Sisteme de operare

Cursul 1 - Generalități

Drăgulici Dumitru Daniel

Facultatea de matematică și informatică,
Universitatea București

2008

Bibliografie

Pentru curs:


Andrew S. Tanenbaum:
Sisteme de operare moderne, Ediția a 2-a,
Editura Byblos, 2004


Octavian Bâscă, Ileana Popescu:
Sisteme de operare,
Tipogr. Univ. București, 1987-1989 (2 vol.)

Pentru laborator:


Jean-Marie Rifflet:

La programmation sous UNIX, 3ème Édition,
EDISCIENCE, 1993


Andrei Baranga:
Dezvoltarea aplicațiilor în C/C++ sub sistemul de operare UNIX,
Editura Albastră


Internet:
www.tldp.org
www.gnu.org/manual

Cuprins

Sisteme de calcul

Sistem de calcul (SC): ansamblu de componente hardware și software care furnizează o formă definită de servicii unui tip de utilizatori.

Un SC este format din:
- partea de hard (**hardware**): totalitatea componentelor fizice (carcasă, procesor, tastatură, etc.);
- partea de soft (**software**): totalitatea componentelor logice (programe, date, etc.).

El trebuie să îndeplinească următoarele **funcții fundamentale**:
- intrare date;
- prelucrare date;
- stocare date;
- ieșire date (furnizare rezultate).

Orice SC definește un **limbaj** în termenii căruia se exprimă întreaga activitate ce se execută pe el. Altfel spus, orice SC este dotat cu posibilitatea reprezentării anumitor **tipuri de date** și **structuri de informație** și implementează o mulțime de **operații primitive** asupra lor.
De asemenea, pentru orice SC se definesc anumite **clase de utilizatori** ce pot exploata (pentru care este destinat spre exploatare) SC respectiv.

Sisteme de calcul

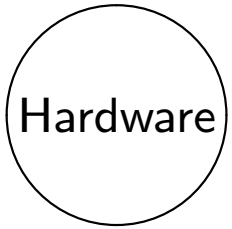
În general, un SC are o structură **stratificată** având la bază hardware-ul și continuând cu mai multe straturi de software.

De asemenea, SC permit în general adăugarea/eliminarea/ modificarea unor componente în diverse straturi ale sale (atât hardware cât și software) schimbându-și astfel modul cum își îndeplinește cele patru funcții fundamentale și, evident, schimbându-și limbajul, clasele de utilizatori cărora le este destinat și scopul pentru care este folosit - devine practic un nou SC, cu o altă configurație hardware și software.

Să analizăm puțin straturile principale ...

Sisteme de calcul

- **Hardware:**

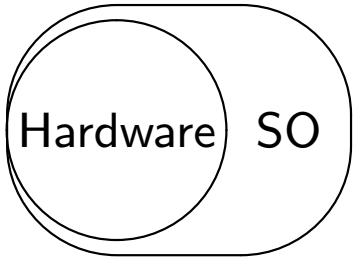


Conține subsisteme fizice specializate pentru îndeplinirea celor 4 funcții fundamentale:
- periferice de intrare (pentru intrare date): tastatură, mouse, etc.;
- procesoare (pentru prelucrare date);
- memorii (pentru stocare date);
- periferice de ieșire (pentru ieșire date): monitor, imprimantă, etc.;
Există dispozitive ce îndeplinesc mai multe funcții, cum ar fi perifericele de intrare și ieșire (ex. modem).

Doar cu stratul hardware, avem un SC cu următorul limbaj și clase de utilizatori:
- tipuri de date: byte, cuvânt de memorie, etc.;
- operații primitive: limbajul mașina;
- utilizatori: realizatorii de sisteme de operare, de drivere, etc.

Sisteme de calcul

- **Sistemul de operare** (SO) **:**



Motive ale prezenței acestui strat:

- stratul hardware oferă un limbaj neintuitiv și greoi (trebuie scris mult pentru a specifica o operație simplă); de exemplu, este greu să programăm o scriere pe dischetă folosind instrucțiuni de pornit/oprit motorul, mutat capul de citire/scriere, citit/scris la anumite adrese fizice, etc.;

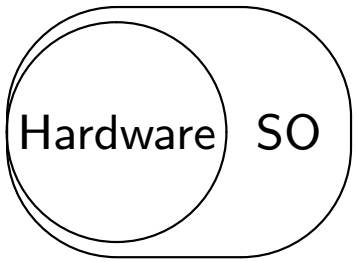
una din funcțiile SO este să gestioneze resursele hardware și software ale SC, oferind o interfață prin care ele sunt prezentate utilizatorilor și aplicațiilor într-o formă mai comod de accesat - de exemplu discheta este prezentată ca o colecție de fișiere organizate în directoare și accesibile doar prin nume și cale;

interfața cu aplicațiile se realizează prin **apelurile sistem** - o colecție de rutine ale SO apelabile din programele de aplicație; de exemplu în Linux o aplicație poate deschide un fișier apelând doar:

open(nume_fișier, mod_deschidere);

Sisteme de calcul

- **Sistemul de operare** (SO) **:**

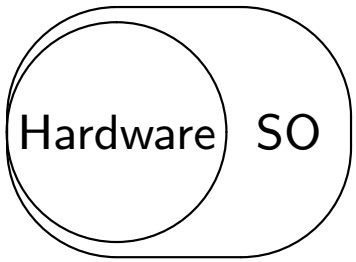


Motive ale prezenței acestui strat:

- o altă funcție a SO este protejarea SC de utilizarea necorespunzătoare din partea utilizatorilor și aplicațiilor - aceștia nu pot accesa resursele sistemului decât apelând apelurile sistem, iar acestea nu execută operația decât dacă ea este posibilă, dacă există drepturile necesare, etc.;
de asemena, SO arbitrează folosirea concurentă a resurselor de către procese care rulează în paralel, comunicarea între procese, protecția datelor fiecărui utilizator de accesul neautorizat, etc.

