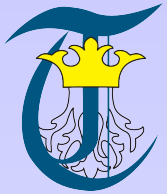


Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare

– curs 2 –

2013.03.06

ș.l. dr. ing. Kertész Csaba-Zoltán

1.3. Concepte de bază ale SO

- Sistemele de operare oferă un mediu de execuție a programelor pe un calculator
  - ușurează munca utilizatorilor
  - ușurează scrierea programelor
  - asigură o rulare eficientă și sigură a programelor
- Serviciile SO sunt definite sub forma unui set de apeluri de sistem care operează asupra unor obiecte software reprezentând componentele principale ale SO

Serviciile SO

- Serviciile SO oferite utilizatorilor pot fi încadrate în:
  - interfața cu utilizatorul
  - executarea programelor
  - operații I/O
  - sistem de fișiere
  - comunicații
  - detecția erorilor
  - alocarea resurselor
  - protecție și securitate

1.3.1. Interfața cu utilizatorul

- orice sistem de operare trebuie să permită accesul utilizatorilor la diferite resurse ale calculatorului prin intermediul unei interfețe cu utilizatorul
- aceasta poate fi de două tipuri:
  - interpretor de comenzi
    - prin intermediul căreia utilizatorul poate introduce comenzi directe către SO
  - interfață grafică
    - prin intermediul căreia utilizatorul poate folosi o reprezentare vizuală și mult mai intuitivă a serviciilor SO

Interpretorul de comenzi

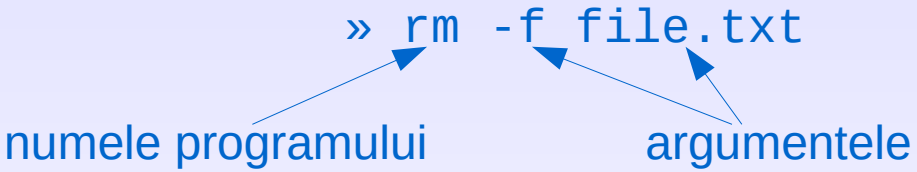
- poate fi o componentă integrată în sistemul de operare
  - sistemul de operare este structurat în jurul unui set de comenzi, acestea putând fi introduse în formă de text și interpretat de nucleul SO
- o altă abordare este folosirea unor interpretoare de comenzi dedicate (shell-uri)
  - sistemul de operare este structurat pe un set de funcții
  - interpretorul de comenzi rulează ca un program simplu, citind comenzile utilizatorilor și traducând în funcțiile SO

Tipuri de shell-uri

- rolul shell-ului este de a executa comenzi
- codul ce trebuie executat poate fi încorporat în shell
  - de ex.: DOS Prompt, BusyBox
  - înțelege un anumit set de comenzi, pentru care este integrat programul ce traduce în apeluri către SO
- shell-ul execută orice comandă primită prin programe dedicate
  - de ex.: shell-urile din Linux – Bourne shell, C shell, Korn shell, Bourne again shell
  - shell-ul execută programul cu numele introdus în linia de comandă

Bash

- cel mai frecvent folosit shell în Linux
- oferă o linie de comandă prin intermediul căreia utilizatorul poate introduce numele unui executabil și argumentele ce trebuie plasate acestuia



- pot fi executate și alte programe în afara celor sistem într-o manieră similară
- comenzile pot fi structurate în funcții (scripturi shell) și pot fi legate împreună (redirectări)

Interfața grafică (GUI)

- oferă o interfață vizuală mai prietenoasă și mai intuitivă pentru utilizator
- comenzile către SO pot fi date printr-o metodă bazată pe mouse și un sistem de ferestre-meniuri-sistem
- interfața grafică este executată similar cu shell, ca un program care traduce acțiunile utilizatorilor în apeluri către sistemul de operare

1.3.2. Apeluri sistem

- programele utilizator comunică cu SO pentru a cere anumite servicii SO prin intermediul apelurilor sistem
- fiecărei serviciu oferit de SO este asociat un set de funcții care rulează pe un nivel privilegiat (cu acces direct la resursele calculatorului)
- apelul sistem este realizat prin intermediul unei întreruperi software (TRAP) prin intermediul căreia se poate trimite o adresă de funcție (asociat serviciului) și parametrii acesteia către SO

