7 SISTEME DE OPERARE

Un sistem de calcul fără software-ul său este inutil. Software-ul exploatează universalitatea sistemului, asigurându-i potenţialul necesar rezolvării problemelor. Sistemul de calcul, prin intermediul software-ului său, poate de exemplu, să faciliteze scrierea unei scrisori, să recunoască forme, să prevadă timpul, să stocheze sau să imprime imagini, să corecteze erori de ortografie, să realizeze simplificarea unei expresii algebrice, optimizarea concepţiei unui circuit electronic, simularea comportamentului unor subansamble pentru diverse tipuri de maşini etc.

De-a lungul timpului software-urile au evoluat în mai multe direcţii:

a) **software-uri de aplicaţie**, care servesc la rezolvarea unor probleme specifice, pot fi scrise de utilizator, sau sunt elaborate de către firme specializate, cum este cazul programelor de utilitate generală: procesoare de texte, software-uri de gestiune şi contabilitate, pentru concepţia asistată de calculator, didactice etc.
b) **software-uri utilitare**, care oferă ajutor pentru dezvoltarea aplicaţiilor, cum ar fi: compilatoarele, asambloarele, linkeditoarele, programele încărcătoare şi depanatoare etc., cuprinzând de asemenea instrumente grafice şi de comunicare.

Sistemul de operare [operating sistem] este un produs software care coordonează ansamblul tuturor activităţilor sistemului de calcul, asigură gestiunea eficientă a resurselor sale şi facilitează munca utilizatorilor prin preluarea unor sarcini complicate cum ar fi, de exemplu, controlul perifericelor sau stocarea şi gestiunea fişierelor.

Sistemul de operare permite interacţiunea directă între om şi sistemul de calcul, oferind în acest sens o interfaţă convenabilă şi organizând prelucrarea şi stocarea programelor şi datelor.

Sarcinile sistemelor de operare s-au amplificat de-a lungul timpului şi astfel, volumul său a crescut considerabil, necesitând eforturi deosebite pentru programare. Un sistem de operare modern este constituit din sute de mii, chiar milioane de instrucţiuni, solicitând utilizarea limbajelor de programare evoluate adaptate scrierii acestui gen de software.

Pentru a reliefa importanţa deosebită a sistemului de operare, în lumea informaticii se spune adesea că **“un elefant este un şoricel împreună cu sistemul său de operare”**.

7.1 Evoluţie şi caracteristici

Calculatoarele primei generaţii erau lipsite de sistem de operare, erau programate în binar, programele erau încărcate în memorie, executate şi puse la punct de la pupitrul de comandă.

Chiar şi după apariţia limbajului de asamblare procedura de utilizare a sistemului de calcul era aceeaşi, utilizatorul era stăpân unic asupra maşinii pe toată durata de execuţie a programului său care putea să fie foarte mare.

Etapele evoluţiei unui sistem de operare corespund introducerii unor noi dispozitive capabile să amelioreze performanţele sistemului de calcul.

Evoluţia sistemelor de operare cuprinde următoarele etape:

a) **Monitorul** sau **executivul** anilor ‘60 este precursorul sistemelor de operare moderne. El este un program însărcinat cu asigurarea bunei desfăşurări a operaţiilor prin secvenţializarea acţiunilor utilizatorilor şi continuitatea operaţiilor;

b) **Sisteme pe loturi** sau **trenuri de lucrări** [batch], sunt sisteme monoutilizator apărute la sfârşitul anilor ‘50, o dată cu introducerea primelor sisteme pe tranzistoare şi dotate cu benzi magnetice, care asigurau succesiunea mai rapidă a informaţiilor. Programarea era în plin avânt datorită apariţiei limbajului Fortran. Separarea operaţiilor de intrare/ieşire de cele de prelucrare se realizează prin utilizarea unui calculator auxiliar. Joburile, prezentate operatorului sub formă de pachete de cartele perforate sunt recopiate pe bandă magnetică de către calculatorul auxiliar. Banda, conţinând un lot de lucrări se constituie ca intrare pentru calculatorul principal care execută acţiunile în ordinea prezentării, iar rezultatele se scriu de asemenea pe bandă pentru a putea fi imprimate în continuare de către calculatorul auxiliar;

c) **Independenţa intrărilor/ieşirilor** este un deziderat atins la mijlocul anilor ‘50, şi care se materializează prin realizarea următoarelor dispozitive:

- **unităţi de discuri** şi **tambururi magnetice** la un preţ abordabil, care oferă un acces aleator şi capacităţi de memorare importante;
- **unităţi de canal** şi **sistemul de întreruperi**, care elimină dependenţa intrărilor/ieşirilor de prelucrări;

Joburile sunt citite şi stocate pe disc în fişiere (spooling), şi sunt utilizate de către monitor în momentul execuţiei, acesta putând alege ordinea de execuţie pe baza unor criterii de exploatare care se prezintă sub forma unui sistem de priorităţi. Monitorul este dotat în acest sens cu un nou modul, **planificatorul** [scheduler], care asigură planificarea lucrului.

d) **Multiprogramarea**. Un sistem de calcul este multiprogramat dacă mai multe programe sunt încărcate în memorie în scopul partajării CPU-ului, asigurându-se exploatarea mai eficientă a sistemului de calcul prin eliminarea perioadei de aşteptare a unităţilor de

prelucrare în timpul operaţiilor de intrare/ieşire. Sistemul de operare va conţine un nou modul, **alocatorul**, care asigură gestiunea CPU ţinând cont de sistemul de priorităţi gestionat de către planificator, dar poate decide de asemenea întreruperea unei execuţii prelungite fără să aştepte o operaţie de intrare-ieşire în vederea unei partajări echitabile a CPU-ului. Multiprogramarea ridică un anumit număr de probleme:

- partajarea unităţii de prelucrare între programe şi salvarea contextului (starea de execuţie) pentru fiecare program;
- gestiunea memoriei centrale încât să permită încărcarea unui număr ridicat de programe într-un spaţiu limitat;
- gestiunea intrărilor/ieşirilor pentru diversele programe, asigurând transferul de date între memorie şi unităţile periferice şi împiedicând sistemul să **"**amestece**"** informaţiile specifice programelor diferite;
- protejarea programelor şi a datelor stocate în memoria centrală şi pe disc, eventualele erori putând avea consecinţe grave asupra derulării operaţiilor;

e) **Sistemele în timp partajat** [time-sharing], numite de asemenea **multiacces** sau **multiutilizator**, sunt o variantă a sistemelor multiprogramate în care timpul CPU este distribuit în mici tranşe egale unui mare număr de utilizatori interactivi, conectaţi la sistem. Sistemele actuale combină prelucrările batch şi time-sharing. Într-un sistem în timp partajat, orice job lansat de la un terminal poate fi direct controlat de către utilizator, care are posibilitatea, de exemplu, să decidă corectarea erorilor, recompilarea şi relansarea în execuţie. Acest mod de exploatare este adaptat în special fazei de punere la punct a unui program, în timp ce modul batch este utilizat în general pentru aplicaţii care vizează activităţi de producţie şi pentru alte sarcini interactive.

Caracteristicile sistemelor de operare pentru sistemele multiprogramate vizează următoarele aspecte:

a) **Exploatarea resurselor** este o sarcină fundamentală care asigură:

- **paralelismul** între diverse activităţi în vederea creşterii performanţelor sistemului de calcul. Noţiunea de procese paralele şi concurente facilitează înţelegerea funcţionării unui sistem de calcul multiprogramat;
- **partajarea resurselor** şi a **informaţiilor** este asigurată prin gestiunea diferitelor unităţi funcţionale ale sistemului (CPU, memorie centrală şi auxiliară, dispozitive de intrare/ieşire), permiţând acccesul simultan la datele comune (baze de date, fişiere etc.) şi la anumite programe (utilitare, biblioteci etc.);

- **interdependenţa între funcţiunile sistemului de operare şi alte funcţiuni**, deoarece nu există o distincţie netă între ele, de exemplu, compilatorul şi editorul de texte sunt considerate programe utilitare, iar în sistemul Unix, modulul de gestiune a fişierelor este tratat ca o aplicaţie oarecare;
- **nedeterminismul operaţiilor** care vizează caracteristica de comportament a sistemelor de operare. Dacă la nivelul unei aplicaţii, execuţii repetate cu aceleaşi date produc aceleaşi rezultate, sistemul de operare trebuie să reacţioneze la situaţii nereproductibile, la evenimente aleatoare, ca de exemplu, întreruperi generate de dispozitive de intrare/ieşire, transferuri de date repetate ca urmare a erorilor detectate în urma verificărilor de paritate, incidente în funcţionare etc.

b) **Virtualizarea sistemului** prezintă utilizatorului, prin intermediul **limbajului de comandă**, o maşină virtuală mai uşor de programat decât cea reală. Limbajul de comandă furnizează modalitatea de a comunica sistemului, prin formularea de cereri, toate informaţiile necesare diferitelor etape de lucru. Deşi există o oarecare asemănare între limbajele de comandă şi cele de programare în sensul formulării unor fraze care specifică într-un mod neambiguu acţiuni de executat, totuşi, instrucţiunile unui limbaj evoluat de programare sunt în mod normal executate de către CPU după ce au fost traduse de către compilator, în timp ce comenzile unui limbaj de comandă sunt interpretate de către sistemul de operare. Maşina virtuală **"**ascunde**"** utilizatorului toate detaliile privind, de exemplu, o operaţie de intrare/ieşire sau o manipulare de fişiere;

c) **Dispozitivele esenţiale pentru sisteme multiprogramate**, ca de exemplu, canale de intrare/ieşire, sistemul de întreruperi, memoriile auxiliare şi terminalele interactive, la care se adaugă alte dispozitive pentru protecţia programelor şi a datelor, pentru relocarea dinamică a programelor sau pentru gestiunea memoriei virtuale stau la baza oricărui sistem de operare modern;

d) **Maşină cu două stări:**
   - **starea supervizor**, rezervată sistemului de operare;
   - **starea utilizator**, în care intră programele de aplicaţie.

