Proiectarea şi analiza sistemelor de operare

1. Introducere

Scopul lucrării este expunerea bazelor contemporane ale metodelor şi mijloacelor de elaborare a resurselor program de sistem (inclusiv, operaţii asincrone, tratarea întreruperilor, interfeţele sistemelor de operare, compromisele dintre dispozitivele tehnice şi resursele program) şi pregătirea cititorului pentru proiectarea şi analiza sistemelor de operare (S.O.).

Resursele fizice ale sistemelor de calcul contemporane posedă caracteristici tehnice extraordinare şi pot fi utilizate în cele mai diverse scopuri. Însă aceste resurse fără compartimentul logic de sistem (software de sistem) întâmpină dificultăţi majore în relaţiile cu mediul în care trebuie să funcţioneze. *Acesta este* ***motivul principal*** *al 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].

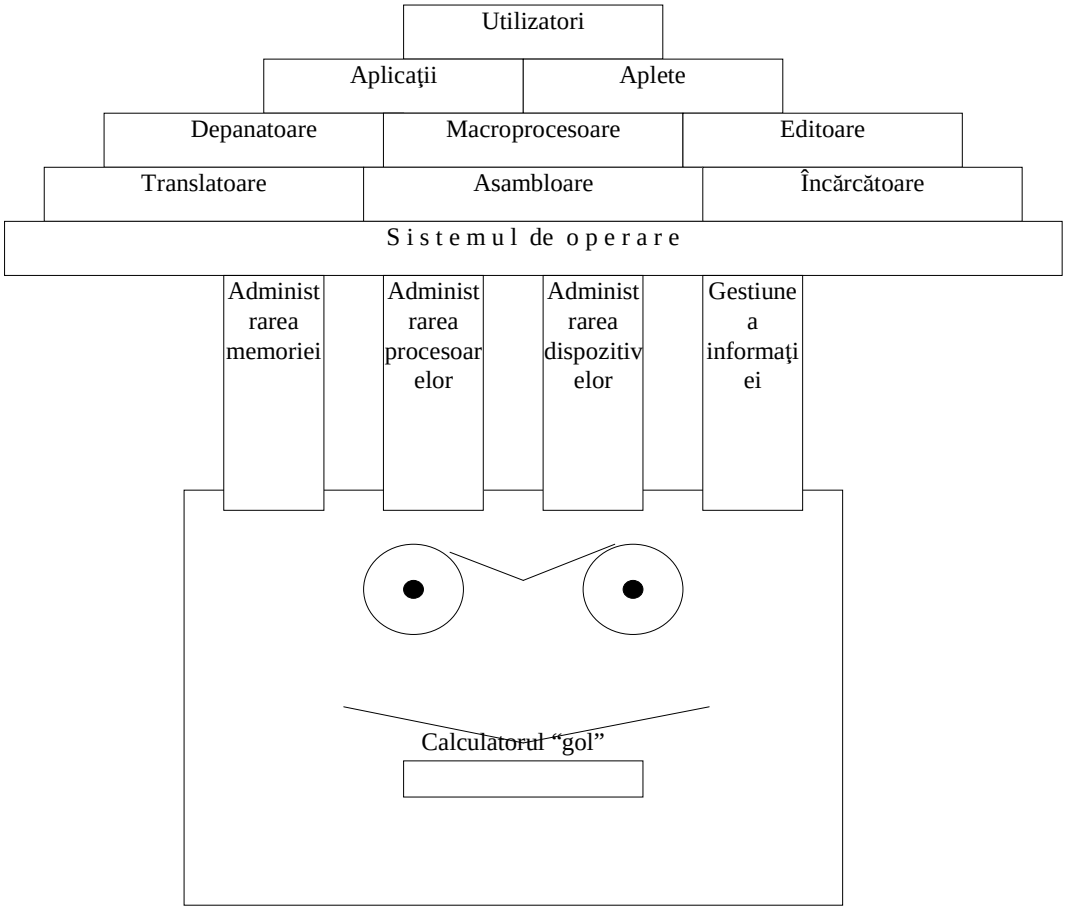

Fig.1.1. Relaţia dintre un sistem de operare şi componentele unui calculator

Există mai multe motivaţii ale necesităţii studierii sistemelor de operare, cele mai importante fiind următoarele:

- pentru utilizarea resurselor hardware în scopuri speciale poate fi necesar să se creeze un sistem de operare propriu sau să se introducă modificări în sistemul 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 sistemelor de operare, pot fi implementate cu succes şi în alte domenii (ultima motivaţie conform ordinii de expunere, dar nu şi ultima ca importanţă).

Prin noţiunea ***sistem de operare*** înţelegem 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, depanatoarele, bibliotecile, mediile integrate (Visual) etc. nu sunt incluse în definiţia unui sistem de operare, fiind considerate şi ele utilizatori ai sistemului de operare.

Noţiuni de bază şi clasificări

Un calculator constă dintr-un ansamblu de componente funcţionale fizice şi logice, care cooperează în satisfacerea cerinţelor utilizatorilor privind introducerea, stocarea, prelucrarea, transmisia şi regăsirea 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.

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 subînţelege componentele (dispozitivele) care formează calculatorul şi legăturile dintre ele. Componentele principale sunt: *procesorul, memoria, 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, dar, istoric prima şi, devenită mai apoi clasică, este **structura John von Neumann,** prezentată în fig.1.2.

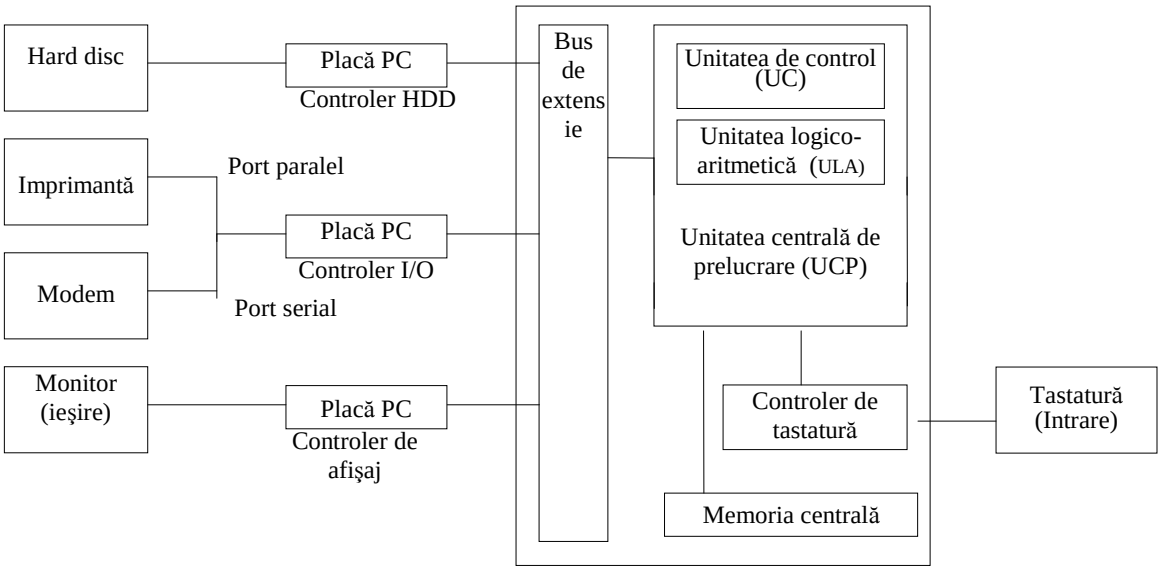

Fig.1.2. Structura unui calculator

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 CPU stochează datele şi programele pentru a le consulta în mod rapid. Memoria poate fi organizată în mod ierarhic, caz în care există cel puţin două nivele de ierarhie – **memoria centrală** (operativă) 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. 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. CU controlează în principal funcţiile I/O, de memorie 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 temporare, numite ***registre***. Datele sunt transferate ontre 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 sau 128).

Mărimea magistralei externe de date măsoară câte date pot fi transferate între procesor şi dispozitivele periferice într-o singură operaţie. 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 către calculator fără eforturi speciale.
Un **ceas** electronic asigură coordonarea corespunzătoare a numeroaselor elemente ale calculatorului. Componentele calculatoarelor cu performanţe superioare pot opera la **frecvenţe de ceas** mai mari. Frecvenţa ceasului indică viteza de operare a CPU. Ea se măsoară în milioane de impulsuri de ceas pe secundă, adică în *megahertzi* (MHz).

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 funcţiuni. 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, generalitatea, 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 respectivă 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 on 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).

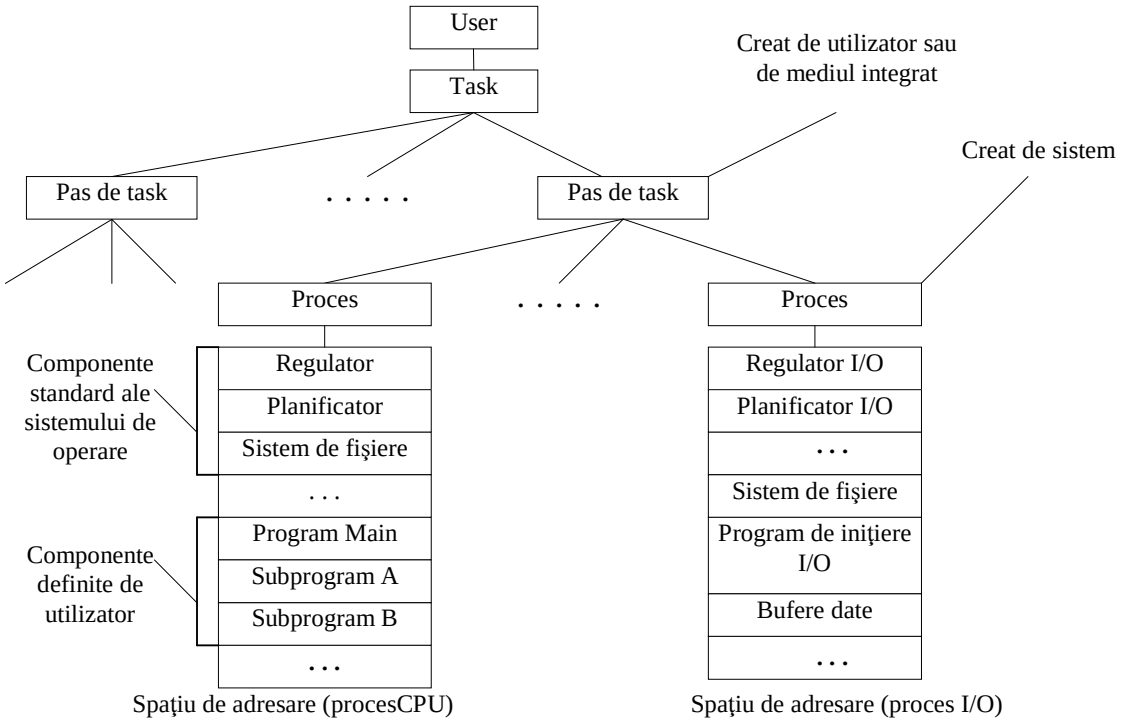

Fig.1.3. Relaţia dintre utilizator, sarcină, pas de task, proces şi spaţiu de adrese

Generalitatea, flexibilitatea, extensibilitatea măsoară gradul în care un sistem poate fi folositor şi adaptabil 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.
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 ontr-un limbaj de programare a unui algoritm, altfel – programul este o secvenţă de instrucţiuni sau simplu **cod**. **Utilizatorul (user)** este oricare doritor să îndeplinească anumite lucrări la calculator. Prin **lucrare (sarcină, task)** vom subînţelege un set de acţiuni, necesare pentru îndeplinirea unui lucru anume. Sarcina poate conţine mai mulţi **paşi**. **Paşi de task** sunt unităţi de lucru, care vor fi îndeplinite consecutiv, de exemplu trei paşi – compilare, încărcare şi executare.
Primind un task de la utilizator, sistemul de operare poate crea câteva **procese**. Până la tehnologia Win32 prin **proces** se considerau **calculele** care puteau fi efectuate paralel cu alte calcule. Relaţia dintre un **user**, **task, proces** şi **spaţiu de adrese** este prezentată în fig.1.3.
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**. În fig.1.3 sunt aduse două exemple de spaţii de adrese – unul pentru procesul de intrare/ieşire (input/output, I/O) şi altul pentru procesul unităţii centrale.
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 stare de execuţie. Un proces se consideră în stare de **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 on starea de **execuţie** să fie şi executat de procesor la un moment dat.
Resursele hardware de **protecţie** sunt utilizate cel mai des pentru controlul accesării memoriei.
Resursele hardware de **ontrerupere** permit sistemului de operare să coordoneze operaţiile care au loc în acelaşi timp; pot fi utilizate şi pentru a schimba ordinea de execuţie a programelor. **Ontreruperea** 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**.