Implementarea apelurilor sistem

- pentru a ușura programarea fiecare SO oferă un API (Application Programming Interface) realizat într-o bibliotecă care oferă funcții standard către serviciile SO
- aceste funcții traduc comenzile utilizator în apeluri sistem și realizează trap-ul corespunzător
- SO realizează serviciul cerut și pune rezultatele în registre speciale ce sunt preluate de funcție
- POSIX – API-ul standard ce trebuie să fie oferit de un SO

1.3.3. Procese

- un proces este un program în execuție
- procesul cuprinde:
  - programul executabil
  - datele și stiva programului
  - registrele de uz general
  - registrele speciale: program counter, stack pointer
- toate informațiile despre un proces sunt stocate într-o tabelă (process table) administrat de SO (câte o intrare pentru fiecare proces)

Procese

- crearea și terminarea proceselor se realizează prin apeluri sistem (astfel SO poate actualiza tabelul de procese)
  - de ex. shell citește comenzile de la linia de comandă și creează procese pentru fiecare comandă prin intermediul unui apel sistem, iar la terminarea execuției acesteia este executat un alt apel sistem de terminare
- un proces poate să creeze procese fiu
  - **`fork()`**

Semnalizări între procese

- SO sau alte procese poate să trimită semnale către un proces
- la apariția unui semnal execuția procesului este suspendat temporar, registrele și stiva este salvată și se execută o procedură de tratare a semnalului; după terminarea procedurii, procesul își reia activitatea din punctul unde a fost suspendat
- semnalele sunt echivalente a întreruperilor hardware

Proprietarii proceselor

- într-un sistem multiprogramat este important să cunoaștem proprietarii proceselor
- fiecare utilizator are un identificator unic: **uid**
- utilizatorii de asemenea pot fi împărțiți în grupe având și acestea un identificator: **gid**
- fiecărei proces i se atribuie uid-ul şi gid-ul proprietarului
- aceste identificatori oferă protecția proceselor

1.3.4. Fișiere

- conțin datele într-o formă grupată
- fișierele pot fi grupate la rândul lor în directoare
- SO asigură apeluri de sistem pentru:
  - crearea unui fișier/director
  - ștergerea unui fișier/director
  - scrierea unui fișier
  - citirea unui fișier
  - adăugare unui fișier într-un director
  - eliminarea unui fișier dintr-un director

Mecanisme de protecție

- SO oferă mecanisme de protecție a fișierelor
- În UNIX
  - 9 biți rwx (pentru user, grup, alții)
  - r: posibilitate de citire
  - w: posibilitate de scriere
  - x: posibilitate de execuție a unui fișier / posibilitatea de intrare într-un director
- la apelurile de sistem de deschidere a unui fișier pentru citire/scriere se verifică acești biți

Fișiere speciale

- dispozitivele IO sunt reprezentate în forma unor fișiere speciale
  - astfel se poate utiliza aceleași apeluri sistem pentru accesul la dispozitive ca și la accesul la fișiere
- tipuri de fișiere speciale:
  - block special file: utilizate la modelare dispozitivelor constând dintr-o colecție de blocuri adresabile aleator
  - character special file: utilizate la modelarea dispozitivelor constând din fluxuri de caractere

Fișiere speciale

- fiecare proces are 3 descriptoare de fișiere deschise mereu
  - stdin (Standard Input)
  - stdout (Standard Output)
  - stderr (Standard Error)
- pipe: este un fișier special care conectează două procese

1.3.5. Operații I/O

- un rol important a SO este de a ascunde hardware-ul de utilizator:
  - siguranță crescută în funcționare
  - independență de platforma hardware
- aplicațiile trebuie să acceseze dispozitivele hardware prin intermediul unor apeluri sistem
- SO oferă și anumite nivele de abstractizare a dispozitivelor hardware – HAL (Hardware Abstraction Layer)
  - nivel superior (ex dispozitive de stocare, afișare)
  - nivel inferior (acces direct)

