contextul.

Operaţiile de armare, dezarmare, mascare, demascare şi declanşare a întreruperilor sunt totdeauna realizate prin intermediul unor instrucţiuni privilegiate.

Figura 2.3 prezintă schematic tranziţiile de stare pentru un nivel de întrerupere.

Schema unui program de întrerupere. O întrerupere forţează procesorul să reacţioneze la un eveniment. Executarea programului curent este suspendată, comanda fiind transmisă programului de tratare a întreruperii. Programul reluat de către procesor după tratarea întreruperii nu este în mod obligator programul întrerupt (întreruperea putea avea drept scop realocarea procesorului). Schema generală de tratare a unei întreruperi este prezentată în fig.2.4.

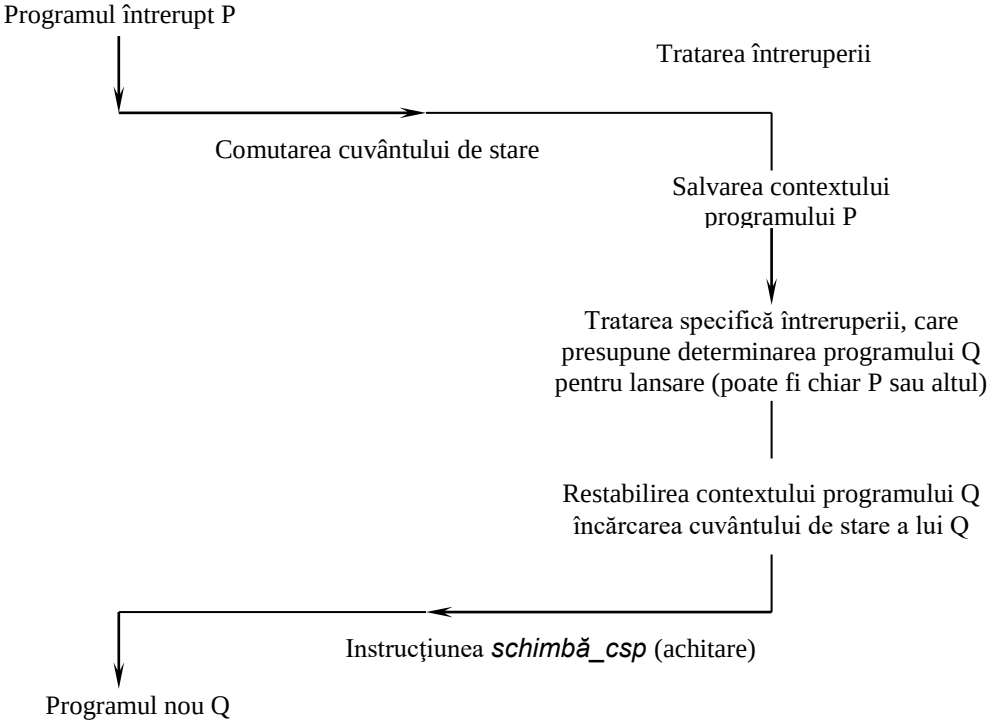


Fig.2.4. Schema generală a tratării unei întreruperi

2.2.4. Devieri şi apelarea supervizorului

Spre deosebire de o întrerupere, o deviere sau un apel al supervizorului sunt provocate de o cauză legată direct de derularea instrucţiunii curente.

O deviere (trap, exeption, eng., dèroutement, exeption, fr.) semnalizează o anomalie în derularea unei instrucţiuni, care prohibitează executarea instrucţiunii. Originile pot fi diverse:

- date incorecte, care conduc la imposibilitatea execuţiei corecte a instrucţiunii (împărţirea la zero, de exemplu),
- tentativa executării unei operaţii interzise de mecanismul de protecţie (violarea protecţiei memoriei, executarea unei instrucţiuni privilegiate în modul program, etc.),
- instrucţiune neexecutabilă (cod neautorizat de operaţie, adresă în afara memoriei existente, utilizarea unui dispozitiv opţional, care este lipsă în configuraţia dată, ș.a.).

Devierile pot fi clasificate, ca şi întreruperile, conform cauzelor care le generează. Efectul unor cauze poate fi suprimat (de exemplu devierile legate de operaţiile aritmetice, erorile fiind semnalate doar de valoarea codului condiţiei). **Natura unei devieri nu permite aplicarea noţiunii de mască. O deviere poate fi suprimată, dar nici intr-un caz retardată.**

Un apel al supervizorului (supervisor call, prescurtat SVC, eng., appel au superviseur, fr.) este o instrucţiune chemată să provoace o comutare a contextului procesorului. Acest efect este analogic apelării unei proceduri, însă modificarea contextului este mai profundă, fiindcă ea afectează întreg cuvântul de stare şi nu numai contorul ordinal.

Destinaţia unui apel al supervizorului este de a permite apelarea unei proceduri a sistemului de operare, pretinzând la drepturi mai mari (modul master, întreruperi mascate, drepturi de acces, etc.), direct din programul utilizatorului. Mecanismul comutării contextului permite asigurarea protecţiei, impunând condiţiile de intrare procedurii apelate. Ca rezultat:

- contextul nou (modul, masca, etc.,) este specificat în noul cuvânt de stare, amplasat intr-o zonă de memorie inaccesibilă utilizatorilor,
- programul de tratare a apelului supervizorului începe cu o secvenţă de verificare a drepturilor utilizatorului de a executa acest apel şi stabilirea validităţii parametrilor transmişi.

Parametrii sunt transmişi ca şi în cazul apelului unei proceduri obişnuite. Returul în programul apelant se face prin restabilirea cuvântului de stare salvat la apelare.

Alegerea procedurii accesibile în cazul unui apel al supervizorului este determinată de un parametru suplimentar, aflat într-un registru sau în partea de adresă a instrucţiunii de apelare. Putem considera, că mulţimea procedurilor accesibile utilizatorului în acest mod constituie o extensie a setului de instrucţiuni, formând astfel o "maşină" nouă. Operaţiile de accesare a fişierelor şi de intrare-ieşire sunt disponibile utilizatorilor sub forma apelării supervizorului.

Figura 2.4, care descrie schematic întreruperile, este aplicabilă şi pentru cazul devierilor sau a apelării supervizorului.

2.2.5. Exemple de sisteme de întreruperi

Funcţionarea sistemelor de întrerupere este ilustrată prin două exemple reprezentative, unul pentru un calculator mare, altul pentru calculatoarele personale.

Exemplul 2.6. IBM 370.

Cuvântul de stare (64 biţi) conţine următoarele câmpuri:

<0-7>	masca întreruperii	<16-31>	codul întreruperii
<8-11>	cheie de protecţie	<32-33>	lungimea instrucţiunii
<12>	codul caracterelor	<34-35>	codul condiţiei
<13>	masca întreruperii	<36-39>	suprimarea devierii
<14>	1 - aşteptare, 0 - activ	<40-63>	contor ordinal
<15>	0 - stăpân, 1 - sclav.		

Sistemul de întreruperi are 5 nivele (în ordinea de descreştere a priorităţilor): eroare hardware, deviere, apelare supervizor, extern şi intrare-ieşire. Fiecărui nivel îi corespunde în memoria operativă un cuplu de amplasamente rezervate cuvintelor de stare vechi şi nou. Fiecare nivel conţine mai multe cauze de întrerupere. Pentru a face o deosebire a acestor cauze este utilizat un cod al întreruperii format din biţii 16 - 31 ai cuvântului de stare. Acest cod este în mod automat pus la zi în cazul unei întreruperi. De exemplu, pentru o întrerupere de intrare-ieşire el va conţine adresa

canalului şi a perifericului care au provocat întreruperea; pentru o întrerupere externă acest cod permite să se stabilească originea - ceasul, apel al operatorului, adresare a unui organ extern, etc. Biţii 0 - 7 şi 13 ai cuvântului de stare permit introducerea măştii întreruperii. Ei permit mascarea nivelelor de întrerupere externă, eroare hardware şi, selectiv, întreruperile de intrare-ieşire, care provin de la diferite canale. Biţii 36 - 39 permit inhibarea a 4 cauze de deviere.