Tipuri de sisteme de operare, obiective şi funcţii

Valorile concrete ale atributelor sistemelor de operare şi combinaţii ale acestora determină diverse tipuri de sisteme şi restricţiile 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 ca un 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, traitement par lots) execută la un moment dat un singur program, care trebuie terminat înainte de a lua un alt program în consideraţie.
Sistemele cu ***multiprogramare*** acceptă la un moment 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 *sarcini (multitasking)* sau mai multe *procese.*
Sistemele on ***timp real*** sunt dedicate, de obicei, funcţionării în cadrul unor sisteme de comandă şi este necesar ca valorile anumitor atribute să se încadreze în limite destul de restrictive, dictate de dinamica proceselor comandate.
Tipurile de sisteme de operare enumerate mai sus nu sunt nici disjuncte şi nici exhaustive. 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 on primul rвnd **maximizarea eficienţei sistemului de calcul şi a satisfacţiei utilizatorilor**. Tot la obiective poate fi trecută şi cererea de **minimizare a posibilităţii de apariţie a unor erori** şi de **maximizare a transparenţei** sistemului de operare, **garantarea securităţii datelor**, **optimizarea controlului comunicaţiilor** în cazul unei reţele de calculatoare. Un obiectiv foarte important este necesitatea de **minimizare** a efortului **concepţie-realizare** a sistemului, ultim on enumerare, dar poate cel mai 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 respectiv să o retragă.

Exemple de sisteme de operare

Exemplele care urmează [6] 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.

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.1.4).

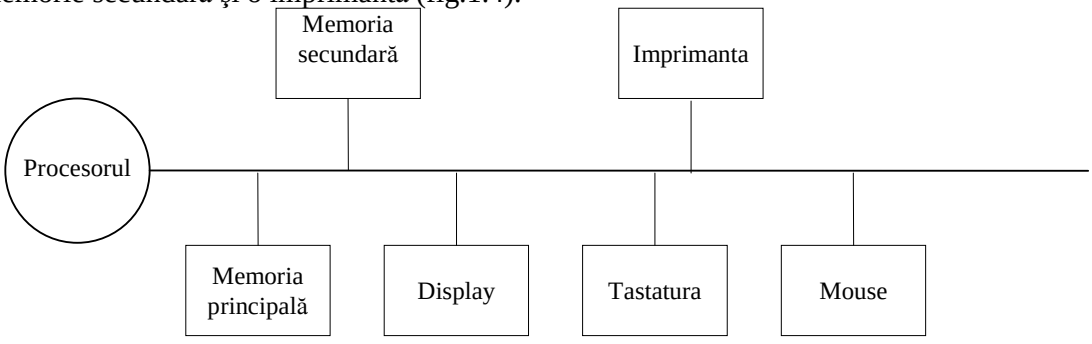

Fig.1.4. Configuraţia unui calculator personal

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

- 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;
- executarea unor programe existente în PC sau introduse sub formă de fişiere; introducerea datelor necesare executării programului (de la tastatură, din fişier sau de la alte surse); listarea rezultatelor la display, imprimantă sau copierea lor într-un fişier.

Sistemul de operare va 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 vor fi introduse in mod interactiv aceste date 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* este lipsă, or PC 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 caracteristice 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).

Comanda unor procese industriale

La o uzină chimică sunt utilizate doua produse iniţiale *A* şi *B* pentru sinteza produsului *C* conform fig.1.5.

Procesul de fabricaţie 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.

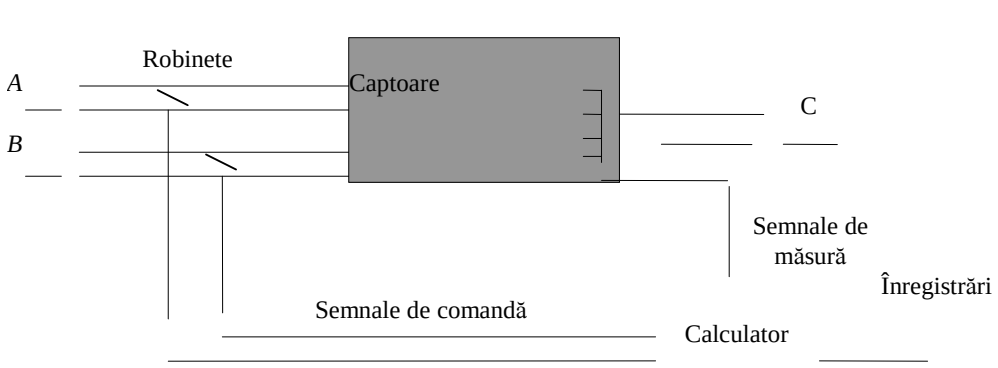

Fig.1.5. Schema unui proces

Acest mod de funcţionare introduce unele restricţii, descrise mai jos.
Măsurările sunt făcute 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 comanda robinetelor). Sistemul va funcţiona doar în cazul respectării relaţiei de restricţionare $\boldsymbol{t \leq T}$.
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 fişierului jurnalului de bord).

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

Sisteme tranzacţionale

Caracteristicile principale ale sistemelor cu tranzacţii sau tranzacţionale sunt următoarele:

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

Ca exemplu pot fi menţionate sistemele de rezervare a biletelor de tren sau avion, de gestionare 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.

Sisteme on timp partajat

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;
- de servicii 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 caracteristicilor resurselor fizice, calitatea interfeţei şi serviciilor utilizatorului, facilitatea adaptării şi extensibilităţii.

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. 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 acesta intră în administrarea procesului (fie el şi al utilizatorului), care a cerut serviciul. Resursele (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 program anume? Pentru a răspunde la aceste întrebări (şi la altele) vom face cunoştinţă cu ciclul de viaţă a unui proces. În fig.1.6 sunt prezentate trei procese (trei sarcini ale utilizatorilor) existente ontr-un sistem cu multiprogramare.

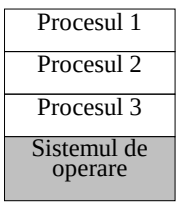

Fig.1.6. Trei procese ontr-un sistem cu multiprogramare

Ciclul de viaţă a unui proces poate fi reprezentat printr-un set de stări ale procesului şi trecerea 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** – procesul aşteaptă să se producă un anumit eveniment, a cărui apariţie este indispensabilă, proces **eligibil** – procesul are la dispoziţie toate resursele necesare lipsă fiind doar procesorul, adică este pregătit să se execute din momentul alocării unităţii centrale (fig.1.7).

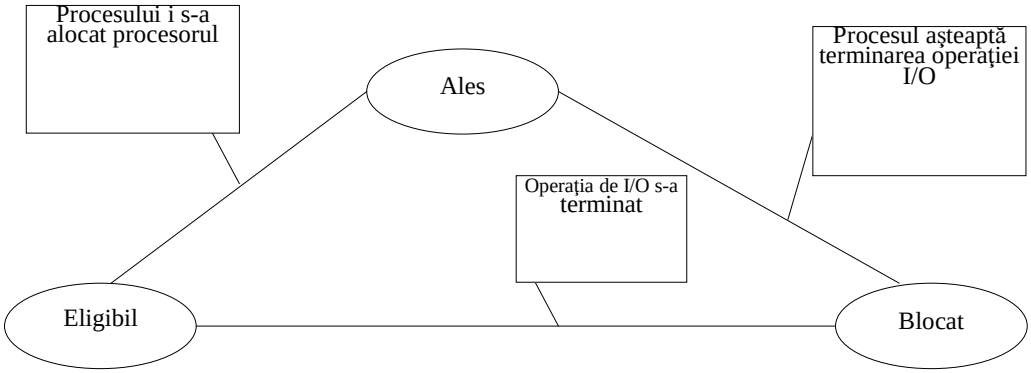

Fig.1.7. Ciclul de viaţă a unui proces

Ciclul de viaţă a unui proces, prezentat în fig.1.7 sugerează existenţa "veşnică" a proceselor. Mai aproape de realitate un proces este reprezentat în fig.1.8. Aici norii semnifică componentele sistemului de operare, care tratează evenimentele respective, iar pentru modelarea procesului real au fost adăugate trei stări suplimentare:

- ***prezentare*** – utilizatorul prezintă sistemului sarcina, sistemul trebuie să reacţioneze la cererea utilizatorului,
- ***păstrare*** – sarcina este reprezentată în formă internă, dar resursele nu sunt încă alocate,
- ***terminare*** – calculele corespunzătoare procesului au luat sfârşit, toate resursele alocate procesului pot fi eliberate şi întoarse sistemului.

Sistemul de operare este acea componentă, care va ghida procesul prin toate aceste stări. În fig.1.8 este prezentată una din variantele posibile ale ciclului de viaţă a procesului. Pot exista abateri de la acest model atât în sensul complexităţii, cât şi în direcţie opusă. Oricum, acest model este foarte util pentru rezolvarea problemei proiectării sistemelor de operare.

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 un calculator fără "hainele program" (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 hardware. 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 on 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

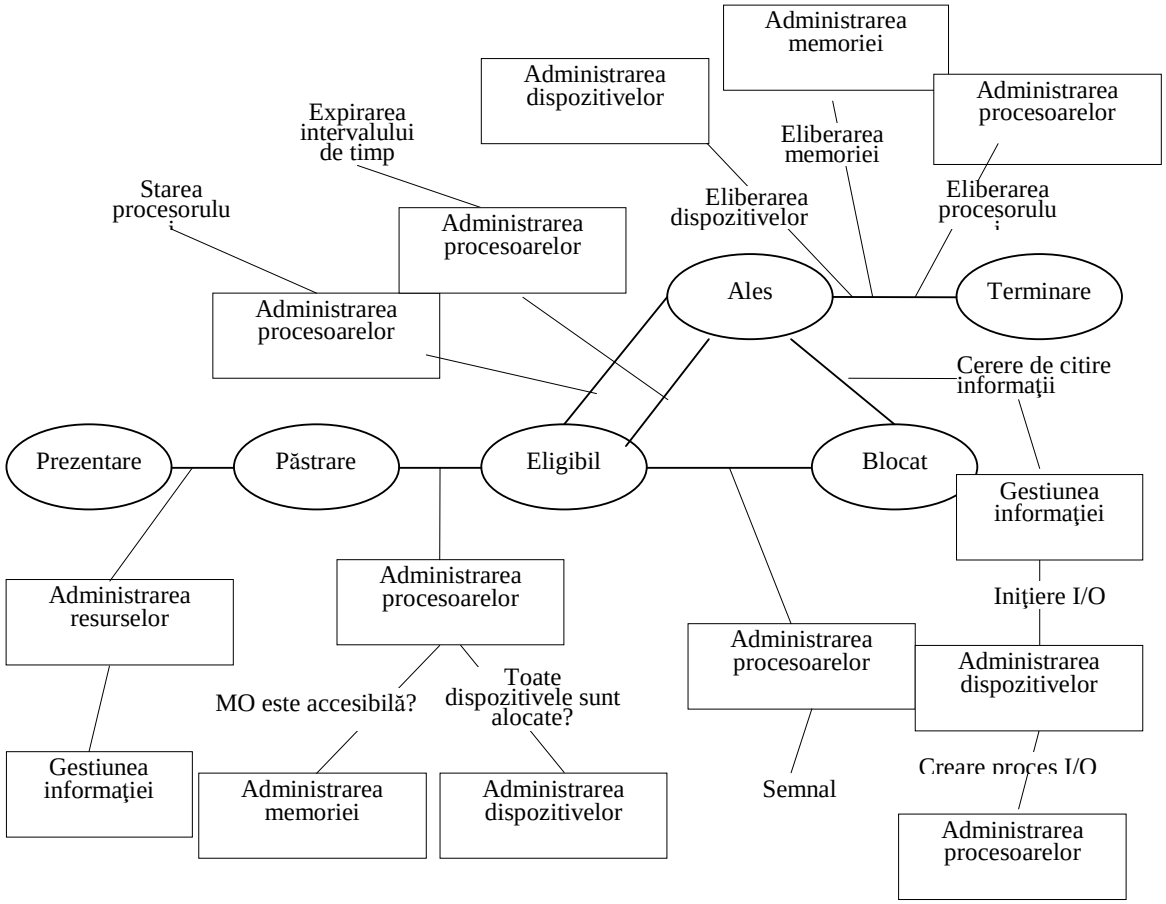

Fig.1.8. Schimbarea stărilor unui proces şi componentele SO responsabile de aceste schimbări

Instrucţiunile 1, 4 şi 5 sunt instrucţiuni standard în multe calculatoare contemporane. Însă execuţia corectă şi eficientă a instrucţiunilor 2 şi 3 poate necesita 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. Din această cauză, 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.1.9.
Nucleul sistemului de operare va fi executat pe maşina "goală", iar programele utilizatorului – pe maşina extinsă. Să facem cunoştinţă acum cu modul în care sunt organizate ontr-un tot ontreg componentele sistemelor de operare.
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, on cazul sistemelor de operare, poate fi utilizat în două nivele (fig.1.10.) şi conduce la noţiunea de ***maşină ierarhică*** [4]:
Primul nivel - funcţiile cheie, utilizate de majoritatea modulelor de sistem, pot fi realizate on cadrul unei *maşini extinse interne* şi

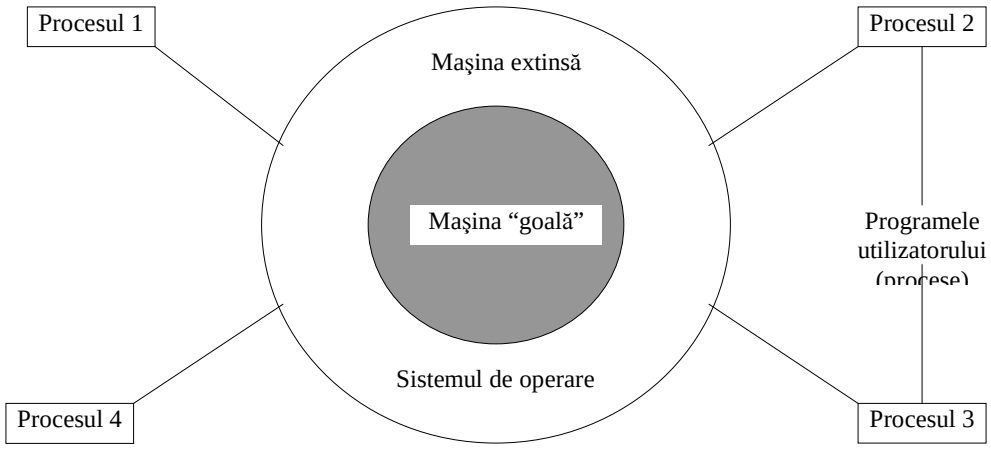

Fig.1.9. Maşina extinsă

Nivelul doi - unele module pot fi executate on cadrul unei *maşini extinse externe*, analogic proceselor utilizatorului.