Accesul direct la dispozitive I/O

- în Linux accesul la dispozitive I/O se realizează prin intermediul sistemului de fișiere
- SO asociază fiecărei dispozitiv un fișier special, utilizatorul poate să scrie/citească la/de la dispozitiv scriind/citind fișierul respectiv
- SO controlează accesul utilizatorului pentru a oferi o fiabilitate crescută a sistemului

1.3.6. Comunicații

- procesele utilizatorilor uneori au nevoie de a interacționa între ele
- SO asigură modalități de comunicații între procese (într-o manieră sigură și fiabilă)
  - procesele nu pot accesa direct datele celuilalt, comunicarea realizându-se prin mesaje ce trec prin SO
  - sincronizare la trimiterea și recepția mesajelor
- comunicația poate fi extinsă și pe procese rulate pe alte calculatoare, pentru care SO integrează și protocoale de comunicații

1.4. Structura SO

- SO este alcătuit dintr-o colecție de proceduri, fiecare putând apela oricare alta
- aceste proceduri sunt compilate și legate împreună
- nu există o protecție a informației între aceste proceduri
- apelurile sistem asigurate de SO sunt apelate plasând parametrii în registrii sau stivă și apoi se execută un trap numit **kernel call**

Structura SO

- programele utilizator se rulează în **user mode**, iar procedurile în **kernel mode**, trecerea făcându-se prin aceste trapuri
- programele din user mode nu pot accesa direct procedurile din SO, astfel oferind o protecție a informațiilor din sistem
- SO examinează registrele încărcate înainte de trap și caută din tabela de proceduri cea corespunzătoare parametrilor

1.4.1. Organizare SO în nivele

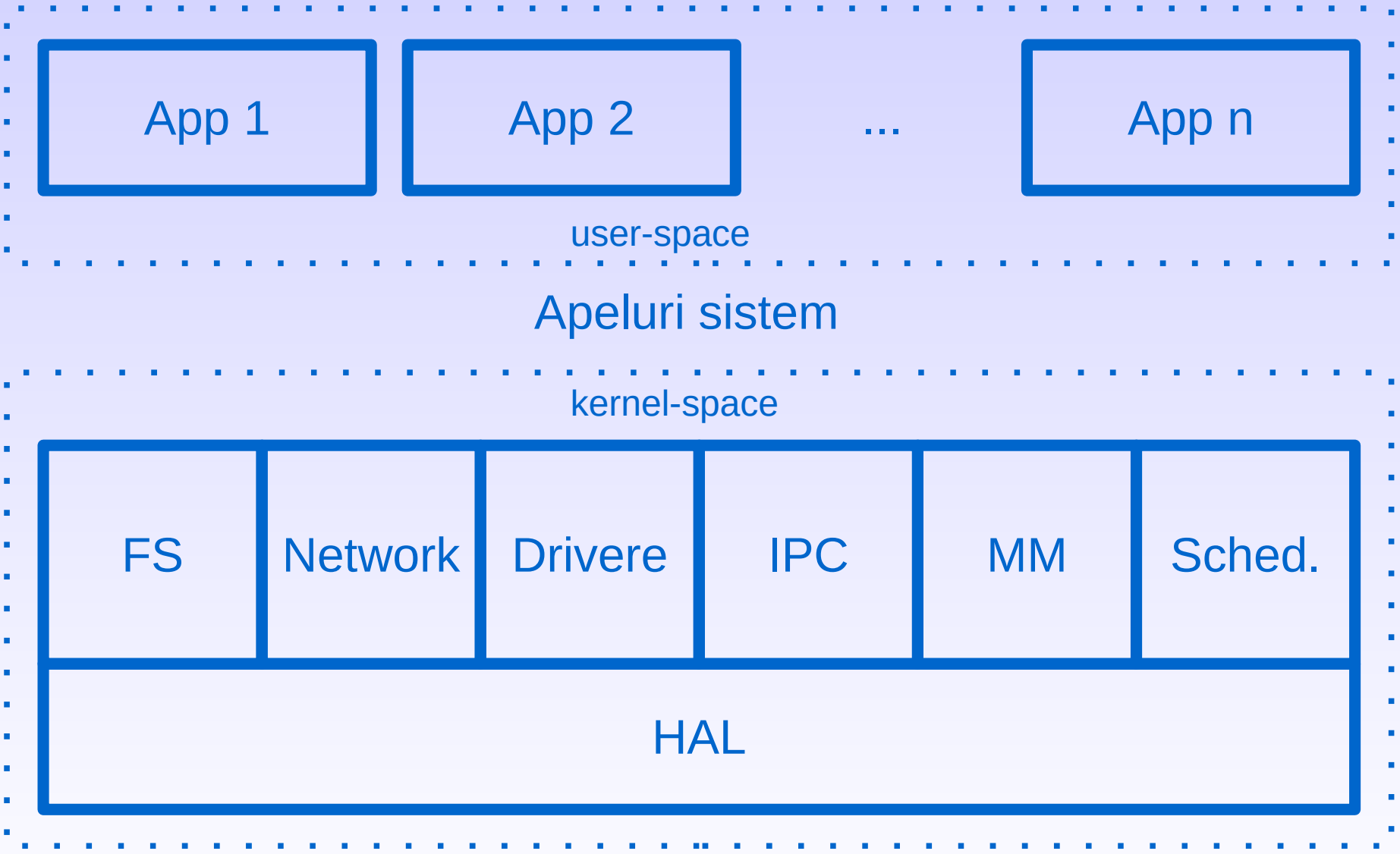
- organizarea SO în nivele a fost propus de Dijkstra în 1968
- accesul de la un nivel mai puțin prioritar către un nivel mai prioritar se face cu un trap