   Această concepţie permite dotarea sistemului cu un set instrucţiuni de bază, executabile în cele două stări, şi câteva instrucţiuni suplimentare, **instrucţiuni privilegiate**, executabile numai în mod supervizor. Starea sistemului este specificată prin poziţionarea unui indicator accesibil pentru modificare doar sistemului de operare (de exemplu, un bit al registrului de stare).

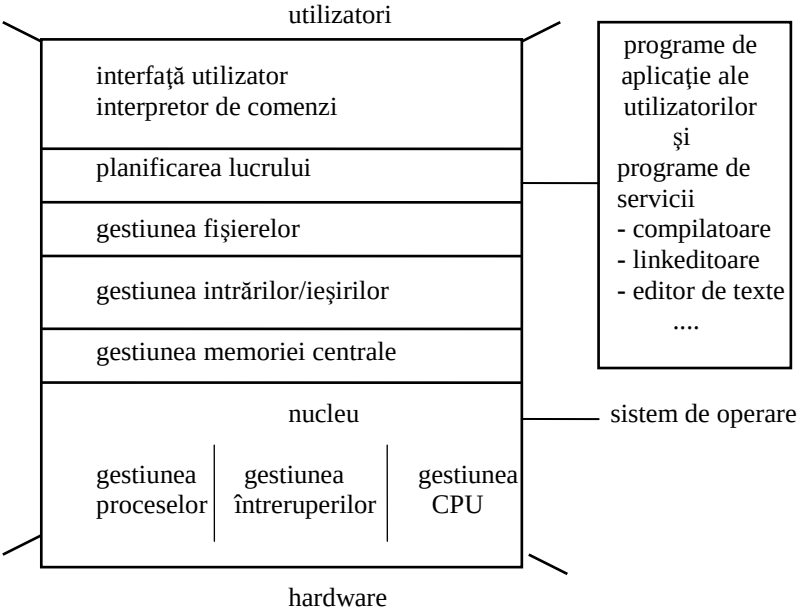
În anumite situaţii, indicatorul de stare trece automat în starea supervizor, de exemplu în cazul unei întreruperi sau în caz de eroare, sau în general, ca urmare a unui eveniment care necesită intervenţia sistemului.
Interfaţa între cele două stări este asigurată prin existenţa unei instrucţiuni neprivilegiate, numită cerere a supervizorului.

e) **Program, procesor, proces.** Un **program** este o secvenţă statică de instrucţiuni. Un **procesor** execută instrucţiunile unui program. Un **proces** este o acţiune, o secvenţă de operaţii care se derulează pentru realizarea unei sarcini determinate, pe scurt, este un program în execuţie.

7.2 Structura unui sistem de operare modern

Examinând funcţiunile unui sistem de operare modern, se poate aprecia că acesta se bazează pe un model structurat pe nivele, de exemplu, sistemul de gestiune a fişierelor face apel la sistemul de gestiune a intrărilor/ieşirilor, care la rândul său, utilizează modulul de prelucrare a întreruperilor etc.

Figura următoare prezintă structura unui sistem de operare conform modelului expus, din care se desprinde ideea complexităţii sistemului de operare.



Se poate realiza un model de sistem de operare bazat pe o suprapunere de nivele funcţionale, nivelele inferioare fiind în interacţiune directă cu hardware-ul, iar cele superioare servesc interfeţei cu utilizatorul. Fiecare nivel utilizează funcţiuni definite de către nivelele inferioare, iar această abordare este asemănătoare cu cea adoptată pentru definirea nivelelor funcţionale privind protocoalele de comunicare într-o reţea de calculatoare (modelul ISO-OSI).

Nucleul sistemului de operare

Funcţiunile principale ale nucleului [kernel] sunt următoarele:

a) alocarea CPU;
b) gestiunea întreruperilor;
c) gestiunea proceselor.

Nucleul este singura componentă a SO în întregime rezidentă în memoria centrală. Funcţiunile sale implică intervenţii frecvente şi rapide. De aceea, şi datorită ocupării permanente a unei părţi din memorie, codificarea nucleului trebuie realizată cu mare atenţie şi optimizată.

Este într-adevăr nivelul cel mai solicitat al sistemului, codificat adesea în limbaj de asamblare, restul sistemului fiind codificat în limbaj de programare evoluat orientat sistem (C, Pascal concurent, Modula-2).

Alocare CPU

Alocatorul (dispecerul) este responsabil cu repartizarea timpului disponibil unităţii de prelucrare (sau unităţilor de prelucrare în cazul arhitecturilor multiprocesor) între diferite procese.

Sarcina sa implică gestiunea unui fir de aşteptare, unde procesele care sunt gata să utilizeze CPU sunt clasate în ordinea **priorităţii.** Prioritatea este atribuită de **planificator** în funcţie de urgenţa prelucrării şi resursele solicitate şi este modificată dinamic pe baza timpului de aşteptare între două execuţii parţiale. Dispecerul alocă CPU procesului care se găseşte în capul cozii în momentul în care CPU devine disponibil.

Dispecerul trebuie de asemenea să salveze starea (contextul) procesului a cărui execuţie s-a întrerupt şi trebuie să furnizeze CPU-ului elementele de context (echipament) ale procesului desemnat ca succesor.

Pentru salvarea informaţiilor privind starea proceselor, se asociază fiecărui proces o zonă de memorie conţinând toate informaţiile esenţiale ca: identificator, paritate, context, statut (de exemplu **activ**, dacă este stăpân al CPU; **gata de execuţie**, dacă este încărcat în memorie şi dispune de toate resursele, fără CPU; **în aşteptare**, dacă este pe disc în aşteptarea posesiei perifericelor şi spaţiului de memorie necesare; **suspendat**, dacă execuţia sa

a fost întreruptă; **terminat**, dacă procesul şi-a realizat sarcinile şi execuţia sa a luat sfârşit), necesităţile în resurse etc.

Acest bloc de informaţii se numeşte **vector de stare**, sau **descriptor**, sau **imaginea procesului**. Aceşti descriptori sunt regrupaţi într-o structură de date şi pot fi accesaţi printr-un pointer pornind de la o tabelă centrală. Această structură este accesibilă programelor nucleului.

Dispecerul este solicitat în toate cazurile în care trebuie schimbat procesul stăpân al CPU. De exemplu când procesul executant declanşează o operaţie de intrare/ieşire, sau când o întrerupere de ceas semnalează că tranşa de timp alocată este epuizată şi trebuie suspendată execuţia, atunci trebuie atribuit CPU unui alt proces.

Dispecerul va fi de asemenea activ când o întrerupere externă modifică starea procesului stăpân al CPU, sau îl face pe moment inoperant (de exemplu tratarea unei erori).

Gestiunea proceselor

Un **proces** (task) este un **calcul** care poate fi executat concurent sau în paralel cu alte calcule. El este o abstractizare a procesorului, fiind considerat ca un program în execuţie.

Existenţa unui proces este condiţionată de existenţa a trei factori:

a) **o procedură** (un set de instrucţiuni) care trebuie executată;
b) un **procesor** care să poată executa aceste instrucţiuni;
c) un **mediu** (memorie, periferice) asupra căruia să acţioneze procesorul conform celor precizate în procedură.

Programare paralelă şi concurentă

Caracteristic **programării paralele** este faptul că procesele paralele nu sunt condiţionate unul de celălalt, nu colaborează între ele, execuţia unuia nu este în nici un moment dependentă de rezultatele parţiale ale celuilalt.

Spunem că avem de-a face cu **programare concurentă** atunci când procesele paralele se intercondiţionează reciproc.

Într-un sistem de calcul, paralelismul proceselor trebuie înţeles astfel:

Dacă I_i şi I_j sunt momentele de început a două procese P_i şi P_j, iar H_i şi H_j sunt momentele lor de sfârşit, P_i şi P_j sunt executate concurent dacă $\max(I_i, I_j) \leq \min(H_i, H_j)$.

Definiţia 7.1 Execuţie paralelă

Despre două instrucţiuni succesive S_1 şi S_2 spunem că *pot fi executate în paralel*, dacă efectul lor asupra mediului este acelaşi, indiferent dacă mai întâi se execută complet S_1 şi apoi S_2, sau se execută complet S_2 şi apoi S_1, sau execuţia uneia începe înaintea terminării execuţiei celeilalte.