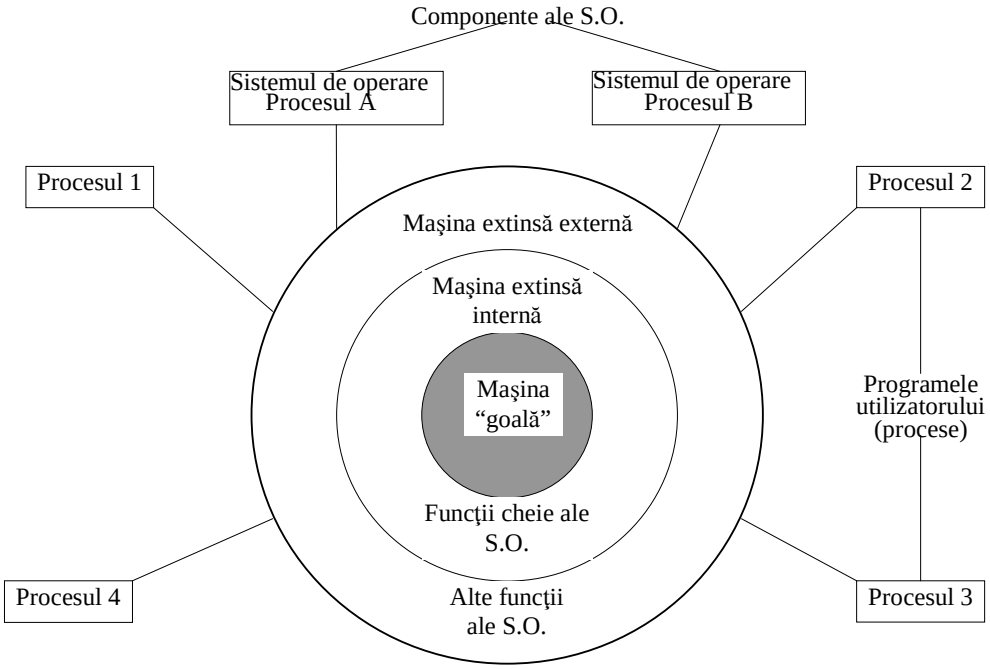

Fig.1.10. Ilustrarea conceptului de maşină ierarhică

On 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 utilizвnd 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.1.11. Procesele (incluse în dreptunghiuri) se adresează către funcţiile nucleului şi utilizează împreună toate resursele 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 ontr-un nivel oarecare, pot accesa (se pot adresa) numai resursele nivelelor inferioare.

Taskuri

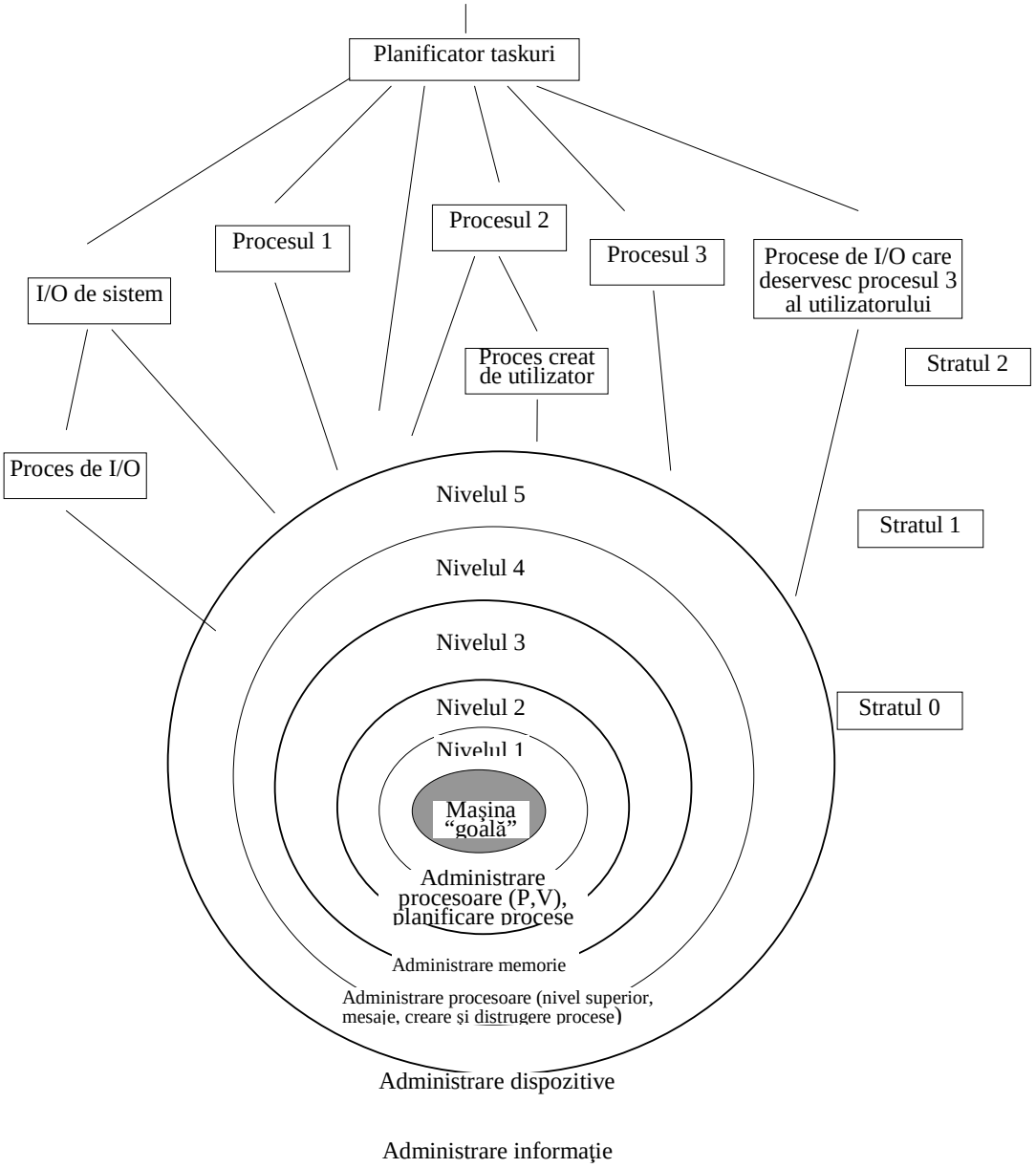

Fig.1.11. Structura ierarhică a sistemului de operare

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, necesită 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*. Pot fi semafoare simple (binare) sau polivalente (care pot avea mai multe valori). 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

comanda. 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 (Win32, de exemplu) noţiunea de proces a fost substanţial modificată, introducându-se alte concepte (thread-uri), care exploatează într-un mod mai eficient ideile multitasking-ului şi multiprogramării.

Alte puncte de vedere asupra sistemelor de operare

On 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.

Abordare funcţională

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 (translatoare, biblioteci, medii integrate, etc.). Pentru a evita posibilitatea apariţiei unei astfel de probleme aceste programe, de obicei, nu sunt considerate componente ale sistemului de operare.

Abordare din punctul de vedere al interfeţei cu utilizatorul

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 utilizatorul era obligat să indice în mod

explicit şi manual fiecare pas, oricât de nesemnificativ ar fi părut. Formularea paşilor cu ajutorul 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 în cunoştinţă de cauză 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, principala fiind lipsa posibilităţii lucrului interactiv.
Microprocesoarele şi memoriile anilor ’70 au pus problema lansării pe piaţă a calculatoarelor personale (PC) cu toate consecinţele respective. 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.

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 ’60 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 taskurilor. 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 mai 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 concordării 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 în viitorul 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.

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. Utilitele 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" [6], era de o eficacitate minimă, dispozitive foarte costisitoare fiind utilizate ineficient. Din această cauză la sfârşitul anilor '50 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 automatizată.
Funcţia principală a unui atare sistem era gestiunea resurselor: memoria, procesorul, intrarea-ieşirea. 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 buclele 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 eliberat 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.

Multiprogramarea şi partajarea timpului

Progresul tehnologic şi conceptual al anilor '60 - '70 a permis excluderea unor limitări caracteristice sistemelor de prelucrare pe loturi. Astfel, au fost introduse procesoare specializate pentru operaţiile de intrare-ieşire (unităţile de schimb sau canalele), care permitea eliberarea procesorului central de gestionarea dispozitivelor de intrare-ieşire. Utilizarea principiului multiprogramării sau partajarea memoriei între mai mulţi utilizatori a permis o utilizare şi mai bună a procesorului central. Exploatarea unui calculator conform principiului timpului partajat oferă utilizatorilor posibilităţi analogice unui calculator individual, permiţând beneficiul unor servicii comune la un preţ redus.

Organizarea intrărilor - ieşirilor în memorii 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.1.12.

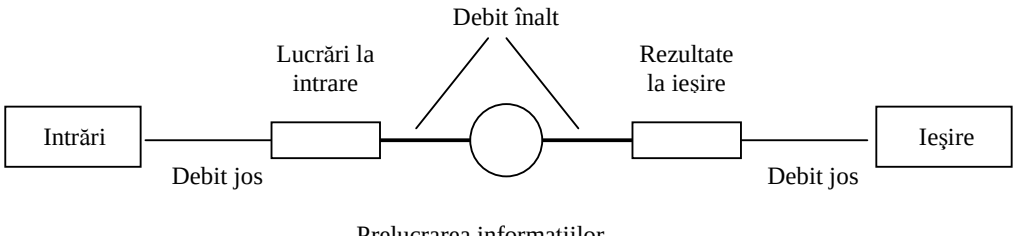

Fig.1.12. Buferizarea intrărilor-ieşirilor

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 nuvele ale memorie fiind de asemenea comandate de un canal.
Deşi utilizarea memoriilor tampon prezintă o serie de avantaje în vederea sporirii randamentului dispozitivelor calculatorului, totuşi două momente negative pot fi menţionate:

- atunci când lucrarea în curs de execuţie are nevoie de nişte 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.

Multiprogramarea

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

- o lucrare aflată în starea *blocat* ar putea folosi unitatea centrală, eliberată de o lucrare care este în aşteptarea datelor,
- unitatea centrală ar putea schimba timpul său de alocare înainte de terminarea unei lucrări în scopul satisfacerii cerinţelor legate de *timpul de răspuns.*

On acest caz este necesar ca valoarea timpului de realocare 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 fig.1.13.

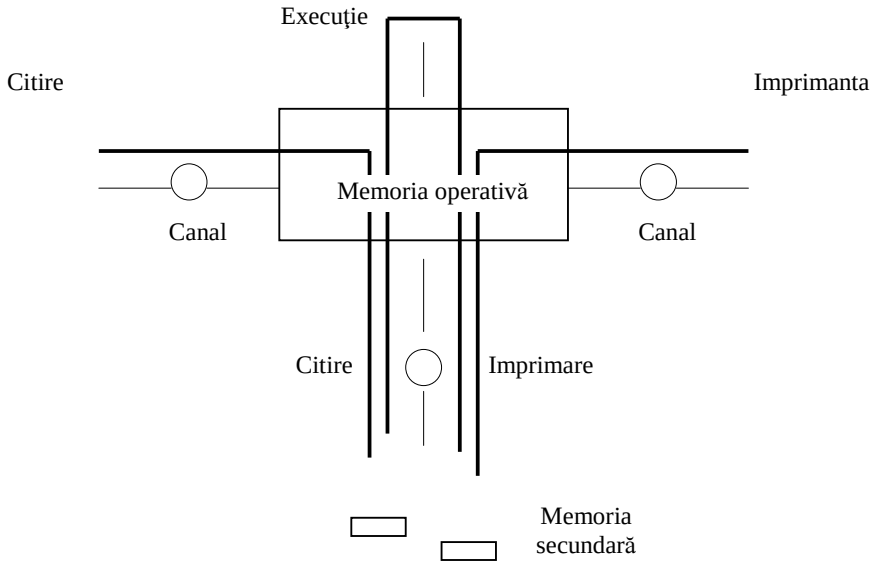

Fig.1.13. Fluxul informaţional într-un sistem cu multiprogramare

Această schemă evidenţiază două momente importante: rolul principal al memoriei operative (în sistemele 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 aceste două nivele ale memoriei devin caracteristici determinante ale performanţei sistemului.
Principalele avantaje şi restricţii ale multiprogramării pot fi prezentate în rezumat după cum urmează:

- un sistem cu multiprogramare este mai complicat pentru că el trebuie să asigure partajarea memoriei şi protecţia reciprocă a programelor;
- multiprogramarea necesită dispozitive speciale pentru reactivarea programelor şi protecţia memoriei;
- un sistem cu multiprogramare asigură o utilizare mai uniformă a diferitor resurse: 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.