5	Operatorul
4	Programe utilizator
3	I/O management
2	Comunicații operator – proces
1	Memory management
0	Alocare procesor și multiprogramare

Implementare SO pe nivele

- avantajul implementării pe nivele este protecția și fiabilitatea crescută a sistemului
  - nivelele superioare nu pot accesa direct sistemul, evitând problemele datorate malfuncționării programelor utilizator
- dezavantajul este greutatea implementării acestei separări între nivele
  - de obicei arhitecturile hardware nu oferă suport pentru multe nivele software
  - arhitectura x86 oferă 4 nivele de privilegiere, din care efectiv și eficient se poate folosi 2, de aceea este preferat o implementare SO pe 2 nivele

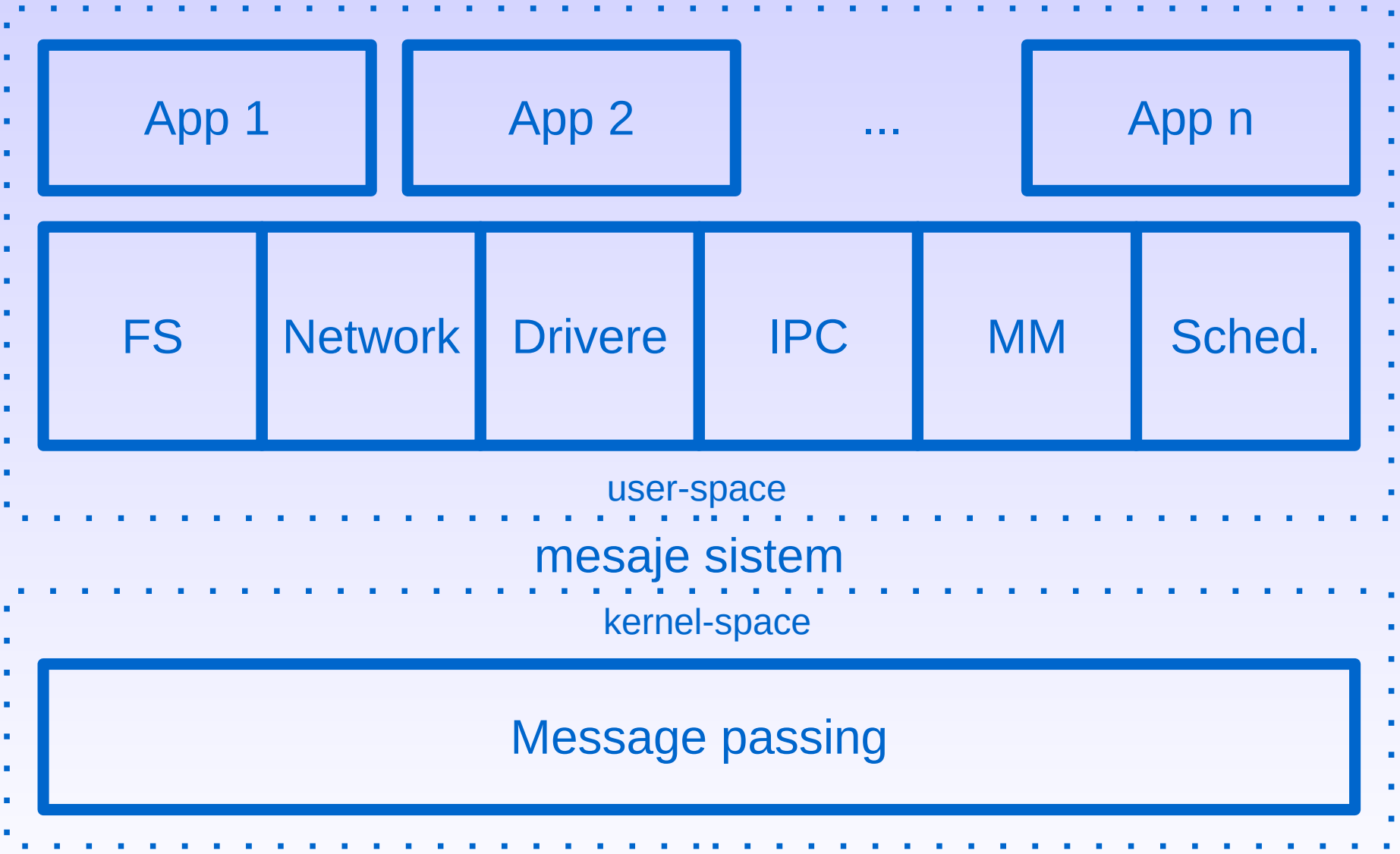
1.4.2. Kernel monolitic



Caracteristici kernel monolitic

- distincție între user-space și kernel-space
  - aplicațiile rulează în user-space
  - nucleul și driverele rulează în kernel-space
  - între ele există un strat de apeluri sistem
- suportă sistem complexe cu multe aplicații
- oferă protecție între aplicații
  - dacă o aplicație se blochează sau funcționează incorect atunci nu afectează pe celelalte
- poate rula și pe arhitecturi fără MMU, dar cu protecție limitată

1.4.3. Microkernel



Caracteristici microkernel

- teoretic cea mai performantă arhitectură de SO
- practic implementările existente prezintă prea multe limitări din cauza complexității
- toate funcțiile SO rulează în user-space în afara unui strat foarte subțire de message passing