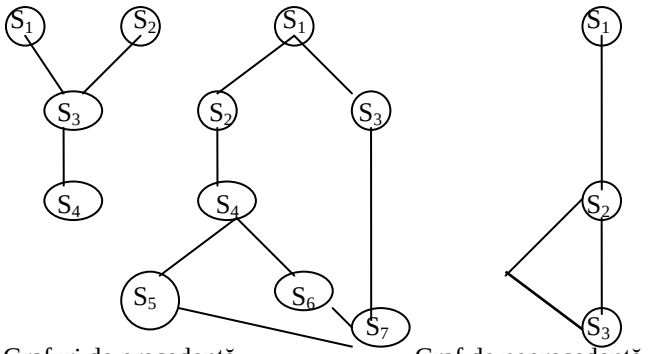
Exemplul 1: Un program citeşte de la două periferice diferite pe *a* şi *b*, după care tipăreşte suma lor *c*. S_1 şi S_2 se pot executa în paralel.

S_1: read (a);
S_2: read (b);
S_3: c := a+b;
S_4: write (c);

Definiţia 7.2 Graf de precedenţă

Un graf aciclic (X, U) este graf de precedenţă asociat unui program, dacă X este mulţimea instrucţiunilor programului, iar mulţimea arcelor U este formată din perechi (S_i, S_j) pentru care S_i precede S_j.

Exemple de grafuri de precedenţă şi neprecedenţă:



Grafuri de precedenţă Graf de neprecedenţă

Mecanisme de specificare a concurenţei

Grafurile de precedenţă sunt un model matematic pentru concurenţă, dar nu pot descrie direct concurenţa în limbajele de programare.

Construcţiile FORK, JOIN, QUIT

Instrucţiunea FORK are sintaxa: **FORK** *etichetă;* şi provoacă execuţia concurentă a două secvenţe de program, deci crează două procese paralele. Instrucţiunile primului proces încep la eticheta *etichetă*, iar pentru al doilea proces instrucţiunile sunt cele care urmează după FORK.

Instrucţiunea JOIN are sintaxa: JOIN *nr, etichetă* şi are rolul de a recombina *nr* procese, toate terminate. După ce a fost executată a *nr*-a oară,

se trece la instrucţiunea cu eticheta *etichetă*. Variabila *nr* indică numărul de procese paralele care mai sunt de aşteptat în vederea reunirii.

Instrucţiunea QUIT are sintaxa: QUIT şi are ca efect terminarea procesului care o execută (de obicei însoţeşte pe JOIN).

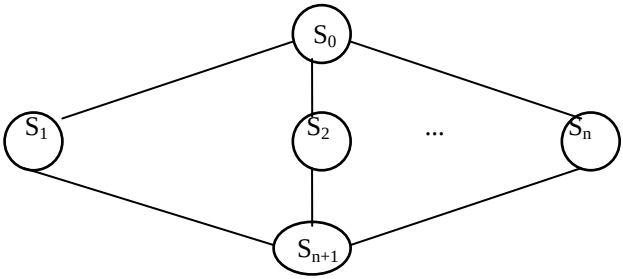
Prezentăm descrierea FORK-JOIN-QUIT pentru grafurile de precedenţă din figura anterioară:

```
        nr := 2;
        FORK L1;
        read(a);
        goto L2;
L1:     read (b);
L2:     JOIN nr, L3;
        QUIT;
L3:     c:= a+b;
        write (c)
```

```
        S1;
        nr := 3;
        FORK L1;
        S2;
        S4;
        FORK L2;
        S5;
        goto L3;
L2:     S6;
        goto L3;
L1:     S3;
L3:     JOIN nr, L4;
        QUIT;
L7:     S7;
```

Mecanismul PARBEGIN-PAREND

O astfel de construcţie are forma: S_0; PARBEGIN S_1 | S_2 | ... | S_n PAREND; S_{n+1}; şi corespunde grafului de precedenţă următor:



Deci instrucţiunile S_1, S_2, ..., S_n sunt lansate în execuţie simultan şi sunt executate concurent. Prezentăm descrierea grafurilor de precedenţă din figura anterioară cu PARBEGIN-PAREND.

```
                                    S1:
                                    PARBEGIN
                                     begin
PARBEGIN                                  S2;
      read (a);                           S4;
      read (b);                           PARBEGIN
 PAREND                                         S5;
 c := a+ b;                                     S6;
 write (c);                                PAREND
                                     end;
                                    S3;
                                    PAREND
                                    S7;
```

Prezentăm în continuare un exemplu de copiere a unui fişier F într-un fişier G, situat pe un suport diferit, folosind instrucţiuni concurente:

```
Type    articol = record . . . end;
Var     F, G: file of articol;
        w, r: articol;
begin
        reset (F);
        rewrite (G);
        read (F, r);
        while not  eof (F)  do  begin
                w:=r;
                PARBEGIN
                        read (F, r); | write (G, w);
                PAREND;
        end;
        write (G,r);
        close (F);
        close (G);
end.
```

Conceptul de semafor

Un *semafor s* este o pereche (v (s), c (s)), unde v (s) este valoarea semaforului (valoarea iniţială v_0 (s)), iar c (s) o coadă de aşteptare care conţine pointeri la procesele care aşteaptă la semaforul s. Pentru gestiunea semafoarelor se definesc două operaţii primitive: WAIT şi SIGNAL.

WAIT(s) apelat de procesul A :
v (s):=v (s) -1;
if v (s) < 0 **then begin**
Trece A în aşteptare;
c(s) < == A;
Trece controlul la dispecer;
end
else
Se trece controlul la A;
{**endif**}
	SIGNAL(s) apelat de procesul A:
	v (s):= v (s) + 1;
	if v (s) <= 0 **then begin**
	c (s) == > B;
	Se extrage din coadă B;
	Trece controlul la dispecer;
	end
	else
	Se trece controlul la A;
	{**endif**}

Deci operaţia SIGNAL (s) incrementează semaforul s cu o unitate, iar WAIT (s) decrementează s cu o unitate dacă s este pozitiv.

Secţiune critică, resursă critică, excludere mutuală

Problema secţiunii critice constă în aceea că două sau mai multe procese concurente încearcă să modifice o aceeaşi variabilă, de un număr neprecizat de ori:

```
PARBEGIN
        P1: . . . v := v + 1; . . .
        P2: . . . v := v + 1; . . .
PAREND
```

Vom spune că porţiunea de program prezentată, v := v + 1 este o *secţiune critică*, deoarece nu este permis ca ea să fie executată simultan de cele două procese. Analog, vom spune că variabila v este o *resursă critică*, deoarece nu poate fi accesată simultan de mai multe procese. Procesele P_1 şi P_2 se *exclud reciproc*, deoarece au acces exclusiv la secţiunea şi la resursa critică.

Folosirea semafoarelor rezolvă complet şi elegant problema secţiunii critice, fiind necesar un singur semafor s care se numeşte *semafor de excludere mutuală*.

```
var s:semafor;
  v0 (s):=1;
  PARBEGIN
        P1:     repeat
                        WAIT (s);
                        secţiune critică;
                        SIGNAL (s);
                        rest program1
                until false
        P2:     repeat
                        WAIT (s);
                        secţiune critică;
                        SIGNAL (s);
                        rest program2
                until false
  PAREND
```

Sincronizarea proceselor

Operaţia de *sincronizare* a două procese se enunţă astfel: Un proces P_1 nu poate trece de un anumit punct A decât după ce un alt proces P_2 ajunge într-un punct B. Prin intermediul semafoarelor, procesul P_1 va aştepta în punctul A, prin operaţia WAIT (s), până când procesul P_2 va efectua în punctul B operaţia SIGNAL (s).

```
  var s:semafor;
  v0 (s) := 0;
  PARBEGIN
        P1:     repeat   . . . A: WAIT (s); . . . until false;
        P2:     repeat   . . . B: SIGNAL (s) . . . until false;
  PAREND
```

Prezentăm în continuare câteva aplicaţii ale sincronizării proceselor:

Problema producătorului şi consumatorului

Să presupunem că există unul sau mai multe procese numite *producătoare*, şi unul sau mai multe procese *consumatoare* (de exemplu conceptele de *pipe* şi *spooling*). Transmiterea informaţiilor de la producători

la consumatori se efectuează prin intermediul unui buffer cu *n* intrări. Problema constă în a dirija cele două tipuri de procese astfel ca:

a) să existe acces exclusiv la buffer (semaforul *exclus*);
b) consumatorii să aştepte când bufferul este gol (semaforul *plin*);
c) producătorii să aştepte când bufferul este plin (semaforul *gol*).

```
Var     plin, gol, exclus: semafor;
v0 (plin) := 0;
v0 (gol) := n;
v0 (exclus) :=1;
PARBEGIN
        Producător:     repeat
                            produce articol;
                            WAIT (gol);
                            WAIT (exclus);
                            depune articol în buffer;
                            SIGNAL (exclus);
                            SIGNAL (plin);
                        until false;
        Consumator:     repeat
                            WAIT (plin);
                            WAIT (exclus);
                            extrage articol din buffer;
                            SIGNAL (exclus);
                            SIGNAL (gol);
                            consumă articol;
                        until false
PAREND
```

Problema impasului

Problema impasului se manifestă în următoarea situaţie când ambele procese sunt blocate.

```
var x,y: semafor;
v0 (x) := 1;
v0 (y) := 1;
PARBEGIN
        A:      . . . WAIT (x);     . . . WAIT (y);
        B:      . . .  WAIT (y);    . . .  WAIT (x);
PAREND
```

Impasul este o stare foarte gravă care poate duce la blocarea sistemului de operare sau la distrugerea unor procese.