Sisteme cu timp partajat

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 foarte mici (*cuante*) 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 executarea de către sistem a serviciului cerut. 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, deci, 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 este neglijabil în comparaţie cu valoarea cuantei. Multiprogramarea devine obligatorie, dacă ţinem cont de raportul dintre acest timp şi timpul necesar pentru încărcarea unui program din 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 nu au făcut acest lucru au avut doar de câştigat.
Apariţia microprocesoarelor cu posibilităţi mari de calcul şi de 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 pentru sistemele informatice ale anilor care au urmat.

Realizări conceptuale

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 sistemelor de operare şi a condus la elaborarea unor metode proprii de concepere.
Debutul unei cercetări ştiinţifice a sistemelor de operare poate fi considerat anul 1964, care a succedat o etapă importantă de dezvoltare tehnică: primele sisteme cu partajare a timpului (Thor, CTSS), anunţarea seriei IBM 360 şi introducerea canalelor de intrare-ieşire, prima memorie paginată (Atlas), etc.
On perioada 1965-1968 au fost elaborate noţiunile necesare conştientizării gestionării activităţilor fizice sau conceptuale paralele: proces secvenţial, formularea şi soluţionarea unor probleme de excludere mutuală, sincronizarea şi inventarea semaforului. Acestea au fost aplicate cu succes în 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 '70. Utilite 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 in memoria fizică, direct adresabilă.
Restricţiile de capacitate şi preţ a memoriei principale a condus foarte repede la apariţia necesităţii unei memorii secundare şi a 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 1962 sistemul Burroughs B5000 utiliza un mecanism de adresare logică prin unităţi de volum variabil sau segmente.

În 1966 este propus un model de arhitectură pentru calculatoare cu partajare a timpului, implementat în IBM 360/67. În acest calculator segmentele sunt doar simulate printr-o dublă paginaţie, din care cauză noţiunea de segmentare va rămâne pentru mult 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 adresare uniform (contiguu). Au fost introduse noţiunile de modularitate şi structurare ierarhică, obiect, etc.

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 sistemelor de operare şi a condus la elaborarea unor metode proprii de concepere.
Debutul unei cercetări ştiinţifice a sistemelor de operare poate fi considerat anul 1964, care a succedat o etapă importantă de dezvoltare 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 elaborate noţiunile necesare conştientizării gestionării activităţilor fizice sau conceptuale paralele: proces secvenţial, formularea şi soluţionarea unor probleme de excludere mutuală, sincronizarea şi inventarea semaforului. Acestea au fost aplicate cu succes în 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 '70. Utilite 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 in memoria fizică, direct adresabilă.
Restricţiile de capacitate şi preţ a memoriei principale a condus foarte repede la apariţia necesităţii unei memorii secundare şi a 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 1962 sistemul Burroughs B5000 utiliza un mecanism de adresare logică prin unităţi de volum variabil sau segmente.
În 1966 este propus un model de arhitectură pentru calculatoare cu partajare a timpului, implementat în IBM 360/67. În acest calculator segmentele sunt doar simulate printr-o dublă paginaţie, din care cauză noţiunea de segmentare va rămâne pentru mult 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 adresare uniform (contiguu). Au fost introduse noţiunile de modularitate şi structurare ierarhică, obiect, etc.
Astăzi sistemele de operare în sensul tradiţional de concepere sunt obiectul unor elaborări industriale, decât de cercetare. Chiar şi lucrările care au avut loc sau mai derulează în SUA capătă tot mai multe trăsături de lucrări semiindustriale. Aceasta este datorat atât de achiziţionării unei rezerve extraordinare de metode şi algoritmi, cât şi standardizării stricte a funcţiilor şi interfeţelor sistemelor de operare. Au totuşi loc unele activităţi de cercetare, ca de exemplu cercetările din domeniul mediilor operaţionale obiect-orientate.

SO UNIX şi standardele sistemelor 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 sistemului de operare UNIX a permis trecerea de la declararea sistemelor deschise

la dezvoltarea 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 (deşi, 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 întreţinerea acestui produs program complicat, unele dificultăţi juridice, etc.), compania AT&T a creat compania USL (UNIX System Laboratories), care avea în şarjă exclusivă dezvoltarea şi susţinerea modulelor sursă ale 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 sistemelor de operare UNIX a fost realizat mecanismul proceselor “legere” (thread, fire), care erau executate în baza unei memorii virtuale comune şi permitea utilizarea posibilităţilor dispozitivelor fizice, numite “arhitectura multiprocesorală simetrică” în care mai multe procesoare au drepturi egale de accesare a memoriei centrale comune.
On 1993 compania USL a fost absorbită de compania Novell, 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 sistem de operare 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 modificaţii şi extensii în această versiune. Ca exemplu, Sun a implementat paralelizarea programelor pentru sistemele multiprocesorale simetrice. Solaris este un mediu de interfaţare 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 total compatibile cu toate standardele de bază

OSF-1 şi alte variante UNIX

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 în sens de licenţiere „curat”, 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 iniţiale şi folosită chiar în fosta URSS pe calculatoarele de tip DEC. 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.

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 propuneau modificări şi chiar variante proprii, a apărut necesitate elaborării unor standarde, care ar conduce la sisteme compatibile. Prin standard al unei interfeţe al SO subînţelegem un set de proprietăţi, mai mult sau mai puţin formale, sintactice sau semantice ale componentelor sistemului de operare. Activitatea de standardizare a început cu mai mult

de 10 ani în urmă, dar este puţin probabil să se ajungă la finiş într-un viitor apropiat. Totuşi rezultatele sunt mai mult decât vizibile, permiţând elaborarea unor sisteme de operare mobile.

Standarde UNIX

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 standard oficial.

Paralel exista direcţia BSD (Berkeley Standard Distribution), susţinută de comunitatea universitară. Deşi majoritatea realizărilor comerciale se baza 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 profesionali 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 resurselor, 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 necesare într-o realizare standard. Deoarece chiar de la apariţie limbajul C şi sistemele de programare respective erau strвns 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.

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 era declarat, 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.

Baze de date

Sistemele informaţionale sunt orientate, în linii mari, spre păstrarea, accesarea şi modificarea informaţiilor existente. Structura informaţiilor este adesea foarte complicată, şi, deşi structurile datelor diferă de la sistem la sistem, ele au foarte multe momente comune. La etapa iniţială de folosire a tenicii de calcul în gestiunea informaţiei problemele structurării datelor erau rezolvate în mod individual pentru fiecare sistem informaţional. Erau elaborate aplicaţii speciale pentru sistemele de gestiune a fişierelor (biblioteci de programe), la fel cum se procedează în cazul compilatoarelor, editoarelor, etc. Dar, fiindcă sistemele informaţionale necesită structuri de date complicate, aceste resurse suplimentare de gestiune a datelor formau partea principală a sistemelor, repetându-se de la un sistem la altul. Tendinţa de evidenţiere şi generalizare a acestei părţi comune a sistemelor informaţionale, responsabile de gestiunea datelor structurate, a fost prima cauză de creare a sistemelor de gestiune a bazelor de date (SGBD). Foarte repede a devenit clară imposibilitatea rezolvării problemei doar cu ajutorul unei biblioteci de programe, care ar realiza metode mai complicate de manipulare a datelor prin intermediul SGF.

Vom exemplifica prin următoarea situaţie. Presupunem că dorim să realizăm un sistem informaţional simplu, care ar duce evidenţa colaboratorilor unei organizaţii. Sistemul va îndeplini următoarele funcţii:

- să formeze lista colaboratorilor unei secţii,
- să permită transferul unui colaborator dintr-o secţie în alta,
- să asigure primirea la lucru a colaboratorilor şi eliberarea acestora.
- Pentru fiecare secţie trebuie susţinută posibilitatea aflării numelui şefului acestei secţii, numărului total de colaboratori din secţie, valoarea ultimului salariu total, etc. Pentru fiecare colaborator trebuie să existe posibilitatea aflării numărului legitimaţiei de serviciu folosind numele şi prenumele colaboratorului şi invers, să se obţină informaţii despre faptul dacă colaboratorul dat corespunde postului ocupat, nivelul salariului, etc.

Presupunem că am hotărât să bazăm acest sistem informaţional pe Sistemul de Gestiune a Fişierelor (SGF) şi să utilzăm pentru aceasta un singur fişier, extinzând posibilităţile de bază ale SGF, creând o bibliotecă specială de funcţii. Unitatea informaţională minimală în acest caz este *colaboratorul* din care cauză vom cere ca în acest fişier să existe o înregistrare pentru fiecare coalborator. Care vor fi câmpurile acestei înscrieri? Numele complet al colaboratorului (COL_NUME), numărul legitimaţiei (COL_NR_LEGIT), informaţia depsre corepunderea postului ocupat ((COL_STARE), pentru simplitate, “da” sau “nu”), mărimea salariului (COL_SAL), numărul secţiei (COL_NR_SEC). Deoarece dorim să avem un singur fişier, aceeaşi înregistrare trebuie să conţină şi numele şefului secţiei (COL_SEF_SEC).

Reieşind din funcţiile sistemului nostru informaţional, trebuie să fie asigurat accesul multicriterial la acest fişier utilizând cheile unice (care nu sunt repetate în diferite înregistrări) COL_NUME şi COL_NR_LEGIT. În afară de aceasta mai este necesar să putem selecta toate înregistrările, care conţin aceeaşi valoare COL_NR_SEC (colaboratorii unei secţii), adică să existe acces conform unei chei multiple. Mai mult, pentru a obţine numărul de colaboratori dintr-o secţie sau salariul total, sistemul informaţional va trebui de fiecare dată să acceseze fiecare colaborator al secţiei şi să calculeze valorile respective.

Observăm, că chiar penru un astfel de sistem simplu, realizarea în baza SGF cere (1) - crearea unei infrastructuri complicate pentru accesul multicriterial, şi (2) - generează o metodă de păstrare evident redundantă (repetarea numelui şefului secţiei pentru fiecare colaborator) şi executarea selectării şi calculelor masive pentru obţinerea informaţiilor sumare despre secţii. În afară de aceasta, dacă dorim, de exemplu, să tipărim lista colaboratorilor, care au un salariu dat, va trebui să cercetăm tot fişierul sau să-l restructurăm în aşa mod încât câmpul COL_SAL să devină cheie.

Prima propunere ar fi să creăm două fişiere cu chei multiple: COLABORATORI şi SECŢII. Primul fişier va conţine câmpurile COL_NUME, COL_NR_LEGIT, COL_STARE, COL_SAL şi COL_NR_SEC, iar al doilea – SEC_NR, SEC_SEF, SEC_SAL (slariul total) şi SEC_NR_COL (numărul de colaboratori dintr-o secţie). Majoritatea inconvenienţelor de mai sus vor fi depăşite. Fiecare

fişier va conţine doar informaţii nedublate, nu sunt necesare calcule suplimentare pentru a afla informaţii de tipul salariului total. Dar în acest caz sistemul nostru capătă proprietăţi noi, care îl apropie de un Sistem de Gestiune a Bazelor de Date (SGBD).

Mai întâi de toate, sistemul trebuie în acest caz să ştie că lucrează cu două fişiere legate (este primul pas în direcţia schemei bazelor de date), să cunoască structura şi sensul fiecărui câmp (de exemplu, că COL_NR_SEC în fişierul COLABORATORI şi SEC_NR în fişierul SECŢII înseamnă acelaşi lucru), şi să înţeleagă că în unele cazuri modificarea informaţiei într-un fişier trebuie să genereze modificarea automată în celălalt fişier, pentru ca să fie ***coordonat*** (nu sunt sigur că este corect spus româneşte, am făcut facultatea în limba rusă) conţinutul lor. De exemplu, dacă la lucru este primit un nou colaborator, se va adăuga o înregistrare în fişierul COLABORATORI, vor fi schimbate şi câmpurile SEC_SAL şi SEC_NR_COL în înregistrarea din fişierul SECŢII, care descrie secţia colaboratorului dat.

Noţiunea de ***coordonare***(?) ***a datelor*** este o noţiune cheie a bazelor de date. De fapt, dacă un sistem informaţional (chiar la fel de simplu ca şi în exemplul nostru) susţine modul coordonat de păstrare a informaţiilor în câteva fişiere, putem afirma că sistemul susţine o bază de date. Iar dacă un sistem auxiliar de gestiune a datelor permite lucrul cu mai multe fişiere, asigurând ***coordonarea*** lor, acesta poate fi numit sistem de gestiune a bazelor de date. Deja singură cererea de susţinere a coordonării datelor în câteva fişiere nu ne mai permite să ieşim din situaţie doar cu ajutoul unei biblioteci de funcţii: un atare sistem trebuie să posede şi unele date proprii (metadate) şi chiar valori, care definesc integritatea datelor.

