Aspecte generale ale sistemelor de operare

Sisteme de Operare

Ciprian Oprișa și Adrian Coleșa

Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Cursul 1

Cuprins

Informații utile I

1. Laborator (2 ore / săptămână / semigrupă)
   - Grupa 30221: vineri 16-20 cu Alexandru Jîrcan
   - Grupa 30222_1: marți 14-16 cu Alexandra Mitrea
   - Grupa 30222_2: marți 12-14 cu Ioan Ghiorghiu
   - Grupa 30223: vineri 8-12 cu Dan Domnița
   - Grupa 30224: marți 16-20 cu Adrian Tirea
   - Grupa 30225: joi 16-20 cu Leonard Pîrvu
   - Grupa 30226: luni 16-20 cu Alexandru Boțolan
   - Grupa 30227: luni 8-12 cu Aralda Păcurar
   - Grupa 30228: luni 12-16 cu Ciprian Oprișa
   - Grupa 30229: miercuri 12-16 cu Vlad Olteanu
   - Grupa 302210: miercuri 16-20 cu Andrei Sântoma

Informații utile II

2. Curs (2 ore / săptămână): Ciprian Oprișa
   - Seria A: miercuri 18-20 (Aula Instalații)
   - Seria B: marți 14-16 (Sala 40)
3. Pagina cursului pe Moodle:
   `https://moodle.cs.utcluj.ro/course/view.php?id=737`
   - dacă nu aveți cont, creați unul
   - atenție la *firstname* și *surname* (nume de familie)
   - înregistrați-vă la cursul "*Sisteme de operare 2025*"
   - chei de înrolare:
     - pentru grupa 3022X: `So.2025@X`, $X \in \{1, 2\_1, 2\_2, 3, \ldots, 10\}$
4. MS Teams: **SO_2024_2025**
   - team code: `qj60fvi`

Scop și obiective

Scop

înțelegerea conceptelor fundamentale și a funcționalității în sistemele de operare moderne

Scop și obiective

Scop

înțelegerea conceptelor fundamentale și a funcționalității în sistemele de operare moderne

- Obiective specifice
  1. înțelegerea rolului SO-ului și funcționalitatea componentelor sale
  2. familiarizarea cu apelurile de sistem dintr-un SO
  3. cunoștințe generale despre mecanismele interne ale SO

Scop și obiective

Scop

înțelegerea conceptelor fundamentale și a funcționalității în sistemele de operare moderne

- Obiective specifice
  1. înțelegerea rolului SO-ului și funcționalitatea componentelor sale
  2. familiarizarea cu apelurile de sistem dintr-un SO
  3. cunoștințe generale despre mecanismele interne ale SO
- Metode
  - prezentarea celor mai importante componente ale SO din punct de vedere **extern** (interfață) și intern (design)
  - rezolvarea de probleme
  - lucrări practice (în general programe C) pe un SO real: **Linux**

Conținutul cursului

1. Introducere
2. Interpretorul de comenzi
3. Interacțiunea programelor cu SO
4. Sistemul de fișiere
5. Procese și thread-uri
6. Sincronizare
7. Mecanisme de comunicare între procese
8. Management-ul memoriei

Surse de documentare

[MOS] A. Tanenbaum și H. Bos, *Modern Operating Systems*, 4th Edition, Pearson Education, 2015

[LAB] C. Oprișa și A. Colesa, *Sisteme de operare - îndrumător de laborator*, UTPress, 2021

[SEM] A. Downey, *The Little Book of Semaphores*, 2016

[OSC] A. Silberschatz, P.B. Galvin și G. Gagne, *Operating System Concepts*, 8th edition, Wiley Publishing, 2009

Abilități vizate

Înainte de curs (*prerequisites*)

- cunoștințe generale despre calculatoare
- programare în C

Abilități vizate

Înainte de curs (*prerequisites*)

