8 Sisteme de operare – prezentare generală

8.1 Tipuri de sisteme de operare (SO); clasificări

In prezent există un mare număr de SO în funcţiune şi numărul acestora este în continuă creştere. Nu există un criteriu unitar de comparare a acestora. Noi vom da mai multe clasificări, în funcţie de următoarele criterii:

- după gradul de partajabilitate a resurselor;
- după tipurile de interacţiuni 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 şi monotasking;
- Sisteme monouser şi 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 execuţia unui singur program.

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

Un sistem monouser şi multitasking este acela în care un singur utilizator este conectat la sistem, dar el poate să îşi lanseze simultan mai multe procese în lucru. Tipice în acest sens sunt SO din familia Windows 9X, ca şi cele din familia NT care nu sunt servere. Tot în această categorie putem îngloba sistemele Linux instalate pe calculatoare neconectate în reţea. Pentru sistemele de acest fel care au memoria internă relativ mică, se aplică aşa-numita tehnică swapping, adică evacuarea temporară a unui program, din memoria internă pe discul magnetic. In timpul evacuării, în memoria internă este încărcat un alt program, care la rândul lui este executat parţial şi este apoi evacuat şi el. Cât timp un program este în memorie, el are acces la toate resursele sistemului. Programele sunt astfel executate pe porţiuni, momentele de execuţie alternând cu cele de evacuare. După unii autori ([4]), SO de acest tip se află la graniţa dintre sistemele monoutilizator şi 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 şi a altor tipuri de resurse, între utilizatorii activi la un moment dat. Vom reveni cu detalii asupra metodelor de partajare a procesorului, timpului şi a memoriei. La astfel de sisteme sunt necesare tehnici mai sofisticate, de gestiune şi protecţie 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 interacţiuni permise

In funcţie de resursele pe care le are un sistem şi în funcţie de destinaţia acestuia, este necesară stabilirea unor strategii de interacţiune cu utilizatorul. Conform acestui criteriu de diferenţiere, distingem [34]:

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

Sisteme seriale. In cadrul acestor sisteme lucrările se execută pe loturi pregătite în prealabil. Practic, din momentul predării unei lucrări la ghişeul centrului de calcul şi până la eliberarea ei, utilizatorul nu poate interveni spre a influenţa execuţia programului său. SO afectate acestor sisteme pot lucra atât monoutilizator, cât şi multiutilizator. Din punct de vedere istoric sunt printre primele sisteme. In prezent, în această categorie intră supercalculatoarele, precum şi 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 şi eventual să pregătească livrarea rezultatelor (listare etc.). In general gradul de concurenţă la acest tip de sisteme este relativ redus. In [10] sunt date mai multe detalii despre acest tip de sisteme.

Sistemele cu timp partajat trebuie să suporte, într-o manieră interactivă, mai mulţi utilizatori. Fiecare utilizator este în contact nemijlocit cu programul său. In funcţie de unele rezultate intermediare, el poate decide modul de continuare a activităţii programului său. SO trebuie să gestioneze printre altele şi 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 şi 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 (şi la celelalte resurse aferente). După epuizarea cuantei, procesul este suspendat, pus la sfârşitul cozii de procese şi un alt proces ocupă procesorul pentru următoarea cuantă ş.a.m.d. Dimensiunea cuantei şi strategiile de comutare sunt astfel alese încât să se reducă timpii de aşteptare şi să se realizeze o servire rezonabilă a proceselor. Tipice acestui tip de sisteme sunt sistemele Unix, dar şi Windows din familia NT / server.

Calculatoare personale şi staţii de lucru. La acest tip de sisteme, toate resursele maşinii sunt dedicate utilizatorului care este conectat. Aşa 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 interfaţă prietenoasă de exploatare a sistemului.

Sistemele autonome sunt sisteme dedicate unui anumit proces industrial, cum ar fi: funcţionarea unui robot, coordonarea activităţilor dintr-un satelit geostaţionar, supravegherea unui baraj de apă, a unei staţii de radiolocaţie 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 comunicaţiilor. Acestea formează cea mai nouă categorie de sisteme. Exemplele tipice sunt telefoane mobile şi PDA-urile. Sunt maşini portabile, de dimensiuni mici şi au facilităţi puternice de comunicare. Facilitatea lor principală este conexiunea wireless printr-una dintre tehnologiile existente: radio, infraroşu, Bluetooth etc. Evident, sistemul de operare aferent trebuie să fie capabil să gestioneze eficient aceste comunicaţii. Din cauza formatului mic, memoria internă, dar mai ales memoria pe suport extern este foarte limitată. Sistemul de operare trebiue să facă faţă acestei penurii de resurse. Securizarea accesului la un astfel de sistem este esenţială. Fiind echipamente mici pot fi uşor pierdute / furate, iar această securizare trebuie să le facă de nefolosit de către persoanele neautorizate.

