

8 Sisteme de operare – prezentare generală

8.1 Tipuri de sisteme de operare (SO); clasificări

În prezent există un mare număr de SO în funcțiune și numărul acestora este în continuă creștere. Nu există un criteriu unitar de comparare a acestora. Noi vom da mai multe clasificări, în funcție de următoarele criterii:

- după gradul de partajabilitate a resurselor;
- după tipurile de interacțiuni permise;
- după organizarea internă a programelor ce compun SO.

8.1.1 Clasificare după gradul de partajabilitate a resurselor

După acest criteriu, distingem trei categorii de sisteme de operare [10]:

- Sisteme monouser și monotasking;
- Sisteme monouser și multitasking;
- Sisteme multiuser.

Sisteme de operare monouser sunt acelea care permit, la un moment dat, unui singur utilizator să folosească sistemul. Un sistem este monotasking dacă admite la un moment dat execuția unui singur program.

În forma cea mai simplă, un sistem monouser și monotasking execută un singur program la un moment dat și acesta rămâne activ din momentul lansării lui și până la terminarea lui completă. Un astfel de sistem este de exemplu cel care deservește un telefon mobil sau un terminal PDA (Personal Digital Assistant).

Un sistem monouser și multitasking este acela în care un singur utilizator este conectat la sistem, dar el poate să își lanseze simultan mai multe procese în lucru. Tipice în acest sens sunt SO din familia Windows 9X, ca și cele din familia NT care nu sunt servere. Tot în această categorie putem îngloba sistemele Linux instalate pe calculatoare neconectate în rețea. Pentru sistemele de acest fel care au memoria internă relativ mică, se aplică așa-numita tehnică swapping, adică evacuarea temporară a unui program, din memoria internă pe discul magnetic. În timpul evacuării, în memoria internă este încărcat un alt program, care la rândul lui este executat parțial și este apoi evacuat și el. Cât timp un program este în memorie, el are acces la toate resursele sistemului. Programele sunt astfel executate pe porțiuni, momentele de execuție alternând cu cele de evacuare. După unii autori ([4]), SO de acest tip se află la granița dintre sistemele monoutilizator și cele multiutilizator.

Sisteme de operare multiutilizator permit accesul simultan al mai multor utilizatori la sistem. Ele trebuie să aibă în vedere partajarea procesorului, a memoriei, a timpului, a perifericelor și a altor tipuri de resurse, între utilizatorii activi la un moment dat. Vom reveni cu detalii asupra metodelor de partajare a procesorului, timpului și a memoriei. La astfel de sisteme sunt necesare tehnici mai sofisticate, de gestiune și protecție a utilizatorilor. La un moment dat aici există mai multe procese active care se execută concurent sub controlul SO.

8.1.2 Clasificare după tipurile de interacțiuni permise

În funcție de resursele pe care le are un sistem și în funcție de destinația acestuia, este necesară stabilirea unor strategii de interacțiune cu utilizatorul. Conform acestui criteriu de diferențiere, distingem [34]:

- Sisteme seriale (care prelucrează loturi de lucrări);
- Sisteme cu timp partajat (time – sharing);
- Calculatoare personale și stații de lucru (workstations);
- Sisteme autonome (embedded systems);
- Sisteme portabile, destinate comunicațiilor (telefoane mobile, PDA-uri etc.);
- Sisteme conectate în rețea.

Sisteme seriale. În cadrul acestor sisteme lucrările se execută pe loturi pregătite în prealabil. Practic, din momentul predării unei lucrări la ghișeul centrului de calcul și până la eliberarea ei, utilizatorul nu poate interveni spre a influența execuția programului său. SO afectate acestor sisteme pot lucra atât monoutilizator, cât și multiutilizator. Din punct de vedere istoric sunt printre primele sisteme. În prezent, în această categorie intră supercalculatoarele, precum și o serie de sisteme medii – mari cum ar fi AS-400. Sistemul de operare trebuie să gestioneze loturi de lucrări pe care să le pregătească spre a fi introduse în sistem și eventual să pregătească livrarea rezultatelor (listare etc.). În general gradul de concurență la acest tip de sisteme este relativ redus. În [10] sunt date mai multe detalii despre acest tip de sisteme.

Sistemele cu timp partajat trebuie să suporte, într-o manieră interactivă, mai mulți utilizatori. Fiecare utilizator este în contact nemijlocit cu programul său. În funcție de unele rezultate intermediare, el poate decide modul de continuare a activității programului său. SO trebuie să gestioneze printre altele și terminalele de teletransmisie la capătul cărora se află utilizatori care-i pot transmite diverse comenzi. Sistemul trebuie să dispună de mecanisme mai sofisticate decât cele seriale pentru partajarea procesorului, a memoriei și a timpului. De regulă, acest tip de sisteme practică partajarea timpului, astfel. Se fixează o cuantă de timp. Pe durata cuantei, un singur proces are acces la procesor (și la celelalte resurse aferente). După epuizarea cuantei, procesul este suspendat, pus la sfârșitul cozii de procese și un alt proces ocupă procesorul pentru următoarea cuantă ș.a.m.d. Dimensiunea cuantei și strategiile de comutare sunt astfel alese încât să se reducă timpii de așteptare și să se realizeze o servire rezonabilă a proceselor. Tipice acestui tip de sisteme sunt sistemele Unix, dar și Windows din familia NT / server.

Calculatoare personale și stații de lucru. La acest tip de sisteme, toate resursele mașinii sunt dedicate utilizatorului care este conectat. Așa că nu se pune problema partajării resurselor între utilizatori, ci doar a datelor între procesele pe care userul le lansează simultan. Practic, sarcina principală a SO este aceea de a oferi userului o interfață prietenoasă de exploatare a sistemului.

Sistemele autonome sunt sisteme dedicate unui anumit proces industrial, cum ar fi: funcționarea unui robot, coordonarea activităților dintr-un satelit geostaționar, supravegherea unui baraj de apă, a unei stații de radiolocație etc. Fiind conectate la diverse procese tehnologice, aceste sisteme trebuie să fie capabile să deservească în timp prestabilit fiecare serviciu care i se cere. Dacă sistemul nu este capabil să dea un răspuns, este posibilă oprirea procesului supravegheat. Astfel de sisteme sunt în prezent în plină dezvoltare, mai ales din cauza dezvoltării rapide a tehnologiilor multimedia.