Observăm, că devierile şi apelările supervizorului sunt tratate ca şi întreruperi particulare.◄

Exemplul 2.7. IBM PC. Este propus ca exerciţiu.

2.3. Implementarea mecanismelor de comutare a contextului

Implementarea mecanismelor descrise mai sus este ilustrată printr-o serie de exemple, legate de utilizarea lor.

Vom preciza notaţia utilizată în schemele programelor:

- conţinutul unui cuvânt de stare este reprezentat prin notaţia <*activitate, mod, mascare, contor ordinal*>. Primele trei câmpuri pot lua, respectiv, valorile: aşteptare/activ, master/slave, mascat/demascat. Dacă *csp* desemnează un cuvânt de stare, câmpurile sale vor fi desemnate de *csp.act, csp.mod, csp.masc* şi *csp.co.*
- operatorul **adr** aplicat unui identificator de variabilă sau de procedură, desemnează adresa amplasamentului unde se află variabila sau prima instrucţiune a procedurii,
- notaţia *Mp[adr]* desemnează conţinutul amplasamentului cu adresa *adr*.
- Notaţiile *svc_vechi, dev_vechi, intr_x_vechi, svc_nou, dev_nou, intr_x_nou* desemnează amplasamentele unde sunt plasate şi de unde sunt încărcate cuvintele de stare.

Vom mai presupune că

- apelarea supervizorului este de forma *SVC <cod>,* unde *<cod>* reprezintă un număr care permite identificarea funcţiei cerute,
- un indicator *<cauză>* permite stabilirea cauzei care provoacă o deviere.

Schemele prezentate conţin un program de iniţializare, executat la prima încărcare a sistemului. Funcţia acestui program este de a iniţializa cuvântul de stare asociat tratării întreruperilor, devierilor şi apelurilor supervizorului.

2.3.1. Utilizarea devierilor şi apelării supervizorului

Simularea instrucţiunilor lipsă. Unele instrucţiuni din setul de bază de instrucţiuni ale procesorului pot fi opţionale şi, deci, pot lipsi în unele configuraţii ale calculatorului. Tentativa executării unei instrucţiuni opţionale lipsă generează o drivere de tipul *instrucţiune inexistentă.* Mecanismul devierilor poate fi utilizat pentru a realiza prin program instrucţiunea inexistentă în setul de bază. De exemplu, pentru o configuraţie în care operaţiile aritmetice în virgulă mobilă nu sunt disponibile ele pot fi simulate prin apelarea procedurilor corespunzătoare. Aceste proceduri trebuie executate în mod slave pentru a permite tratarea explicită a erorilor: ele sunt apelate prin încărcarea cuvântului de stare respectiv. O adresă de retur trebuie să fie pregătită de către programul de tratare a devierii în top-ul stivei pentru a asigura returul la instrucţiunea care urmează instrucţiunii de operaţie aritmetică simulată.

Realizarea prin program:

```
procedura iniţializare;
    dev_nou:=<activ,master,mascat, adr tratare_deviere>;
    csp[ADD_FL]:= <activ,slave,demascat, adr ADD_FL>;
    ……
    csp [DIV_FL]:= <activ,slave,demascat, adr DIV_FL>;

procedura tratare deviere;
    begin
    save(zonă);
    case cauză of
    ……
instrucţiune inexistentă:
            cod_op:=Mp[dev_vechi.co].cod_operaţie;
            case cod_op of
            ……
    ADD_FL: determinarea operanzilor;
            introdu_în_stivă(stivă_utilizator,dev_vechi.co+1);
```

```
            încarcă_csp(csp[ADD_FL])
            ……
            else            -- caz de eroare
            <tratarea erorii>
            end;
    end;
restabilire(zonă);
încarcă_csp(dev_vechi)
end
```

Operanzii vor fi determinaţi prin analiza conţinutului instrucţiunii, care conţine adresele lor; returul rezultatelor utilizează convenţiile proprii instrucţiunii în cauză (plasarea într-un registru, etc.).

Măsurarea capacităţii memoriei. Un sistem de operare presupune a fi utilizat pentru diverse configuraţii ale calculatorului. Capacitatea memoriei, numărul şi natura dispozitivelor periferice, etc., pot fi diferite. Sistemul de operare trebuie să se adapteze configuraţiei concrete, ca şi condiţiilor particulare de utilizare (număr de utilizatori, priorităţi, etc.). Crearea unei asemenea versiuni specifice se numeşte **generarea** sistemului de operare. Pentru a reduce frecvenţa generărilor S.O. unii parametri de configurare pot fi determinaţi în mod automat la iniţializare.

Astfel, poate fi determinată capacitatea memoriei operative utilizând o deviere pentru adresa amplasamentului de memorie inexistent. Memoria constă dintr-un număr n_{bloc} de blocuri (care trebuie determinat), fiecare bloc conţinând *p* cuvinte. Un program executat la iniţializarea sistemului încearcă să acceseze ciclic primul cuvânt al fiecărui bloc; tentativa de accesare a primului bloc inexistent în configuraţia curentă provoacă, prin deviere, părăsirea buclei. Numărul de blocuri este egal cu numărul primului bloc neimplantat (blocurile fiind numerotate începând cu 0).

Realizarea prin program:

```
program principal:
                begin
                   dev_nou:= <activ,master,mascat, adr măsurare>;
                   i:=1;
                   ciclu
                     <accesare cuvânt cu adresa i.p>;
                     i:=i+1;
                   endciclu;
ieşire ciclu:   nbloc:=i;
                dev_nou := <activ,master,mascat, adr tratare_deviere>;
                end
procedura măsurare;             -- apelare prin deviere
if cauză = adresă în afara memoriei then
csplucrare:= <activ,master,mascat, adr ieşire ciclu>;
încarcă_csp(csplucrare)
endif
```

La ieşirea din programul de măsurare a capacităţii memoriei se va restabili tratarea normală a devierilor (procedura *tratare_deviere*).

Gestionarea devierilor de către utilizatorul sistemului. Reacţia standard la o deviere care are loc în timpul execuţiei unui program utilizator este apelarea unei proceduri de sistem, care provoacă emiterea unui mesaj de eroare, oprirea programului curent şi trecerea la programul următor.

Este de dorit ca fiecare utilizator să aibă posibilitatea să asocieze fiecărei cauze distincte de deviere *i* o procedură proprie, tratarea căreia ar înlocui reacţia standard. Pentru a evita o buclă infinită, o deviere din aceeaşi cauză *i* în interiorul acestei proceduri trebuie să apeleze procedura standard.

Asocierea unei proceduri *proc* de reluare a cauzei *i* de deviere se face cu ajutorul unui apel al supervizorului (SVC *asociere_deviere*) cu parametrii *i* şi *proc*.

Pentru trecerea la programul următor se va apela o procedură a sistemului de operare, apel realizat prin încărcarea unui cuvânt de stare, numit *schimbare*.

Realizarea prin program:

```
const ndev=<numărul de cauze de deviere>;
var C: array[0..ndev-1] of boolean;
    adrtratare: array[0..ndev-1] of adresă;

procedura iniţializare;
    begin
    svc_nou:=<activ,master,mascat, adr tratare_svc >;
        for i:=0  to ndev-1 do          -- tratare standard
        C[i]:=false;
        endfor;
    dev_nou:=<activ,master,mascat, adr tratare_dev >;
    end

procedura tratare_svc;
    begin
        save(zonă);
        case cod of
          ……
asociere_deviere:                                   -- parametrii i, proc
        C[i]:=true;
        adrtratare[i]:=adr proc;
        restabilire(zonă);
        încarcă_csp(svc_vechi)
        ……
        endcase
    end

procedura tratare_dev(i);
    begin
    save(zonă);
    if C[i] then
        C[i]:=false;                                -- restabilire tratare standard
        introdu_în_stivă(stivă_utilizator,dev_vechi.co);  -- pentru retur
        dev_vechi.co:= adrtratare[i];
        restabilire(zonă);
        încarcă_csp(dev_vechi);
    else
        <tratare standard[i]>;
încarcă_csp(schimbare);                             -- programul următor
    endif
    end
```

Programul de tratare a devierii, specificat de către utilizator se va executa în modul *slave*, iar returul se va face cu ajutorul mecanismului standard de retur dintr-o procedură, deoarece adresa de retur a fost pusă în vârful stivei de execuţie.