Sisteme de calcul

- **Sistemul de operare** (SO) **:**

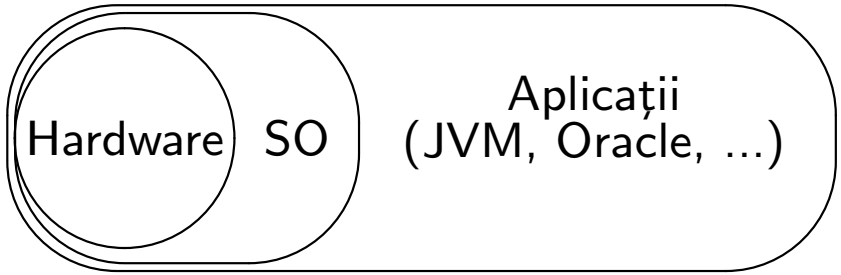


Adăugând stratul SO obținem un nou SC, cu următorul limbaj și clase de utilizatori:
- tipuri de date: fișier, director, etc.;
- operații primitive: apeluri sistem;
- utilizatori: realizatorii de aplicații (aplicațiile fiind de multe tipuri, ei se împart în mai multe clase: realizatori de compilatoare, de S.G.B.D., etc.).

În general, dintr-un program se pot efectua și operații scrise în cod mașină, cu date de tip byte, word, etc., dar nu orice operații și numai într-o formă pe care o permite SO.
Cu alte cuvinte, stratul SO oferă **transparență** pentru unele elemente ale stratului inferior, dar controlează modul în care se face acces la ele - accesul se face doar prin interfața SO.
De aceea, în limbajul acestui SC (hardware + SO) vom considera și elementele preluate prin transparență.

Sisteme de calcul

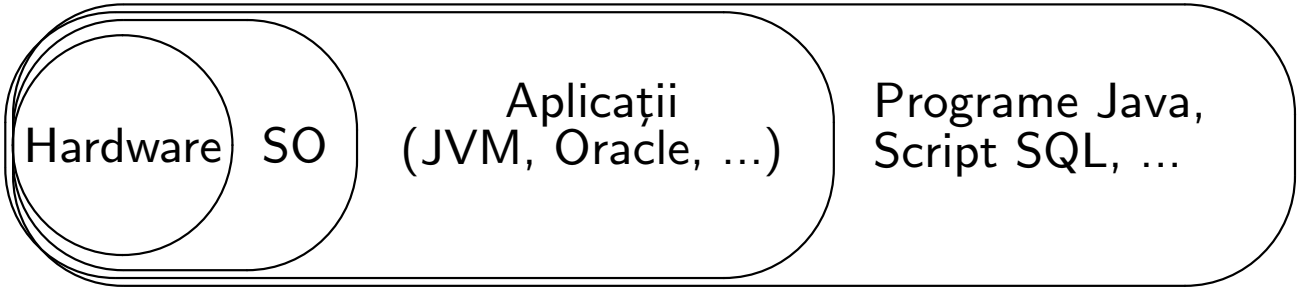
- **Aplicații:**



În funcție de aplicațiile instalate, obținem diverse SC, capabile să îndeplinească diverse activități. De fiecare dată, SC oferă un alt limbaj și este destinat unei alte clase de utilizatori. De exemplu:
- dacă am instalat un program de contabilitate, SC este destinat economiștilor, care pot comunica cu sistemul (utilizând programul respectiv) într-un limbaj specific meseriei lor - au ca tipuri de date conturi bancare, contracte, etc., ca operații tranzacții de sume de bani, ș.a.m.d.;
- dacă am instalat doar player-e de muzică și filme, obținem un sistem multimedia destinat divertismentului; limbajul său are ca tipuri de date melodii, albume, etc., ca operații "play", "pause", specificarea unor parametri de volum, luminozitate, ș.a.m.d..
Evident, în toate cazurile sistemul ne permite și crearea/ștergerea/modificarea de fișiere și directoare, deoarece anumite elemente ale straturilor hardware și SO rămân vizibile prin transparență și la acest nivel.

Sisteme de calcul

- **Alte straturi:**



Unele aplicații oferă un limbaj cu ajutorul căruia pot fi adăugate alte straturi. De exemplu:
- prezența unui compilator Pascal oferă un limbaj ce conține noi tipuri de date (array, boolean, etc.), noi instrucțiuni (for, while, etc.), dar și unele mai vechi prin transparență (tipurile byte, fișier, apeluri sistem sau chiar instrucțiuni assembler), în care putem scrie programe sursă; ele sunt compilate și apoi integrate în stratul aplicațiilor;
- prezența unui motor Java ne permite să scriem aplicații Java (într-un limbaj specific) care sunt rulate chiar de motorul Java (care funcționează ca o mașină virtuală), deci nu sunt la același nivel cu aplicațiile din stratul 3; ele aparțin unui nou strat;
- asemănător în cazul script-urilor adresate unui S.G.B.D. sau interpretor de comenzi shell.

Sisteme de calcul

Sintetizând cele spuse mai devreme, putem spune:

În general, un SC are o structură **ierarhică**, pe mai multe niveluri (straturi), primul fiind nivelul hardware.

Fiecare nivel este un SC **generator** pentru nivelul următor, oferind în acest scop un **limbaj**, format din anumite **tipuri de date**, **structuri de informație** și **operații primitive**.

Trecerea de la un nivel inferior la unul superior se face prin **abstractizarea** unor proprietăți ale nivelului inferior.

De exemplu SO nu lasă aplicațiile să acceseze discul în limbaj de adrese fizice, mișcări ale capetelor magnetice, etc., ci le oferă conceptele abstracte de fișier și director, care nu depind de hardware - chiar dacă instalăm un alt disc, cu alt limbaj mașină de accesare, SO îl va prezenta aplicațiilor tot ca pe o colecție de fișiere și directoare, accesibile prin aceleași apeluri sistem (evident, SO trebuie modificat a.î. să recunoască noul disc, instalând alte drivere).

Există și **abstractizări în sens invers** - de exemplu SO trebuie scris a.î. să nu depindă de aplicațiile care se vor instala ulterior în el.

Sisteme de calcul

Între nivelurile i și $i+2$ există o **interfațare** dată de nivelul i; ea oferă o anumită **transparență** pentru unele elemente ale nivelului i.