Sisteme portabile, destinate comunicațiilor. Acestea formează cea mai nouă categorie de sisteme. Exemplele tipice sunt telefoane mobile și PDA-urile. Sunt mașini portabile, de dimensiuni mici și au facilități puternice de comunicare. Facilitatea lor principală este conexiunea wireless printr-una dintre tehnologiile existente: radio, infraroșu, Bluetooth etc. Evident, sistemul de operare aferent trebuie să fie capabil să gestioneze eficient aceste comunicații. Din cauza formatului mic, memoria internă, dar mai ales memoria pe suport extern este foarte limitată. Sistemul de operare trebuie să facă față acestei penurii de resurse. Securizarea accesului la un astfel de sistem este esențială. Fiind echipamente mici pot fi ușor pierdute / furate, iar această securizare trebuie să le facă de nefolosit de către persoanele neautorizate.

Sisteme conectate în rețea. Practic, astăzi marea majoritate a sistemelor nu mai sunt independente, ci sunt conectate la rețele de calculatoare, inclusiv la rețeaua publică Internet. Din această cauză, una dintre sarcinile fundamentale ale sistemelor de operare din această categorie trebuie să fie gestiunea accesului la rețea. De asemenea, accesul din exterior la resursele locale trebuie să fie bine protejat, spre a interzice accesese neautorizate.

În legătură cu acest criteriu de clasificare, trebuie arătat că el este relativ. Astfel, există sisteme seriale care permit interacțiunea cu programele, sistemele interactive au facilități puternice de lucru serial etc. Mai mult, același SO permite în același timp să lucreze, spre exemplu, în timp real, dar în același timp să servească, evident cu o prioritate mai mică, și alte programe de tip serial sau interactiv, în regim de multiprogramare clasică.

8.1.3 Clasificare după organizarea internă a programelor ce compun SO.

Să vedem cum arată un SO văzut din "interior". Vom vedea patru tipuri de structuri, în evoluția lor istorică.

- Sisteme monolitice;
- Sisteme cu nucleu de interfață hardware;
- Sisteme cu structură stratificată;
- Sisteme organizate ca mașini virtuale

Sisteme monolitice. Un astfel de SO este o colecție de proceduri, fiecare dintre acestea putând fi apelată după necesități. Execuția unei astfel de proceduri nu poate fi întreruptă, ea trebuie să-și execute complet sarcina pentru care a fost apelată. Un program utilizator rulat sub un sistem monolitic se comportă ca o procedură apelată de SO. La rândul lui, programul poate apela în maniera apelurilor sistem (vezi 4.5 pentru Unix) diverse rutine din SO. Singura structură posibilă aici este legată de cele două moduri de lucru: nucleu și user (vezi pentru Unix 4.5 și pentru Windows 7.1.3). Esențial pentru aceste tipuri de sisteme este legătura de tip apel-revenire, fără posibilitatea de întrerupere a fluxului normal al execuției. În prezent, aceste tipuri de SO sunt pe cale de dispariție.

Sisteme cu nucleu de interfață hardware. Un astfel de sistem concentrează sarcinile vitale de nivelul cel mai apropiat de hardware într-o colecție de rutine care formează nucleul SO. În acesta sunt înglobate inclusiv rutinele de întrerupere. Componentele nucleului se pot executa concurrent. Toate acțiunile utilizatorului asupra echipamentului hard trec prin acest nucleu. Unele SO mai plasează între nucleu și utilizator încă o interfață. Exemple de nuclee ale SO pot fi considerate componentele BIOS ale mașinilor PC deși fac parte din hardware.

Sisteme cu structură stratificată este o generalizare a organizării cu nucleu. Un astfel de sistem este construit nivel după nivel, componentele fiecărui nivel folosind toate serviciile oferite de nivelul inferior. În prezent aceste sisteme sunt cele mai răspândite. "Părinții" lor sunt SO THE (Dijkstra, 1968) și SO MULTICS. Ultimul dintre ele oferă o organizare mai interesantă, și unică în felul ei, aceea a inelelor de protecție. Pentru detalii se pot consulta [48], [49]. În secțiunea următoare vom ilustra structura stratificată a unui SO ipotetic.

Sisteme organizate ca mașini virtuale. Echipamentul hard al acestor tipuri de sisteme servește, în regim de multiprogramare, eventual în time-sharing, un număr de procese. Fiecare proces dispune, în mod exclusiv, de o serie de resurse, cea mai importantă fiind memoria. Fiecare dintre procesele deservite este un sistem de operare, care are la dispoziție toate resursele alocate procesului respectiv de către echipamentul hard. În acest mod, pe același echipament hard se poate lucra simultan sub mai multe SO. Primele sisteme de acest tip au fost VM/370 (Virtual Machines for IBM-370), care coordonează mai multe SO conversaționale monoutilizator de tip CMS (Conversational Monitor System). Fiecare utilizator lucrează sub propriul CMS, pentru el fiind transparent faptul că lucrează sub CMS singur, pe un calculator mic, sau că este legat, împreună cu alții la același echipament hard. Sistemele actuale Windows oferă mașini virtuale (ferestre) pentru lucru sub DOS. Implementările Linux, FreeBSD și Solaris actuale oferă mașina virtuală Wine care emulează platforme Windows. Pentru detalii, se pot consulta [4], [49].

8.2 Structura și funcțiile unui sistem de operare

8.2.1 Stările unui proces și fazele unui program

8.2.1.1 Stările unui proces

În secțiunile precedente am folosit destul de des termenul de proces, privit intuitiv ca un program în execuție. Am văzut în capitolele precedente noțiunea de proces sub Unix și sub Windows. În fig. 5.18 din 5.2.3 sunt prezentate stările unui proces Unix. Există [62] și stări în care se poate afla un proces Windows.