Dar aceasta încă nu este totul ce este cerut de la un SGBD. În primul rând, chiar şi în exemplul nostru este incomodă îndeplinirea unor interpelări de tipul “să se pună la dispoziţie numărul de colaboratori din secţia în care lucrează Maria Mirabela Popescu”. Ar fi mult mai simplu dacă SGBD ar permite formularea unor astfel de interpelări într-un limbaj apropiat de cel natural. Astfel de limbaje se numesc *limbaje de interpelare a bazelor de date*. De exemplu, în limbajul *SQL* interpelarea noastră poate fi exprimată astfel:

```
SELECT SEC_NR_COL
FROM COLABORATORI, SECŢII
WHERE COL_NUME = "Maria Mirabela Popescu"
AND COL_NR_SEC = SEC_NR
```

- Formularea unei interpelări în SQL permite să nu ne deranjeze modul cum va fi îndeplinită această interpelare. Printre metadatele acesteia va fi prezentă informaţia conform căreia câmpul COL_NUME este câmp cheie pentru fişierul COLABORATORI, iar SEC_NR – pentru fişierul SECŢII şi sistemul se va folosi de aceasta.

Dacă va fi necesar să obţinem lista colaboratorilor, care nu corespund postului ocupat vom proceda astfel:

```
SELECT COL_NUME, COL_NR_LEGIT
FROM COLABORATORI
WHERE COL_STARE = "nu",
```

şi sistemul va îndeplini căutarea necesară în fişierul COLABORATORI, deoarece câmpul COL_STARE nu este câmp cheie.

Acum să ne închipuim, că în prima realizare a sistemului informaţional, bazată pe utilizarea unor biblioteci de extensii a metodelor de accesare a fişierelor, are loc operaţia de înregistrare a unui nou colaborator. Conform proprietăţii de modificare coordonată, sistemul a introdus o nouă înregistrare în fiţirerul COLABORATORI şi trebuia să treacă la modificarea înregistrării fişierului SECŢII, dar anume în acest moment a avut loc decontectarea accidentală a alimentării cu curent electric. Evident, după

relansarea sistemului baza de date se va afla într-o stare necoordonată. Va fi necesar să controlăm în mod explicit fişierele noastre şi să introducem modificările necesare. O aplicaţie poate su nu-şi facă griji în acest sens, dar SGBD-urile adevărate au în şarjă astfel de funcţii. Mai mult, sistemele de gestiune a bazelor de date rezolvă o serie de probleme, care nu pot fi în principiu rezolvate cu ajutorul sistemelor de gestiune a fişierelor (accesarea paralelă a informaţiilor, de ex.).

Ca şi concluzie: există aplicaţii pentru care sunt suficiente fişierele, aplicaţii pentru care trebuie să hotărâm care este nivelul necesar de lucru cu datele în memoria externă, şi aplicaţii care au evident nevoie de baze de date.

2. Funcţiile unui SGBD. Organizarea-tip a SGBD

După cum am observat, posibilităţile tradiţionale ale sistemelor de gestiune a fişierelor sunt insuficiente pentru construirea unor sisteme informaţionale, chiar foarte simple. Aceasta este cauza principală a apariţiei SGBD.

2.1. Funcţiile principale ale unui SGBD

2.1.1. Administrarea explicită (directă) a datelor în memoria externă

Această funcţie presupune susţinerea structurilor necesare în memoria externă atât pentru păstrarea datelor care sunt parte componentă a BD, cât şi pentru scopuri de service, de exemplu, pentru accelerarea accesului la date (de obicei sunt utilizaţi idicii în acest scop). În unele realizări ale SGBD pentru aceasta sunt utilizate posibilităţile SGF, altele pot lucra chiar şi cu dispoziltivele de memorie externă. Utilzatorii nu sunt obligaţi să cunoască dacă un SGBD utilizează SGF şi, în caz afirmativ, cum sunt organizate fişierele. SGBD susţine un sistem propriu de desemnare a obiectelor BD.

2.1.2. Administrarea tampoanelor (buferelor) din memoria operativă

SGBD lucrează cu BD de dimensiuni semnificative, de obicei cel puţin aceste dimensiuni sunt mult mai mari decât volumul memoriei operative prezente. Evident, dacă la accesarea fiecărui element de date va fi nevoie de schimb de informaţii cu memoria externă, tot sistemul va lucra cu viteza suportului extern de memorie. Unica posibilitate de sporire a acestei viteze este buferizarea datelor în memoria operativă. Chiar dacă SO execută buferizarea de sistem (cum este în cazul SO UNIX), aceasta nu este suficient pentru scopurile SGBD. Din această cauză SGBD concurente susţin un set propriu de bufere în memoria operativă cu propriul algoritm de dispecerizare.

Notăm, că există o direcţie întreagă în domeniul SGBD, dedicată prezenţei constante a întregii BD în memoria operativă (cazul sistemelor de căutare, tranzacţionale în Internet, de exemplu). Aici presupunem că capacitatea memoriei operative poate fi suficient de mare şi nu trebuie să ne facem probleme cu buferizarea.

2.1.3. Administrarea tranzacţiilor

Tranzacţiile sunt o succesiune de operaţii asupra BD, considerate de SGBD un tot întreg. Dacă tranzacţia este executată în totalitate SGBD fixează (COMMIT) modificările în BD, în caz contrar nu are loc nici o modificare. Noţiunea de ***tranzacţie*** este necesară pentru asigurarea integrităţii logice a BD. În exemplul precedent atunci când este îndeplinită operaţia de primire la lucru a unui colaborator nou, această operaţie este uniunea operaţiilor elementare asupra fişierelor COLABORATORI şi SECŢII într-o tranzacţie. Noţiunea dată este şi mai importantă pentru SGBD multiuser. Proprietatea că fiecare tranzacţie începe atunci când starea BD este consistentă şi lasă această stare la fel consistentă după terminarea sa, face folosirea noţiunii *tranzacţie* on calitate de unitate de activitate a user-ului. Dacă tranzacţiile sunt gestionate corect de către SGBD, fiecare utilizator are impresia că este singurul care foloseşte baza de date (dacă nu-s prea mulţi, evident!).

Străns legate de administrarea tranzacţiilor în SGBD multiuser sunt noţiunile de *serializare a tranzacţiilor* şi *planul secvenţial (serializat) de îndeplinire a tranzacţiilor*. Prin serializarea unor tranzacţii care ar trebui să fie îndeplinite în mod paralel înţelegem o astfel de planificare a executării lor, în care efectul final este acelaşi, deşi tranzacţiile nu sunt îndeplinite paralel, ci secvenţial (din cauza resurselor limitate, de exemplu).

2.1.4. Jurnalizarea

Una din proprietăţile de bază ale SGBD este fiabilitatea păstrării datelor în memoria externă, prin aceasta înţelegându-se că, dacă a avut loc orice cădere hardware sau software, SGBD-ul trebuie să fie în stare să restabilească ultima stare coordonată a BD. Evident, pentru a restabili BD trebuie să existe nişte informaţii suplimentare. Altfel spus, susţinerea fiabilităţii păstrării datelor în BD necesită o redundanţă a informaţiilor, iar aceea parte a informaţiilor, care este folosită pentru restabilire trebuie păstrată extrem (straşnic!☺) de fiabil. Cea mai răspândită metodă de susţinere a unei astfel de redundanţe este întreţinerea unui *jurnal de bord* în care să fie fixate schimbările BD. Jurnalul este o parte specială a BD, inaccesibilă utilizatorilor BD şi întreţinută în mod foarte pedant (adesea există două c**o**pii ale jurnalului, care se află pe discuri fizice diferite), în care vin informaţiile despre toate înscrierile în BD (partea principală, fără Jurnal, adică). În dependenţă de SGBD şi înscrierile în jurnal se fac în mod diferit. În unele cazuri înscrierea în jurnal corespunde unei operaţii logice oarecare (eliminarea unei linii, de ex.), în alte cazuri unei operaţii interne minime de modificare a unei pagini de memorie externă, sau combinaţia acestora.

On toate cazurlie este utilizată strategia înscrierii “profilactice” în jurnal (protocolul Write Ahead Log - WAL). În linii mari asta ar însemna, că înscrierea despre modificarea oricărui obiect al BD trebuie să nimerească în memoria externă a jurnalului înainte ca obiectul modificat să ajungă în memoria externă a părţii principale a BD. Dacă SGBD-ul respectă protocolul WAL, jurnalul permite rezolvarea tuturor problemelor de restabilire a BD după oricare tip de cădere.

2.1.5. Susţinerea limbajelor BD

Pentru lucrul cu bazele de date sunt utilizate limbaje speciale, numite *limbaje ale bazelor de date*. On SGBD-urile timpurii erau susţinute câteva limbaje specializate funcţional. Cel mai frecvent importante erau două limbaje – *limbajul de definire a schemei BD (SDL - Schema Definition Language)* şi *limbajul de manipulare a datelor (DML - Data Manipulation Language).* Primul era utilizat pentru determinarea structurii logice a BD (cum îşi închipuie utilizatorul baza de date). *DML* conţinea instrucţiuni de manipulare a datelor (introducerea datelor în BD, eliminarea, modificarea sau selecatarea datelor existente). În SGBD contemporane este susţinut un limbaj integrat unic. Cel mai răspândit este limajul SQL (Structured Query Language). Acest limbaj combină posibiltăţile SDL şi DML. Desemnarea obiectelor BD (desemnarea tabelelor şi coloanelor lor, în cazul unor BD de tip relaţional) este susţinută la nivelul limbajului în sens că compilatorul SQL transformă numele externe ale obiectelor în identificatori interni în baza unor tabele-catalog de serviciu. Nucleul SGBD în genere nu are “treabă” cu numele tabelelor şi coloanelor acestora. Limbajul SQL conţine resurse speciale pentru determinarea restricţiilor de consistenţă a BD. Aceste restricţii sunt păstrate în tabele-catalog speciale, iar garantarea consistenţei are loc la nivel de limbaj, adică la compilarea instrucţiunilor de modificare a BD compilatorul SQL în baza restricţiilor de consistenţă generează codul respectiv.

Autorizarea accesului la BD la fel se realizează cu ajutorul unui set de instrucţiuni SQL, fiind posibil accesul diferenţiat în dependenţă de statutul utilizatorului.

2.2. Organizarea tipică a unui SGBD concurent

Organizarea unui SGBD tip şi setul de componente ale acestuia corespunde funcţiillor stabilite mai sus. Noi am stabilit următoarele funcţii de bază:

- administrarea datelor în memoria externă,
- administrarea buferelor în memoria externă,
- adminstrarea tranzacţiilor,
- jurnalizarea şi restabilirea BD după căderi,
- susţinerea limbajelor BD.

La nivel logic on cadrul unui SGBD concurent putem evidenţia nivelul cel mai intern – nucleul SGBD (numit adesea Data Base Engine), compilatorul SGBD (de obicei SQL), subsistemul Run-Time şi un set de utilitare.

- Nucleul SGBD este responsabil de administrarea datelor în memoria externă, gestiunea buferelor memoriei operatve, adminstrarea tranzacţiilor şi jurnalizare. Respectiv, pot fi evidenţiate astfel de componente ale nucleului cum ar fi managerul datelor, managerul buferelor, managerul tranzacţiilor şi managerul jurnalului. Funcţiile acestor componente sunt interdependente din care cauză ele trebuie să funcţioneze şi să interacţioneze conform unor protocoale bine gândite şi strict validate. Nucleul SGBD posedă o interfaţă proprie, dar care este utilizată doar de programele SQL şi utilitarele BD. Nucleul SGBD este partea rezidentă principală a sistemului de gestiune a BD. În arhitectura "client-server" nucleul este componenta principală a părţii server a sistemului.
- Destinaţia principală a compilatorului este translatarea instrucţiunilor scrise în limbajul BD în cod executabil. Problema principală constă în faptul că aceste limbaje (de obicei SQL) sunt neprocedurale, adică în operatorul unui atare limbaj sunt specificate nişte acţiuni asupra BD, dar această specificare nu este procedură, ci doar descrie într-o formă oarecare condiţiile de îndeplinire a acţiunii dorite. Compilatorul trebuie să hotărască cum să execute operatorul limbajului înaintea generării codului executabil. În rezultatul compilării se obţine un program executabil, reprezentat de obicei în cod intern independent de maşină. În acest caz executarea reală a operatorului are loc utilizând un subsistem de susţinerea a regimului run-time, care nu este altceva decât un interpretor al acestui limbaj intern.

Din categoria utilitarelor BD fac parte unele proceduri, îndeplinirea cărora nu este binevenită utilizând limbajul BD. Dintre acestea aminitim încărcarea şi unload-ul unei BD, culegere date statistice, controlul global al consistenţei BD, etc.

3. Despre calculul relaţional

Presupunem că lucrăm cu BD care are schema COLABORATORI(NR_COL, NUME_COL, SAL_COL, NR_SEC_COL) (aici NR_COL este numărul legitimaţiei colaboratorului, NUME_COL – numele colaboratorului, SAL_COL - salariul, iar NR_SEC_COL – numărul secţiei în care colaboratorul concret lucrează) şi SECŢII(NUMĂRUL_SEC, NR_COL_IN_SECŢIE, SEF_SECŢIE). Dorim să aflăm numele şi numerele colaboratorilor, care sunt şefii secţiilor cu numărul total de colaboratori mai mare ca 50. În sensul algebrei relaţionale am fi obţinut expresia, care ar fi citită aproximativ astfel:

- Să se unească relaţiile COLABORATORI şi SECŢII cu condiţia NR_COL= SEF_SECŢIE;
- Relaţia obţinută să fie restricţionată cu condiţia NR_COL_IN_SECŢIE > 50;
- Rezultatul să fie proiectat pe atributul NR_COL, NUME_COL.