Principiul modularității spune că resursele și funcțiile oferite de nivelul i trebuie să formeze o **bază completă** pentru generarea nivelului $i+1$. Astfel, avem o mai mare flexibilitate în a modifica un anumit nivel fără să le alterăm pe altele (software-ul de acolo rămâne același).

Sisteme de calcul

Regulile de mai sus nu sunt întotdeauna respectate cu strictețe, sau sunt, dacă le interpretăm într-un sens mai larg.

De exemplu MS-DOS permite aplicațiilor să acceseze hardware-ul și direct, nu numai prin apelurile sistem. Putem încadra însă acest fenomen în teoria generală gândind că sistemul de operare MS-DOS preia toată interfața hardware în interfața proprie (oferă transparență totală).

De asemenea, un utilizator poate accesa mai multe niveluri (ex: dăm și comenzi SO, scriem și un program Pascal), iar o modificare se poate face la mai multe niveluri deodată (ex: instalăm un webcam și un microfon (la nivel hardware), instalăm driverele lor (la nivel SO) și instalăm și un program de videoconferință (la nivel de aplicații)). Totul se încadrează în teoria generală dacă ne gândim că fiecare nivel include (via acea transparență) nivelurile anterioare.

Sisteme de calcul

Delimitarea între straturi este de asemenea discutabilă - de exemplu anumite aplicații sunt livrate odată cu SO pe post de utilitare (fac parte din distribuție): editoare de text, compilatoare uzuale, etc. Alte aplicații sunt vândute separat și nu sunt prezente în orice instalare a SO respectiv.

Stratul "SO" din desenul anterior este alcătuit de fapt din așa-numitul **kernel** (**nucleu**), care implementează funcționalitățile de bază ale SO și care rulează într-un mod special, privilegiat, numit **kernel mode** (sau **mod supervizor**). Celelalte programe rulează în **user mode** (sau **mod utilizator**), mai puțin privilegiat, și sunt incluse în stratul "Aplicații".

Sisteme de calcul

O clasificare împarte software-ul în 2 categorii:

- **programe de sistem**: gestionează funcționarea corectă și eficientă a calculatorului; cel mai important este **SO**, care controlează resursele calculatorului și oferă baza pe care sunt scrise programele de aplicație;

- **programe de aplicație**: execută operațiile pe care le dorește utilizatorul (procesare de text, procesare de tabele, programe de gestiune a salariilor, etc.).

Cuprins

Istoria SO - primul calculator

Istoria sistemelor de operare se împletește cu istoria calculatoarelor (hardware-ului).

• **Primul calculator numeric**:
A fost "motorul analitic" al lui Charles Babbage (matematician englez, 1792 - 1871).

Era mecanic. Nu a funcționat niciodată corect, deoarece tehnologia acelor vremuri nu permitea construirea unor roți dințate suficient de precise.

Nu avea SO, dar Babbage a intuit necesitatea existenței unor programe pentru motorul analitic, motiv pentru care a angajat-o pe Ada Lovelace (fiica poetului englez lord Byron) ca prima programatoare din lume - după numele ei a fost denumit limbajul de programare ADA.

Istoria SO - prima generație

- **Prima generație (1945 - 1955) - tuburi elecronice și plăci de conexiuni**:

Pe la jumătatea anilor '40 Howard Aiken (Harvard), John von Neumann (institutul de studii avansate din Princetown), J. Presper Eckert și William Mauchley (Univ. Pennsylvania) și Konrad Zuse (Germania) au reușit construirea calculatoarelor cu relee, apoi cu tuburi electronice - erau f. mari (ocupau mai multe camere), deși erau de milioane de ori mai lente decât un calculator ieftin de azi.

Același grup de oameni proiectau, construiau, programau, operau și mențineau în funcție propria mașină.

Programarea se realiza în limbaj mașină absolut, prin inserarea de plăci de conexiuni realizate cu fire. Ulterior (începutul anilor '50) s-au folosit cartele perforate.

Nu existau limbaje de programare (nici măcar limbaje de asamblare), nici SO.

Problemele rezolvate erau în general calcule simple, de ex. tabele de sin, cos, logaritmi.

Istoria SO - generația a II-a

- **Generația a II-a (1955 - 1965) - tranzistoare și prelucrare pe loturi**:

Odată cu introducerea tranzistoarelor (jumătatea anilor '50) calculatoarele au devenit suficient de fiabile ca să fie fabricate și vândute unor clienți. Practic a început să se facă distincție între proiectanți, constructori, programatori, personalul de întreținere.

Mașinile, numite **mainframe** (sisteme mari de calcul) erau puse în camere speciale, cu aer condiționat, și erau manevrate de operatori calificați.
Erau scumpe (milioane de dolari) și nu și le puteau permite decât marile corporații, agenții guvernamentale, universitățile.

Istoria SO - generația a II-a

Lucrul era organizat pe job-uri (**job** = program sau grup de programe). Programatorul scria programul pe hârtie, apoi pe cartele perforate, dădea pachetul operatorului, apoi aștepta rezultatele; operatorul lua pachetul, îl introducea în calculator, apoi lua de la imprimantă listing-ul cu rezultatele și le preda programatorului; apoi lua alt pachet, etc. Dacă era nevoie de un anumit compilator, îl lua din dulap și îl transfera prin citire în calculator. Astfel operatorii făceau multe drumuri prin camerele din jurul mașinii și se pierdea timp.

Pentru a se câștiga timp s-a trecut la prelucrarea pe **loturi** de programe: în alte camere mai multe job-uri erau transferate de pe cartele pe o bandă magnetică cu un calculator ieftin (IBM 1401), apoi banda era dusă la calculatorul principal (IBM 7094) împreună cu un strămoș al SO de astăzi (care citea succesiv job-urile de pe bandă și le rula), rezultatele erau scrise pe altă bandă, care era dusă mai apoi la calculatorul ieftin pentru tipărire la imprimantă.