Sintetizând stările proceselor sub diverse sisteme de operare, putem defini stările unui proces într-un sistem de operare. Aceste stări sunt:

- HOLD – proces pregătit pentru intrarea în sistem;
- READY – procesul este în memorie și este gata de a fi servit de (un) procesor;
- RUN – un procesor execută efectiv instrucțiuni (mașină) ale procesului;
- WAIT – procesul așteaptă terminarea unei operații de intrare / ieșire;
- SWAP – imaginea procesului este evacuată temporar pe disc;
- FINISH – procesul s-a terminat, trebuie livrate doar rezultatele.

HOLD este starea unui proces la un sistem serial în care procesului îi sunt citite datele de lansare din cadrul lotului din care face parte, se face o analiză preliminară a corectitudinii lor și apoi procesul este depus într-o coadă pe disc (numită HOLD) spre a fi preluat spre prelucrare. De obicei, aceste sisteme au o componentă specializată pentru astfel de prelucrări – SPOOL-ing de intrare (Simultaneous Peripheral Operations OnLine – vezi [10]).

FINISH este, de asemenea, stare a unui proces la un sistem serial. Procesul se execută și rezultatele lui sunt plasate pe disc într-o coadă FINISH. După terminare, acestea vor fi listate,

fie pe o imprimantă locală, fie pe una aflată la distanță. Componenta din SO care face preia din coada FINISH și listează se numește SPOOL-ing de ieșire (vezi [10]).

READY este starea în care un proces se află în memoria internă dar nu este servit de procesor. Este posibil ca el să fi fost servit parțial, dar pe moment să fie suspendat în defavoarea altui proces, urmând să i se continue execuția mai târziu.

SWAP este starea în care un proces este evacuat temporar într-un spațiu rezervat pe disc (SWAP) pentru a face temporar loc altui proces în memoria internă. Ulterior procesul va fi reîncărcat în memoria internă și i se va continua execuția.

WAIT este starea în care un proces a cerut executarea unei operații de intrare / ieșire. Cât timp procesul așteaptă să se termine operația, cedă altor procese procesorul.

RUN este starea principală, aceea în care procesorul execută efectiv instrucțiuni mașină ale programului procesului. Într-un sistem monoprosesor doar un singur proces se află în starea RUN. Dacă sistemul are n procesoare, atunci maximum n procese se vor afla în starea RUN.

Aceste stări nu se vor întâlni la toate tipurile de sisteme de operare. De exemplu, la un sistem monoutilizator (vezi 8.1) nu va fi starea SWAP, numai sistemele seriale (vezi [10]) au stările HOLD și FINISH etc.

8.2.1.2 Fazele unui program

Fazele unui program reprezintă etapele prin care trece acesta din momentul în care s-a început proiectarea lui și până când devine cod mașină executabil spre a fi integrat într-un proces. Aceste faze, bine cunoscute de către programatori, sunt:

1. Editarea textului sursă într-un limbaj de programare, executată de către programator cu ajutorul unui editor de texte.
2. Compilarea, executată de un compilator specializat în limbajul respectiv. Ca rezultat se obține un fișier obiect.
3. Editarea de legături, fază în care se grupează mai multe module obiect rezultate din compilări și se obține un fișier executabil. Editarea se face cu un program specializat, numit editor de legături
4. Execuția programului înseamnă mai întâi încărcarea fișierului executabil în memorie cu ajutorul unui program încărcător (loader) și lansarea lui în execuție.
5. Testarea și depanarea programului, fază care alternează cu cele de mai sus până când proiectantul are certitudinea că programul lui este corect. Uneori această depanare poate fi asistată de către un program specializat numit depanator.

În secțiunile următoare vom reveni cu detalii, atât asupra stărilor unui proces cât și asupra fazelor unui program.

Menirea unui SO, indiferent de tipul lui, este pe de o parte de a facilita accesul la sistem a unuia sau mai multor utilizatori, iar pe de altă parte de a asigura o exploatare eficientă a echipamentului de calcul. Vom încerca să arătăm mai în detaliu care sunt funcțiile posibile ale unui sistem, independent de tipul acestuia. Desigur, dependentă de tip va fi ponderea în cadrul sistemului a uneia sau alteleia dintre funcții. Din motive care se vor clarifica mai târziu, vom folosi în loc de termenul program termenul de proces sau task. Printr-un proces vom înțelege un calcul care poate fi efectuat concurent cu alte calcule (asupra acestei noțiuni vom reveni).

S-au proiectat și se proiectează încă multe tipuri de SO. Fiecare dintre acestea trebuie să aibă ca obiective fundamentale:

- optimizarea utilizării resurselor;
- minimizarea efectului uman de programare și exploatare;
- automatizarea operațiilor manuale în toate etapele de pregătire și exploatare a SC;
- creșterea eficienței utilizării SC prin scăderea prețului de cost al prelucrării datelor.

Principalele funcții ale SO sunt legate de acțiunile acestuia pentru tranziția între diverse stări ale procesului. În fig. 8.1 [10], [47] sunt schematizate stările unui proces. În dreptul fiecărei tranziții sunt trecute principalele acțiuni efectuate de către SO. Caracteristicile fiecăreia dintre cele șase stări posibile (HOLD, READY, RUN, SWAP, WAIT, FIHISH) le-am prezentat în secțiunea precedentă.