Fiecare pas corespunde unei operaţii relaţionale. Dacă vom formula aceeaşi interpelare cu ajutorul calculului relaţional (ceea ce va urma în continuare), vom obţine o formulă, care ar putea fi citită astfel: *să se determine NR_COL, NUME_COL pentru colaboratorii pentru care există secţie cu aceeaşi valoare SEF_SECŢIE (ca şi NUME_COL (sau NR_COL)) şi cu valoarea NR_COL_IN_SECŢIE mai mare decât 50.*

Aici am arătat numai caracteristicile relaţiei rezultante, nu am spus nimic despre modalitatea de formare a acesteea. Sistemul va trebui singur să decidă, care operaţii şi în care ordine să fie îndeplinite peste relaţiile COLABORATORI şi SECŢII. Spunem că formularea algebrică este procedurală, adică defineşte regulile de îndeplinire a interpelării, iar formularea logică (a doua) este descriptivă sau

declarativă, deoarece ea doar descrie proprietăţile rezultatului dorit. Aceste două mecanisme sunt echivalente şi există reguli, relativ simple, care permit transformarea unui formalism în celălalt.

3.1. Relaţii şi proprietăţile lor

Se numeşte **relaţie** *n*-ară pe *M* submulţimea $R \subseteq M^n$. Vom zice că $a_1,...,\ a_n$, sunt în relaţia *R* dacă $(a_1,...,a_n) \in R$. O relaţie unară este o parte a mulţimii *M* şi determină o proprietate a elementelor unei submulţimi a mulţimii *M* din care cauză pentru *n = 1* denumirea de relaţie practic nu se utilizează. Un interes mai mare prezintă cazul când *n = 2* - relaţiile binare. Dacă *a* şi *b* se află în relaţia *R*, aceasta se va scrie *aRb*.

Exemplul 3.1. Pentru mulţimea *N*: relaţia "<" are loc pentru perechea *(3,9)* şi nu are loc pentru *(6,4)*. Relaţia *"a fi divizor"* are loc pentru perechea *(7,35)* şi nu are loc pentru *(18,2)* sau *(4,9)*. Pentru o mulţime de oameni pot fi relaţii de tipul *"a fi prieteni", "a locui in acelaşi oraş", "a fi fiu"*, etc. <

Restricţia lui *R* pe $M_1 \subseteq M$ este $R^1 = R \cap M_1^2$.

Pentru definirea unei relaţii pot fi utilizate oricare din metodele de definire a mulţimilor. O posibilitate suplimentară este matricea de relaţie. Pentru mulţimea $M = \{a_1, a_2,..., a_m\}$ aceasta este o matrice *mxm* on care

$$a(i,j) = \begin{cases} 1 \text{ dacă } a_i R a_j, \\ 0 \text{ on caz contrar.} \end{cases}$$

Deoarece relaţia este în ultimă instanţă o mulţime, pot fi executate aceleaşi operaţii (reuniune, intersecţie etc.) şi cu relaţiile.

O relaţie se numeşte inversa relaţiei *R* (se va nota R^{-1}) dacă $a_i R a_j$ are loc atunci şi numai atunci cвnd are loc $a_j R^{-1} a_i$.

Relaţia poate poseda o serie de proprietăţi dintre care vom menţiona **reflexivitatea, simetria** şi **tranzitivitatea**. Dacă pentru $\forall a \in M$ are loc *aRa* relaţia *R* se numeşte **reflexivă**. Diagonala principală a matricei relaţiei *R* conţine numai unităţi. Relaţia *R* se numeşte **antireflexivă** dacă nu există $a \in M$ pentru care ar avea loc *aRa*. Diagonala principală a matricei unei astfel de relaţii conţine numai zerouri. Relaţiile *"≥", "a avea un divizor comun"* sunt reflexive. Relaţiile *"a fi fiu", ">"* - sunt antireflexive.

Dacă pentru o pereche $(a,b) \in M^2$ din *aRb* rezultă *bRa* (relaţia are loc în ambele părţi sau nu are loc de fel), relaţia *R* se numeşte **simetrică**. Pentru astfel de relaţii *c(i,j) = c(j,i)*: matricea este simetrică faţă de diagonala principală. Relaţia se va numi **antisimetrică**[*]**,** dacă din $a_i R a_j$ şi $a_j R a_i$ rezultă că $a_i = a_j$. Relaţia "≤" este antisimetrică, iar *"a locui în acelaşi oraş"* - simetrică.

Relaţia *R* se numeşte **tranzitivă** dacă pentru oricare a, *b* şi *c* din *aRb* şi *bRc* rezultă *aRc*. Relaţiile *"a locui în acelaşi oraş", "egal", "<"* sunt tranzitive, iar *"a fi fiu"* nu este tranzitivă.

Pentru oricare relaţie *R* poate fi definită noţiunea de închidere tranzitivă R^*: aR^*b (*a* se află în relaţia R^* cu *b*), dacă în *M* există o secvenţă de *n* elemente $a=a_1,...,\ a_{n-1},\ a_n = b$ on care pentru elementele vecine are loc *R:* $aRa_2,\ a_2Ra_3,...,\ a_{n-1}Rb$. Dacă *R* este tranzitivă, atunci $R^*=R$. Pentru relaţia *"a fi fiu"* relaţia *"a fi descendent direct"* este închidere tranzitivă (este reuniunea relaţiilor *"a fi fiu", "a fi nepot", "a fi strănepot"* s.a.m.d.).

O relaţie care posedă proprietăţile reflexivitate, simetrie şi tranzitivitate se numeşte relaţie de ***echivalenţă****.*

Se numeşte **relaţie de ordine** oricare relaţie care posedă proprietăţile reflexivitate, antisimetrie şi tranzitivitate. O relaţie antireflexivă, antisimetrică şi tranzitivă se numeşte **relaţie de ordine strictă**.

[*] Termenul *antisimetric* nu a fost ales prea bine, deoarece se poate crede că o relaţie, care nu este simetrică este totdeauna antisimetrică, ceea ce nu este corect.

Două elemente *a* şi *b* se numesc **comparabile** conform relaţiei de ordine *R* dacă are loc *aRb* sau *bRa*. O mulţime *M* cu o relaţie de ordine definită pe *M* se numeşte **total ordonată** dacă oricare două elemente din *M* sunt comparabile şi **parţial ordonată,** on caz contrar.

Exemplul 3.2. Relaţiile *"≤"*, *"≥"* pentru o mulţime de numere sunt relaţii de ordine, iar *"<"*, *">"* - de ordine strictă. Relaţia de subordonare în cadrul unei întreprinderi defineşte o ordine strictă (dar parţială - nu pot fi comparaţi colaboratorii diferitor departamente).

În alfabetul latin literele sunt aranjate într-o ordine binecunoscută: se află în relaţia de precedare a literelor. Conform acestei relaţii poate fi stabilită relaţia de precedare a cuvintelor - ordinea lexicografică a cuvintelor (utilizată de exemplu în dicţionare). Relaţia de ordine lexicografică poate fi definită şi pentru informaţii numerice. De exemplu, în calculator data şi anul sunt memorizate sub forma *"anul, luna, ziua"* pentru ca ordinea de creştere a datei totale să coincidă cu ordinea lexicografică. <

Exemplul 3.3. B^* este mulţimea tuturor cuvintelor binare de orice lungime şi ε desemnează unicul cuvânt binar de lungime *0*. Dacă *m* este un cuvвnt binar de lungime *p*, vom nota biţii acestuia $m_1, m_2,..., m_p$. Relaţia de ordine $\mathfrak{R}$, definită peste B^*, se numeşte de ordine lexicografică, dacă:

a) $\varepsilon \mathfrak{R} m, \forall m$

b) pentru *m* de lungime *p* şi *n* de lungime *q* cu proprietatea $0 < p \le q$

$m \mathfrak{R} n$, dacă $m_1=n_1, m_2=n_2,..., m_p=n_p$,

$m \mathfrak{R} n$, dacă $m_1=n_1, m_2=n_2,..., m_{s-1}=n_{si1}, \quad m_s < n_s$, pentru $1 \le s \le p$,

$n \mathfrak{R} m$, dacă $m_1=n_1, m_2=n_2,..., m_{s-1}=n_{si1}, \quad m_s > n_s$, pentru $1 \le s \le p$. <

3.2. Operaţii şi algebre. Proprietăţile operaţiilor

Funcţia de tipul $\varphi: M^n \to M$ se va numi **operaţie** *n*-ară pe *M*. Setul $A = <M, \Omega>$, on care Ω este o mulţime de operaţii definite pe *M*, se numeşte **algebră**. Mulţimea *M* se va numi mulţime de bază sau suportul, iar $\Omega = \{\varphi_1, \varphi_2,..., \varphi_m,...\}$ - signatura algebrei *A*. Vectorul, componentele căruia sunt arităţile operaţiilor $\varphi_1, \varphi_2,...$ se numeşte tipul algebrei *A*.

Operaţia φ se numeşte:

a) comutativă, dacă $a\varphi b = b\varphi a$,
b) asociativă, dacă pentru oricare *a, b, c* are loc $(a\varphi b)\varphi c = a\varphi(b\varphi c)$,
c) idempotentă, dacă $a\varphi a = a$,
d) distributivă stânga faţă de operaţia *g*, dacă pentru oricare *a, b, c* are loc relaţia
e) $a\varphi(bgc) = (a\varphi b)g(a\varphi c)$, şi distributivă dreapta dacă $(agb)\varphi c = (a\varphi c)g(b\varphi c)$.

Dacă există un element *e* pentru care are loc $a\varphi e = e\varphi a = a$, atunci acest element se numeşte neutru (sau unitate).

Exemplul 3.4. a) Pentru o mulţime arbitrară *U* şi mulţimea tuturor părţilor $\boldsymbol{B}(U)$, algebra $A = \{\boldsymbol{B}(U), \cup, \cap, \overline{\ }\}$ se numeşte algebră booleana a mulţimilor. Tipul ei este *(2,2,1)*.

b) Algebra $A = \{R,+,x\}$ se numeşte câmp al numerelor reale. Ambele operaţii sunt binare, deci tipul este *(2,2)*.<

Algebrele $L = \{M,\cup,\cap\}$ (cu două operaţii binare - reuniunea şi intersecţia) se numesc ***latice***, dacă au loc axiomele:

P1: $a\cup b = b\cup a$, $a\cap b = b\cap a$ - **comutativitate**,
P2: $a\cup(b\cup c) = (a\cup b)\cup c$, $a\cap(b\cap c) = (a\cap b)\cap c$ - **asociativitate**,
P3: $a\cup(b\cap a) = a$, $a\cap(b\cup a) = a$ - **absorbţie**,

pentru oricare $a, b, c \in M$.

3.3. Modele si sisteme algebrice. Algebra relaţiilor

Noţiunea de **model** este una din noţiunile de bază în matematica discretă. Se va numi **model** *M* setul care constă din mulţimea *D* - suportul modelului, şi o mulţime de relaţii *S* definite pe *D:*

$$M = <D, S>.$$

Aici $S = \{R_{11}, R_{12},..., R_{1n1}, R_{21}, R_{22},..., R_{2n2},..., R_{m1}, R_{m2},..., R_{mnm}\}$ este **signatura** modelului, $R_{ij} \in M^i$. Exponenta suportului determină aritatea relaţiei. Două relaţii R_i şi R_j care au aceeaşi aritate se numesc **compatibile.**

Setul care conţine mulţimea *D*, operaţiile şi relaţiile definite pe D

$$A = <D, F, S>.$$

se numeşte **sistem algebric**.

Modelul este un caz particular al sistemului algebric, când mulţimea *F* este vidă, iar pentru o algebră mulţimea *S* este vidă.

Un alt caz particular al sistemelor algebrice ol constituie **algebra relaţiilor** şi extensia acesteea - **algebra relaţională**. Pentru o algebră a relaţiilor drept suport serveşte mulţimea relaţiilor considerate, iar signatura o formează operaţiile de reuniune, intersecţie, diferenţă şi produsul cartezian extins al relaţiilor. Să facem cunoştinţă cu aceste operaţii.

Reuniunea $R_i \cup R_j$ a două relaţii compatibile R_i şi R_j este mulţimea tuturor cortejurilor, fiecare dintre care aparţine cel puţin uneia din relaţii.

Intersecţia $R_i \cap R_j$ a două relaţii compatibile R_i *şi* R_j se va numi mulţimea tuturor cortejurilor care aparţin ambelor relaţii în acelaşi timp.

Diferenţa $R_i \backslash R_j$ a două relaţii compatibile R_i şi R_j se numeşte mulţimea tuturor cortejurilor care aparţin lui R_i şi nu aparţin relaţiei R_j .

Produs cartezian extins $R_i x R_j$ a două relaţii R_i şi R_j se va numi mulţimea tuturor cortejurilor formate prin concatenarea lui $a \in R_i$ şi a lui $b \in R_j$.