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

In legătură cu acest criteriu de clasificare, trebuie arătat că el este relativ. Astfel, există sisteme seriale care permit interacţiunea cu programele, sistemele interactive au facilităţi puternice de lucru serial etc. Mai mult, acelaşi SO permite în acelaşi timp să lucreze, spre exemplu, în timp real, dar în acelaşi timp să servească, evident cu o prioritate mai mică, şi 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 evoluţia lor istorică.

- Sisteme monolitice;
- Sisteme cu nucleu de interfaţă hardware;
- Sisteme cu structură stratificată;
- Sisteme organizate ca maşini virtuale

Sisteme monolitice. Un astfel de SO este o colecţie de proceduri, fiecare dintre acestea putând fi apelată după necesităţi. Execuţia unei astfel de proceduri nu poate fi întreruptă, ea trebuie să-şi 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 şi user (vezi pentru Unix 4.5 şi pentru Windows 7.1.3). Esenţial pentru aceste tipuri de sisteme este legătura de tip apel-revenire, fără posibilitatea de întrerupere a fluxului normal al execuţiei. In prezent, aceste tipuri de SO sunt pe cale de dispariţie.

Sisteme cu nucleu de interfaţă hardware. Un astfel de sistem concentrează sarcinile vitale de nivelul cel mai apropiat de hardware într-o colecţie de rutine care formează nucleul SO. In acesta sunt înglobate inclusiv rutinele de întrerupere. Componentele nucleului se pot executa concurent. Toate acţiunile utilizatorului asupra echipamentului hard trec prin acest nucleu. Unele SO mai plasează între nucleu şi utilizator încă o interfaţă. Exemple de nuclee ale SO pot fi considerate componentele BIOS ale maşinilor PC deşi 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. In prezent aceste sisteme sunt cele mai răspândite. "Părinţii" lor sunt SO THE (Dijkstra, 1968) şi SO MULTICS. Ultimul dintre ele oferă o organizare mai interesantă, şi unică în felul ei, aceea a inelelor de protecţie. Pentru detalii se pot consulta [48], [49]. In secţiunea următoare vom ilustra structura stratificată a unui SO ipotetic.

Sisteme organizate ca maşini virtuale. Echipamentul hard al acestor tipuri de sisteme serveşte, 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 dispoziţie toate resursele alocate procesului respectiv de către echipamentul hard. In acest mod, pe acelaşi 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 conversaţionale 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 alţii la acelaşi echipament hard. Sistemele actuale Windows oferă maşini virtuale (ferestre) pentru lucru sub DOS. Implementările Linux, FreeBSD şi Solaris actuale oferă maşina virtuală Wine care emulează platforme Windows. Pentru detalii, se pot consulta [4], [49].

8.2 Structura si funcţiile unui sistem de operare

8.2.1 Stările unui proces şi fazele unui program

8.2.1.1 Stările unui proces

In secţiunile precedente am folosit destul de des termenul de proces, privit intuitiv ca un program în execuţie. Am văzut în capitolele precedente noţiunea de proces sub Unix şi sub Windows. In fig. 5.18 din 5.2.3 sunt prezentate stările unui proces Unix. Există [62] şi 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 şi este gata de a fi servit de (un) procesor;
- RUN – un procesor execută efectiv instrucţiuni (maşină) ale procesului;
- WAIT – procesul aşteaptă terminarea unei operaţii de intrare / ieşire;
- 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 şi 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 şi 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 distanţă. Componenta din SO care face preia din coada FINISH şi listează se numeşte SPOOL-ing de ieşire (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 parţial, dar pe moment să fie suspendat în defavoarea altui proces, urmând să i se continue execuţia mai târziu.

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

WAIT este starea în care un proces a cerut executarea unei operaţii de intrare / ieşire. Cât timp procesul aşteaptă să se termine operaţia, cedă altor procese procesorul.