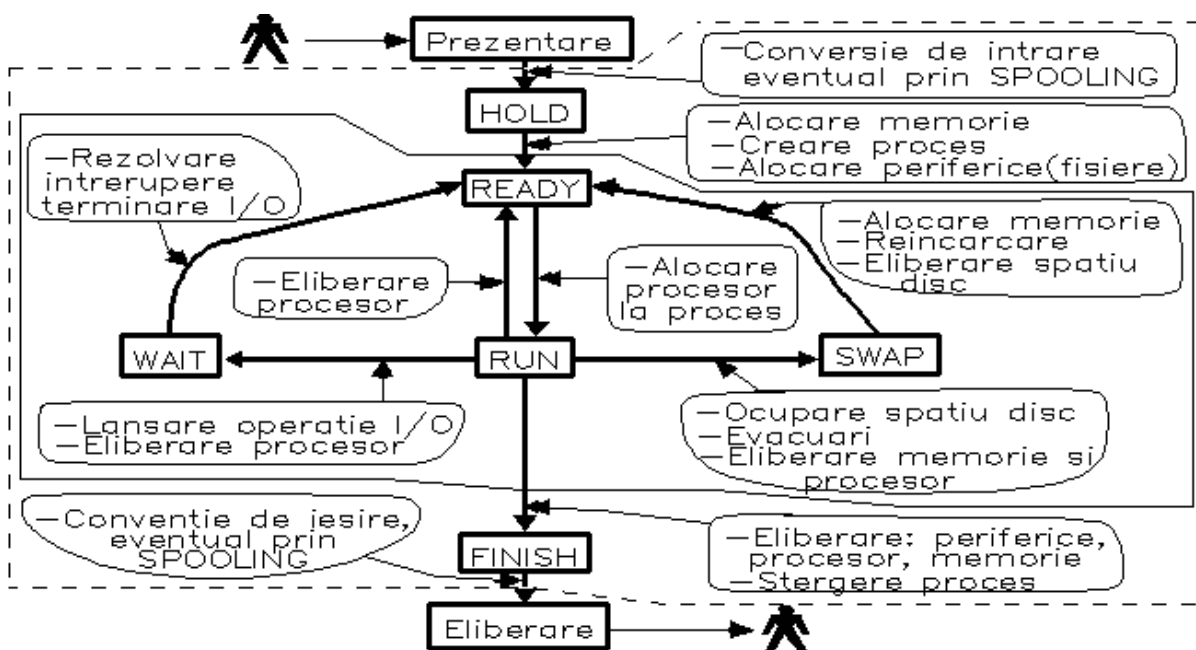


Figura 8.1 Stările unui proces și acțiunile SO aferente

Acțiunile SO ilustrate în fig. 8.1 sunt executate de către componenta nucleu a sistemului de operare. La nivelul componentei user, o mare parte dintre acțiunile SO sunt focalizate mai ales în sprijinul utilizatorului pentru dezvoltarea de programe, adică de asistare a userului la trecerea programului prin diverse faze (vezi secțiunea precedentă 8.2.1.2).

Am exclus de aici, de fapt am amânat numai până la o secțiune viitoare, o funcție primordială: punerea în lucru a SO. După cum vom vedea, în mod (aparent) paradoxal, această sarcină revine tot SO! La prima vedere pare a fi vorba de o antinomie, cum este posibil ca ceva (cineva) să-și dea viață singur? Vom vedea.

8.2.2 Structura generală a unui sistem de operare

În această secțiune încercăm să oferim o imagine asupra structurii unui SO ipotetic. Un astfel de sistem nu va fi regăsit nicăieri în practică, însă structura fiecărui SO real va fi apropiată de a acestuia. Evident, în funcție de tipul de SO real, unele dintre prezentele componente vor fi atrofiate sau vor lipsi cu desăvârșire, iar altele vor putea fi eventual mult mai dezvoltate decât

aici. Fără pretenția de "gen-proxim" și "diferență-specifică", încercăm să definim principalele noțiuni, structuri și componente cu care operează orice SO.

Să trecem acum la structura propriu-zisă.

8.2.2.1 Structura unui SO

Schema generală a unui SO este ilustrată în fig. 8.2.

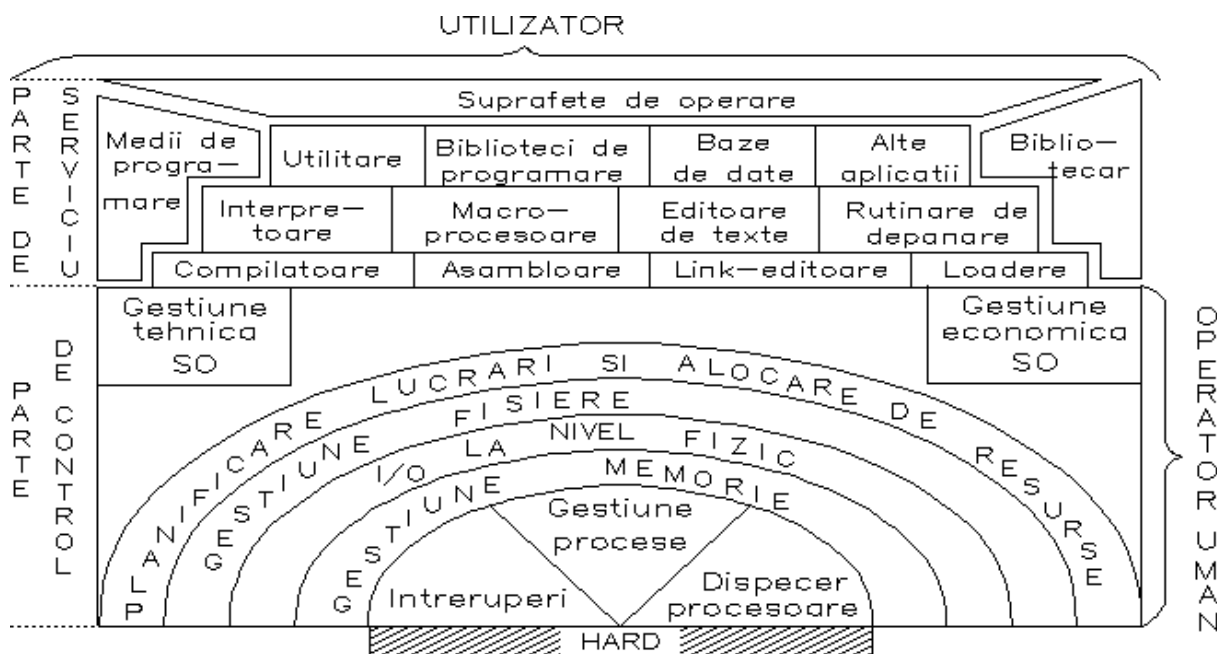


Figura 8.2 Structura generală a unui SO

Orice SO conține două părți mari: partea de control și partea de servicii.

PARTEA DE CONTROL este în legătură directă cu operatorul uman și indirectă cu utilizatorul. SO execută în mod master componentele acestei părți. Principalul ei rol este de a realiza legătura cu SC, deci cu echipamentul hard.

PARTEA DE SERVICIU lucrează în mod slave, folosind facilitățile părții de control. Este în legătură directă cu utilizatorul și indirectă cu operatorul uman. Sarcina de bază a ei este de a ține legătura cu utilizatorul, facilitându-i acestuia accesul la resursele SC și SO.