Istoria SO - generația a II-a

Structura unui job (pe cartele):

```
$JOB,durata_max_de_rulare,cont,nume_programator
$FORTRAN ⟵ cere SO să încarce compilatorul de FORTRAN
  .                de pe banda sistem
  .
  .      ⟵ programul în FORTRAN (va fi compilat și salvat pe benzi
  .                auxiliare)
  .
$LOAD      ⟵ cere SO să încarce programul obiect proaspăt compilat
$RUN       ⟵ cere SO să ruleze programul
  .
  .       ⟵ date pentru program
  .
$END       ⟵ sfarșitul programului
```

Istoria SO - generația a II-a

Deci se lucra neinteractiv, introducând deodată, la început, pachetul de cartele conținând și programul și datele și comenzile SO.

Limbaje de programare folosite frecvent: FORTRAN, limbajul de asamblare.

SO tipice: FMS (the FORTRAN Monitor System), IBSYS (SO de la IBM pentru calculatorul IBM 7094).

Problemele rezolvate erau de obicei calcule științifice și inginerești, ex: rezolvarea ecuațiilor diferențiale și cu derivate parțiale.

Istoria SO - generația a III-a

• **Generația a III-a (1965 - 1980) - circuite integrate și multiprogramare**:

Circuitele integrate au permis îmbunătățirea raportului performanță/preț.

Prima serie de calculatoare cu circuite integratea fost IMB 360 - o familie de mașini diferite ca performanță și preț, dar compatibile software (e mai ieftin decât existența unor linii diferite de producție pentru IBM 1401 și IBM 7094).

Pentru ele a fost scris sistemul de operare OS/360. El a avut multe probleme de funcționare, a cunoscut multe versiuni, deoarece era greu de scris un SO adecvat pentru diversele caracteristici și scopuri ale mașinilor din seria 360 - să lucreze eficient și cu periferice multe și cu periferice puține, și în medii comerciale și în medii științifice, etc. Avea câteva milioane de instrucțiuni.

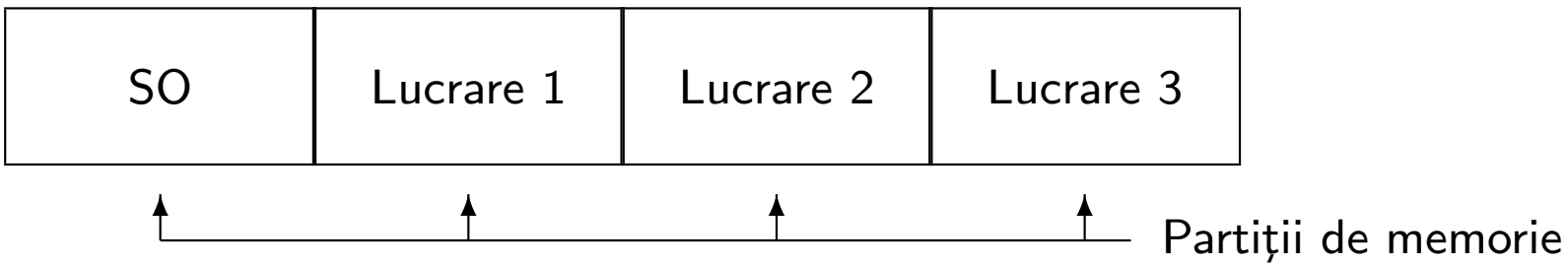
Istoria SO - generația a III-a

Noutăți introduse de OS/360:

• **multiprogramarea**: la IBM 7094, când job-ul curent făcea o pauză așteptând ca banda sau alt dispozitiv de I/O să-și termine activitatea, unitatea centrală (UCP) era neutilizată;

soluția a fost partiționarea memoriei în mai multe zone, încărcarea câte unui program în fiecare partiție, iar când o lucrare aștepta terminarea activității unui dispozitiv de I/O, o altă lucrare putea utiliza UCP;

prezența simultană a mai multor lucrări în memorie a impus și existența unui modul hardware care să protejeze fiecare program de a fi spionat sau modificat de celelalte; mașinile seriei 360, precum și altele din generația a III-a, aveau un asemenea modul.



Istoria SO - generația a III-a

Noutăți introduse de OS/360:

• posibilitatea de a transfera programele de pe cartele pe disc imediat ce erau aduse camera calculatorului; când un program se termina de executat, SO încărca altul nou de pe disc în zona de memorie rămasă liberă și apoi îl executa; această tehnică se numește **virtualizare** (**spooling** - Simultaneous Peripheral Operation On Line - operare simultană online a perifericelor) și era folosită și pentru ieșire;

folosind virtualizarea, mașinile auxiliare (IBM 1401) nu au mai fost necesare iar plimbatul benzilor a dispărut.

Istoria SO - generația a III-a

Deși SO din generația a III-a erau potrivite pentru calcule științifice complicate sau prelucrarea unor cantități mari de date comerciale, ele rămâneau sisteme de prelucrare pe loturi de programe.

Dorința de a avea un timp redus de răspuns a determinat apariția **partajării în timp** (**timesharing**), o variantă de multiprogramare în care fiecare utilizator are un terminal online - dacă sunt 20 de utilizatori conectați și 17 se gândesc, UCP este distribuită pe rând celor 3 job-uri care solicită serviciul.

Primul SO cu partajare în timp a fost CTSS (Compatibile Time Sharing System), dezvoltat de M.I.T. pe o mașină 7094 adaptată; partajarea în timp a adevenit însă populară abia după introducerea unui modul hardware pentru protecție.

Istoria SO - generația a III-a

CTSS a avut însă succes, ceea ce a determinat M.I.T., Bell Labs și General Electric să dezvolte MULTICS (MULTiplexed Information and Computing Service - serviciu multiplexat pentru calcule și informații) - el a fost proiectat să deservească o mașină cu un număr mare de dispozitive de I/O.

MULTICS a introdus multe idei de perspectivă și a condus ulterior la realizarea sistemului UNIX, devenit popular în lumea academică, agenții guvernamentale, diverse companii.

Diversele organizații și-au creat propriile versiuni de UNIX, de exemplu:
- System V (de la AT&T)
- BSD (Berkeley Soft Distribution)
- HP-UX

De aceea, IEEE a dezvoltat un standard pentru UNIX numit POSIX, care definește o interfață minimală de apeluri sistem și care pot fi portate ușor de la un SC la altul. În prezent și alte SO suportă interfața POSIX.