RUN este starea principală, aceea în care procesorul execută efectiv instrucţiuni maşină ale programului procesului. Intr-un sistem monoprocesor 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 şi 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 şi până când devine cod maşină 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 obţine un fişier obiect.
3. Editarea de legături, fază în care se grupează mai multe module obiect rezultate din compilări şi se obţine un fişier executabil. Editarea se face cu un program specializat, numit editor de legături
4. Execuţia programului înseamnă mai întâi încărcarea fişierului executabil în memorie cu ajutorul unui program încărcător (loader) şi lansarea lui în execuţie.
5. Testarea şi 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.

In secţiunile următoare vom reveni cu detalii, atât asupra stărilor unui proces cât şi 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 funcţiile posibile ale unui sistem, independent de tipul acestuia. Desigur, dependentă de tip va fi ponderea în cadrul sistemului a uneia sau alteia dintre funcţii. Din motive care se vor clarifica mai târziu, vom folosi în loc de termenul program termenul de proces sau task. Printr-un proces vom înţelege un calcul care poate fi efectuat concurent cu alte calcule (asupra acestei noţiuni vom reveni).

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

- optimizarea utilizării resurselor;
- minimizarea efectului uman de programare şi exploatare;
- automatizarea operaţiilor manuale în toate etapele de pregătire şi exploatare a SC;
- creşterea eficienţei utilizării SC prin scăderea preţului de cost al prelucrării datelor.

Principalele funcţii ale SO sunt legate de acţiunile acestuia pentru tranziţia între diverse stări ale procesului. In fig. 8.1 [10], [47] sunt schematizate stările unui proces. In dreptul fiecărei tranziţii sunt trecute principalele acţiuni efectuate de către SO. Caracteristicile fiecăreia dintre cele şase stări posibile (HOLD, READY, RUN, SWAP, WAIT, FIHISH) le-am prezentat în secţiunea precedentă.

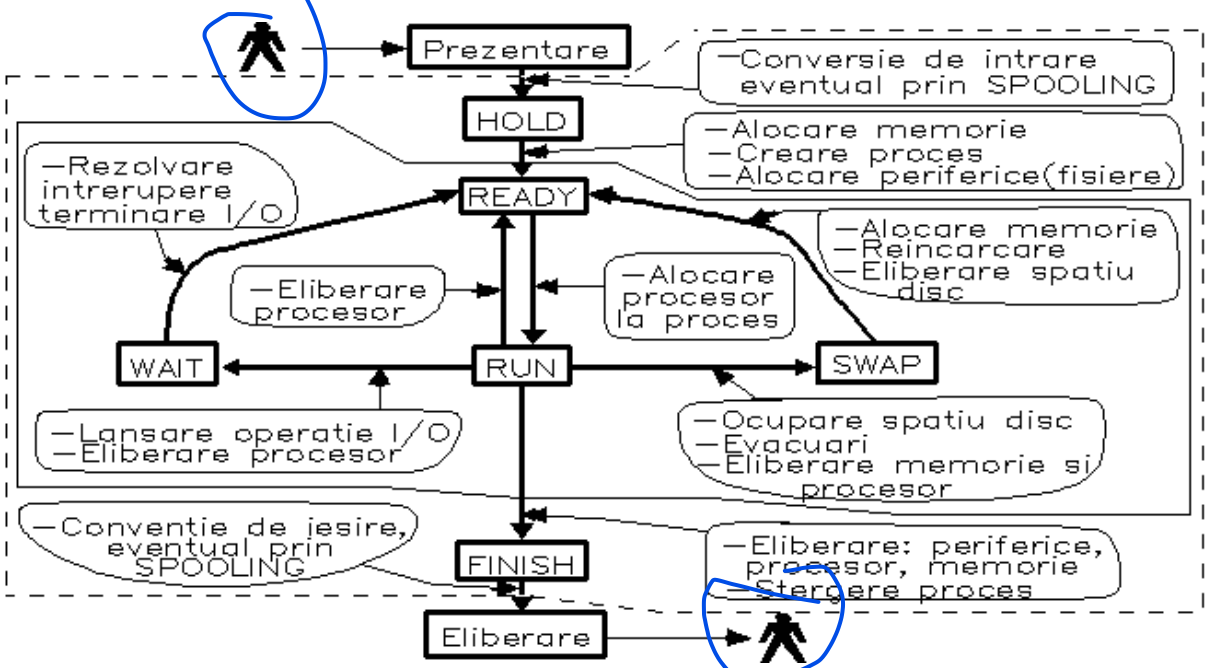

Figura 8.1 Stările unui proces şi acţiunile SO aferente

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

Am exclus de aici, de fapt am amânat numai până la o secţiune viitoare, o funcţie 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-şi dea viaţă singur? Vom vedea.