Idea schemei din fig. 8.2 este preluată din [10]. În continuare vom descrie succint rolul fiecărei componente.

8.2.2.2 Partea de control

INTRERUPERI. Un ansamblu de rutine, fiecare dintre ele fiind activată la apariția unui anumit semnal fizic de întrerupere. Corespunzător, SO efectuează acțiunile aferente întreruperii respective. Semnificația conceptului de întrerupere este descrisă în fiecare manual de arhitectura calculatoarelor. Pentru întreruperile 8086, se poate consulta [52].

Funcționarea concretă a componentei (sistemului) de întreruperi depinde foarte mult de particularitățile SC, deci nu se pretează prea bine la abordări cu caracter general. Pe de altă parte, detalii despre aceste componente interesează în primul rând pe programatorii și inginerii de sistem, mai puțin pe utilizatorii obișnuiți., cărora ne adresăm noi. Din aceste motive vom schița numai mecanismul de funcționare a întreruperilor.

Fiecare tip de întrerupere are asociată o locație fixă de memorie. În această locație se află o adresă. Această adresă indică locul din memorie la care se află o secvență de instrucțiuni, numită handler, secvență care deservește întreruperea respectivă.

La apariția semnalului de întrerupere, după ce instrucțiunea mașină în curs s-a executat, se execută, în această ordine, următoarele activități:

- se salvează într-o zonă de memorie starea programului în curs de desfășurare;
- se activează handlerul asociat întreruperii respective;
- handlerul execută acțiunile necesare servirii întreruperii;
- se restaurează starea programului care a fost întrerupt.

GESTIUNE PROCESSE. Creează procese și rezolvă problemele privind cooperarea și concurența acestora. La terminarea unui proces, acesta este șters din sistem. Asupra conceptului de proces vom mai reveni.

DISPECER PROCESOARE. La sistemele multiprocesor, repartizează sarcinile de calcul solicitate de procese, procesoarelor care sunt libere și care pot executa sarcinile cerute.

GESTIUNEA MEMORIEI. Alocă necesarul de memorie internă solicitat de procese și asigură protecția memoriei interprocese. Componenta este parțial realizată prin hard și parțial este realizată soft, de către o componentă a SO. Vom reveni și asupra problemelor legate de gestiunea memoriei.

I/O LA NIVEL FIZIC. Asigură efectuarea operațiilor elementare de I/O cu toate tipurile de periferice din sistem realizând, acolo unde este posibil, desfășurarea simultană a uneia sau mai multor operații I/O, cu activitatea procesorului central. Această componentă este, de asemenea, puternic dependentă hard, în special de multitudinea de echipamente periferice conectate la sistem. Așa după cum am văzut, există niște rutine de interfață numite drivere, care realizează compatibilizarea convențiilor de I/O între SC și echipamentele periferice. Driverele printre altele, fac parte din această componentă. Vom reveni într-o secțiune viitoare cu detalii.

GESTIUNEA FISIERELOR. O colecție de module prin intermediul cărora se asigură: deschiderea, închiderea și accesul utilizatorului la fișierele rezidente pe diferite suporturi de informații. Componentă de bază a SO, este cel mai des invocată de către utilizatori și de către operator. Vom reveni și asupra ei într-un capitol special.

PLANIFICAREA LUCRARILOR SI ALOCAREA RESURSELOR. Resursele unui SC se împart în: resurse fizice (procesoare, memorie internă, periferice etc.) și resurse logice (proceduri comune, tabele ale sistemului etc.). Fiecare proces solicită la un moment dat anumite resurse. Prin această componentă a SO se asigură planificarea proceselor astfel, încât ele să poată obține de fiecare dată resursele necesare, în mod individual sau partajate cu alte procese. Elemente ale planificării se răsfrâng, după cum vom vedea mai târziu, asupra gestiunii proceselor, gestiunii memoriei, intrărilor și ieșirilor etc. Atunci când vom discuta despre procese, memorie și I/O la nivel fizic vom aborda și problematica planificărilor și a alocărilor de resurse.

GESTIUNEA TEHNICA a SO. Tine evidența erorilor hard apărute la echipamentele SC. La cerere, furnizează informații statistice asupra gradului de utilizare a componentelor SC.

GESTIUNEA ECONOMICA a SO. Tine evidența utilizatorilor care folosesc sistemul, lucrările executate de către aceștia și resursele consumate de aceste lucrări (timp de rulare, memorie internă ocupată, echipamente și suportați folosiți etc). Periodic (zilnic, lunar, anual) editează liste cuprinzând lucrările rulate și facturi de plată către beneficiari.

8.2.2.3 *Partea de servicii*

COMPILATOARELE sunt programe care traduc texte sursă scrise în limbaje de nivel înalt (C, C++, Java, FORTRAN, COBOL, Pascal, Ada etc.) în limbaj mașină. La această oră există o mare varietate de tehnici de compilare, având ca nucleu teoria limbajelor formale. Există de asemenea sisteme automate de elaborare a diverselor componente ale compilatorului. Rezultatul unei compilări este un modul obiect memorat într-un fișier obiect. Detalii privind compilatoarele se pot obține consultând [33].

ASAMBLORUL este un program care traduce texte scrise în limbajul de asamblare propriu SC într-un modul obiect. Din punct de vedere al SO problemele legate de asamblor sunt incluse în problemele compilatoarelor. De exemplu, programarea în limbaj de asamblare 8086 este descrisă în [52].

LINK-EDITORUL sau **EDITORUL DE LEGATURI**. Grupează unul sau mai multe module obiect, împreună cu subprograme de serviciu din diverse biblioteci, într-un program executabil. De cele mai multe ori această componentă oferă posibilitatea segmentării programelor mari, adică posibilitatea ca două componente diferite a programului să poată ocupa în execuție, la momente diferite de timp, aceeași zonă de memorie. Rezultatul editării de legături este un fișier executabil. În 5.1 este descris formatul executabil de sub Unix, iar în [52] fișierele executabile COM și EXE de pe platformele Microsoft.

LOADER (PROGRAM DE INCARCARE) este un program al SO care are trei sarcini:

- citirea unui program executabil de pe un anumit suport;
- încărcarea acestuia în memorie;
- lansarea lui în execuție.