Exemplu: dacă $R_i = \{(a,b),(a,c),(a,e)\}$, iar $R_j = ((a,b,c),(c,d,e)\}$, atunci

$R_i x R_j = \{(a,b,a,b,c), (a,b,c,d,e), (a,c,a,b,c), (a,c,c,d,e), (a,e,a,b,c), (a,e,c,d,e)\}$.

Algebra relaţiilor şi modelele sunt utilizate pentru formalizarea unor obiecte reale. Vom exemplifica prin folosirea algebrei relaţiilor în cazul bazelor relaţionale de date.

O **bază de date** de tip relaţional este un tablou bidimensional în care coloanele determină aşa numitele *domene* (atribute), iar liniile sunt cortejuri de valori concrete ale atributelor, care se află în relaţia *R*.

Exemplul 3.5. Relaţia R_5 - "examene" (v.tab. 3.1). Relaţia R_5 este o submulţime a produsului cartezian $D_1 x D_2 x D_3 x D_4 x D_5$. Elemente ale domenului D_i sunt valorile atributelor:

$D_1 = \{3\text{-}101, 3\text{-}501, 3\text{-}502, 3\text{-}310\}$ - numerele auditoriilor unde au loc examenele;

D_2 = *{Matematica discretă în inginerie, Microelectronica, Fizica, Circuite integrate, Electrotehnica}* - denumirea disciplinelor;

$D_3 = \{conf.V.Beşliu, conf.\ V.Negură, conf.A.Diligul, conf.V.Şontea, prof.I.Samusi\}$ -examinatorii;

$D_4 = \{3$ *iunie, 4 iunie, 8 iunie, 13 iunie} - data examenului;*

$D_5 = \{TI\text{-}961, TI\text{-}962, TI\text{-}963, C\text{-}941, C\text{-}942, C\text{-}951, C\text{-}952\}$ - denumirea grupei.

Tabelul 3.1. Relaţia “examene”.

R_5	D_1	D_2	D_3	D_4	D_5
1	3-101	Matematica discretă în inginerie	conf. V.Beşliu	3 iunie	TI-961
2	3-202	Microelectronica	conf. V.Şontea	4 iunie	C-951
3	3-310	Fizica	prof. I.Samusi	3 iunie	TI-962
4	3-101	Circuite integrate	conf. V.Negură	4 iunie	C-941
5	3-104	Electrotehnica	conf.	3 iunie	TI-951

R_5	D_1	D_2	D_3	D_4	D_5
			A.Diligul		
6	3-101	Matematica discretă în inginerie	conf. V.Beşliu	8 iunie	TI-962
7	3-101	Matematica discretă în inginerie	conf. V.Beşliu	13 iunie	TI-963
8	3-202	Microelectronica	conf. V.Şontea	8iunie	C-952
9	3-310	Fizica	prof. I.Samusi	8iunie	TI-961
10	3-101	Circuite integrate	conf. V.Negură	8iunie	C-942

Numerele *1, 2,..., 10* din prima coloană identifică elemente ale relaţiei R_5. <

Algebra relaţională este o extensie a algebrei relaţiilor în sens că signatura *S* în afară de cele 4 operaţii descrise anterior mai conţine câteva operaţii speciale, de exemplu ***proiecţia, selecţia*** şi ***joncţiunea***.

Operaţia **selecţie** permite evidenţierea unei submulţimi de cortejuri care posedă o proprietate dată. De exemplu, operaţia selecţie permite evidenţierea relaţiei *orarul conf. V.Beşliu* - liniile on care valoarea domenului D_3 este conf. V.Beşliu:

Tabelul 3.2. Rezultatul operaţiei *selecţie* pentru valoarea "conf. V.Beşliu"

R_5	D_1	D_2	D_3	D_4	D_5
1	3-101	Matematica discretă în inginerie	conf. V.Beşliu	3 iunie	TI-961
6	3-101	Matematica discretă în inginerie	conf. V.Beşliu	8 iunie	TI-962
7	3-101	Matematica discretă în inginerie	conf. V.Beşliu	13 iunie	TI-963

Operaţia **proiecţie** se defineşte introducând pentru suportul *D* al algebrei relaţionale o partiţie de *n* submulţimi (*n* este aritatea relaţiei) $R_n \in D^n$.

Proiecţia relaţiei binare $R_2 \in AxB$ pe *A* (PrR_2/A) se numeşte mulţimea $\{a_i \mid (a_i,b_i) \in R_2\}$.

Proiecţia $PrR_n/A_{i1},...,A_{im}$ a relaţiei *n*-are $R_n \in A_1xA_2x...xA_n$, $m \leq n$, *pe* A_{i1}, A_{i2},..., A_{im} se numeşte mulţimea cortejurilor $(a_{i1}, a_{i2},..., a_{im})$, on care $a_{i1} \in A_{i1}$, $a_{i2} \in A_{i2}$,..., $a_{im} \in A_{im}$ şi fiecare cortej este parte a unui element al relaţiei *n*-are R_n. Cu alte cuvinte, operaţia *proiecţie* permite construirea unei submulţimi *verticale* a relaţiei (a unei mulţimi de submulţimi de atribute care se obţine prin alegerea unor domene concrete). De exemplu, $Pr(R_5/D_2,D_3)$ determină denumirea examenelor şi numele examinatorilor (liniile care coincid se scriu o singură dată, v.tab. 3.3).

Tabelul 3.3. Rezultatul operaţiei "*proiecţie*".

D_2	D_3
Matematica discretă în inginerie	conf. V.Beşliu
Microelectronica	conf. V.Şontea
Fizica	prof. I.Samusi
Circuite integrate	conf. V.Negură
Electrotehnica	conf. A.Diligul

Operaţia **joncţiune** (join) a două tabele care au un domen comun permite construirea unui tabel nou în care fiecare linie se va obţine din unirea a două linii din tabelele iniţiale. Aceste linii corespund aceluiaşi atribut din domenul comun. Domenul comun se va scrie o singură dată.

De exemplu, pentru tabele 3.4 şi 3.5 domenul comun este D_5, rezultatul operaţiei de joncţiune este prezentat în tabelul 3.6.

Tabelul 3.4.

D_1	D_2	D_3	D_4	D_5
3-202	Microelectronica	conf. V.Şontea	4 iunie	C-951
3-310	Fizica	prof. I.Samusi	3 iunie	TI-962
3-104	Electrotehnica	conf. A.Diligul	3 iunie	TI-951

Tabelul 3.5.

D_1	D_2	D_3	D_4	D_5
3-104	Electrotehnica	conf. A.Diligul	13 iunie	C-951
3-310	Matematica	conf. L.Dogotaru	13 iunie	TI-962
3-202	Microelectronica	conf. V.Şontea	14 iunie	TI-951

Tabelul 3.6. Rezultatul operaţiei "*join*".

D_1	D_2	D_3	D_4	D_{11}	D_{21}	D_{31}	D_{41}	D_5
3-202	Microelectronica	conf. V.Şontea	4 iunie	3-104	Electrotehnica	conf. A.Diligul	13 iunie	C-951
3-310	Fizica	prof. I.Samusi	3 iunie	3-310	Matematica	conf. L.Dogotaru	13 iunie	TI-962
3-104	Electrotehnica	conf. A.Diligul	3 iunie	3-202	Microelectronica	conf. V.Şontea	14 iunie	TI-951

Operaţia *join* este definită nu numai pentru condiţia de *egalitate* a două domene, ci pot fi şi alte condiţii de comparare, de exemplu, $>$, $\geq$, $<$, $\leq$, etc.

4. Proiectarea bazelor de date relaţionale

Proiectarea bazelor de date rezolvă două probleme:

- Cum să fie reprezentate obiectele domeniului obiectiv în obiecte abstracte ale modelului datelor astfel ca această reprezentare să nu contrazică semantica domeniului obiectiv şi să fie, în măsura posibilităţilor, cea mai bună (eficientă, comodă, etc.)? Adesea această problemă este numită **proiectarea logică a BD**.
- Cum să se asigure executarea eficientă a interpelărilor BD? Altfel, luând în consideraţie specificul unui SGBD concret, cum vor fi plasate datele în memoria externă, care structuri vor fi create suplimentar (de ex., indici), etc. Această problemă este numită **proiectarea fizică a BD**.

Este dificil să fie propuse nişte reţete de ordin general pentru rezolvarea problemei proiectării fizice. Prea multe depind de SGBD folosit. De exemplu, cu ajutorul SGBD Ingres poate fi aleasă una dintre metodele de organizare fizică a relaţiilor, iar lucrând cu System R ar trebui să ne gândim în primul rând la clasterizarea relaţiilor şi la setul necesar de indici, etc. Din această cauză vom cerceta mai departe doar momentele, legate de proiectarea logică a BDR, care au importanţă mare atunci când utilizăm orice

SGBDR. Mai mult, nu vom discuta un aspect foarte important al proiectării – determinarea cererilor de consistenţă (cu excepţia restricţiei cheii primare). Aceasta din cauza, că atunci când este utilizat un SGBD cu mecanisme de asigurare a consistenţei datelor (de ex., sistemele SQL orientate) este foarte dificil să fie propusă o abordare generală de determinare a cererilor de consistenţă. Aceste cereri (restricţii) pot fi de ordin foarte general, iar formularea lor ţine mai repede de artă, decât de măiestria inginerească. În literatură în această direcţie este propus cel mult controlul automat al necontradicţiei setului de restricţii al consistenţei.

Vom considera, deci, că problema proiectarii unei baze de date de tip relaţional constă în luarea unei decizii motivate în vederea obţinerii unor răspunsuri la următoarele întrebări:

- Care sunt relaţiile din care este formată baza de date?
- Care vor fi atributele acestor relaţii?

4.1. Proiectarea logică

Vom face cunoştinţă mai întâi cu abordarea clasică, când întreg procesul de proiectare are loc în termeni din domeniul modelului relaţional de date, folosind metoda determinării consecvente a setului satisfăcător de relaţii. Punctul de plecare este reprezentarea domeniului obiectiv sub forma uneia sau mai multor relaţii şi la fiecare pas al procesului de proiectare se va produce un set oarecare de scheme ale relaţiilor, care posedă proprietăţi mai bune decât la pasul precedent. Procesul de proiectare se transformă într-un proces de normalizare a schemelor relaţiilor, iar fiecare formă normală care urmează posedă proprietăţi mai bune decât forma precedentă.

Fiecărei forme normale îi corespunde un set anumit de cerinţe, iar relaţia se află într-o formă normală, dacă satisface setul de cerinţe caracteristic ei. Ca exemplu de set de cerinţe poate fi considerată restricţia primei forme normale – valorile tuturor atributelor relaţiei sunt atomare. Deoarece restricţia primei forme normale este o cerinţă fundamentală a modelului de date relaţional clasic, vom considera că setul iniţial de relaţii deja corespunde acestei cerinţe. În teoria bazelor de date relaţionale este evidenţiat următorul şir de forme normale:

- Prima formă normală (*1FN*);
- A doua formă normală (*2FN*);
- A treia formă normală (*3FN*);
- Forma normală Boyce-Codde (*BCNF*);
- A patra formă normală (*4FN*);
- A cincea formă normală sau forma normală *proiecţie-reuniune* (*5FN* sau *PJ/NF*).

Proprietăţile principale ale formelor normale:

- Fiecare următoare formă normală este într-un anumit sens mai bună decât precedenta;
- Trecerea la următoarea formă normală păstrează proprietăţile formelor normale precedente.

La baza procesului de proiectare se află metoda normalizării - decompoziţia relaţiei, care se află în forma normală precedentă în două sau mai multe relaţii, care vor respecta cerinţele următoarei forme normale.

În practică, cele mai importante forme normale ale relaţiilor sunt bazate pe noţiunea fundamentală, numită în teoria BDR *dependenţă (relaţie) funcţională.*

Definiţia 1. *Dependenţă funcţională.* În relaţia *R* atributul *y* depinde funcţional de atributul *x* (*x* şi *y* pot fi compuse) dacă şi numai dacă fiecărei valori a lui *x* îi corespunde exact o valoare a lui *y*: *x R y*.

Definiţia 2. *Dependenţă funcţională totală.* Dependenţă funcţională *x R y* se numeşte totală, dacă atributul *y* nu depinde funcţional de nici o submulţime proprie a lui *x*.

Definiţia 3. *Dependenţă funcţională tranzitivă.* Dependenţă funcţională *x R y* se numeşte tranzitivă, dacă există un atribut *z*, astfel încât au loc dependenţele funcţionale totale *x R z* şi *z R y* şi lipseşte

dependenţa funcţională $R.z \rightarrow R.x$. (Dacă ultima cerinţă nu va fi îndeplinită vom avea în orice relaţie, care posedă mai multe chei, dependenţe tranzitive “neinteresante”).

Definiţia 4. *Atribut simplu*. Numim atribut simplu orice atribut al relaţiei, care nu intră în componenţa cheii primare.

Definiţia 5. *Atribute reciproc independente*. Două şi mai multe atribute sunt reciproc independente dacă nici unul dintre aceste atribute nu este dependent funcţional de altele.

4.1.1. A doua formă normală

Considerăm următoarea schemă a relaţiei:

COLABORATORI–SECŢII-PROIECTE(Nr_COL, Salariu_COL, SECŢIA_Nr, Nr_PROIECT, SARCINA_COL).
Cheie primară: Nr_COL, Nr_PROIECT.