- cunoștințe generale despre calculatoare
- programare în C

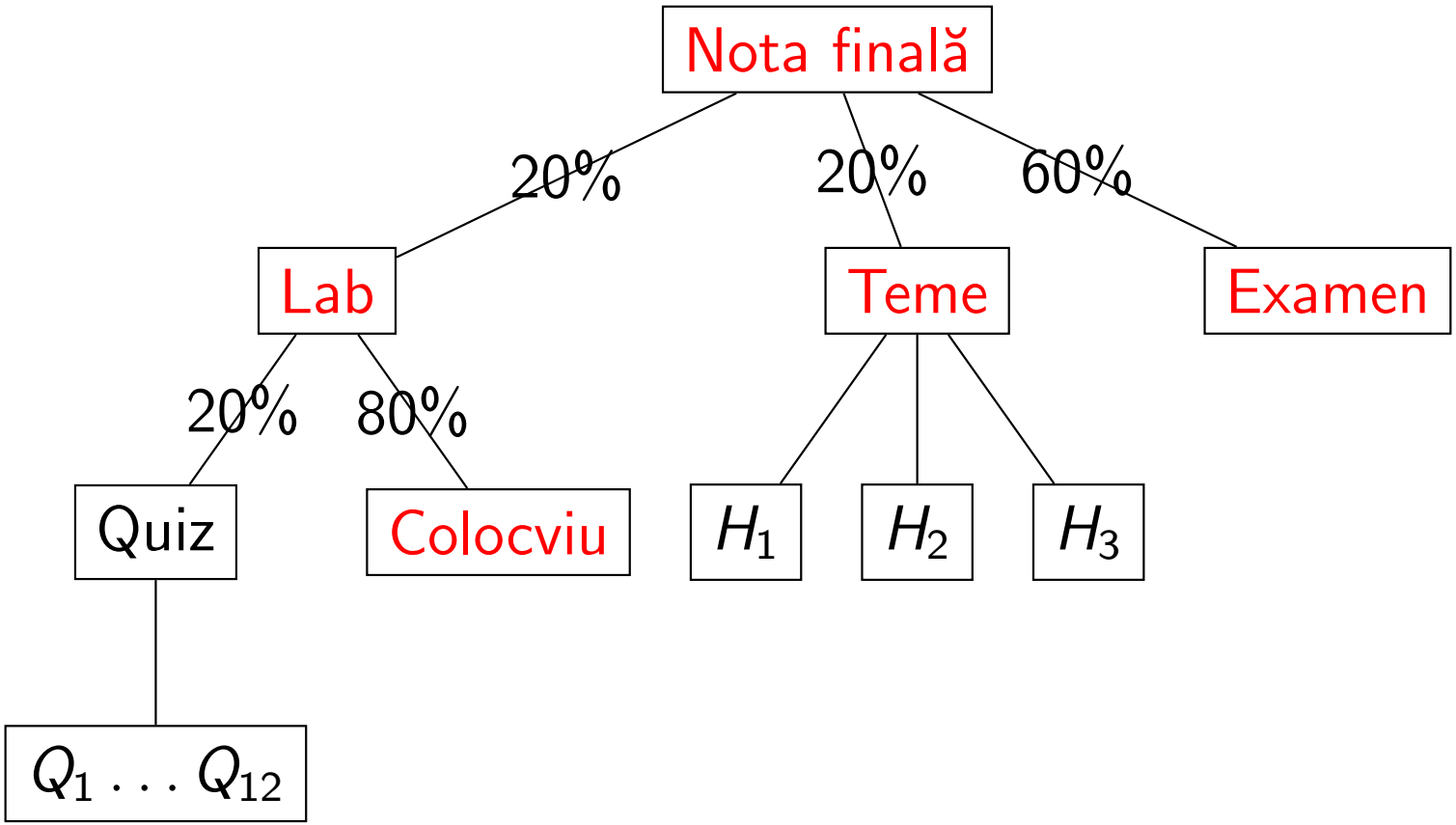
La finalul cursului

- abilitatea de a explica rolul și conceptele fundamentale ale unui sistem de operare
- familiaritatea cu componentele principale ale unui SO; înțelegerea rolului și a funcționalității lor
- familiaritatea cu cele mai comune apeluri de sistem din Linux
- abilitatea de a scrie programe C care folosesc apeluri de sistem
- abilitatea de a folosi terminalul și de a scrie script-uri simple