Modelarea matematică a impasului

Să considerăm *n* procese şi *m* tipuri de resurse. Un proces oarecare va fi notat cu litera i şi $i \in \{1, 2, ..., n\}$, iar o resursă oarecare cu litera j, şi $j \in \{1, 2, ..., m\}$.

Notăm cu x[j] cantitatea din resursa j existentă în sistem;

Notăm cu c [i, j] (t) cantitatea din resursa j cerută de procesul i la momentul t;

Notăm cu a [i, j] (t) cantitatea din resursa j alocată procesului i la momentul t;

Notăm cu r [j] (t) cantitatea din resursa j care mai este liberă la momentul t;

Definiţie 5.3 Stare realizabilă

Numim *stare realizabilă* a alocării resurselor la momentul t, dacă au loc condiţiile:

$c [i, j] (t) \leq x [j], \forall i, \forall j;$

$a [i, j] (t) \leq c [i, j] (t), \forall i, \forall j;$

$a [1, j] (t) + a [2, j] (t) + ... + a [n, j] (t) \leq x [j], \forall i, \forall j;$

Problemele impasului şi rezolvarea lor

a) Ieşirea din impas se rezolvă de regulă adoptând următoarele strategii:
   - reîncărcarea sistemului de operare;
   - alegerea unui proces **"**victimă**"** care este distrus;
   - crearea unui *punct de reluare,* care este o fotografie a memoriei pentru procesul **"**victimă**"** şi pentru procesele cu care colaborează;

b) Detectarea unui impas dacă sistemul posedă un mecanism de prevenire a impasului (de exemplu, detectarea ciclurilor în graful de alocare a resurselor);

c) Prevenirea impasului se face prin urnmătoarele metode:
   - Metoda 1: **"**totul sau nimic**"**. Procesul cere resurse în momentul încărcării, iar SO întocmeşte graful alocării resurselor;
   - Metoda a 2-a: cererea de resurse într-o anumită ordine, prin numerotarea resurselor;
   - Metoda a 3-a: alocare şi suspendare controlată.

Gestiunea întreruperilor

Întreruperile sunt constituite dintr-un ansamblu de rutine, fiecare dintre ele fiind activată la apariţia unui semnal fizic de întrerupere.

Sarcina modulului de gestiune a întreruperilor este determinarea sursei întreruperii şi activarea rutinei de serviciu sau de răspuns corespunzătoare.

Am văzut ce este un program în limbaj maşină. Putem presupune că execuţia unei instrucţiuni maşină nu poate fi întreruptă.

Prin starea unui program la un moment dat, notată prescurtat PSW [Program Status Word] înţelegem o pereche formată din:

a) adresa următoarei instrucţiuni de executat;
b) conţinutul registrelor maşină la terminarea ultimei instrucţiuni.

Fiecare sursă posibilă a unei întreruperi are asociată o locaţie fixă de memorie. În această locaţie se află o adresă care indică locul din memorie la care se găseşte o secvenţă de instrucţiuni, numită *handler*, care deserveşte întreruperea respectivă.

La apariţia semnalului de întrerupere, după ce instrucţiunea maşină în curs s-a executat, se derulează în această ordine, următoarele activităţi:

a) se salvează într-o zonă de memorie (în stivă, sau o zonă prestabilită) PSW-ul programului în curs de desfăşurare;
b) se restaurează PSW al handlerului asociat întreruperii;
c) handlerul execută acţiunile necesare deservirii întreruperii;
d) se salvează, numai dacă este necesară corelarea a două acţiuni succesive ale aceluiaşi handler, PSW al handlerului;
e) se restaurează PSW al programului care a fost întrerupt.

Printre întreruperile care trebuie tratate la acest nivel se includ **întreruperile interne**, provocate, de exemplu, de detectarea unei erori sau printr-o acţiune care solicită trecerea în starea supervizor, ca şi toate **întreruperile externe**.

Ca exemplu de acţiune care cauzează trecerea în starea supervizor se poate cita cazul unui utilizator care încearcă să execute o instrucţiune privilegiată sau caută să acceseze o informaţie protejată etc.

Dacă întreruperea implică o schimbare de alocare a procesorului, acest modul al nucleului va activa dispecerul.

Gestiunea memoriei centrale

Programele au nevoie de memorie pentru execuţia lor (pentru stocarea instrucţiunilor şi datelor). Numai instrucţiunile stocate în memoria centrală pot fi executate de CPU.

Dacă sistemul de calcul (SC) este exploatat în **monoprogramare**, problema se reduce la partajarea memoriei între programul de aplicaţie de executat şi partea sistemului de operare rezidentă în memorie, care ocupă în

mod normal o zonă de adrese pornind de la adresa 0, care se numeşte adesea partea de jos a memoriei.

Dacă sistemul ocupă zona de adrese de la 0 la N, programul utilizator va avea la dispoziţie spaţiul de adrese de la N + 1 la extremitatea superioară a memoriei.

Dacă programul utilizator are o talie mai mare decât spaţiul disponibil, programatorul trebuie să **"**decupeze**"** programul în module care să se poată succede în zona de memorie pusă la dispoziţia lor.

Sistemul nu este în mod obligatoriu stocat în memoria RAM, în anumite cazuri se preferă utilizarea unei memorii ROM separate, pentru raţiuni de protecţie şi nevolatilitate.

Memoria centrală este o resursă costisitoare şi deci ea este limitată şi constituie un element critic al performanţei unui calculator.

Dimensiunea memoriilor a crescut considerabil, o dată cu adoptarea memoriilor electronice, dar şi cea aprogramelor a urmat aceeaşi tendinţă.

Spaţiul de memorie trebuie gestionat eficient, iar al doilea nivel al SO este asigură partajarea memoriei între mai multe procese în aşteptare.

Partiţii cu talie fixă

Ideea cea mai simplă de partiţionare a memoriei ca ea să poată conţine un număr maxim de programe constă în **"**decuparea**"** memoriei fizice disponibile în **partiţii fixe**, dar nu obligatoriu de talie identică, fixate la generarea sistemului.

Aceste partiţii fiind fixate în avans şi o dată pentru totdeauna, alocarea lor prezintă câteva probleme:

a) trebuie gestionate mai multe cozi de procese în aşteptare, sortând procesele în funcţie de talia lor;
b) se produce o oarecare risipă de memorie, deoarece nu poate fi prevăzută talia joburilor de executat şi rareori se va găsi un proces a cărui talie să corespundă unei partiţii prestabilite.

Partiţii cu talie variabilă

Inevitabila risipă de memorie a sistemelor cu partiţii fixe conduce la concepţia partiţiilor adaptabile taliei programelor, numite **partiţii cu talie variabilă.**

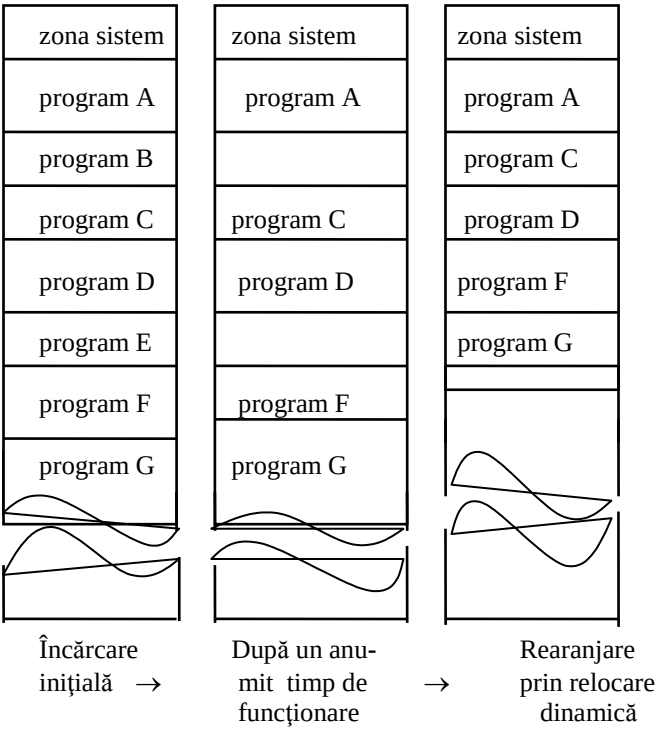
Dacă unul sau mai multe programe îşi termină execuţia, apar zone de memorie neocupate, deci ar fi de dorit să se ofere posibilitatea deplasării programelor în memorie. În acest sens, sistemul, din când în când suspendă execuţia programelor şi efectuează o compactare a spaţiului de memorie care permite rearanjarea memoriei rămase şi crearea de spaţiu pentru programele în aşteptare.