2.3.2. Exemple de utilizare a întreruperilor

Principala utilizare a întreruperilor este măsurarea timpului şi administrarea operaţiilor de intrare-ieşire. Prezentăm mai jos câteva exemple de utilizare a ceasului calculatorului.

Un ceas de calculator este realizat cu ajutorul unui oscilator cu cuarţ, care emite impulsuri periodice. Perioada de emitere, de obicei poate fi aleasă dintr-o plajă de valori prestabilită, de exemplu de la 0.1 până la 100 microsecunde. Aceste impulsuri sunt utilizate pentru a decrementa valoarea unui contor, conţinut într-un registru special sau într-un cuvânt de memorie, rezervat acestui scop. Când acest contor trece prin zero este declanşată o întrerupere, numită întrerupere de ceas, care corespunde unui nivel rezervat acestui efect.

Pentru aplicaţiile care urmează vom presupune, că fiecare impuls decrementează cu *1* conţinutul unei locaţiuni de memorie, notat *ceas*.

Limitarea timpului de execuţie a unei proceduri. Este necesar să fie executată o procedură *p* astfel, încât dacă după un interval de timp prestabilit *q*, numit *timeout*, executarea nu se va termina, ea va fi întreruptă şi o procedură specificată *tratare_eroare* este apelată.

Realizarea prin program:

```
procedura iniţializare;
    intr_ceas_nou:=<activ,master,mascat,adr intr_ceas>;
    svc_nou      :=<activ,master,mascat,adr tratare_svc >;
    dezarmare(intr_ceas);
procedura tratare_svc;
    save(zonă);
    case cod of
    …
apel_lim_timp_ex:                          -- parametrii p, q, tratare_eroare
            ceas:=q;
            cspretur:=svc_vechi;           -- salvare pentru retur
            csplucrare:= <activ,slave,demascat,adr p>;
        csperoare:= <activ,slave,demascat,adr tratare_eroare>;
        armare(intr_ceas);
        restabileşte(zonă);
        încarcă_csp(csplucrare);
        …
    retur:                                 -- datorat terminării procedurii p
        dezarmare(intr_ceas);
        încarcă_csp(cspretur);
        …
     endcase
procedura intr_ceas;                       -- expirarea timpului limită
        dezarmare(intr_ceas);
        încarcă_csp(csperoare);
```

Primul eveniment, care va avea loc (expirarea timpului *q*, rezervat executării, sau returul din *p*), dezarmează ceasul şi activează procedura adecvată pasându-i contextul curent. Returul din *p* trebuie să se facă prin apelarea supervizorului şi nu printr-o instrucţiune obişnuită de retur. La fel se va proceda şi cu returul din *tratare_eroare*, dacă se cere revenirea la contextul apelării iniţiale a procedurii *p*.

Culegerea unor date de măsurare. Un calculator are obligaţiunea să preia periodic datele unor măsurări legate de o instalaţie industrială. Perioada ceasului este de 5 μs. Datele trebuie culese fiecare 100 ms (perioada de eşantionare). Deoarece timpul necesar culegerii datelor este cu mult inferior perioadei de eşantionare, calculatorul poate executa alte lucrări în regim de fond.

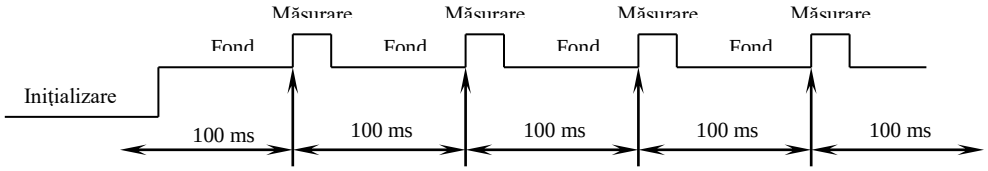
Realizarea prin program:

```
const q = 20000                            -- 100ms/5μs
procedura iniţializare;
    begin
        intr_ceas_nou:=<activ,master,mascat,adr intr_ceas>;
        csplucrare:= <activ,slave,demascat,adr fond>;
        cspmăsurare:= <activ,slave,demascat,adr măsurare>;
        ceas:=q;
        încarcă_csp(cspdev)
    end
procedura intr_ceas;
    begin
        save(zonă);                        salvarea contextului de fond
        cspsave:=intr_ceas_vechi;          salvarea cuvântului de stare pentru fundal
        ceas:=q;
        încarcă_csp(cspmăsurare)           programul de culegere a datelor
    end
```

Programul de măsurare trebuie să se termine printr-o apelare a supervizorului *SVC retur* efectul căreia este de a restabili contextul lucrului în fond. Acest apel execută secvenţa:

restabileşte (zonă);
încarcă_csp(cspsave);

Schema de mai jos ilustrează funcţionarea sistemului descris:



Remarcă. Putem include procedura *măsurare* direct în programul de tratare a întreruperii, care ar fi avut în forma

save(zonă);
ceas:=q;
măsurare;
restabileşte(zonă);
încarcă_csp(intr_ceas_vechi)

Dar în acest caz programul de măsurare va trebui să fie executat în modul *master* şi întreruperile să fie mascate, ceea ce nu ar permite modificarea sau înlocuirea lui de către un utilizator neprivilegiat. Această remarcă ilustrează un principiu general, care cere îndeplinirea independentă (decuplată) a programelor de tratare a întreruperilor (secvenţe de scurtă durată, executate cu privilegii înalte) şi programele activate de către procedurile de tratare a întreruperilor şi apelate prin încărcarea unui cuvânt de stare (tratare primară şi secundară a întreruperii).

Administrarea lucrărilor în timp partajat. O mulţime de programe sau lucrări trebuiesc executate pe un calculator, utilizând procesorul în regim de partajare a timpului: procesorul este alocat succesiv lucrărilor într-o ordine prestabilită, pentru tranşe de timp de durată fixă *q* (cuanta de timp). Dacă o lucrare este încheiată înainte de expirarea cuantei *q*, procesorul este imediat alocat lucrării următoare.

Lucrările sunt numerotate de la *0* la *n-1*. Este definită o zonă de amplasare a cuvântului de stare a procesorului *csp[i]* pentru fiecare lucrare *i* şi o altă zonă *reg[i]* pentru registrele acesteia. Aceste zone permit salvarea contextului procesorului de fiecare dată când o lucrare este întreruptă pentru a permite reluarea ulterioară a lucrării întrerupte.

Realizarea prin program:

Pentru început, presupunem că lucrările nu se vor termina niciodată. Vom utiliza un fir circular de lucrări, notat *fir_lucr* şi gestionat de trei proceduri *intrare, ieşire* şi *urm*.

```
procedura iniţializare;
    begin
    for i:=0 to n-1 do
        csp[i] := <activ,slave,demascat,adr lucrare[i]>;
        intrare(i, fir_lucr);
    endfor
    intr_ceas_nou:=<activ,master,mascat,adr intr_ceas>;
    svc_nou:=<activ,master,mascat,adr trattare_svc>;
    lucr_ales:=0;                                  -- începem cu lucrarea 0
    încarcă_csp(csp[lucr_ales]);
    end

procedura intr_ceas;
    begin
    csp[lucr_ales]:=intr_ceas_vechi;
    save_registre(reg[lucr_ales]);
    lucr_ales:=urm(lucr_ales);
    ceas:=q;
    încarcă_registre(reg[lucr_ales]);
    încarcă_csp(csp[lucr_ales]);
    end
```

Pentru a lua în consideraţie terminarea lucrărilor, fiecare lucrare trebuie să înştiinţeze sistemul atunci când execuţia sa a luat sfârşit printr-un apel al supervizorului *SVC end* după cum este prezentat mai jos.

```
procedura tratare_svc;
    ...
    case cod of
    ...
term:
      suc:=urm(lucr_ales);
      ieşire(lucr_ales, fir_lucr);
      lucr_ales:=suc;
      ceas:=q;
      încarcă_registre(reg[lucr_ales]);
      încarcă_csp(csp[lucr_ales]);
    endcase
```

Am presupus că firul de lucrări nu va deveni niciodată vid. În practică aceasta poate fi obţinut prin trecerea la o lucrare de fond, care poate eventual fi redusă la o buclă.