Cerințe

1. interes pentru domeniu
2. receptivitate la întrebările/propunerile profesorului
3. perseverență de-a lungul semestrului
4. participare la cursuri
   - se va da un quiz scurt după fiecare curs
   - $\geq 40\%$ din punctaj pentru a intra în examen (vară)
   - $\geq 35\%$ pentru a putea da examenul în sesiunea de restanțe
   - $< 35\% \Rightarrow$ recontractare!
5. participare la laboratoare
   - o absență: se face lucrarea acasă și se prezintă data viitoare
   - 2-3 absențe: recuperare în săpt. 13 sau în sesiunea de restanțe
   - 4 sau mai multe absențe $\Rightarrow$ recontractare!

Evaluare și notare



Elementele cu roșu trebuie să fie ≥ 5 pentru a trece.

Cuprins

Ce este un sistem de operare?

- un sistem de calcul modern este foarte complex
  - doar pentru a putea interacționa direct cu un hard-disk SATA ar trebui să citim un manual de 450+ pagini

Sistemul de operare este un **software** (*system software*) care:

- abstractizează sistemul de calcul și oferă programelor o interfață mai bună / simplă / curată
- gestionează resursele hardware

Ce este un sistem de operare?

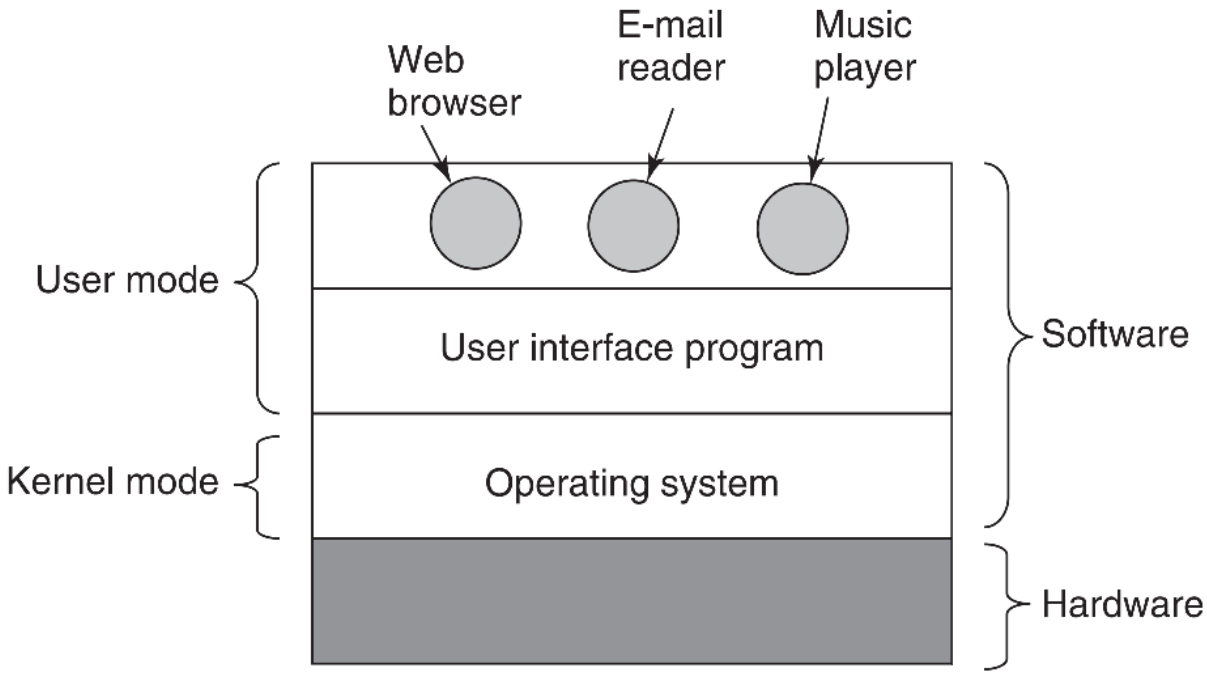
- un sistem de calcul modern este foarte complex
  - doar pentru a putea interacționa direct cu un hard-disk SATA ar trebui să citim un manual de 450+ pagini