Dacă programul executabil este segmentat, atunci loaderul încarcă la început numai segmentul rădăcină al programului, după care, la cerere, în timpul execuției, încarcă de fiecare dată segmentul solicitat în memorie. În [52] este descris mecanismul de încărcare de sub MS-DOS.

INTERPRETOR este un program care execută pas cu pas instrucțiunile descrise într-un anumit limbaj. Tehnica interpretării este aplicată la diferite limbaje de programare, cum ar fi Java, BASIC, LISP, PROLOG etc. Pentru SO, cea mai importantă categorie de interpretoare sunt "interpretoarele de comenzi". În 2.1.1 am descris interpretoarele de comenzi Shell de sub Unix, iar în [10] [52] interpretorul de comenzi COMMAND.COM de pe platformele Microsoft.

MACROPROCESOR (PREPROCESOR) este un program cu ajutorul căruia se transformă o machetă de program, pe baza unor parametri, într-un program sursă compilabil. Spre exemplu, fie macheta:

```

MACRO    ADUNA    &1, &2, &3
LOAD     &1
ADD      &2
STORE    &3
ENDMACRO

```

Dacă într-un program (dintr-un limbaj de asamblare) se scrie:

```
ADUNA  A, B, C
```

atunci prin macroprocesoare se generează:

```

LOAD    A
ADD     B
STORE   C

```

Practic, macroprocesarea este tehnica prin care se realizează o scriere mai compactă a programelor și o posibilă parametrizare a acestora pentru realizarea comodă a unor implementări înrudite.

Astfel de macroprocesoare există la toate SO moderne. Remarcăm în mod deosebit interpretorul de comenzi "shell" de sub Unix, descris în 2.2. Acesta dispune, printre altele, de cele mai puternice facilități de macroprocesare dintre toate interpretoarele de comenzi cunoscute.

Unele limbaje au prevăzute în standardele lor specificații de macroprocesare. În acest sens amintim preprocesorul "C" care tratează în această manieră liniile care încep cu "#" și construcțiile "typedef", precum și facilitățile generice ale limbajului ADA .

EDITORUL DE TEXTE este un program interactiv destinat introducerii și corectării textelor sursă sau a unor texte destinate tipării. Orice SO interactiv dispune cel puțin de câte un astfel de editor de texte. Până vom ajunge la capitolul dedicat editoarelor de texte, să enumerăm câteva editoare "celebre" unele prin vechimea lor, iar altele prin performanțele lor:

- ed și vi sub Unix;
- Notepad și EDIT sub Windows;
- xedit și emacs de sub mediul X WINDOWS.

RUTINELE DE DEPANARE (DEPANATOARELE) asistă execuția unui program utilizator și-l ajută să depisteze în program cauzele erorilor apărute în timpul execuției lui. Intreaga execuție a programului utilizator se face sub controlul depanatorului. La prima apariție a unui eveniment deosebit în execuție (depășire de indici, adresă inexistentă, cod de operație necunoscut etc.) depanatorul primește controlul. De asemenea, el poate primi controlul în locuri marcate în prealabil de către utilizator. Un astfel de loc poartă numele de breakpoint. La primirea controlului, depanatorul poate afișa valorile unor variabile sau locații de memorie, poate reda, total sau parțial istoria de calcul, poate modifica valorile unor variabile sau locații de memorie etc. Unele depanatoare mai performante permit execuția reversibilă, concept tratat teoretic dar mai puțin aplicat practic. La această oră sunt cunoscute două tipuri de depanatoare:

- depanatoare mașină;
- depanatoare simbolice.

Depanatoarele mașină operează cu elemente cum ar fi: adrese fizice, configurații de biți, locații de memorie etc. Fiecare SO dispune de câte un astfel de instrument de depanare. Pentru

exemplificări se pot consulta documentațiile MS_DOS pentru DEBUG, precum și documentația Unix despre depanatoarele adb și gdb [47].

Depanatoarele simbolice operează cu elemente ale textelor sursă ale programelor. Întâlnim aici termeni ca linie sursă, nume de variabilă, nume de procedură, stivă de apel a procedurilor etc. Evident, depanatoarele simbolice sunt mult mai tentante pentru utilizator, și este bine că este așa. De obicei, un astfel de depanator însoțește implementarea unui limbaj de programare de nivel înalt. Primul și cel mai răspândit limbaj la care au fost atașate depanatoare simbolice este limbajul Pascal. Amintim aici depanatorul Pascal de pe lângă implementarea TURBO Pascal, un depanator simbolic complet interactiv, cu o grafică destul de bună pentru momentul implementării lui și cu facilități puternice. Firma BORLAND, un adevărat vârf de lance în acest domeniu, a realizat printre altele produsul TURBO DEBUGER, care recunoaște în vederea depanării toate limbajele pentru care s-au realizat medii TURBO: Pascal, "C", limbajul de asamblare TASM, Basic, PROLOG etc. Produsul lucrează atât simbolic, cât și la nivel mașină.

BIBLIOTECARUL este un program cu ajutorul căruia utilizatorul poate comanda păstrarea programelor proprii în fișiere speciale de tip bibliotecă, poate copia programe dintr-o bibliotecă în alta, șterge, inserează și modifică programe în aceste biblioteci. Exemple de programe biblioteci sunt LBR sub MS-DOS, ar (archive) sub Unix etc.

MEDIILE DE PROGRAMARE sunt componente complexe destinate în principal activității de dezvoltare de programe. O astfel de componentă înglobează, relativ la un limbaj, cel puțin următoarele subcomponente:

- editor de texte;
- compilator interactiv;
- compilator serial (clasic);
- editor de legături sau un mecanism echivalent acestuia destinat creerii de programe executabile;
- interpretor pentru execuția rezultatului compilării;
- depanator, de regulă simbolic;
- o componentă de legătură cu mediul exterior, de regulă cu SO sub care se lucrează.

De regulă, filozofia de manevrare a mediului are un caracter unitar, este simplă și puternică. Facilitățile grafice ale acestor medii sunt destul de mult utilizate, făcând deosebit de agreabil lucrul cu ele.