8.2.2 Structura generală a unui sistem de operare

In această secţiune 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 funcţie de tipul de SO real, unele dintre prezentele componente vor fi atrofiate sau vor lipsi cu desăvârşire, iar altele vor putea fi eventual mult mai dezvoltate decât

aici. Fără pretenţia de "gen-proxim" şi "diferenţă-specifică", încercăm să definim principalele noţiuni, structuri şi 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.

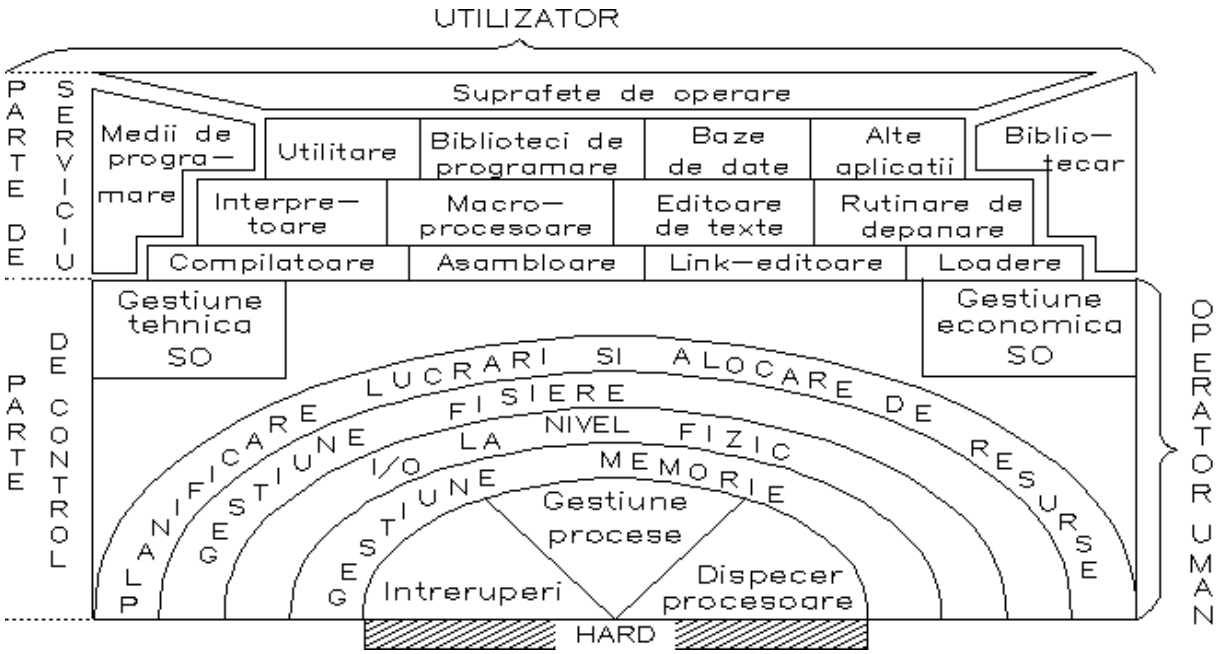

Figura 8.2 Structura generală a unui SO

Orice SO conţine două părţi mari: partea de control şi partea de servicii.

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

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

Ideea schemei din fig. 8.2 este preluată din [10]. In 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 apariţia unui anumit semnal fizic de întrerupere. Corespunzător, SO efectuează acţiunile aferente întreruperii respective. Semnificaţia conceptului de întrerupere este descrisă în fiecare manual de arhitectura calculatoarelor. Pentru întreruperile 8086, se poate consulta [52].

Funcţionarea concretă a componentei (sistemului) de întreruperi depinde foarte mult de particularităţile 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 şi inginerii de sistem, mai puţin pe utilizatorii obişnuiţi., cărora ne adresăm noi. Din aceste motive vom schiţa numai mecanismul de funcţionare a întreruperilor.
Fiecare tip de întrerupere are asociată o locaţie fixă de memorie. In această locaţie se află o adresă. Această adresă indică locul din memorie la care se află o secvenţă de instrucţiuni, numită handler, secvenţă care deserveşte întreruperea respectivă.