Sistemul de operare este un **software** (*system software*) care:

- abstractizează sistemul de calcul și oferă programelor o interfață mai bună / simplă / curată
- gestionează resursele hardware

- probabil ați interacționat deja cu Windows / OS X / Ubuntu / Android / iOS
  - a nu se confunda interfața utilizator (grafică sau text) cu SO

Plasarea sistemului de operare

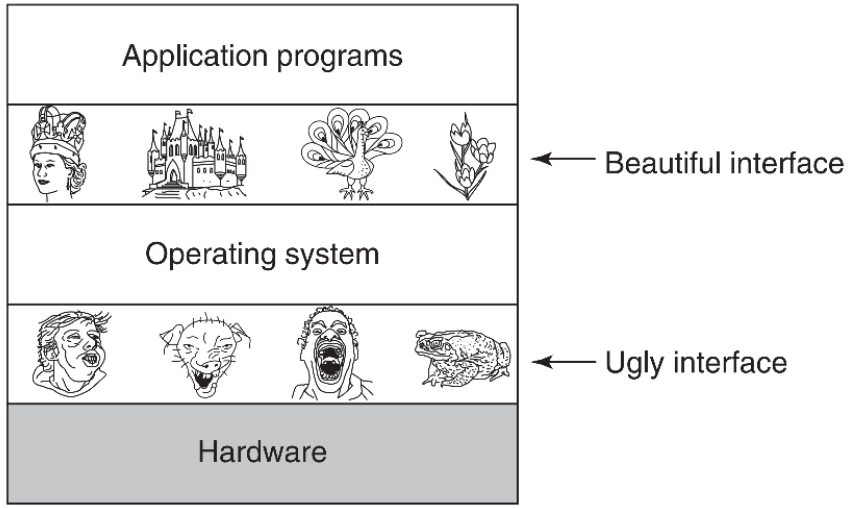


figură preluată din [MOS]

- *kernel mode* (mod privilegiat) – acces complet la hardware, se poate executa orice instrucțiune
- *user mode* – set de instrucțiuni limitat (ex. nu avem `in`, `out`)

Sistemul de operare ca mașină extinsă

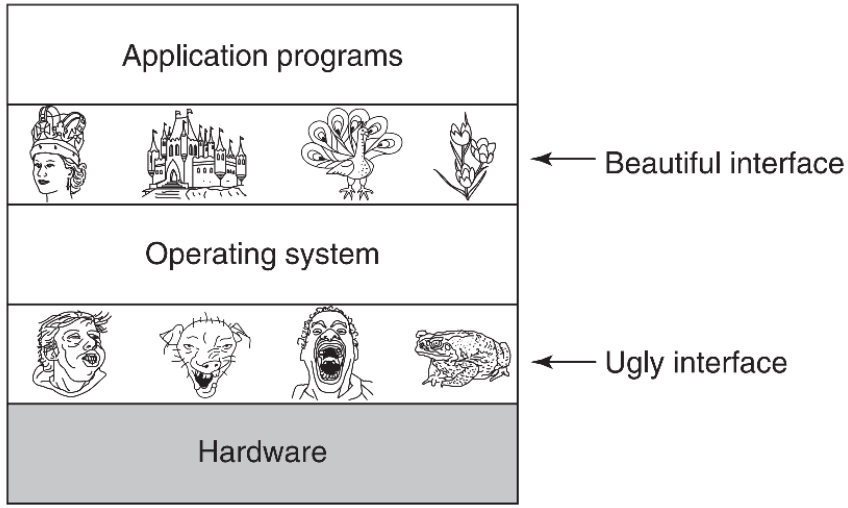
- aplicațiie “văd” niște **abstractizări** ale sistemului de calcul
- se ascunde complexitatea hardware-ului