2.3.3. Concluzii

Exemplele precedente ilustrează modurile elementare de comunicare dintre programele unui sistem de operare şi mediul exterior. Programele sistemului sunt activate:

1) fie de către programul utilizatorului, care semnalează o anomalie (deviere) sau cere achitarea unui serviciu (apelarea supervizorului),

2) fie de către un dispozitiv extern (ceas, proces comandat, organ periferic sau linie de comunicaţii), care semnalează producerea unui eveniment şi care cere o reacţie imediată a sistemului.

Ca şi rezultat putem afirma că un sistem de operare este pilotat de evenimente (*event-driven,* eng.). Notăm de asemenea, că pentru comunicarea internă între programele unui sistem se foloseşte apelarea supervizorului dacă un anumit tip de protecţie trebuie să fie activat (program executat în modul slave, de exemplu) şi apelarea unei proceduri obişnuite în caz contrar. O secvenţă de execuţie a unui program de sistem se termină prin încărcarea unui cuvânt de stare a procesorului care permite controlul total al contextului restabilit.

2.4. Programarea operaţiilor de intrare-ieşire

Prin noţiunea de **intrare-ieşire** (prescurtat I/E; eng. input/output, I/O; fr. entrée-sortie, E/S) numim orice transfer de informaţii din sau spre nucleul calculatorului. Operaţiile de I/E semnifică:

- transferurile de informaţii dintre diferite nivele ierarhice ale memoriei,
- transferurile de informaţii din sau spre mediul exterior (organe periferice locale sau la distanţă, captoare sau dispozitive de acţionare, alte calculatoare, etc.).

2.4.1. Organizarea generală

2.4.1.1. Periferice, controlere, canale

Un organ de intrare-ieşire este un dispozitiv capabil să transfere informaţii între procesorul sau memoria calculatorului şi un suport extern de informaţie. Acest transfer este comandat de către procesorul central. În cel mai simplu caz, o instrucţiune specială a procesorului permite transferarea informaţiei între suportul extern şi un registru al procesorului, care va fi deci ocupat pe toată perioadă transferului informaţiei. Odată cu evoluţia calculatoarelor, în scopul unei mai bune utilizări a procesorului s-a ajuns la necesitatea acordării unei autonomii organelor de intrare-ieşire încredinţându-le funcţii tot mai complicate de înlănţuire şi comandă, procesorului central lăsându-i-se doar iniţiativa de lansare şi de control a operaţiilor. Din considerente economice mai apoi s-a trecut la separarea dispozitivelor de comandă a perifericelor de perifericele propriu-zise, pentru ca dispozitivele de comandă să poată fi partajate între mai multe periferice.

Câteva scheme de organizare a perifericelor sunt prezentate în fig.2.5. Schema (c) este proprie unui calculator de putere mare, (a) şi (b) corespund configuraţiilor calculatoarelor de putere mai mică. Precizăm funcţiile organelor reprezentate în această figură.

1) Un **canal** (sau unitate de schimb) este un procesor specializat în operaţiile de intrare-ieşire. El poate fi lansat doar de un procesor central, nu posedă întreruperi, dar poate întrerupe un procesor central. Setul de instrucţiuni ale canalului îi permite să acţioneze controlerele şi perifericele, care-i sunt conectate. Mini- şi microcalculatoarele pot poseda organe, numite *Unităţi de acces direct la memorie (ADM)*, care sunt nişte canale simplificate.

2) Un **contróler** este un dispozitiv de comandă adaptat la un tip concret de echipament periferic. Autonomia sa este limitată de operaţii foarte elementare. Destinaţia principală a unui controler este de a permite conectarea a mai multor periferice de acelaşi tip la un singur controler. Un singur dispozitiv periferic poate transmite informaţii prin intermediul controlerului la un moment de timp dat. În acelaşi timp, este posibilă executarea simultană a unor operaţii pe alte periferice, operaţii care nu implică transferul de informaţii (rebobinarea unei bande magnetice, deplasarea braţului unui disc, etc.). Partajarea funcţiilor între periferic şi controler depinde de tipul perifericului. De obicei, funcţiile logice (înlănţuirea şi sincronizarea operaţiilor, emiterea unor semnale în caz de accident sau terminarea normală a execuţiei) sunt încredinţate controlerului, iar funcţiile fizice (transferul propriu-zis) - perifericului. În programarea intrărilor-ieşirilor nu este necesar, de obicei, să se facă o deosebire între un controler şi un periferic.

3) Un **periferic** este un organ capabil să transfere informaţii din sau spre un suport extern. Controlerul este legat de periferic printr-o interfaţă, care conţine un set de funcţii (intrare, ieşire, semnalizări de comandă şi de accident) şi o linie de comunicaţie, care serveşte la transferul informaţiilor.

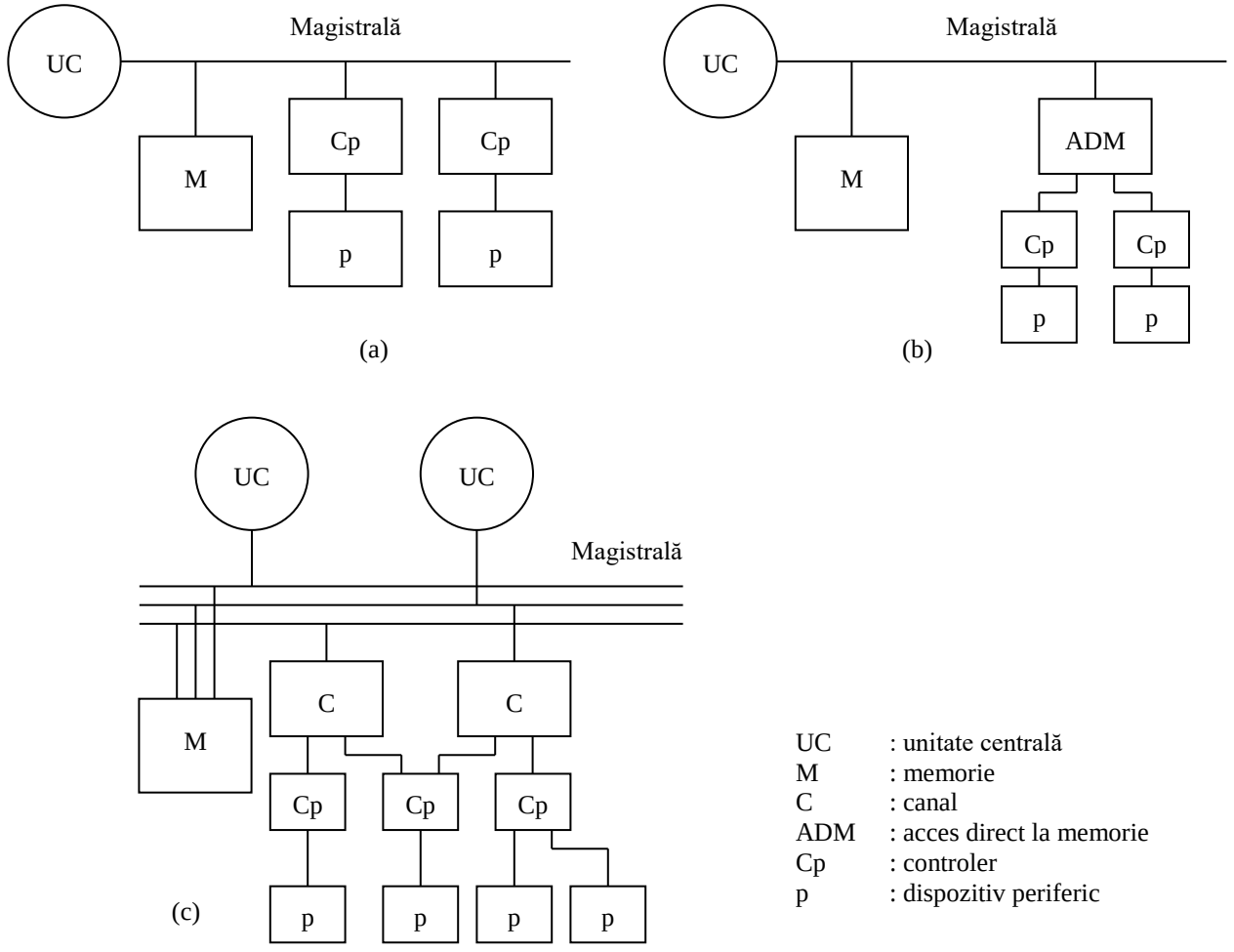


Fig. 2.5. Organizarea intrărilor-ieşirilor

Un canal poate comanda un singur dispozitiv periferic cu debit ridicat (disc, de ex.) sau poate fi multiplexat pentru mai multe periferice cu debitul mai mic. Încercarea mai multor procesoare (unităţi centrale sau canale) de a accesa simultan memoria operativă poate genera conflicte. Conflictele sunt reglate de către dispozitivul hardware de accesare,

care impune o ordine de acces conform unor priorităţi prestabilite. Canalele au prioritate faţă de unităţile centrale, deoarece ele trebuie să reacţioneze cât se poate de rapid la evenimentele externe.