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

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

GESTIUNE PROCESE. Creează procese şi rezolvă problemele privind cooperarea şi concurenţa acestora. La terminarea unui proces, acesta este şters 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 şi care pot executa sarcinile cerute.

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

I/O LA NIVEL FIZIC. Asigură efectuarea operaţiilor elementare de I/O cu toate tipurile de periferice din sistem realizând, acolo unde este posibil, desfăşurarea simultană a uneia sau mai multor operaţii I/O, cu activitatea procesorului central. Această componentă este, de asemenea, puternic dependentă hard, în special de multitudinea de echipamente periferice conectate la sistem. Aşa după cum am văzut, există nişte rutine de interfaţă numite drivere, care realizează compatibilizarea convenţiilor de I/O între SC şi echipamentele periferice. Driverele printre altele, fac parte din această componentă. Vom reveni într-o secţiune viitoare cu detalii.

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

PLANIFICAREA LUCRARILOR SI ALOCAREA RESURSELOR. Resursele unui SC se împart în: resurse fizice (procesoare, memorie internă, periferice etc.) şi 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 obţine 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 si ieşirilor etc. Atunci când vom discuta despre procese, memorie şi I/O la nivel fizic vom aborda şi problematica planificărilor şi a alocărilor de resurse.

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

GESTIUNEA ECONOMICA a SO. Tine evidenţa utilizatorilor care folosesc sistemul, lucrările executate de către aceş- tia şi resursele consumate de aceste lucrări (timpi de rulare, memorie internă ocupată, echipamente şi suporţi folosiţi etc). Periodic (zilnic, lunar, anual) editează liste cuprinzând lucrările rulate şi 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 maşină. 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 fişier obiect. Detalii privind compilatoarele se pot obţine 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 execuţie, la momente diferite de timp, aceeaşi zonă de memorie. Rezultatul editării de legături este un fişier executabil. In 5.1 este descris formatul executabil de sub Unix, iar în [52] fişierele executabile COM şi 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 execuţie.

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

INTERPRETOR este un program care execută pas cu pas instrucţiunile 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". In 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 şi 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 facilităţi de macroprocesare dintre toate interpretoarele de comenzi cunoscute.

Unele limbaje au prevăzute în standardele lor specificaţii de macroprocesare. In acest sens amintim preprocesorul "C" care tratează în această manieră liniile care încep cu "#" şi construcţiile "typedef", precum şi facilităţile generice ale limbajului ADA .

EDITORUL DE TEXTE este un program interactiv destinat introducerii şi corectării textelor sursă sau a unor texte destinate tipăririi. Orice SO interactiv dispune cel puţin 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 performanţele lor:

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

RUTINELE DE DEPANARE (DEPANATOARELE) asistă execuţia unui program utilizator şi-l ajută să depisteze în program cauzele erorilor apărute în timpul execuţiei lui. Intreaga execuţie a programului utilizator se face sub controlul depanatorului. La prima apariţie a unui eveniment deosebit în execuţie (depăşire de indici, adresă inexistentă, cod de operaţie necunoscut etc.) depanatorul primeşte 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 afişa valorile unor variabile sau locaţii de memorie, poate reda, total sau parţial istoria de calcul, poate modifica valorile unor variabile sau locaţii de memorie etc. Unele depanatoare mai performante permit execuţia reversibilă, concept tratat teoretic dar mai puţin aplicat practic. La această oră sunt cunoscute două tipuri de depanatoare:

- depanatoare maşină;
- depanatoare simbolice.

Depanatoarele maşină operează cu elemente cum ar fi: adrese fizice, configuraţii de biţi, locaţii de memorie etc. Fiecare SO dispune de câte un astfel de instrument de depanare. Pentru

exemplificări se pot consulta documentaţiile MS_DOS pentru DEBUG, precum şi documentaţia Unix despre depanatoarele adb şi gdb [47].

Depanatoarele simbolice operează cu elemente ale textelor sursă ale programelor. Intâlnim 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, şi este bine că este aşa. De obicei, un astfel de depanator însoţeşte implementarea unui limbaj de programare de nivel înalt. Primul şi cel mai răspândit limbaj la care au fost ataşate 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 şi cu facilităţi puternice. Firma BORLAND, un adevărat vârf de lance în acest domeniu, a realizat printre altele produsul TURBO DEBUGER, care recunoaşte 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 şi la nivel maşină.