SUPRAFETELE DE OPERARE au apărut tot la calculatoarele IBM-PC și a celor compatibile cu ele. Ele au rolul principal de a "îmbrăca" sistemul de operare (Windows sau Unix), pentru a-i face mai ușoară operarea. Inițial au fost destinate utilizatorilor neinformaticieni. La această oră, nici profesioniștii nu se mai pot "dezlipi" de o suprafață de operare cu care s-a obișnuit să lucreze. De regulă, suprafețele de operare înlocuiesc limbajul de control (de comandă) al SO respectiv. Utilizatorul, în loc să tasteze o comandă a SO respectiv, poartă un dialog cu suprafața de operare, iar la finele dialogului suprafața generează sau execută comanda respectivă. Ponderea dialogului o deține suprafața însăși, utilizatorul răspunde de regulă prin apăsarea uneia sau maximum a două taste, sau unul sau două "clicuri" cu un mouse. Suprafața deține și un mecanism deosebit de puternic de HELP, conducând utilizatorul din aproape în aproape spre scopul dorit. Deși se adresează în primul rând neinformaticienilor, ele sunt din ce în ce mai mult folosite și de către specialiști.

O remarcă specială trebuie făcută pentru WINDOWS, care deține un mecanism propriu de dezvoltare a unor aplicații.

Din rațiuni impuse de standarde, sub Unix se folosește aproape peste tot suprafața X-WINDOWS [43].

Din scurta prezentare a componentelor principale ale unui SO rezultă caracterul modular al SO și organizarea lui ierarhică. Modularitatea este termenul prin care se descrie partiționarea unui program mare în module mai mici, cu sarcini bine precizate. Caracterul ierarhic este reflectat în faptul că orice componentă acționează sub controlul componentelor interioare acesteia, apelând primitivele oferite de acestea. Aceste facilități asigură o implementare mai ușoară a SO, o înțelegere mai facilă a SO de către diverse categorii de utilizatori, o întreținere și depanare mai ușoară a sistemului și creează premisele simulării lui pe alt sistem.

8.3 Incărcarea (lansarea în execuție) a unui sistem de operare

Una dintre sarcinile de bază ale unui SO este aceea de a se putea autoîncărca de pe disc în memoria internă și de a se autolansa în execuție. Această acțiune se desfășoară la fiecare punere sub tensiune a SC, precum și ori de câte ori utilizatorul (operatorul uman) crede de cuviință să reîncarce SO. Acțiunea este cunoscută sub numele de încărcare a SO sau lansare în execuție a SO.

În cele ce urmează dorim să punem în evidență mecanismul de lansare. O vom face însă pentru un SO ipotetic, fără să ne legăm de un anumit sistem de operare concret. Simplificând la maximum expunerea.

Pentru lansare se utilizează un mecanism combinat hard și soft, numit bootstrap. Pentru a putea înțelege mai bine modul lui de funcționare, să considerăm un exemplu.

Să notăm cu $M[0]$, $M[1]$, $M[2]$, . . . locațiile de memorie ale unui calculator ipotetic. Presupunem că în fiecare locație $M[i]$ încapă un număr întreg. Presupunem, de asemenea că sistemul nostru dispune de o memorie ROM și un disc (dischetă, CD, DVD, hard disc, partiție disc etc.) numit disc de boot, de la care se citesc, secvențial, începând din primul sector, numere întregi.

Prin $read(x)$ vom înțelege că se citește următorul număr de la intrarea standard și valoarea lui se depune în locația x .

Prin $transferla(y)$ înțelegem că următoarea instrucțiune mașină de executat este cea din (al cărei cod se află în) locația y .

Mecanismul bootstrap intră în lucru la apăsarea butonului de pornire <START>. Ca prim efect, sistemul extrage din ROM un șir de octeți reprezentând echivalentul în limbaj mașină al programului din fig. 8.3.

```
APASASTART:
for (i = 0; i < 100 ; i++) {
    read(M[i]);
}
transferla(M[0]);
```

Figura 8.3 Programul APASASTART

Sirul de octeți citit este depus în memorie la o adresă pe care o vom nota APASASTART. După depunerea la adresa respectivă, sistemul execută instrucțiunea:

```
transferla(APASASTART);
```

Cu aceasta, sistemul transferă controlul programului din fig. 8.3. Execuția acestui program înseamnă că sistemul citește de pe discul de boot primele 100 de numere, le depune în primele 100 de locații de memorie, după care trece la executarea instrucțiunii al cărei cod se află în prima locație.

Este limpede că în 100 de locații nu se poate scrie decât un program simplu. Ori pentru lansarea în execuție a unui SO, probabil că este necesară punerea în lucru a unui program mai sofisticat. Un astfel de program poate fi scris pe mai multe sectoare consecutive, folosind, de exemplu reprezentarea din fig. 8.4. Să presupunem că într-un sector se pot înregistra S numere întregi.

AM1	n1	n1 întregi de program sau date	$S - n1 - 2$ întregi zero
AM2	n2	n2 întregi de program sau date	$S - n2 - 2$ întregi zero
---	---	---	---
AMk	Nk	nk întregi de program sau date	$S - nk - 2$ întregi zero
ADL	-1		$S - 2$ întregi zero

Figura 8.4 Structura unui fișier ce conține un program executabil

Deci fișierul executabil din fig. 8.4 poate avea un număr oarecare k de înregistrări a câte un sector, conținând numere întregi. Semnificațiile câmpurilor sunt:

- AM_i adresa din memorie de unde începe introducerea porțiunii de cod de program sau date din înregistrarea i ;
- n_i este numărul efectiv de locații ocupat de porțiunea de program sau date a înregistrării i . Valoarea -1 indică ultimul sector al a programului.
- ADL se află în ultimul sector al programului și este adresa de lansare în execuție a programului reprezentat: un număr între AM_1 și $AM_k + n_k - 1$.