2.4.1.2. Adresarea dispozitivelor periferice

Diferite organe sunt desemnate, la fiecare nivel, printr-o adresă care permite să evidenţiem:

- canale legate de memorie,
- controlere ataşate fiecărui canal,
- dispozitive periferice ataşate fiecărui controler.

La un nivel anume adresa este un simplu număr de ordine. Un periferic este desemnat printr-o adresă compusă

<numărul canalului, numărul controlerului, numărul dispozitivului periferic>.

Un controler poate fi conectat la mai multe canale, iar un periferic la mai multe controlere, care vor fi, obligator, de acelaşi tip. Astfel, un periferic are mai multe căi de acces la memorie (doar una singură poate fi activă la un moment de timp dat) şi poate fi desemnat prin mai multe adrese (fig.2.6).

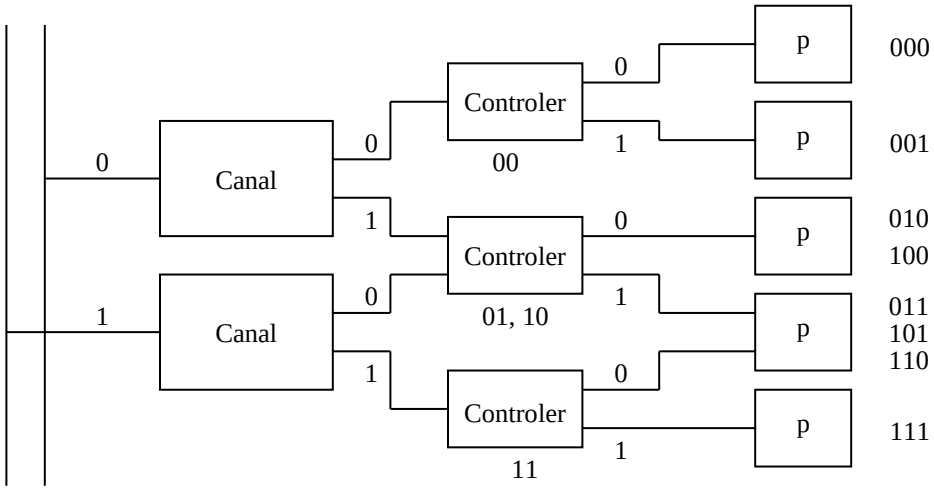


Fig.2.6. Adresarea perifericelor

Această organizare permite ameliorarea performanţelor şi disponibilităţii sistemului. Existenţa mai multor căi de acces la un periferic diminuează riscul unor indisponibilităţi din cauza saturaţiei sau ieşirii din funcţiune a unui canal sau controler.

În cazul unor calculatoare, cum ar fi de exemplu DEC, adresarea perifericelor utilizează amplasamente de memorie rezervate, care sunt asociate cuvântului de stare sau registrului de date a unui controler specificat. Accesarea cuvântului de stare sau a registrului de date este realizată prin intermediul instrucţiunilor obişnuite de accesare a memoriei. Această schemă simplifică desemnarea perifericelor şi programarea intrărilor-ieşirilor.

2.4.2. Metode de comandă a perifericelor

Programul, care controlează funcţionarea elementară a unui dispozitiv periferic se numeşte **driver**. Driverul gestionează în mod direct interfaţa controlerului perifericului, tratează întreruperile generate de acesta, detectează şi tratează cazurile de eroare. El este, de obicei, invizibil unui utilizator obişnuit al sistemului, care apelează funcţii de intrare-ieşire prin intermediul unor servicii de înalt nivel, realizate prin apelări ale supervizorului. Vom prezenta mai jos modurile principale de control ale perifericelor şi primitivele elementare de intrare-ieşire.

2.4.2.1. Intrări-ieşiri sincrone

În acest caz nu există nici un fel de paralelism între procesare şi transferul informaţiilor. Procesorul este ocupat pentru toată durata transferului.

Schema care urmează nu descrie doar un caz particular, ci este reprezentativă pentru organizarea intrărilor-ieşirilor sincrone pentru mai multe tipuri de calculatoare.

1) Specificarea mecanismului

Operaţia de bază constă în transferarea unor informaţii elementare de volum fix între un amplasament de memorie şi un dispozitiv periferic. Presupunem că volumul informaţiei transmise, care depinde de perifericul considerat, este un octet. Starea ansamblului controler-periferic este definită printr-un cuvânt de stare, aflat în controler. Acest cuvânt de stare conţine o mulţime de indicatori booleeni, dintre care doar 3 ne vor interesa aici:

- *preg* : perifericul este pregătit (gata) de funcţionare,
- *term* : transferul este terminat; perifericul este gata să transfere un nou caracter,
- *err* : în cursul transferului a avut loc o eroare; natura ei este precizată de un cod care face parte din cuvântul de stare.

Controlerul este comandat de unitatea centrală cu ajutorul a trei instrucţiuni de intrare-ieşire, efectul cărora este descris mai jos:

IN(adr_mem, adr_perif) : cerere de transfer tip intoducere a unui octet,

OUT(adr_mem, adr_perif) : cerere de transfer tip extragere a unui octet,

TEST(adr_perif) : copie cuvântul de stare într-un registru cu scopul de a consulta indicatorii.

Prin *adr_mem* şi *adr_perif* sunt desemnate adresele amplasamentului memoriei şi perifericului.

2) Programul driver-ului

Fie că vrem să extragem o secvenţă de *n* caractere care se află într-un masiv *T[0..n-1]*. Caracterul *T[i]*, dacă *i>0*, poate fi extras doar dacă a fost transferat *T[i-1]*. Pentru a verifica această condiţie se testează iterativ (aşteptare activă) indicatorul *term*.

```
iniţializare : term:=false;
        TEST(adr_perif);
        if preg = false then   // perifiericul nu este pregătit
            <tratare eroare>
        endif;
for i:=0 to n-1 do
        OUT(adr T[i], adr_perif);
 ciclu : TEST(adr_perif);
        if err = true then
            <tratare eroare>
        endif;
        if term = false then      goto ciclu
        endif;
endfor
```

Intrările-ieşirile sincrone sunt folosite la microprocesoarele cele mai simple sau atunci când procesorul nu poate fi exploatat în mod util în timpul transferului (înregistrări de măsurare, încărcarea iniţială a unui sistem).

2.4.2.2. Intrări-ieşiri asincrone cu întreruperi

1) Specificarea mecanismului.

Terminarea procesului de transferare a caracterului *T[i]* este semnalizată printr-o întrerupere. Dacă acest caracter nu este ultimul (*i<n-1*), următorul caracter poate fi transmis. Procesul este declanşat de transferul primului caracter. Acest transfer este cerut de către programul principal prin intermediul unui apel a supervizorului *(SVC intrare* sau *ieşire)*, care are drept parametri:

- adresa perifericului,
- numărul de caractere care vor fi transferate,
- adresa de origine sau de destinaţie a primului caracter,
- booleanul *term*, valoarea *TRUE* a căruia semnalizează sfârşitul transferului.