BIBLIOTECARUL este un program cu ajutorul căruia utilizatorul poate comanda păstrarea programelor proprii în fişiere speciale de tip bibliotecă, poate copia programe dintr-o bibliotecă în alta, şterge, inserează şi 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 activităţii de dezvoltare de programe. O astfel de componentă înglobează, relativ la un limbaj, cel puţin 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 execuţia 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 şi puternică. Facilităţile 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 şi a celor compatibile cu ele. Ele au rolul principal de a "îmbrăca" sistemul de operare (Windows sau Unix), pentru a-i face mai uşoară operarea. Iniţial au fost destinate utilizatorilor neinformaticieni. La această oră, nici profesioniştii nu se mai pot "dezlipi" de o suprafaţă de operare cu care s-a obişnuit să lucreze. De regulă, suprafeţele 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 suprafaţa de operare, iar la finele dialogului suprafaţa generează sau execută comanda respectivă. Ponderea dialogului o deţine suprafaţa însăşi, utilizatorul răspunde de regulă prin apăsarea uneia sau maximum a două taste, sau unul sau două "clicuri" cu un mouse. Suprafaţa deţine şi un mecanism deosebit de puternic de HELP, conducând utilizatorul din aproape în aproape spre scopul dorit. Deşi se adresează în primul rând neinformaticienilor, ele sunt din ce în ce mai mult folosite şi de către specialişti.

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

Din raţiuni impuse de standarde, sub Unix se foloseşte aproape peste tot suprafaţa X-WINDOWS [43].

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

8.3 Incărcarea (lansarea în execuţie) 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 şi de a se autolansa în execuţie. Această acţiune se desfăşoară la fiecare punere sub tensiune a SC, precum şi ori de câte ori utilizatorul (operatorul uman) crede de cuviinţă să reîncarce SO. Acţiunea este cunoscută sub numele de încărcare a SO sau lansare în execuţie a SO.

In cele ce urmează dorim să punem în evidenţă mecanismul de lansare. O vom face însă pentru un SO ipotetic, fără să ne legăm de un anume sistem de operare concret. simplificând la maximum expunerea.

Pentru lansare se utilizează un mecanism combinat hard şi soft, numit bootstrap. Pentru a putea înţelege mai bine modul lui de funcţionare, să considerăm un exemplu.

Să notăm cu M[0], M[1], M[2], . . . locaţiile de memorie ale unui calculator ipotetic. Presupunem că în fiecare locaţie M[i] încape un număr întreg. Presupunem, de asemenea că sistemul nostru dispune de o memorie ROM şi un disc (dischetă, CD, DVD, hard disc, partiţie disc etc.) numit disc de boot, de la care se citesc, secvenţial, începând din primul sector, numere întregi.

Prin read(x) vom înţelege că se citeşte următorul număr de la intrarea standard şi valoarea lui se depune în locaţia x.

Prin transferla(y) înţelegem că următoarea instrucţiune maşină de executat este cea din (al cărei cod se află în) locaţia y.

Mecanismul bootstrap intră în lucru la apăsarea butonului de pornire <START>. Ca prim efect, sistemul extrage din ROM un şir de octeţi reprezentând echivalentul în limbaj maşină 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 octeţi citit este depus în memorie la o adresă pe care o vom nota APASASTART. După depunerea la adresa respectivă, sistemul execută instrucţiunea:

transferla(APASASTART);

Cu aceasta, sistemul transferă controlul programului din fig. 8.3. Execuţia acestui program înseamnă că sistemul citeşte de pe discul de boot primele 100 de numere, le depune în primele 100 de locaţii de memorie, după care trece la executarea instrucţiunii al cărei cod se află în prima locaţie.

Este limpede că în 100 de locaţii nu se poate scrie decât un program simplu. Ori pentru lansarea în execuţie 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 fişier ce conţine un program executabil

Deci fişierul executabil din fig. 8.4 poate avea un număr oarecare k de înregistrări a câte un sector, conţinând numere întregi. Semnificaţiile câmpurilor sunt:

- AMi adresa din memorie de unde începe introducerea porţiunii de cod de program sau date din înregistrarea i;
- ni este numărul efectiv de locaţii ocupat de porţiunea de program sau date a înregistrării i. Valoarea -1 indică ultimul sector al a programului.
- ADL se află în ultimul sector al programului şi este adresa de lansare în execuţie a programului reprezentat: un număr între AM1 şi AMk+nk-1.