În 1987 A.S. Tanenbaum a lansat MINIX, o clonă redusă de UNIX, gratuită, pentru scopuri educaționale. Dorința de a realiza o versiune de MINIX, tot gratuită, dar pentru scopuri de producție, l-a determinat pe Linus Torvalds să realizeze Linux.

Istoria SO - generația a III-a

Tot în cadrul generației a III-a s-a manifestat și evoluția fenomenală a minicalculatoarelor, ca DEC PDP-1, ..., PDP-11 (incompatibile cu familia IBM). Ele erau tot mari, dar mai accesibile, acum fiecare departament al unei instituții putea avea propriul lui minicalculator.

Istoria SO - generația a IV-a

• **Generația a IV-a (1980 - prezent) - calculatoare personale**:

Calculatoarele personale au apărut datorită dezvoltării circuitelor integrate LSI (Large Scale Integration) - puteau conține mii de tranzistori pe cm^2.
Erau asemănătoare ca arhitectură cu minicalculatoarele PDP-11 dar mai ieftine. Astfel, era posibil ca fiecare utilizator să aibe propriul calculator (nu doar departamentele).

SO folosite la început au fost:
- CP/M (Control Program for Microcomputer) - pentru procesoare 8080 (introduse de Intel în 1974), Z80 (de la Zilog), etc.;
- MS-DOS pentru IBM-PC (calculator introdus de IBM în 1981), realizat de Microsoft preluând anumite idei din UNIX într-o formă foarte simplistă.

Istoria SO - generația a IV-a

Piața procesoarelor a fost din ce în ce mai dominată de familiile (compatibile) Intel (286, 386, 486, Pentium-urile) și AMD.

Pentru calculatoarele personale, cele mai folosite SO sunt:

- familia Windows: a preluat idei de interfață prietenoasă de la SO pentru calculatoarele Apple Macintosh (care oferă o interfață grafică cu pictograme, lucru cu mouse, etc.);
  verisiunile până la 3.11 erau doar medii de programare - aplicații ce se lansau din MS-DOS și ofereau o interfață prietenoasă pentru utilizator și cu mai multe facilități pentru rularea programelor;
  de la Windows 95 (98, NT, 2000, Me, XP, Vista) au devenit SO;

- familiile UNIX (comerciale) și Linux (gratuite).

În practică, sistemele Windows sunt folosite extensiv pentru utilizare casnică și entertainment iar sistemele UNIX/Linux pentru servere.

Istoria SO - generația a IV-a

Deși în decursul evoluției calculatoarelor s-a manifestat o tendință de miniaturizare și utilizare la nivel din ce în ce mai restrâns (până la nivel personal), în prezent există o tendință de reapariție a mainframe-urilor - ca servere pentru Internet, pentru comerț electronic, etc.
Acestea conțin tot software-ul necesar iar utilizatorii se pot conecta la ele cu un calculator slab (și o conexiune la Internet bună), având doar rol de terminal.
Un exemplu este proiectul Google de a crea un SO online (GooOS) - pe un calculator f. puternic să se creeze un sistem în care orice om de pe planetă să poată avea propriul cont; sistemul găzduiește fișierele utilizatorului și conține tot software-ul necesar, iar mașina de unde operează utilizatorul este doar un simplu terminal.

Această tendință se explică prin faptul că tot mai mulți utilizatori vor să poată utiliza calculatorul ușor și repede, fără a fi nevoiți să învețe toate cunoștințele necesare administrării unui sistem complex - administrarea este lăsată pe seama profesioniștilor.

Cuprins

Structura SO - sistemul monolitic

Există mai multe modalități de organizare a unui SO:

- **Sistemul monolitic**:

Este cea mai des întâlnită organizare.

Sistemul este o colecție de proceduri în care fiecare procedură poate apela pe fiecare - nu neapărat o și face, depinde cum e scrisă, ideea e că daca vrea să apeleze orice, modul de organizare a sistemului îi permite.

Nu e posibilă ascunderea de informații - fiecare procedură este vizibilă oricărei alta.

Structura SO - sistemul monolitic

Există totuși o structurare minimală și anume:

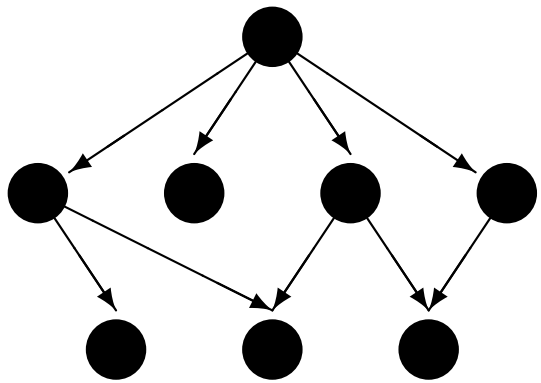
un program cere serviciile (apelurile sistem) oferite de SO punând parametrii într-un loc bine definit (de ex. pe stivă), apoi executând o instrucțiune **trap**, care face ca sistemul să treacă din user mode în kernel mode și transferă controlul către SO;

apoi SO preia parametrii, determină adresa procedurii care implementează apelul sistem și o apelează;

acea procedură poate la rândul ei să apeleze alte proceduri utilitare.

Rezultă următoarea structură:

1. O procedură principală care apelează procedurile de serviciu;
2. Un set de proceduri de serviciu ce implementează apelurile sistem;
3. Un set de proceduri utilitare, apelate eventual de procedurile de serviciu.



procedura principală

proceduri de serviciu

proceduri utilitare

Structura SO - sistemul structurat pe niveluri

- **Sistemul structurat pe niveluri**:

Este o generalizare a abordării din figura precedentă.
Sistemul este organizat ca o ierarhie de niveluri, fiecare fiind construit deasupra nivelului precedent.