Programul principal

```
SVC ieşire { term:=FALSE
             i:=0;
             OUT(adr T[i], adr_perif);
```

programul de tratare a întreruperii

```
(la părăsirea lui T[i])

save context;
```

```
                                    TEST(adr_perif);
…                                   if err then
                                        <tratare eroare>
                                    else
                                        if i<n-1 then
<tratare paralelă>                          i:=i+1;
                                            OUT(adr T[i], adr_perif)
…                                       else
                                            term:=TRUE
if term then                            endif
…                                   endif;
                                    <restabilire context>
```

2.4.2.3. Intrări-ieşiri programate pe canal sau acces direct la memorie

Un canal este un procesor capabil să execute o secvenţă de operaţii de intrare-ieşire, definite de un program din memorie, pentru un periferic conectat la acest canal. El conţine un cuvânt de stare în care este inclus un contor ordinal ca şi un set de indicatori despre starea operaţiei curente. Programul unui canal este compus dintr-o suită de instrucţiuni, fiecare specificând o operaţie elementară de intrare-ieşire. Acest program trebuie să fie pregătit anticipat de procesorul central. Principiul de funcţionare a unui canal este următorul:

1) Un canal poate fi lansat doar de un procesor central cu ajutorul unei instrucţiuni *SIO* (*Start Input-Output,* eng.). Aceasta iniţializează cuvântul de stare a canalului, specificând adresa primei instrucţiuni care trebuie executată şi adresa perifericului origine sau destinaţie a transferului. După aceasta, activitatea canalului şi a procesorului central decurg în mod paralel.

2) Odată lansat, canalul execută programul său. Această execuţie se poate termina în trei moduri:

   a) terminarea execuţiei programului,

   b) terminare cauzată de o eroare (pană hardware, program incorect, etc.),

   c) la comanda unui procesor central, care poate în orice moment stopa execuţia programului canalului cu ajutorul unei instrucţiuni speciale *stop_canal (HIO, Halt Input-Output,* eng.).

Cazul a) este cel al funcţionării normale. Procesorul central este informat despre terminarea executării programului canalului prin intermediul unei întreruperi. Toate datele necesare derulării intrării-ieşirii sunt prezente în cuvântul de stare a canalului, care poate fi consultat de către procesorul central cu ajutorul unei instrucţiuni *testare_canal* (adesea notată prin *TIO,* eng. *Test Input Output*). În cazul b) o întrerupere oricum are loc, cu excepţia, evidentă, a cazului în care însuşi dispozitivul întreruperii a fost afectat. Cuvântul de stare a canalului conţine toate informaţiile necesare despre cauza opririi. Pentru o oprire de urgenţă, dacă procesorul central detectează o situaţie anormală, este folosită instrucţiunea *stop_canal* (cazul c).

3) Procesorul central, la orice moment de timp, poate consulta cu ajutorul instrucţiunii *testare_canal* starea canalului. Această consultare nu împiedică executarea programului procesorului central. Starea obţinută este starea corespunzătoare ultimului punct observabil.

Programul unui canal reflectă specializarea acestuia în operaţii de intrare-ieşire. O instrucţiune a canalului specifică următoarele:

- natura operaţiei care va fi executată (citire, scriere, deplasarea braţului unui disc, etc.),
- adresa de început a zonei de memorie origine sau destinaţie,
- numărul caracterelor care urmează a fi transmise.

Există o instrucţiune, care permite un salt condiţionat în cadrul programului. Pot fi de asemenea programate iteraţii.

Exemplul 2.8. Canalele calculatoarelor IBM 370. Instrucţiunile canalului conţin informaţiile menţionate mai sus la care sunt adăugaţi trei indicatori:

- un bit de **înlănţuire a instrucţiunilor**. Dacă acest bit este pus în *1*, la terminarea instrucţiunii curente se va executa instrucţiunea următoare, în caz contrar execuţia se opreşte,
- un bit de **înlănţuire a datelor**. Dacă acest bit este pus în *1*, canalul execută, la terminarea instrucţiunii curente, o instrucţiune, care are acelaşi cod al operaţiei şi este ataşată aceluiaşi periferic, ca şi instrucţiunea precedentă, dar care utilizează o zonă de memorie, specificată în

instrucţiunea următoare. Aceasta permite executarea unui transfer spre sau dintr-o zonă non-contiguă de memorie,

- un bit de **cerere a unei întreruperi**. Dacă valoarea acestui indicator este *1,* o întrerupere are loc atunci când canalul începe execuţia instrucţiunii date. Aceasta permite culegerea unor date statistice.

Un canal este lansat cu ajutorul unei instrucţiuni SIO care specifică adresa perifericului. În prealabil adresa primei instrucţiuni, care va fi executată şi cheia de protecţie utilizată în timpul transferului, trebuiesc plasate într-o locaţiune rezervată de memorie.◄

Mai sus au fost prezentate doar aspectele esenţiale ale funcţionării unui canal. Există însă câteva mecanisme, care complică în mod substanţial descrierea unui sistem real de intrare-ieşire:

- **multiplexarea canalelor şi a contolerelor**. Suntem impuşi să ştim a determina imediat momentul de timp în care canalul sau controlerul devin disponibile, pentru a le utiliza în mod eficient. Pentru aceasta sunt folosiţi indicatori şi întreruperi speciale,
- **necesitatea de a ţine cont de caracteristicile diverselor periferice,** ceea ce complică în mod substanţial relaţiile de ordonanţare,
- **detectarea şi tratarea erorilor**, cărora sunt consacrate majoritatea programelor unui administrator al intrărilor-ieşirilor.

Un ADM este o formă simplificată a unui canal: "programul" unui ADM conţine o instrucţiune unică, care specifică o adresă în memorie, o adresă de periferic, numărul octeţilor care vor fi transferaţi, sensul transferului şi semnalul terminării transferului (să se ceară sau nu o întrerupere). Acest program nu este plasat în memorie, ci într-un registru al ADM, utilizat ca şi cuvânt de stare, care poate fi consultat în timpul transferului pentru a duce evidenţa octeţilor rămaşi pentru a fi transmişi.

2.4.3. Intrări-ieşiri buferizate în memorie

2.4.3.1. Specificarea problemei

Diferenţa considerabilă dintre viteza de lucru a unităţii centrale şi cea a organelor periferice impune "buferizarea" intrărilor-ieşirilor, adică introducerea unei zone de memorie-tampon între periferic şi programul utilizatorului. Scopul este de a reduce timpul de inactivitate a unităţii centrale, dezlegând funcţionarea acesteia de periferice. Programul utilizatorului va transfera informaţiile din sau spre zona-tampon, iar această zonă-tampon va servi, în mod paralel, drept sursă sau destinaţie la schimbul de date cu perifericul. Dezlegarea unităţii centrale de periferic este cu atât mai bine realizată cu cât este mai mare capacitatea zonei-tampon. Adesea, pentru a nu supraîncărca memoria principală, zona-tampon este situată pe discul fix.

Pentru descrierea modelului sunt necesare următoarele specificaţii:

- schimburile de informaţii cu perifericele se va face prin înregistrări de lungime fixă,
- zona-tampon este de volum fix egal cu *N* înregistrări,
- pentru un program utilizator, schimbul de informaţii cu zona-tampon trebuie să simuleze schimbul cu perifericul, drept consecinţă, ordinea de depozitare sau de extragere a informaţiilor trebuie să fie bine stabilită şi nici o înregistrare să nu fie pierdută,
- aşteptările inutile trebuie reduse la minimum, atât în cazul unităţii centrale, cât şi a perifericului,
- capacitatea suporturilor nu este limitată: în cazul unei operaţii de citire organul de intrare va fi totdeauna alimentat; în cazul unei operaţii de scriere, organul de ieşire poate accepta întotdeauna o înregistrare,
- transferul se va efectua fără eroare.

2.4.3.2. Descrierea algoritmilor

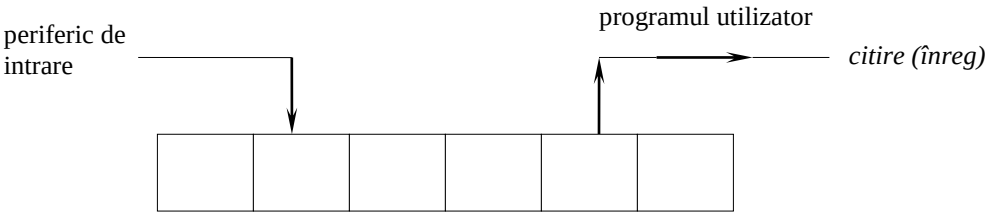
1) Citire

Un program de citire este o procedură *citire(înregistrare),* care prezintă drept rezultat valoarea înregistrării citite. Această procedură, realizată printr-o apelare a supervizorului, are ca efect citirea din zona-tampon a înregistrării următoare. Citirea este posibilă doar dacă zona-tampon conţine cel puţin o înregistrare, care nu a fost încă citită. În caz contrar, activitatea apelantă trece în starea de aşteptare.