Pentru a realiza această operaţie de reaşezare sau compactare este necesară o tehnică de deplasare corectă a programelor încât acestea să-şi poată continua execuţia. Această tehnică se numeşte **relocare (realocare)**, sau **translatare dinamică** şi este realizată cu ajutorul registrelor de bază şi a dispozitivului de calcul al adresei efective în momentul execuţiei.

Adresa efectivă = adresa de bază + adresa începutului de program

Relocarea dinamică a spaţiului de adrese poate fi realizată deplasând programul şi modificând conţinutul registrului de bază asociat, iar execuţia se va efectua corect, adresele fiind calculate la execuţie, în funcţie de adresa conţinută în registrul de bază.

Figura următoare ilustrează această idee care rezolvă problema numai în faza iniţială de încărcare a programelor.



Translatarea dinamică şi protecţia

În sistemele multiprogramate trebuie protejat fiecare program contra eventualelor greşeli ale altor programe, greşeli susceptibile de a periclita

execuţia corectă a unui program prin posibilitatea de a pătrunde în zona sa de memorie.

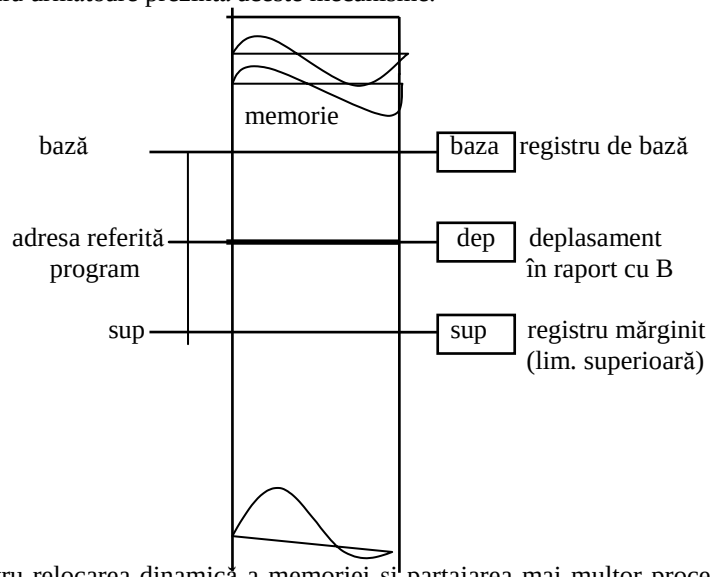
În principiu, este suficientă verificarea ca orice adresă calculată în timpul execuţiei unui program să fie internă intervalului de adrese alocate programului respectiv.

Această verificare se realizează cu ajutorul unui dispozitiv special care compară adresa efectiv calculată cu adresele extreme ale zonei alocate programului, stocate în registre **"**borne**"** sau mărginite (este necesar un singur registru mărginit, deoarece adresa primei locaţii de memorie este stocată în registrul de bază).

Pentru a găsi adresa referită se calculează adresa efectivă **AE = baza + dep** şi se testează dacă **baza ≤ AE ≤ sup**.

Numai SO are dreptul să modifice conţinutul registrelor de bază şi al celor mărginite, ceea ce implică instrucţiuni privilegiate care nu pot fi executate decât în mod supervizor, iar dacă un program face referire la o adresă din exteriorul bornelor sale, dispozitivul de verificare generează o întrerupere alertând SO.

Figura următoare prezintă aceste mecanisme.



Pentru relocarea dinamică a memoriei şi partajarea mai multor procese se utilizează dispozitivele următoare:

a) registru de bază;
b) registru mărginit;

c) dispozitiv de calcul al adresei efective pentru orice referinţă la memorie;
d) dispozitiv de verificare a apartenenţei adresei efective la zona de memorie a procesului.

Datorită acestor dispozitive, SO poate decide deplasarea unui program şi poate efectua din când în când o reaşezare a memoriei. Procesele care devin temporar inactive pot fi puse în aşteptare pe disc şi înlocuite în memorie de alte procese (**swapping**).

Principalele limite ale acestei tehnici sunt:

a) timp deloc neglijabil petrecut cu reaşezarea;
b) necesitatea alocării de spaţiu contiguu de memorie.

În evoluţia conceptuală a sistemelor de gestiune a memoriei, etapa următoare constă în a căuta metode de fragmentare a programelor şi memoriei, astfel încât un program să poată fi încărcat în zone necontigue de memorie.

Segmentarea

Segmentarea constă în divizarea unui program în module sau segmente, fiecare segment corespunzând unei entităţi logice cum ar fi o procedură sau un bloc de date, independentă de alte segmente.

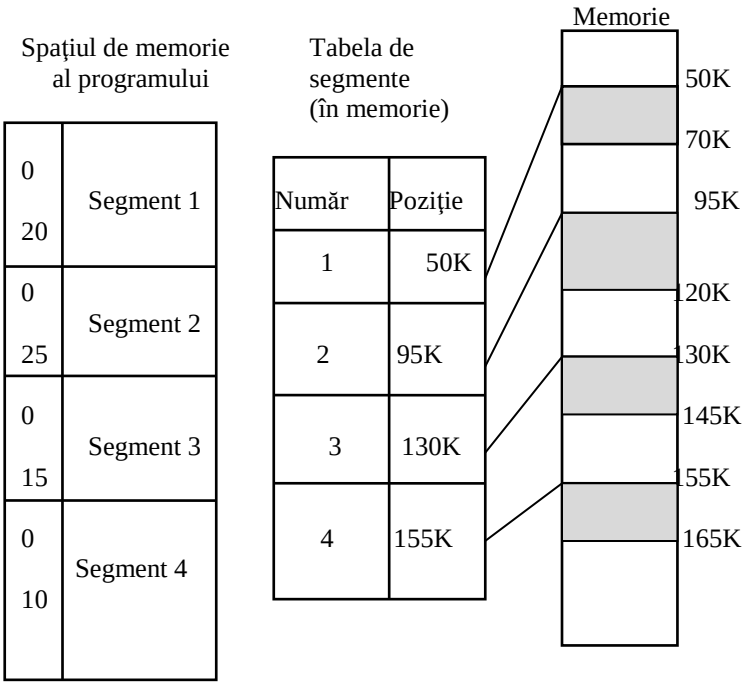
Sistemul de operare se ocupă de plasarea în memorie a segmentelor necesare execuţiei programelor gata să utilizeze CPU. Sistemul de operare trebuie să ştie unde sunt stocate diferitele segmente, şi pentru aceasta el organizează şi gestionează un ansamblu de **tabele de segmente**, câte o tabelă pentru fiecare program, conţinând adresele de încărcare ale segmentelor programului respectiv.

Adresa este structurată şi conţine 2 câmpuri:

a) numărul segmentului;
b) deplasamentul în cadrul segmentului.

Calculul adresei efective este realizat ca de obicei cu ajutorul unui dispozitiv special, adăugând offsetul adresei de încărcare a segmentului care este stocat în tabele de segmente.

Protecţia poate fi asigurată la nivelul tabelei de segmente adăugându-i talia fiecărui segment sau ultima adresă a segmentului.



Noţiunea de memorie virtuală

Ideea de memorie virtuală este simplă, elegantă şi constă în a trata diferenţiat adresele referite prin program (adrese virtuale) şi adresele memoriei fizice (adrese reale).

Mulţimea adreselor virtuale este independentă de implementarea programului în memoria fizică, este vorba deci de un spaţiu de memorie virtual (care nu există), dar poate fi utilizat de către programator ca model de memorie centrală la dispoziţia sa.

Totul se petrece ca şi când utilizatorul ar avea acces la o memorie având 2^n locaţii, *n* fiind numărul de biţi pentru adresele programului.

Acest număr *n* poate fi mult mai mare decât numărul *m* care reprezintă talia memoriei fizice. O tabelă de corespondenţă numită **tabelă de pagini** sau **topografie de memorie,** gestionată de către sistemul de operare pune în corespondenţă cele două spaţii de adrese, iar un dispozitiv special asigură transformarea unei adrese virtuale în adresă fizică.

Spaţiul virtual de adrese, care rezultă de regulă din linkeditarea programului este gestionat de sistemul de operare, care este obligat să

utilizeze memorii auxiliare (discuri, tambururi, memorii de sprijin etc), ca extensii ale memoriei centrale.

Conceptul de memorie virtuală dezvoltat în perioada anilor ‘60 a făcut obiectul mai multor proiecte de cercetare, iar cea mai mare parte a calculatoarele anilor ‘70 erau dotate cu memorie virtuală.

Realizarea conceptului de memorie virtuală se bazează pe tehnica **paginării**.

Paginare

Conceptul de paginare constă în a decupa cele două spaţii de adrese în pagini de aceeaşi talie (1024 sau 2048 cuvinte) şi de a evidenţia un mecanism de transfer de pagini între memoria virtuală şi cea reală.

În orice moment, un program gata de execuţie va avea plasate în memorie câteva copii ale paginilor sale virtuale, alese în mod corespunzător pentru a permite execuţiei sale să avanseze.