Să considerăm acum programul din fig. 8.5, în care presupunem că începând de la locaţia M[r] există S locaţii 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 şi să lansăm în execuţie, 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: In cele 100 de numere care vor fi citite în momentul startului cu programul APASASTART (din fig. 8.3), se trece codul maşină al programului LOADERABSOLUT (din fig. 8.5). Să vedem acum, pe etape, cum are loc încărcarea şi lansarea în execuţie a SO. In fig. 8.6 sunt ilustrate zonele de memorie folosite în faza de încărcare şi lansare în execuţie a SO.

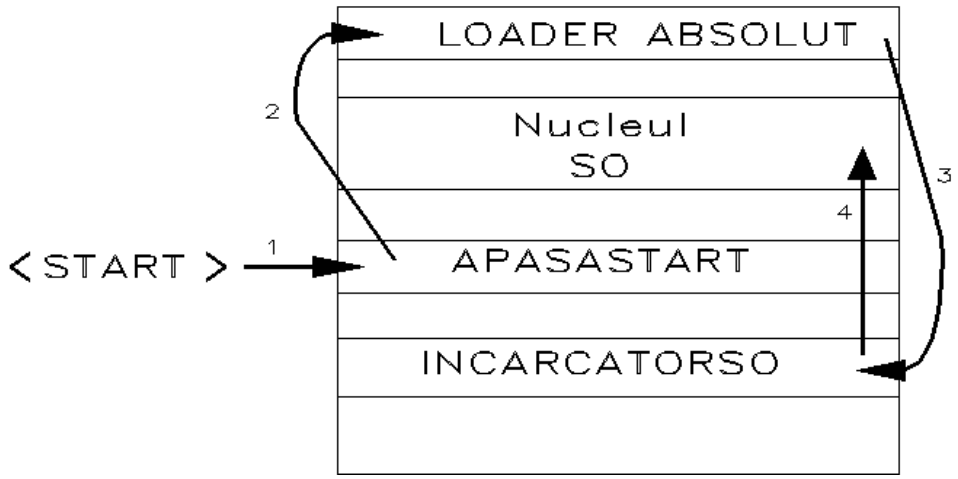

Figura 8.6 Incărcarea unui SO

Etapele încărcării sunt:

1) Se apasă <START>. Ca efect, programul APASASTART citeşte, în primele 100 de locaţii de memorie, codul maşină al programului LOADERABSOLUT.
2) După transferla(M[0]) intră în lucru programul LOADERABSOLUT.
3) Acesta citeşte 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 şi ultima instrucţiune transferla(A) a programului LOADERABSOLUT, intră în lucru programul INCARCATORSO. Dat fiind faptul că formatul de reprezentare a codului lui maşină este cel din fig. 8.4, acesta poate fi practic oricât de mare. Aşa cum îi spune şi numele, acest program va încărca de pe disc programele şi rutinele (nucleului) sistemului de operare.

La încheierea încărcării SO, INCARCATORSO execută următoarele acţiuni:

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

In sfârşit, toate SO moderne lansează la terminarea încărcării un fişier căruia o să-i spunem fişier de comenzi iniţiale. Acesta este un fişier de comenzi, care la rândul lui poate lansa alte fişiere de comenzi ş.a.m.d. text în care sunt trecute toate comenzile pe care administratorul le doreşte a fi executate la pornirea sistemului. Printre ultimele acţiuni desfăşurate la lansarea So amintim:

- montarea partiţiilor locale;
- efectuarea unor operaţii de întreţinere asupra sistemului de fişiere (filesystem cleanup);
- pornirea diferitelor servicii (de exemplu serviciile de printare);
- setare nume sistem şi iniţializare ceasuri în funcţie de zona geografică;
- configurează reţeaua;
- montează sisteme de fişiere aflate la distanţă (aşa-numitele sisteme de fişiere 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.

Faţă 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 îşi aduc aminte- cu pornirea în anii '50 a celebrului tractor KD, sau a unei drujbe, pornire care se face trăgând de o sfoară. ☺)

In 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 şi 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.