Să considerăm acum programul din fig. 8.5, în care presupunem că începând de la locația $M[r]$ există S locații libere.

```

LOADERABSOLUT:
do {
  for ( i=0; i < S; i++) {
    read(M[r+i]);
  }
  A = M[r];
  n = M[r+1];
  if (n== -1)
    break;
  for ( i=0; i < n; i++) {
    M[A+i] = M[r+2+i]
  }
  while (FALSE);
LANSARE:
transferla(A);

```

Figura 8.5 Incărcător (loader) de programe cu structura din fig. 8.4

Avem acum posibilitatea să citim de pe disc și să lansăm în execuție, cu programul din fig. 8.5, programe oricât de mari cu formatul din fig. 8.4. De fapt, în fig. 8.5 avem de-a face cu un exemplu simplu de program de încărcare (loader).

Cheia (saltul calitativ al) încărcării: În cele 100 de numere care vor fi citite în momentul startului cu programul APASASTART (din fig. 8.3), se trece codul mașină al programului LOADERABSOLUT (din fig. 8.5). Să vedem acum, pe etape, cum are loc încărcarea și lansarea în execuție a SO. În fig. 8.6 sunt ilustrate zonele de memorie folosite în faza de încărcare și lansare în execuție a SO.

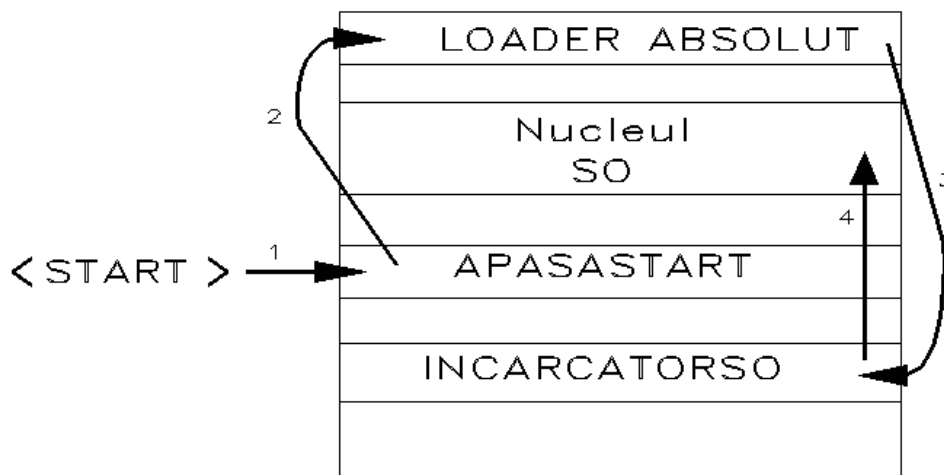


Figura 8.6 Incărcarea unui SO

Etapele încărcării sunt:

- 1) Se apasă <START>. Ca efect, programul APASASTART citește, în primele 100 de locații de memorie, codul mașină al programului LOADERABSOLUT.
- 2) După transferla(M[0]) intră în lucru programul LOADERABSOLUT.
- 3) Acesta citește la rândul lui un program, care de această dată poate fi oricât de sofisticat, cu formatul din fig. 8.4. Acest program îl vom numi INCARCATORSO.
- 4) După ce se execută și ultima instrucțiune transferla(A) a programului LOADERABSOLUT, intră în lucru programul INCARCATORSO. Dat fiind faptul că formatul de reprezentare a codului lui mașină este cel din fig. 8.4, acesta poate fi practic oricât de mare. Așa cum îi spune și numele, acest program va încărca de pe disc programele și rutinele (nucleului) sistemului de operare.

La încheierea încărcării SO, INCARCATORSO execută următoarele acțiuni:

- șterge din memorie codul programului APASASTART;
- șterge din memorie codul programului LOADERABSOLUT;
- eliberează spațiul de memorie ocupat de el, cu excepția unei scurte porțiuni, despre care vorbim mai jos;
- execută o instrucțiune transferla(ASSO), unde ASSO este adresa de start a sistemului de operare. Codul necesar acestei instrucțiuni este cel pe care nu-l poate, în mod firesc, elibera. Însă dacă acest cod este plasat într-o zonă folosită ca buffer, practic nu mai rămâne memorie ocupată.
- după aceasta, încărcătorul își încheie activitatea.

În sfârșit, toate SO moderne lansează la terminarea încărcării un fișier cărui o să-i spunem fișier de comenzi inițiale. Acesta este un fișier de comenzi, care la rândul lui poate lansa alte fișiere de comenzi ș.a.m.d. text în care sunt trecute toate comenzile pe care administratorul le dorește a fi executate la pornirea sistemului. Printre ultimele acțiuni desfășurate la lansarea SO amintim:

- montarea partițiilor locale;
- efectuarea unor operații de întreținere asupra sistemului de fișiere (filesystem cleanup);
- pornirea diferitelor servicii (de exemplu serviciile de printare);
- setare nume sistem și inițializare ceasuri în funcție de zona geografică;
- configurează rețeaua;
- montează sisteme de fișiere aflate la distanță (așa-numitele sisteme de fișiere remote)
- permite conectarea utilizatorilor;

Proiectantul SO va avea grijă ca programul încărcător să fie depus într-o zonă de memorie care să nu fie ocupată de către programele SO pe care el le încarcă.

Față de modelul de încărcare exemplificat de noi mai sus, realitatea, conceptual, este foarte aproape. Primele calculatoare executau APASASTART citind o cartelă, pe care era înregistrat programul LOADERABSOLUT. Acesta realiza citirea programului INCARCATORSO de pe bandă sau de pe disc. Multe calculatoare foloseau în acest scop banda perforată (care uneori trebuia "ajutată" cu mâna să defileze prin cititor ☺. Imaginea "arhaică" a unei astfel de porniri este similară -cei în vârstă probabil că își aduc aminte- cu pornirea în anii '50 a celebrului tractor KD, sau a unei drujbe, pornire care se face trăgând de o sfoară. ☺)

În prezent suportul magnetic de încărcare este primul sector (sectorul zero) al unui disc magnetic. Programele care realizează încărcarea se vor deosebi de cele de mai sus, numai prin înlocuirea lui 100 cu 128, 256, 512 sau 1024, deci cu lungimea unui sector.

Unele SO pretind o modalitate prin care i se indică sistemului de pe ce disc să încarce programele LOADERABSOLUT și INCARCATORSO: specificarea se face în memoria CMOS, la cheile pupitrului calculatorului etc. Alte sisteme stabilesc singure, în mod automat, de pe ce disc să facă încărcarea.