- sistemul are un overhead mare datorită mulțimii de mesaje ce trebuie să treacă din user-space în kernel-space și invers

1.4.4. Modelul client-server

- SO oferă funcțiile sale în forma unui server care rulează în user mode
- se minimizează nucleul sistemului de operare
- pentru a realiza un system call, procesul apelant trimite cererea către un alt proces (server) ce va realiza acțiunea dorită și întoarce răspunsul
- este ușor adaptabil sistemelor distribuite: SO și programele nici nu trebuie să fie pe aceleași calculatoare

Implementare SO client-server

- pentru a putea avea o rulare pseudoparalelă a mai multor procese pe un singur procesor, trebuie să existe măcar un planificator de procese
- implementările reale de obicei se bazează pe un SO existent, care este extins cu funcționalități client-server
- de ex.: Linux poate fi extins pe o structură client-server, în care majoritatea funcționalității este realizat de servere (fișiere, I/O, rețea)

1.5. Mașini virtuale

- abstractizarea hardware-ului poate fi extins la un nivel la care practic sunt create medii de execuție complete pentru diferite programe
  - programele astfel vor avea iluzia că dețin un calculator complet
  - chiar și părți din SO pot fi transferate în aceste medii de execuție
- permite o implementare mai simplă a programelor:
  - nu trebuie să aibă grijă de resurse partajate
  - platforma virtuală pe care rulează este identică indiferent de platforma reală