În ambele cazuri, continuitatea alimentării zonei-tampon este asigurată de introducerea unei înregistrări de la perifericul de intrare, dacă un transfer nu are deja loc. Terminarea transferului unei înregistrări în zona-tampon provoacă o întrerupere. Pentru a menţine perifericul în stare activă, programul de tratare asociat acestei întreruperi trebuie să

relanseze transferul următor, dacă mai există loc în zona-tampon. Plus la aceasta, dacă activitatea apelantă era în aşteptarea sosirii unei înregistrări, ea trebuie relansată.

Acest mod de funcţionare poate fi schematic prezentat conform schemei programului de mai jos.



Zonă-tampon cu *N* locaţiuni

procedura *citire(înreg)*
(apelare supervizor)

if *tamponul_nu este_vid* **then**
extragere(înreg, tampon);
lansare_transfer în caz de necesitate;
else
lansare_transfer în caz de necesitate;
aşteptare
endif

tratare întrerupere

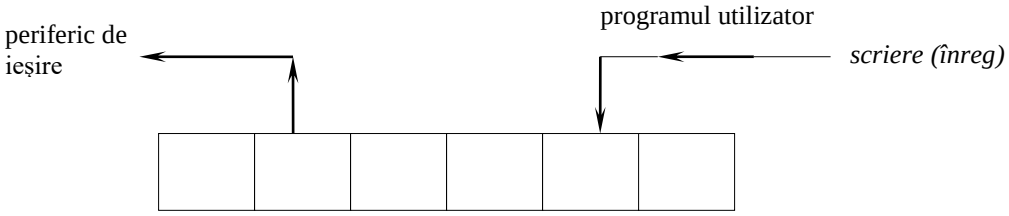
if *tamponul_nu este_plin* **then**
lansare_transfer;
if *aşteptare* **then** // aşteptare pentru citire
reluare citire
endif

În acest program "*în caz de necesitate*" semnifică "*dacă un transfer nu este deja în curs*". Sistemul este iniţializat de citirea primei înregistrări, care provoacă transferuri succesive până la umplerea zonei-tampon.

2) Scriere

Schema care corespunde unei scrieri se obţine din precedenta, înlocuind după cum urmează:

citire	prin	*scriere*
extragere	prin	*introducere*
tampon_vid	prin	*tampon_plin*
tampon_plin	prin	*tampon_vid*



Zonă-tampon cu *N* locaţiuni

procedura *scriere(înreg)*
(apelare supervizor)

if *tamponul_nu este_plin* **then**
introducere(înreg, tampon);
lansare_transfer în caz de necesitate;
else
lansare_transfer în caz de necesitate;
aşteptare
endif

tratare întrerupere

if *tamponul_nu este_vid* **then**
lansare_transfer;
if *aşteptare* **then**
reluare scriere
endif

3) Programe

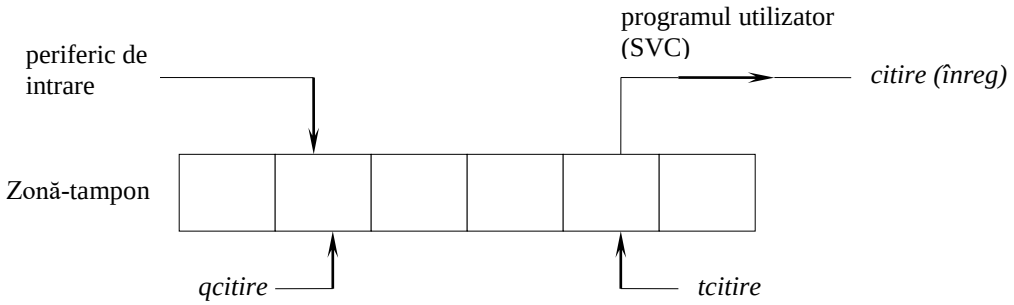
Programele sunt o transcriere a schemelor de mai sus. Zona-tampon este un fir de aşteptare, gestionat circular cu ajutorul a doi pointeri: *tcitire* (top-ul FA) - indică următoarea înregistrare care urmează a fi preluată, *qcitire* (coada FA) -

indică amplasamentul unde se va introduce următoarea înregistrare. Funcţia *suc* realizează înlănţuirea amplasamentelor; o realizare simplă pentru o zonă-tampon secvenţială, care are *N* înregistrări a câte *L* amplasamente este de forma:

suc(ptr) = (ptr + L) **mod** *(L*N)*

Vom presupune, că activitatea care cere operaţia *citire* este unica care va utiliza procesorul; trecerea acesteia în starea de aşteptare se va realiza trecând în starea de aşteptare procesorul. Această soluţie este impracticabilă în cazul unor activităţi multiple (este necesar de a relansa o altă activitate), dar o vom accepta din motive de simplitate.

Mai subliniem că transferul este comandat de un canal sau un ADM, fără a detalia programul respectiv, pregătit de obicei anticipat; numai valorile parametrilor se vor schimba de la o execuţie la alta. Procedurile *lansare_citire* şi *lansare_imprimare* realizează pregătirea programului şi lansarea transferului.



Variabile	Semnificaţie	Valoare iniţială
L	lungimea unei casete (înregistrări)	const.
Ncitire	numărul de casete a zonei-tampon	const
nvcitire	numărul de casete vide	*Ncitire*
citire_activ	indicator al activităţii perifericului de intrare	*false*
tcitire, qcitire	pointeri top şi coadă	0, 0
suc()	pointer spre caseta următoare	…

citire(înreg)

```
save(zonă_svc);
început:
if nvcitire<Ncitire then
    înreg:=tampon[tcitire];
    tcitire:=suc(tcitire);
nvcitire:=nvcitire+1;
    if non citire_activ then
      citire_activ:=true;
      lansare_perif_de_intrare(qcitire)
    endif
else
    if  non citire_activ then
      citire_activ:=true;
      lansare_perif_de_intrare(qcitire)
    endif;
    csplucrare:=<aşteptare,slave,demascat, adr început>;
    aşteptare_citire:=true;
    încărcare_csp(csplucrare);
endif;
restabilire(zonă_svc);
încărcare_csp(svc_vechi);
```

tratare_întrerupere_perif_de_intrare

```
save(zonă_întrerupere);
qcitire:=suc(qcitire);
nvcitire:=nvcitire-1;
if nvcitire>0 then
    lansare_perif_de_intrare(qcitire)
else
    citire_activ:=false
endif;
if aşteptare_citire then
    întrerupere_veche.act:=activ;
    aşteptare_citire:=false
endif;
restabilire(zonă_întrerupere);
încărcare_csp(întrerupere_veche);

lansare_perif_de_intrare(qcitire)
elaborare prog.canal:
<sens=citire
  adr.memorie=adr tampon[qcitire]
  contor_octeţi=L>
SIO
  <prog=prog.canal
   perif= perif_de_intrare>
```

Execuţia programelor de mai sus ne impun să facem o remarcă foarte importantă. Trebuie să notăm, că executarea tratării întreruperii este declanşată de sfârşitul transferului fizic al unei înregistrări, activitate, care are loc paralel cu executarea programului primitivei *citire(înreg)* pe procesor. Drept rezultat, este posibil ca întreruperea să survină într-un

punct oarecare (observabil) al executării acestei primitive, ceea ce va însemna intercalarea executării programului de tratare a întreruperii *între două puncte oarecare* ale programului *citire(înreg).* Consecinţele pot fi catastrofale, cum ne arată exemplul următor.