Sistemul de operare trebuie să rezolve câteva probleme importante:

a) dacă o anumită pagină se găseşte deja în memorie şi unde;
b) cum se convertesc adresele virtuale ale programului în adrese reale;
c) ce pagină va fi înlocuită pentru a face loc alteia;
d) cum se poate şti dacă pagina evacuată din memorie a fost modificată şi trebuie deci recopiată în memoria auxiliară.

Tabela paginilor

Tabela paginilor constituie mecanismul esenţial care face să corespundă fiecărei pagini virtuale o serie de informaţii, actualizate de către sistem. Un bit indicator specifică prezenţa (sau absenţa) paginii în memoria principală.

Dacă pagina este în memoria principală, numărul paginii reale este înscris în tabelă.

Un alt bit indică dacă pagina a fost modificată în timpul execuţiei.

Mecanismul de transformare a adreselor este următorul: câmpul de adresă este divizat în două părţi:

a) numărul de pagină, singurul element care se modifică;
b) poziţia în cadrul paginii.

Tabela paginilor conţine de asemenea **biţi de protecţie** specificând un anumit nivel al protecţiei (protejare în citire, scriere, execuţie etc.).

Tabela paginilor oferă de asemenea protecţia informaţiilor în sensul apartenenţei adreselor la domeniul alocat paginii respective.

Paginarea la cerere

Această metodă de gestiune a memoriei constă în încărcarea unei pagini din memoria virtuală numai dacă ea a fost referită.

Spaţiul de lucru

O alternativă a paginării la cerere constă în a plasa în memorie a unui mic număr de pagini oportun selecţionate, acest ansamblu de pagini numindu-se spaţiu de lucru.

Paginare şi segmentare

Anumite arhitecturi permit combinarea segmentării cu paginarea. Spaţiul adreselor virtuale este segmentat şi segmentele sunt decupate în pagini. Structura adresei trebuie să prevadă atunci trei câmpuri:

a) numărul segmentului ;
b) numărul paginii în cadrul segmentului;
c) deplasamentul în pagină.

Segmentarea fiind o divizare logică a spaţiului de adrese al programului, este sarcina programatorului să definească segmentele. Paginarea este realizată prin intermediul hardware-ului şi a sistemului de operare, ea este deci transparentă pentru utilizator.

Paginarea sistemului de operare

Sistemul de operare este un program chiar dacă el nu este chiar ca toate celelalte (instrucţiuni privilegiate, mod supervizor etc.), iar cea mai mare parte a sistemului de operare este supusă paginării, deoarece, datorită taliei sale el nu poate fi stocat în întregime în memoria principală. Fac excepţie de la această regulă nucleul şi o parte a programelor de gestiune a memoriei şi intrările/ieşirile.

Programele de serviciu sunt de asemenea paginate. Ca regulă generală, programele care se execută într-un sistem multiprogramat modern nu au decât câteva pagini în memoria centrală.

Tendinţele actuale sunt pe de o parte de a trata majoritatea funcţiunilor sistemului de operare ca procese utilizator şi pe de altă parte de a profita de microelectronică şi de a încredinţa hardware-ului sau microprogramării sarcinile frecvente.

Organizarea intrărilor/ieşirilor

Intrările/ieşirile constituie domeniul cel mai delicat în concepţia şi realizarea unui sistem de operare. Importanţa lor este fundamentală, dar orice generalizare devine dificilă prin varietatea unităţilor periferice şi a procedurilor de intrare/ieşire utilizate în sistemele de calcul actuale.

Dificultăţile se referă atât la aspectele materiale cât mai ales la necesitatea de a asigura o gestiune optimală a resurselor sistemului precum şi simplificarea sarcinii utilizatorilor acestuia.

Iată câteva aspecte ale problemelor care se ridică:

a) **diversitatea funcţională a unităţilor periferice**: imprimante laser, unităţi de discuri magnetice şi optice, cartuşe magnetice etc.
b) **diferenţa de viteză între unităţile periferice**: de la câteva caractere pe secundă ale tastaturii sau terminalului, la câteva milioane de caractere într-o secundă pentru o unitate de discuri;
c) **diversitatea de codificare şi structurare a informaţiilor transferate**: ASCII, EBCDIC, binar, cuvinte, octeţi, blocuri;
d) **diferenţa între metodele de acces ale unităţilor**: acces secvenţial sau aleator, adresaj complet diferit pentru unităţile de bandă, discuri sau mari memorii de arhivare;
e) **diferenţa între condiţiile de partajare a perifericelor**: o unitate de discuri sau ecranul terminalului pot fi utilizate în acelaşi timp de către mai multe procese, în timp ce o imprimantă nu poate fi partajată decât în timp şi numai pentru job-uri întregi;
f) **marea diversitate a condiţiilor de eroare**: eroare de paritate, imprimantă fără hârtie, eroare de poziţionare a unui cap de citire/scriere pe disc, unitate deconectată, eroare de adresare etc.;
g) **complexitatea sistemelor de legătură între periferice şi unitatea centrală**: DMA, canale, bus, unităţi de comandă, unităţi de telecomunicaţii etc.;
h) **gradul înalt de paralelism al operaţiilor**;
i) **necesitatea protejării utilizatorilor: "**ascunderea**"** anumitor detalii legate de operaţiile de intrare/ieşire**;**
j) necesitatea de a asigura **independenţa programelor** faţă de tipul perifericelor utilizate**.**

Pentru rezolvarea acestor probleme, sistemul de operare încearcă să trateze toate perifericele de o manieră uniformă (periferice virtuale, codificare internă a caracterelor etc.), încredinţând toate prelucrările particulare modulelor specializate, numite **gestionari de unităţi periferce** [device handlers, drivers].

Ca structură internă, acest nivel al sistemului de operare care se ocupă cu gestiunea intrărilor/ieşirilor, se poate detalia pe subnivele, astfel:

a) **proceduri standard** (programe de bibliotecă) utilizate de către aplicaţii şi conţinând cereri către supervizor;
b) un **software** de intrare/ieşire independent de unităţile periferice;
c) **drivere**, comandând fiecare unitate în detaliu;
d) **programe de serviciu ale întreruperilor**, acţionând în colaborare cu funcţiunile nucleului.

Utilizatorul dispune de un set de instrucţiuni de intrare/ieşire virtuale, spre deosebire de instrucţiunile de intrare/ieşire în cod maşină, specifice sistemului de operare.

Instrucţiunile de intrare/ieşire virtuale se prezintă într-un limbaj de programare evoluat sub forma unui apel de procedură sistem (**read**, **write**, **print** etc), cu argumentele necesare de apel (nume variabile sau fişier, talie, format, unitate logică etc.).

Aceste instrucţiuni sun înlocuite prin procedurile sistem corespunzătoare în timpul fazei de compilare.

Există de asemenea probleme de protecţie la nivelul intrărilor/ieşirilor. În acest context, sistemul de operare verifică corectitudinea convertirii adreselor virtuale în adrese reale în momentul iniţializării, împiedicând modificarea acestora înaintea terminării operaţiei de intrare/ieşire.

Fişiere şi gestiunea fişierelor. SGF.

Gestiunea fişierelor este serviciul cel mai vizibil oferit utilizatorului de către un sistemul de operare. Cea mai interesantă situaţie apare atunci când memorarea se face pe suport magnetic, cu accent pe suportul disc, cel mai convenabil tip de suport magnetic.

Gestiunea fişierelor privită de utilizator

Un **volum** poate fi o rolă de bandă, un ansamblu formând un disc de masă, un disc cartuş, un disc flexibil (discheta).

În cadrul unui volum, informaţiile sunt grupate de către SO la indicaţia utilizatorului în ansambluri distincte numite **fişiere**.

Utilizatorul are acces prin programele sale la o (mică) entitate din cadrul unui fişier, cunoscută sub numele de **articol**. Putem spune că un articol este o subdiviziune a unui fişier care are un înţeles de sine stătător.

Informaţiile dintr-un articol sunt de regulă grupate în subdiviziuni numite **câmpuri** sau **atribute**. Fiecare câmp sau atribut are în fişier o anumită **valoare**. O pereche (atribut, valoare) o vom numi **cheie**. O cheie unică se numeşte **index de articol**.

Lungimea de reprezentare a unui articol poate fi constantă pentru toate articolele sau poate să varieze de la un articol la altul, deci putem avea articole de format fix sau variabil.

Sistemul de gestiune a fişierelor (SGF) este un ansamblu de rutine de legătură între utilizatori şi componenta sistemului de intrare/ieşire la nivel fizic pentru operarea cu fişiere. Utilizatorul dipune de o serie de operaţii primitive pe care le poate solicita SGF-ului pentru serviciile dorite.

Conceptul de fişier abstract

Se poate defini un fişier ca o funcţie f: $\mathbf{N} \rightarrow \mathbf{T}$, unde **N** este mulţimea numerelor naturale iar **T** mulţimea valorilor posibile pentru un tip de dată

deja definit. Pentru tipurile de date obişnuite (**integer**, **char**, **real**) avem de-a face cu un fişier obişnuit.