Primul SO de acest tip a fost THE, realizat de E.W.Dijkstra și studenții săi ('68) la Technische Hogeschool Eindhoven în Olanda - are 6 niveluri:

Nivel	**Funcție**
5	Operatorul
4	Programe utilizator
3	Administrarea intrărilor/ieșirilor
2	Comunicarea operator - proces
1	Gestiunea memoriei și a rolei magnetice
0	Alocarea procesorului și multiprogramara

Structura SO - sistemul structurat pe niveluri

O generalizare mai extinsă a conceptului de organizare pe niveluri a fost prezentă în sistemul MULTICS (strămoș al UNIX).

El prevedea o serie de inele concentrice (sunt același lucru ca nivelurile), cele interioare fiind mai privilegiate ca cele exterioare.
Când o procedură dintr-un inel exterior dorea să apeleze o procedură dintr-un inel interior trebuia să facă echivalentul unui apel sistem: să execute o instrucțiune TRAP ai cărei parametri erau verificați pentru a fi valizi înainte de permite începerea apelului.
Hardware-ul făcea posibilă protejarea individuală a diverselor proceduri (de fapt segmente de memorie) la citire, scriere, execuție - deci mecanismul inelelor era sprijinit de hardware.

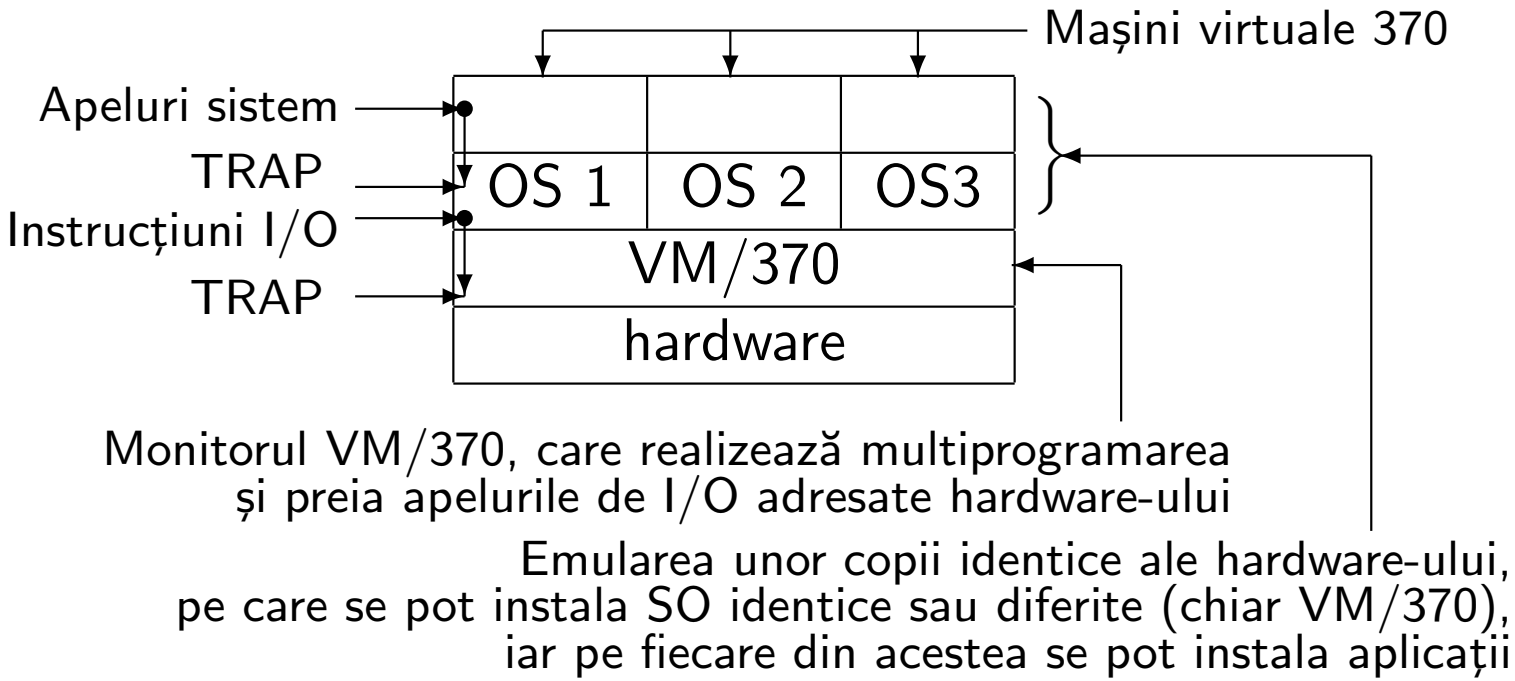
Mecanismul inelelor putea fi extins și pentru a structura programele utilizator: de ex. un profesor putea scrie un program de testare și notare a programelor studenților și să-l ruleze la un nivel n, iar programele studenților ar rula în inelul $n + 1$, astfel încât ei să nu-și poată modifica notele.

Structura SO - mașini virtuale

- **Mașini virtuale**:

Aceste SO au o parte principală, numită **monitorul mașinilor virtuale**, care rulează direct pe hardware, realizează multiprogramarea și oferă mai multe mașini virtuale nivelului următor, pe care se pot instala alte SO ca și când ar fi calculatoare de sine stătătoare.

Primul asemenea sistem a fost VM/370 și avea următoarea structură:



Structura SO - mașini virtuale

Ideea mașinii virtuale este utilizată și azi pentru rularea programelor MS-DOS pe procesoare Pentium sau mai noi - procesoarele Intel pe 32 biti dispun de un mod de lucru numit Virtual 8086, care se comportă ca un procesor vechi 8086; acest mod este folosit de Windows pentru rularea programelor MS-DOS.