Să considerăm instrucţiunea a doua

```
if non citire_activ then
   citire_activ:=true;
   lansare_perif_de_intrare(qcitire)
endif
```

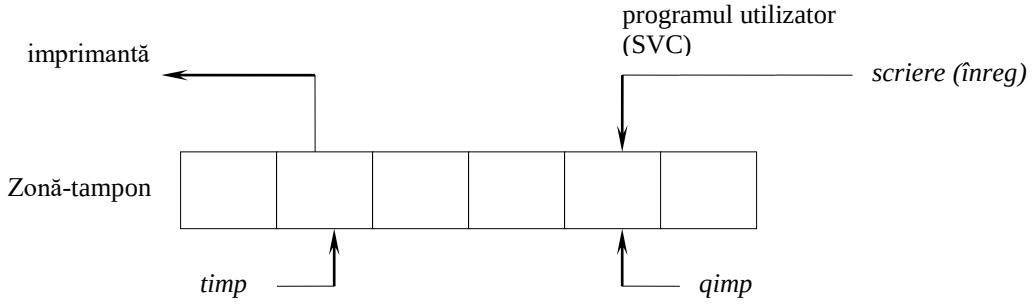
din programul *citire(înreg).* Ea se descompune în următoarele instrucţiuni elementare:

```
(a)   load R, citire_activ
(b)   jifz R, urm                    -- ramificare dacă true
(c)   …
urm:  …
      <aşteptare>
```

Presupunem că întreruperea soseşte între instrucţiunile (a) şi (b), valoarea citită în *R* de (a) este *true*; programul de tratare a întreruperii pune *citire_activ* în *false*. După prelucrarea întreruperii, instrucţiunea (b) este executată cu *R* având valoarea *true* şi activitatea apelantă trece în starea de aşteptare în care rămâne un timp indefinit, deoarece perifericul nu a fost relansat. Pot exista şi alte cazuri de funcţionare incorectă, legată de momentul sosirii unei întreruperi.

Putem afirma, că validitatea variabilelor comune (de ex. *citire_activ, nvcitire etc.,*) nu poate fi garantată dacă autorizăm sosirea întreruperilor de terminare a operaţiilor de intrare-ieşire în orice moment de timp. Apelarea supervizorului pentru a realiza secvenţa de cod *citire_înreg* trebuie executată cu *întreruperile de intrare-ieşire mascate.* Situaţia dată este cunoscută sub numele de **excludere mutuală** şi este o consecinţă a execuţiei activităţilor asincrone.

Programele pentru un periferic de ieşire (imprimare, de ex.) sunt analogice. Remarca despre mascarea întreruperilor rămâne valabilă.



Variabile	Semnificaţie	Valoare iniţială
L	lungimea unei casete (înregistrări)	const.
Nimp	numărul de casete a zonei-tampon	const
nvimp	numărul de casete vide	*Nimp*
imp_activ	indicator al activităţii perifericului de intrare	*false*
timp, qimp	pointeri top şi coadă	0, 0
suc()	pointer spre caseta următoare	…

scriere(înreg)

```
save(zonă_svc);
început:
if nvimp>0 then
    tampon[timp]:= înreg;
    timp:=suc(timp);
nvimp:=nvimp-1;
```

tratare_întrerupere_imp

```
save(zonă_întrerupere);
qimp:=suc(qimp);
nvimp:=nvimp+1;
if nvimp<Nimp then
    lansare_imp(qimp)
else
```

```
    if non imp_activ then                                    imp_activ:=false
      imp_activ:=true;                                     endif;
      lansare_imp(qimp)                                    if aşteptare_imp then
    endif                                                      întrerupere_veche.act:=activ;
else                                                           aşteptare_imp:=false
    if  non imp_activ then                                 endif;
      imp_activ:=true;                                     restabilire(zonă_întrerupere);
      lansare_imp(qimp)                                    încărcare_csp(întrerupere_veche);
    endif;
    csplucrare:= :=<aşteptare,slave,demascat, adr început>;lansare_imp(qimp)
    aşteptare_imp:=true;                                   elaborare prog.canal:
    încărcare_csp(csplucrare);                             <sens=ieşire
endif;                                                       adr.memorie=adr tampon[qimp]
restabilire(zonă_svc);                                       contor_octeţi=L>
încărcare_csp(svc_vechi);                                  SIO
                                                             <prog=prog.canal
                                                              perif= imprimantă>
```

2.4.4. Încărcarea iniţială

Pentru a începe lucrul maşina "goală" trebuie să aibă încărcat în memoria operativă programul sistemului de operare şi să execute un salt la prima instrucţiune a acestui program. Aceasta presupune prezenţa în memorie a unui program de citire, care el însuşi trebuie încărcat în prealabil. La primele calculatoare, un program de câteva instrucţiuni, introdus manual în memorie cu ajutorul unor comutatoare ale pupitrului de comandă la o adresă fixă, permitea încărcarea unui program de citire mai elevat capabil să citească versiunea completă a sistemului de operare. Acest proces era numit **încărcare** (bootstrapping, eng.).

În calculatoarele contemporane, programul de iniţializare se află permanent într-o memorie constantă (BIOS). Execuţia acestui program este declanşată odată cu punerea sub tensiune a calculatorului (autoîncărcare).

Exerciţii

Exerciţiul 2.1.

Elaboraţi schema unui program de intrare-ieşire cu tratare a întreruperilor conform schemei date în 2.4.2.2. cu modificările următoare:

a) în cazul detectării unei erori de transfer, se va încerca repetarea operaţiei până la un număr maxim de încercări n_{max}, înainte de a anula intrarea-ieşirea,

b) dacă transferul nu a fost realizat într-o perioadă de timp t_{max} specificată, să se anuleze operaţiile de intrare-ieşire.

Pentru ambele cazuri să se apeleze o procedură specificată de tratare a erorii.

Exerciţiul 2.2.

Un calculator este utilizat pentru culegerea unor date de măsurare. El execută periodic (cu perioada T) un ciclu de activităţi:

- culegerea rezultatelor de măsurare de la captoare (durata t_m)
- procesarea rezultatelor de măsurare (durata t_p)
- vidajul pe disc al rezultatelor (durata t_v)

Toate cele trei operaţii de mai sus trebuie să fie executate consecutiv pentru fiecare ciclu. Există un ceas, care provoacă o întrerupere la trecerea unui contor prin zero. Să se elaboreze programele acestui sistem (programul principal, programul de tratare a întreruperilor) pentru următoarele trei cazuri (formulând pentru fiecare caz, dacă va avea loc, condiţia posibilităţii tratării în funcţie de durata diferitor operaţii).

a) Captoarele şi controlerul discului sunt comandate de unitatea centrală. Ciclurile succesive de măsurare trebuie executate consecutiv şi un task de fond este executat pentru perioada timpului mort.

b) Presupunem că $t_m+t_p < T < t_m+t_p+t_v$. Vidajul ciclului i trebuie efectuat paralel cu preluarea datelor ciclului $i+1$. Se presupune că discul este comandat de ADM.

c) Presupunem că $t_m < T < t_m+t_p$. Trebuie executat în paralel şi preluarea datelor şi vidajul pentru ciclurile consecutive.

Exerciţiul 2.3.

O instrucţiune *citire_poziţie(x,y)* permite citirea poziţiei curente pe ecran a mouse-ului, definită prin coordonatele sale carteziene *x* şi *y*. O altă instrucţiune *init_mouse* permite să definim poziţia curentă a mouse-ului drept origine a sistemului de coordonate.

Să se elaboreze un program, care va permite urmărirea deplasării mouse-ului pe ecran.

a) Admitem că viteza maximă de deplasare a mouse-ului este de 10 *cm/s*. Care este frecvenţa de preluare a coordonatelor dacă se cere să localizăm poziţia mouse-ului definită printr-un pixel pe un ecran de *30x30* cm şi care conţine 1024x1024 pixeli?

b) Mouse-ul este gestionat de un program activat periodic, cu perioada *T* aleasă în funcţie de rezultatul obţinut.

a) Să se elaboreze acest program presupunând disponibile două proceduri:

- *afişare_cursor(x,y)*, care afişează un cursor centrat pe *x, y,*
- *deplsare_cursor(d_x,d_y),* care deplasează cursorul, presupus afişat, la distanţa dictată de vectorul *(d_x,d_y)*.

Avem la dispoziţie un ceas cu întrerupere la trecerea prin zero.