Prin f (i) notăm mulţimea valorilor câmpurilor articolului al i-lea din fişier.

Dacă tipul de dată este la rândul său un tip fişier, avem de-a face cu o bază de date.

Principalele **operaţii** asupra unui fişier notat f, presupunând că înaintea efectuării operaţiei, fişierul are *n* articole, sunt următoarele:

a) **Citirea** [Read] articolului k înseamnă obţinerea valorii f (k);

b) **Scrierea** [Write] înseamnă adăugarea unui nou articol la fişier. Deci fişierul f cu n articole se transformă în fişierul f' cu n +1 articole definite astfel:

$$f'(i) = f(i), \forall i \in \{1, 2, ..., n\};$$
$$f'(n + 1) = x, \text{ unde x este valoarea articolului adăugat.}$$

c) **Inserarea** [Insert] unui nou articol cu valoarea x, după articolul cu numărul de ordine k înseamnă obţinerea unui nou fişier f' cu n+1 articole:

$$f'(i) = f(i), \forall i \in \{1, 2, ..., k\};$$
$$f'(k + 1) = x;$$
$$f'(i + 1) = f(i), \forall i \in \{k+1, ..., n\}.$$

Se observă că **scrierea** este echivalentă cu inserarea la sfârşitul fişierului iar inserarea după poziţia 0 înseamnă adăugare la începutul fişierului.

d) **}tergerea** [Delete] articolului k înseamnă obţinerea unui nou fişier f':

$$f'(i) = f(i), \forall i \in \{1, 2, ..., k - 1\};$$
$$f'(i - 1) = f(i), \forall i \in \{k+1, ..., n\}.$$

e) **Modificarea** [Modify, Update, Replace] articolului k cu un altul de valoare x înseamnă obţinerea fişierului f' cu n articole, astfel:

$$f'(i) = f(i), \forall i \in \{1, 2, ..., k - 1\};$$
$$f'(k) = x;$$
$$f'(i) = f(i), \forall i \in \{k+1, ..., n\}.$$

Tipuri de acces la articole

a) **Accesul secvenţial** la un articol f (i) presupune i - 1 accese în ordine, la articolele cu numerele de ordine 1, 2, ..., i - 1.

b) **Accesul direct** [random access] presupune existenţa unui mecanism de obţinere a articolului căutat fără a parcurge secvenţial toate articolele care-l preced. Mecanismul de acces este de două feluri:

- acesul direct prin număr de poziţie (adresă) are loc atunci când i se furnizează SGF-ului o valoare i şi acesta returnează f (i) (acces relativ);
- prin conţinut, are loc atunci când i se furnizează SGF-ului o cheie (a, v) şi acesta returnează acel articol i pentru care f (i). a = v**.**

Clasificarea fişierelor

a) După lungimea unui aricol:
   - fişiere cu articole de **format fix;**
   - fişiere cu articole de **format variabil**.

b) După posibilitatea de afişare sau tipărire:
   - **fişiere text** al căror conţinut poate fi afişat pe ecran sau la imprimantă;
   - **fişiere binare**, formate din şiruri de octeţi consecutivi, fără nici-o semnificaţie pentru afişare.

c) După suportul pe care este rezident fişierul
   - fişiere pe disc magnetic;
   - fişiere pe bandă magnetică;
   - fişiere pe imprimantă;
   - fişiere tastatură;
   - fişiere pe ecran;
   - fişiere pe plotter, digitizor;
   - fişiere pe cartele perforate;
   - fişiere pe bandă de hârtie perforată.

d) După modurile de acces
   - fişiere secvenţiale;
   - fişiere în acces direct:
     `i)` acces direct prin adresă;
     `ii)` acces direct prin conţinut:
       - secvenţial-indexat;
       - selectiv;
       - multilistă;
       - B - arbore;

Referirea unui fişier

Utilizatorul se referă la un fişier în conformitate cu anumite reguli sintactice impuse de către SO. În general, referirea la un fişier se face

printr-un şir de caractere, diferite de spaţiu. }irul de referinţă conţine 5 zone, plasate de la stânga la dreapta, astfel:

Periferic	:	Cale	Nume	.	Tip	;	Versiune

Periferic indică numele perifericului suport al fişierului, impus de sistemul de operare.

Cale desemnează subdirectorul din cadrul discului unde se află fişierul căutat.

Nume este o succesiune de litere şi cifre date de creatorul fişierului.

Tip sau **extensie** a numelui este dată tot de utilizator şi ea reflectă conţinutul fişierului.

Versiune este o noţiune specifică unui anumit SO şi diferenţiază mai multe versiuni ale aceluiaşi fişier.

Acţiunile SGF la nivel de articol

a) **citire** pentru suporturile disc, bandă magnetică, cartele, tastatură;
b) **scriere** pentru suporturile disc, bandă magnetică, ecran, imprimantă, plotter;
c) **inserare, ştergere, modificare**, pentru suporturile de tip disc.

Acţiunile SGF la nivel de fişier

a) funcţiile de deschidere şi închidere (open, close). **Open** este operaţia de deschidere şi conţine acţiunile SGF efectuate înaintea primului acces la fişier, iar **close** (închidere) după ultimul acces.
b) alte operaţii globale:
   - crearea unui fişier;
   - ştergerea unui fişier;
   - copierea unui fişier în alt fişier;
   - schimbarea numelui unui fişier (redenumire, mutare);
   - listarea unui fişier;
   - concatenarea mai multor fişiere;
   - compararea a două fişiere;
   - fuzionarea a două sau mai multe fişiere;
   - sortarea (ordonarea) articolelor unui fişier;
   - operaţii de filtrare a unui fişier.

Descriptorul de fişier este un articol special care conţine informaţiile de descriere ale unui fişier. Locul său de memorare este în directorul fişierului. Apar patru grupe de informaţii:

a) identificarea fişierului **(**N, I), unde N este numele simbolic al fişierului (specificatorul de fişier fără periferic), iar I este un număr prin care descriptorul este reperat pe disc) în mod direct;
b) adresele fizice ocupate de fişier;
c) controlul accesului la fişier;
d) informaţii de organizare calendaristice.

Pentru controlul accesului la fişiere se foloseşte **matricea de control** unde a (i, j)=1 dacă utilizatorul *i* are drept de acces la fişierul *j*, şi 0 în rest.

Clase de utilizatori:

a) **proprietar**: cel care crează fişierul şi stabileşte drepturi de acces la fişier;
b) **utilizatori specificaţi**: nominalizaţi de proprietar, cu statut special în raport cu fişierul;
c) **grup sau proiect**: care pot folosi împreună un fişier;
d) **public**: restul utilizatorilor.

Moduri de organizare a fişierelor

a) organizare secvenţială;
b) fişiere în acces direct prin **poziţie**;
c) fişiere în acces direct prin **conţinut**;
d) **fişiere inverse:** pe lângă fişierul de bază se crează automat un fişier invers care conţine pentru fiecare cheie specificată, adresele disc la care se află articolele care conţin cheia respectivă. Dacă $n_1, n_2, ..., n_p$ sunt numerele de articole care conţin cheile 1, 2, ..., p, iar a_{ij} adresa articolului j cu cheia i, atunci, dacă $j \neq k$, $a_{ij} \neq a_{ik}$ $\forall i$.
e) **fişiere multilistă**: se definesc atribute şi valori cheie. SGF ataşează fiecărei chei un pointer către articolul următor care conţine aceeaşi cheie. Există atâtea liste câte chei sunt. Fiecare articol participă la c liste, unde c este numărul de scrieri ale articolului, adică numărul de apariţii ale valorii unei chei. Se crează o zonă cu cheile fişierului şi adresele primelor articole (fişierul director). Avem că :
   - dacă $j \neq k$ atunci $a_{ij} \neq a_{ik}$ $\forall i$;
   - dacă $i \neq k$ atunci $a_{ij} \neq a_{kj}$, care apare de c ori.
f) **fişiere secvenţial-indexate:** articolele sunt scrise pe suport în acces secvenţial şi plasate în ordinea crescătoare a valorilor indexului, rezultă că se crează o tabelă de indecşi care are o **parte principală** şi o **parte de depăşire.** Există trei nivele pentru tabelele de indecşi:
   - nivelul 3: **fişier**: numărul intrărilor este egal cu cel al volumelor;

- nivelul 2: **volum**: numărul intrărilor este egal cu numărul cilindrilor din cadrul volumului;
- nivelul 1: **cilindru**, conţine atâtea intrări câte pagini sunt în cilindrul respectiv.

g) **fişiere selective:** au funcţia de regăsire materializată printr-un calcul al CPU, numită **funcţie de randomizare** *f : {mulţimea valorilor posibile pentru index}→{0, 1, ..., n - 1}* Articolele care au aceeaşi valoare pentru funcţie se numesc **sinonime**.

h) **fişiere organizate** în **B - arbori regulari.** Fiecare nod este prevăzut cu *m* căsuţe (pentru m chei unice). În fiecare nod pot fi maxim *m* şi minim *m div 2* chei. Înainte de prima cheie, între 2 chei şi după ultima cheie sunt pointeri spre noduri subordonate.