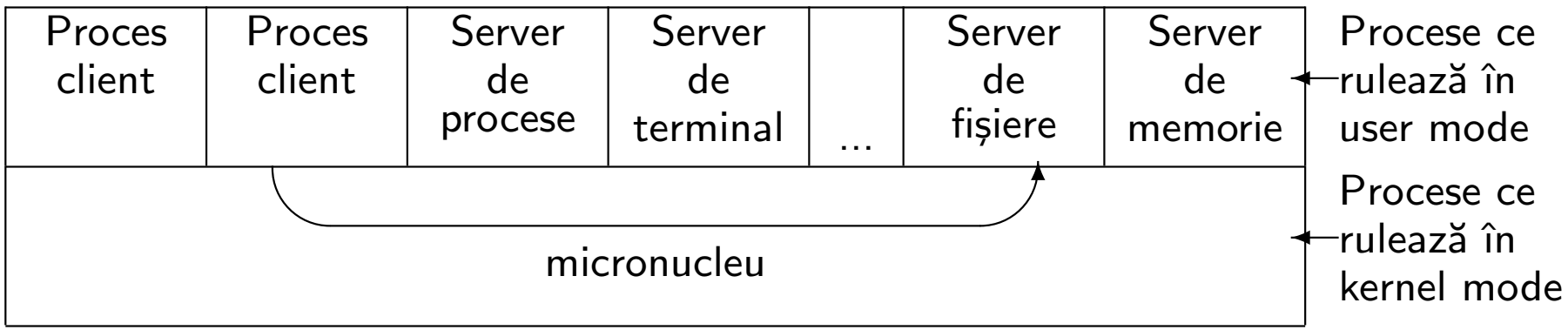
O variantă de SO cu mașini virtuale sunt cele cu **exokernel**.

Structura SO - modelul client - server

- **Modelul client - server**:

O tendință actuală în domeniul SO este de a muta codul în straturile superioare (în procesele utilizator) a.î. kernel-ul să devină cât mai mic (microkernel).

De exemplu, pentru a citi un bloc de date de pe disc, un proces utilizator, numit acum proces **client**, trimite cererea unui alt proces utilizator care funcționează ca **server** de fișiere, iar acesta face citirea și-i trimite înapoi rezultatul.
Rolul kernel-ului este doar de a gestiona comunicarea între procesele client și cele server. Doar kernel-ul ruleaza în kernel mode, procesele client și server rulează în user mode și astfel n-au acces direct la hardware.



Astfel, daca apare o eroare în sistemul de fișiere, serverul de fișiere poate pica, dar restul mașinii funcționează. Un alt avantaj al modelului client - server este că se poate folosi în sisteme distribuite, unde procesele rulează pe mai multe mașini legate în rețea.

Cuprins

Funcțiile SO

Există mai multe puncte de vedere privind setul de funcții minimale pe care trebuie să le ofere un sistem de operare. Un asemenea set ar fi:

• **Gestiunea utilizatorilor și securitatea sistemului**:
Există sisteme:
• **monouser**: nu fac distincție între utilizatorii fizici care se pot conecta; orice persoană care operează în sistem are acces necondiționat la toate resursele oferite de SO, ca și când ar exista un singur utilizator care lucrează mereu - de fapt, conceptul de utilizator logic nu este implementat;
exemplu: MS-DOS;
• **multiuser**: fac distincție între utilizatorii fizici care se pot conecta, prin intermediul conturilor;
în sistem sunt definite un set de conturi, având au un nume de cont (username), o parola (password), niște drepturi, etc., și orice persoană (utilizator fizic), dacă vrea să se conecteze la sistem (logare), trebuie sa indice mai întâi un cont și parola corespunzătoare; odată acceptat, sistemul îl asimilează cu persoana pentru care s-a creat contul respectiv (chiar daca nu este, dar îi cunoaște contul și parola);
conturile sunt utilizatorii logici ai sistemului - în cele ce urmează, prin "utilizator" vom înțelege de fapt un cont.
exemple: UNIX, Linux, Windows, Novell Netware, etc.

Funcțiile SO

Într-un sistem multisuer, orice proces, fișier, etc., are printre caracteristicile sale un proprietar (un cont) și niște drepturi, iar sistemul împiedică o acțiune a unui utilizator să acceseze un obiect al altui utilizator, dacă proprietarul obiectului nu a marcat acolo permisiunile necesare - astfel se asigură securitatea datelor fiecărui utilizator.

De exemplu un proces având ca proprietar utilizatorul $U1$ nu poate citi dintr-un fișier ce are ca proprietar utilizatorul $U2$ dacă $U2$ nu a setat pentru acel fișier permisiunea de citire pentru categoria de utilizatori din care face parte $U1$ (când procesul va încerca să deschidă fișierul în citire cu apelul sistem "open", acest apel va eșua și va returna procesului un cod de eroare).

În general există însă și utilizatori privilegiați (ca de exemplu "root" în sistemele UNIX și Linux) care au automat toate drepturile. Aceștia pot de exemplu crea și distruge alti utilizatori. Un utilizator obișnuit poate face doar modificări minimale sieși, de exemplu schimbarea parolei proprii.

Funcțiile SO

- **Gestiunea fișierelor**:

Fișierele sunt resursa abstractă a dispozitivelor de stocare.

Ele au un **nume** și diverse caracteristici:
- atribute, în sistemele MS-DOS și Windows: Read-only, Archive, System, Hidden;
- proprietar și drepturi de acces, în sistemele multiuser.

Fișierele figurează în **directoare** - structuri de date ce conțin informații despre diverse fișiere și alte directoare; în UNIX, Linux, directoarele sunt implementate tot ca niște fișiere, de un tip special.

Sistemul de fișiere și directoare, legate prin relația de apartenență la un director, formează arborescențe. Un dispozitiv de stocare este văzut ca un ansamblu de **discuri** logice, fiecare conținând câte o asemenea arborescență.

Întrucât numele fișierelor dintr-un sistem nu sunt neapărat unice, un fișier este în general desemnat cu ajutorul unui **specificator** format din disc, **calea** către el în arborescența discului respectiv și apoi numele fișierului.

Funcțiile SO

• **Gestiunea proceselor**:

Procesul este un concept cheie, fundamental, în sistemele de operare.

Proces: execuție a unui program.

Nu trebuie confundată noțiunea de proces cu cea de fișier executabil sau de program.
Mai multe procese diferite pot executa același fișier - atunci, deși procesele execută același program, ele pot avea la un moment dat alte valori pentru variabilele declarate în program, pot fi la o altă instrucțiune curentă, etc.