Dependenţe funcţionale:
Nr_COL → Salariu_COL
Nr_COL → SECŢIA_Nr
SECŢIA_Nr → Salariu_COL
Nr_COL, Nr_PROIECT → SARCINA_COL.

Observăm, că deşi în calitate de cheie primară a fost luat atributul compus Nr_COL, Nr_PROIECT, atributele Salariu_COL şi SECŢIA_Nr depind funcţional de o parte a cheii primare, atributul Nr_COL. Drept rezultat, nu vom putea introduce în relaţia COLABORATORI-SECŢII-PROIECTE cortejul, care descrie un colaborator, care nu este încă implicat în realizarea vreunui proiect (cheia primară nu poate conţine valori indefinite). Atunci când acest cortej va fi eliminat va fi distrusă nu numai legătura colaboratorului cu proiectul dat, dar va fi pierdută şi informaţia, că acest colaborator lucrează într-o secţie oarecare. Dacă colaboratorul este transferat într-o altă secţie vom fi nevoiţi să modificăm toate cortejurile, care descriu acest colaborator, în caz contrar vom obţine rezultate necoordonate. Aceste fenomene neplăcute se numesc **anomalii** ale schemei relaţiei. Ele sunt eliminate prin normalizare.

Definiţa 6. *A doua formă normală* (aici vom considera, că unica cheie a relaţiei este cheia primară). Relaţia *R* se află în a doua forma normală (*2FN*) atunci şi numai atunci, când aflându-se în prima formă normală, fiecare atribut simplu depinde total de cheia primară.

Ca exemplu va fi următoarea decompoziţie a relaţiei COLABORATORI-SECŢII-PROIECTE în două relaţii COLABORATORI-SECŢII şi COLABORATORI-PROIECTE:

COLABORATORI-SECŢII(Nr_COL, Salariu_COL, SECŢIA_Nr)

Cheia primară: Nr_COL.

Dependenţe funcţionale:
Nr_COL → Salariu_COL
Nr_COL → SECŢIA_Nr
SECŢIA_NR → Salariu_COL

COLABORATORI-PROIECTE(Nr_COL, Nr_PROIECT, SARCINA_COL)

Cheia primară: Nr_COL, Nr_PROIECT

Dependenţe funcţionale:
Nr_COL, Nr_PROIECT → SARCINA_COL

Fiecare dintre aceste relaţii se află în a doua formă normală şi în ele sunt eliminate anomaliile evidenţiate mai sus.

Dacă se va permite prezenţa mai multor chei, definiţia 6 va avea următoarea formulare:

Relaţia *R* se află în a doua forma normală (*2FN*) atunci şi numai atunci, când ea se află în prima formă normală şi fiecare atribut simplu depinde total de fiecare cheie a relaţiei *R*. Nu ne vom opri la exemple de relaţii cu mai multe chei, acestea fiind prea complicate şi rar întâlnite în practică.

4.1.2. A treia formă normală

Fie relaţia COLABORATORI-SECŢII, care se află în forma a doua normală. Observăm, că dependenţa funcţională Nr_COL → Salariu_COL este tranzitivă, fiind consecinţa dependenţelor funcţionale Nr_COL → SECŢIA_Nr şi SECŢIA_Nr → Salariu_COL. Altfel spus, salariul unui colaborator nu este o caracteristică a colaboratorului, ci a secţiei în care acesta lucrează (presupunere poate nu chiar corectă, dar acceptabilă pentru exemplu).

Drept rezultat, nu vom putea introduce în BD informaţii despre salariul unei secţii până când în această secţie nu va apare cel puţin un colaborator (cheia primară nu poate conţine valori indefinite). Eliminând cortejul, care descrie ultimul colaborator al secţiei date se va pierde informaţie despre salariul secţiei. Pentru a modifica în mod coordonat salariul secţiei vom fi nevoiţi să găsim mai înrâi toate cortejurile, care descriu colaboratorii secţiei date. Asta înseamă, că în relaţia COLABORATORI-SECŢII există în continuare anomalii. Acestea pot fi eliminate prin aceeaşi procedură de normalizare.

Definiţa 7. *A treia formă normală.* (Din nou vom defini presupunând existenţa unei chei unice). Relaţia *R* se află în a treia forma normală (*3FN*) atunci şi numai atunci, când se află în a doua formă normală, şi fiecare atribut simplu depinde netranzitiv de cheia primară.

Iată decompoziţia relaţiei COLABORATORI-SECŢII în două relaţii COLABORATORI şi SECŢII:

COLABORATORI(Nr_COL, SECŢIA_Nr)

Cheia primară: Nr_COL

Dependenţe funcţionale: Nr_COL → SECŢIA_Nr

SECŢII(SECŢIA_Nr, Salariu_COL)

Cheia primară: SECŢIA_Nr

Dependenţe funcţionale: SECŢIA_Nr → Salariu_COL

Fiecare dintre aceste două relaţii se află în a treia formă normală şi nu are anomaliile de mai sus.

Dacă vrem să omitem restricţia, că fiecare relaţie are o singură cheie, atunci definiţia *3FN* va fi următoarea:

Definiţia 7^{1} . Relaţia *R* se află în a treia forma normală (*3FN*) atunci şi numai atunci, când se află în a doua formă normală, şi fiecare atribut simplu depinde netranzitiv de o cheie oarecare a relaţiei *R*.

În practică, forma a treia normală în marea majoritate a cazurilor este cea finală şi cu determinarea acestei forme, de obicei, procesul de proiectare a BDR ia sfârşit. Dar câteodată, totuşi, procesul de normalizare trebuie continuat.

4.1.3. Forma normală Boyce-Codd

Considerăm următoarea schemă a relaţiei:

COLABORATORI-PROIECTE (Nr_COL, Nume_COL, Nr_PROIECT, SARCINA_COL)

Chei posibile:
Nr_COL, Nr_PROIECT
Nume_COL, Nr_PROIECT

Dependenţe funcţionale:
Nr_COL → Nume_COL
Nr_COL → Nr_PROIECT

Nume_COL → Nr_COL
Nume_COL → Nr_PROIECT
Nr_COL, Nr_PROIECT → SARCINA_COL
Nume_COL, Nr_PROIECT → SARCINA_PROIECT

Am presupus în acest exemplu, că persoana colaboratorului este definită în totalitate atât de numărul acestuia, cât şi de numele lui (nu ontotdeauna are loc on realitate).

În conformitate cu definiţia 7^{1} relaţia COLABORATORI-PROIECTE se află în forma normală 3. Însă, deoarece există dependenţe funcţionale ale atributelor relaţiei de un atribut, care este parte a cheii primare, apar anomalii. De exemplu, pentru a schimba în mod consistent numele colaboratorului cu numărul dat va fi necesar să modificăm toate cortejurile, care conţin numărul lui.

Definiţia 8.*Determinant*. Numim determinant orice atribut de care deăpinde funcţional în totalitate un oarecare alt atribut.

Definiţa 9. *Forma normală Boyce-Codde.* Relaţia *R* se află în forma normală Boyce-Codde atunci şi numai atunci când determinantul ei este o cheie posibilă.

Este evident, că această cerinţă nu este îndeplinită pentru relaţia COLABORATORI-PROIECTE. Putem realiza decompoziţia ei în relaţiile COLABORATORI şi COLABORATORI-PROIECTE:

COLABORATORI(Nr_COL, Nume_COL)

Chei posibile:
Nr_COL
Nume_COL

Dpendenţe funcţionale:
Nr_COL → Nume_COL
Nume_COL → Nr_COL

COLABORATORI_PROIECTE(Nr_COL, Nr_PROIECT, SARCINA_COL)

Cheie posibilă: Nr_COL, Nr_COL

Dependenţe funcţionale: Nr_COL, Nr_PROIECT → Sarcina_COL

Este posibilă şi o altă decompoziţie, dacă am lua la bază Nume_COL. În ambele cazuri relaţiile obţinute COLABORATORI şi COLABORATORI-PROIECTE se află în BCNF cu anomaliile evidenţiate.

4.1.4. A patra formă normală

Fie următorul exemplu de schemă a relaţiei:

PROIECTE(Nr_PROIECT, COL_PROIECT, PROIECT_SARCINĂ)

Relaţia PROIECTE conţine numerele proiectelor, lista colaboratorilor fiecărui proiect şi lista lucrărilor prevăzute în proiect. Colaboratorii pot participa în mai multe proiecte, iar proiecte diferite pot avea lucrări de acelaşi fel.

- Fiecare cortej al relaţiei legă un proiect oarecare de colaboratorul care participă în acest proiect şi de lucrarea pe care colaboratorul o îndeplineşte în cadrul proiectului dat (presupunem că orice colaborator participant la proiect îndeplineşte toate lucrările prevăzute de proiect). Din această cauză unica cheie posibilă a relaţiei este atributul compus Nr_PROIECT, COL_PROIECT, PROIECT_SARCINĂ şi nu există alţi determinanţi. Relaţia PROIECTE se află, deci, în BCNF. Ea are un şir de neajunsuri. De exemplu, dacă un colaborator oarecare este delegat la un proiect dat, va fi necesar să inserăm în relaţia PROIECTE atâtea cortejuri, câte lucrări sunt în proiect.

Definiţia 10. *Dependenţe multiple*. În relaţia *R(A, B, C)* există o dependenţă multiplă *R.A* > > *R.B* dacă şi numai dacă mulţimea valorilor lui *B*, care corespunde unei perechi de valori *A* şi *C* depinde numai de *A* şi nu depinde de *C*.

În relaţia PROIECTE există două dependenţe multiple

Nr_PROIECT > > COL_PROIECT

Nr_PROIECT > > PROIECT_SARCINĂ.

În caz general, în relaţia *R(A, B, C)* există o dependenţă multiplă *R.A* > > *R.B* dacă şi numai dacă există o dependenţă multiplă *R.A* > > *R.C*.

Normalizarea relaţiilr asemănătoare relaţiei PROIECTE se bazaează pe următoarea teoremă:

Relaţia *R(A, B, C)* poate fi proiectată fără pierderi în relaţiile $R_1(A,B)$ şi $R_2(A,C)$ atunci şi numai atunci când există *MVD A* > > *B*|*C*.

Prin proiectare fără pierderi se consideră decompoziţia uei relaţii, când relaţia iniţială poate fi restabilită integral şi fără redundanţă prin joncţiunea simplă a relaţiilor obţinute prin decompoziţie.

Definiţia 11. *A patra formă normală.* Relaţia *R* se consideră în a patra formă normală atunci şi numai atunci când în cazul existenţei dependenţei multiple *A*> > *B* toate celelalte atribute ale *R* depind funcţional de *A*.

În exemplul nostru putem propune decompoziţia relaţiei PROIECTE în două relaţii PROIECTE_COLABORATORI şi PROIECTE_SARCINI:

PROIECTE_COLABORATORI(Nr_PROIECT, COL_PROIECT)

PROIECTE_SARCINI(Nr_PROIECT, PROIECT_SARCINĂ).

Ambele relaţii sunt în a patra formă normală şi nu au anomaliile de mai sus.

4.1.5. A cincea formă normală

În toate operaţiile de normalizare petrecute până acum o relaţie a fost descompusă în două. Nu întotdeauna este posibil acest lucru, dar este posibilă decompoziţia în mai multe relaţii, fiecare având proprietăţi mai bune. Considerăm relaţia COLABORATORI_SECŢII_PROIECTE(Nr_COL, Nr_SECŢIE, Nr_PROIECT).

Presupunem că un colaborator poate lucra în mai multe secţii, iar în fiecare secţie – cu mai multe proiecte. Cheie primară pentru această relaţie este întreaga totalitate a atributelor, lipsind dependenţe funcţionale şi multiple. Din această cauză relaţia se află în *4NF*. Însă pot exista anomalii, care pot fi eliminate prin decompoziţia relaţiei iniţiale în trei relaţii.

Definiţia 12. *Dependenţa joncţiunii*. Relaţia *R(X, Y,…, Z)* satisface dependenţele joncţiunii **(X, Y,…, Z)* atunci şi numai atunci când *R* poate fi restabilită fără pierderi prin joncţiunea proiecţiilor sale pe *X*, *Y,…, Z*.

Definiţia 13. *A cincea formă normală.* Relaţia *R* se află în a cincea formă normală (proiecţie-joncţiune – PJ/NF) atunci şi numai atunci când orice dependenţă a joncţiunii în *R* rezultă din existenţa unei oarecare chei posibile în *R*.

Introducem următoarele nume pentru atributele compuse:

CS = {Nr_COL, Nr_SECŢIE}

CP = {Nr_COL, Nr_PROIECT}

SP = {Nr_SECŢIE, Nr_PROIECT}

Presupunem că în relaţia COLABORATORI_SECŢII_PROIECTE există dependenţa joncţiunii **(CS, CP, SP)*: Prin exemple poate fi arătat, că la inserarea şi eliminarea cortejurilor pot să apară probleme. Acestea pot fi evitate prin decompoziţia relaţiei iniţiale în trei relaţii noi:

COLABORATORI_SECŢII(Nr_COL, Nr_SECŢIE)

COLABORATORI_PROIECTE(Nr_COL, Nr_PROIECT)

SECŢII_PROIECTE(Nr_SECŢIE, Nr_PROIECT)

A cincea formă normală este ultima, care poate fi obţinută prin decompoziţie. Condiţiile acestei forme sunt departe de a fi triviale din care cauză 5NF în practică se foloseşte relativ rar