Acţiunile SGF la nivel de suport disc

SGF are trei sarcini principale:

a) sistem de regăsire a fişierelor pe disc;
b) evidenţa spaţiului neutilizat pe disc;
c) utilizarea acestui spaţiu pentru crearea sau extinderea fişierelor.

Sisteme de cataloage (directoare)

Sunt tabele care conţin fişierele existente şi informaţii necesare accesului la aceste fişiere. Fiecare intrare în director conţine un descriptor de fişier. Principalele operaţii asupra cataloagelor sunt următoarele: căutare, inserare, ştergere, listare, salvare/restaurare etc.

Integritatea informaţiilor

Deoarece utilizatorul depinde de sistemul de fişiere pentru tot ceea ce vizează aspecte legate de lucrul cu datele sale, este esenţial ca sistemul să fie dotat cu mecanisme de salvare a informaţiilor permiţând eventual reconstituirea fişierelor pierdute într-un accident software sau hardware. Printre metodele comune utilizate în acest sens, sunt următoarele:

a) **salvare completă** [backup], care permite recopierea pe bandă magnetică a fişierelor disc, o dată la două sau trei zile. În caz de accident, se pot reconstitui fişierele care existau în momentul ultimului backup;

b) **salvare incrementală**, care permite recopierea doar a informaţiilor modificate după ultimul backup. Această abordare permite reducerea frecvenţei backup-urilor masive (de exemplu, o dată pe săptămână sau pe lună). În cazul reconstrucţiei fişierelor, procedurile de lucru sunt mai complexe;

c) **dublarea sistematică a fişierelor disc**, care constă în a păstra întotdeauna două copii ale fiecărui fişier pe două unităţi de disc diferite. Această metodă necesită mai puţine intervenţii din partea operatorilor, dar implică dublarea spaţiului disc disponibil.

Servere de fişiere

O abordare modernă a gestiunii fişierelor, foarte la modă datorită avântului informaticii distribuite, constă în a încredinţa unui calculator independent întreaga gestiune a fişierelor specifice unei comunităţi de utilizatori conectaţi la o reţea locală. Un astfel de calculator se numeşte **server de fişiere**. În această arhitectură, partea majoră a spaţiului disc este concentrată în jurul serverului, celelalte calculatoare din reţea nu au nevoie de capacitate disc locală pentru fişierele lor.

O altă realizare recentă, datorată integrării tot mai pronunţate a calculatoarelor şi a reţelelor este aceea a **dispersate** într-o reţea. Utilizatorul unui astfel de sistem nu ştie şi nu are nevoie să ştie în care calculator sunt stocate fişierele sale. În momentul în care o cerere de acces este adresată sistemului de fişiere, acesta determină poziţia fişierului căutat şi, utilizând serviciile reţelei, pune la dispoziţia solicitantului o copie a fişierului respectiv.

Alocarea resurselor

Într-un mod general, definim o resursă ca fiind un element necesar unui procesor pentru a asigura execuţia sa în bune condiţiuni.

Resursele materiale ale unui sistem de calcul (CPU, memorii, dispozitive de intrare/ieşire etc.), sunt disponibile în cantitate limitată şi trebuie să fie partajate între diferite procese.

Produsele software şi fişierele, dacă pot fi partajate, fac de asemenea parte dintre resursele pe care sistemul trebuie să le gestioneze.

Mecanismele de alocare ale unei resurse particulare sunt realizate pe diferitele nivele ale sistemului de operare. De exemplu, dispecerul decide dacă alocarea CPU şi alocarea unui fişier al unui proces este implementată la nivelul gestiunii fişierelor.

Strategia de repartizare şi de alocare a resurselor trebuie să fie determinată global, pentru tot sistemul. La nivelul sistemului, se iau decizii în legătură cu planificarea globală a activităţii, se decide, de exemplu, crearea de noi procese, sau aşteptarea, pe baza resurselor disponibile, nivelul de prioritate al unui job.

Obiectivele acestui nivel al sistemului pot fi rezumate astfel:

a) **asigurarea unei bune utilizări a resurselor**: contabilizarea şi furnizarea de statistici asupra resurselor principale;

b) **crearea de noi procese şi atribuirea unui nivel de prioritate corespunzător**: se permite fiecărui proces existent în sistem să obţină resursele necesare în limite de timp rezonabile;

c) **excluderea mutuală a proceselor care solicită aceeaşi resursă nepartajabilă şi şi evitarea situaţiilor de blocare** (aşteptare fără sfârşit a unei resurse de către mai multe procese).

Procesul sistem care se ocupă de toate aceste probleme se numeşte **planificator** [scheduler]. Planificatorul determină ordinea de execuţie a job-urilor lansate de utilizatori, el alege momentul pentru lansarea unei execuţii şi refuză accesul unui utilizator interactiv dacă numărul de utilizatori conectaţi poate conduce la o degradare inacceptabilă a timpului de răspuns. Scopul planificatorului este de a asigura o exploatare echilibrată şi în consecinţă un serviciu satisfăcător pentru toţi utilizatorii.

Interfaţa utilizator-sistem

Interfaţa între utilizator şi sistemul de operare se efectuează prin intermediul unui limbaj, numit **limbaj de comandă.** Natura acestui limbaj depinde de sistemul considerat.

Sistemele batch sunt dotate cu un limbaj relativ suplu şi puternic, permiţând utilizatorului să specifice în avans succesiunea prelucrărilor de realizat, ţinând cont de toate alternativele posibile.

Sistemele interactive oferă interfeţe mai simple prin care utilizatorul poate urmări derularea job-ului său şi decide succesiunea operaţiilor pe măsură ce se prezintă situaţiile posibile.

În majoritatea sistemelor actuale, modurile de lucru batch şi multiacces coexistă, iar limbajul de comandă este adaptat în mod corespunzător.

Tendinţa actuală este de a simplifica sarcina utilizatorului, propunându-i un repertoar de comenzi uşor de utilizat. Aceste comenzi se exprimă sub forma cuvintelor cheie (**Login**, **Logout**, **Edit**, **Fortran**, **Run**, **File**, **Copy**, **Help** etc.), urmate de anumiţi parametri. Este de asemenea normală procedura de lucru prin care se realizează prescurtarea comenzilor (de exemplu, **fl** în loc de **file list**) sau regruparea acestora în fişiere executabile, un fel de macro-comenzi, de exemplu, se poate înlocui secvenţa **Compile**, **Link**, **Load**, **Run** prin procedura **Execute**.

Directivele pe care utilizatorul le furnizează sistemului cu ajutorul limbajului de comandă sunt interpretate de către **interpretorul de comenzi** [command interpreter]. Acesta citeşte comenzile provenind de la terminal şi după interpretarea acestora realizează serviciile solicitate. Datorită dialogului, sistemul trebuie să semnaleze faptul că el **"**ascultă**"** şi este gata să primească instrucţiunile utilizatorului, răspunzând comenzilor şi

comunicând disponibilitatea sa prin afişarea pe ecran a unui caracter special [prompt], invitând utilizatorul de a formula noi cereri.

Comenzile trimise către sistem sunt o formă de cereri la supervizor [sistem calls], cu deosebirea că, în loc să provină dintr-un program sau dintr-o procedură de bibliotecă, aceste comenzi sunt comunicate direct sistemului de către utilizator.

Majoritatea limbajelor de comandă reflectă structura internă a sistemului de operare, neputând fi schimbate decât cu mare greutate. Totuşi, în cazul sistemului Unix, interpretorul de comenzi, numit **shell**, poate fi modificat sau chiar înlocuit de către utilizator, care poate să comunice astfel cu sistemul într-un limbaj convenabil ales.

Shell-ul interpretează comenzile provenind de la un terminal sau de la un fişier [shell script] şi posedă structuri de control puternice, permiţând execuţia condiţionată sau repetată a unei succesiuni de comenzi.

Cu abordarea shell, este uşor de a combina proceduri existente şi diverse elemente de programare; adesea se utilizează shell-ul pentru a se evita scrierea de noi programe.

Interfaţa utilizator grafică

Până la debutul anilor ‘80, toate interfeţele utilizator erau bazate pe limbaje de comandă ca shell-ul sistemului Unix. Pentru fiecare acţiune de efectuat, utilizatorul trebuia să cunoască şi să tasteze numele comenzii.

În laboratoarele Xerox Park, a fost elaborat un nou tip de interfaţă, şi anume interfaţa grafică [GUI: Graphical User Interface], bazată pe utilizarea unui ecran grafic în locul unui ecran alfanumeric.

Au apărut noi concepte, principalele fiind acelea de ferestre, pictograme, meniuri care defilează, mouse etc.

Acest tip de interfaţă a devenit la modă prin intermediul familiei Macintosh elaborată de firma Apple.

La ora actuală, majoritatea sistemelor de calcul utilizează interfeţe utilizator grafice.

Introducerea grafismului în interfeţele utilizator a revoluţionat lumea informaticii, în principal prin aceea că permite publicului larg să utilizeze calculatoarele prin imagine, fără a cunoaşte un jargon specific.

După revoluţia imaginii se estimează că va veni revoluţia sunetului, recunoaşterea vocală din cadrul interfeţelor om-maşină.