Există sisteme:
• **monotasking**: pot rula doar un singur proces la un moment dat; deci procesele trebuie rulate pe rând;
  exemplu: MS-DOS;
• **multitasking**: pot rula mai multe procese simultan (multiprogramare); procesele pot rula pe procesoare diferite, sau mai multe pe un același procesor, într-o manieră întrețesută (anume alternativ, câte o porțiune din fiecare);
  exemple: UNIX, Linux, Windows, Novell Neteware;

Funcțiile SO

Caracteristici ale proceselor:
- fișierul pe care îl execută;
- **spațiul de adresare** - o zonă de memorie pe care o poate accesa cu instrucțiunile obișnuite din fișierul (programul) executat;
  aici se rezervă locații pentru valorile proprii curente ale variabilelor din program și pentru stiva proprie;
- diverse informații asociate de SO: proprietarul (un cont), prioritatea, adresa instrucțiunii curente (reținută în registrul IP (arhitec. Intel) sau PC (arhitec. MIPS)), adresa vârfului curent al stivei (reținută în registrul SP), etc.

SO poate întrerupe temporar un proces (de exemplu pentru a executa o parte din alt proces în cadrul multitasking); atunci informațiile folosite la gestionarea lui (inclusiv valorile acelor regiștri) sunt salvate în **tabela proceselor** (un vector de structuri, câte una pentru fiecare proces).

De asemenea, SO gestionează comunicarea între procese și arbitrează accesul concurent al acestora la resurse, pentru a evita **interblocajele**.

Procesele pot lansa procese fiu, organizându-se dupa această filiație în **arborescențe**; pentru aceasta folosesc niște apeluri sistem (de exemplu "fork" în UNIX, Linux). În general SO oferă instrumente pentru crearea, distrugerea, blocarea, rularea proceselor.

Funcțiile SO

- **Gestiunea memoriei**:

O alocare a memoriei la nivel de octet sau cuvânt este ineficientă;
SO implementează algoritmi de gestionare și livrare a memoriei la nivel de blocuri, a căror evidență e menținută în niște liste.

SO arbitrează cererile diverselor procese pentru o aceeași zonă de memorie și asigură că un proces nu poate accesa zonele de memorie alocate altui proces (este un aspect de **securitate**).

SO implementează de asemenea extensii ale memoriei principale sub formă de **memorie virtuală**, folosind capacitățile de memorare ale altor dispozitive (de obicei memoria principală este RAM iar cea virtuală este stocată pe disc) - astfel încât memoria principală să pară mai mare.
Memoria virtuală este văzută logic de procese ca și când ar face parte din memoria principală. În funcție de locul unde se află informațiile de care are nevoie un proces, SO swap-ează (interschimbă) blocuri din memoria RAM cu blocuri de pe disc.

SO permite unui proces să partajeze memoria folosită cu alte procese de pe aceeași mașină sau mașini diferite (prin rețea).

Funcțiile SO

- **Gestiunea dispozitivelor de I/O**:

De obicei SO privesc toate dispozitivele externe (discuri, benzi, terminale, imprimante, etc.) într-un același fel, generic.

El se ocupă de alocarea, izolarea și acordarea de dispozitive în funcție de o anumită politică, ținând cont de anumite priorități.

Când adăugăm la sistemul de calcul un dispozitiv nou, trebuie adăugat la SO driverul necesar; dacă nu avem codurile sursă, nu putem recompila SO a.î. să includă noul driver; această limitare a dus la dezvoltarea unor drivere reconfigurabile.

Funcțiile SO

- **O interfață utilizator**:

Poate fi:
- În mod **linie de comandă**;
- În mod **grafic**;

ea se poate implementa și cu ajutorul unor utilitare instalate în sistem.

MS-DOS are în mod nativ o interfață linie de comandă; i se poate adăuga ca aplicație o interfață grafică (de exemplu Windows 3.11 sau mai vechi).
Windows are interfață grafică, dar are și una linie de comandă prin "Command prompt".
UNIX, Linux implementează ambele interfețe ca aplicații adăugate kernel-ului:
pentru modul linie de comandă utilizează programele shell;
pentru modul grafic utilizează subsistemul X-Windows.

Funcțiile SO

Alte funcții:

- **Un set de apeluri sistem** (pentru interfațarea cu aplicațiile).
- **Facilități privind lucrul în rețea**.

În cursurile următoare vom detalia modul cum sunt realizate toate aceste funcții, conceptele implementate, algoritmii folosiți, atât la modul teoretic general cât și în cazul particular al unor SO concrete.

Cuprins

Considerații de implementare

Procesoarele moderne au un registru de stare în care se reține informații despre procesul curent; printre aceste informații există și un bit care specifică modul de lucru al procesului:
- supervizor (kernel mode)
- utilizator (user mode)

(deci există un dispozitiv hardware care reține modul).
În kernel mode, procesul poate executa orice instrucțiune hardware, iar în user mode doar o parte din ele.
Instrucțiunile care se pot executa doar în kernel mode se numesc instrucțiuni **supervizor** (alte denumiri: **privilegiate**, **protejate**).

De exemplu instrucțiunile de I/O de la periferice sunt privilegiate.
Astfel, un proces care rulează în user mode nu poate executa direct instrucțiuni de I/O; pentru aceasta ele apelează un apel sistem - în acest scop este invocată o instrucțiune hardware specială ce comută procesul în kernel mode și apoi începe executarea driverului dispozitivului de I/O respectiv (care efectuează acele instrucțiuni de I/O).

Considerații de implementare

Partea din SO care este critică pentru efectuarea corectă a operațiilor se execută în kernel mode și formează așa-numitul **kernel** (**nucleu**), iar programele de aplicație se execută în user mode.

Kernel-ul funcționează ca un soft de încredere, adică i s-au incorporat mecanisme de protecție ce nu pot fi schimbate prin acțiuni ale unui alt soft, executat în user mode.
Extensiile SO se execută în user mode a.î. SO nu se bazează pe corectitudinea lor pentru o funcționare corectă.

De aceea, o decizie fundamentală de proiectare pentru orice funcție a SO este dacă trebuie implementată în kernel.
Dacă da, ea se execută în spațiul supervizor și va avea acces la celelalte părți ale kernel-ului.
Dacă nu (și se va executa în user mode), ea nu va avea acces la structurile de date ale nucleului.