figură preluată din [MOS]

Sistemul de operare ca mașină extinsă

- aplicațiile "văd" niște **abstractizări** ale sistemului de calcul
- se ascunde complexitatea hardware-ului



figură preluată din [MOS]

- exemplu: o aplicație vrea să scrie într-un fișier
  - aplicația folosește apelul de sistem `write`
  - SO preia cererea de scriere și verifică dacă e validă
  - SO calculează pe ce sector de pe disc trebuie să ajungă datele
  - SO comunică efectiv cu hard-disk-ul pentru a realiza scrierea

Sistemul de operare ca manager de resurse

- resurse comune într-un sistem de calcul: procesor, memorie, disc, mouse, tastatură, interfețe de rețea, imprimantă, cameră video, etc.

Sistemul de operare ca manager de resurse

- resurse comune într-un sistem de calcul: procesor, memorie, disc, mouse, tastatură, interfețe de rețea, imprimantă, cameră video, etc.
- ce se întâmplă dacă mai multe procese vor să folosească imprimanta în același timp?

Sistemul de operare ca manager de resurse

- resurse comune într-un sistem de calcul: procesor, memorie, disc, mouse, tastatură, interfețe de rețea, imprimantă, cameră video, etc.
- ce se întâmplă dacă mai multe procese vor să folosească imprimanta în același timp?
- multiplexarea resurselor
  - în spațiu (ex. fiecare proces folosește o porțiune din memorie)
  - în timp (ex. fiecare proces folosește pe rând procesorul)
- SO trebuie să ofere protecție
  - hardware-ul trebuie protejat de aplicații
  - SO trebui protejat de aplicații
  - aplicațiile trebuie protejate una de alta

Avem nevoie de SO?

Avem nevoie de SO?

- **teoretic NU**
  - putem scrie aplicații care rulează direct peste hardware
  - ex: o stație meteo (simplă)

Avem nevoie de SO?

- **teoretic NU**
  - putem scrie aplicații care rulează direct peste hardware
  - ex: o stație meteo (simplă)
- în practică avem nevoie de beneficiile oferite de SO
  - interacțiunea cu hardware-ul e dificilă
  - rularea simultană a mai multor aplicații e dificilă
  - programatorul ar trebui să se poată concentra doar pe logica aplicației
- $\Rightarrow$ orice sistem de calcul de complexitate non-trivială are nevoie de un SO

Caracterisitici dezirabile pentru SO

- **Util**
  - multe funcționalități
  - să furnizeze tot ce e necesar pentru aplicații
- **Invizibil**
  - eficient: să nu aducă un impact de performanță
  - ușor (*light*): să nu folosească resursele
  - flexibil: fără restricții în materie de interfețe sau abstracții
  - generic: să suporte perfect orice tip de aplicație

Caracterisitici dezirabile pentru SO

- **Util**
  - multe funcționalități
  - să furnizeze tot ce e necesar pentru aplicații
- **Invizibil**
  - eficient: să nu aducă un impact de performanță
  - ușor (*light*): să nu folosească resursele
  - flexibil: fără restricții în materie de interfețe sau abstracții
  - generic: să suporte perfect orice tip de aplicație

Nu există un SO bun la toate!

Scurtă istorie a sistemelor de operare

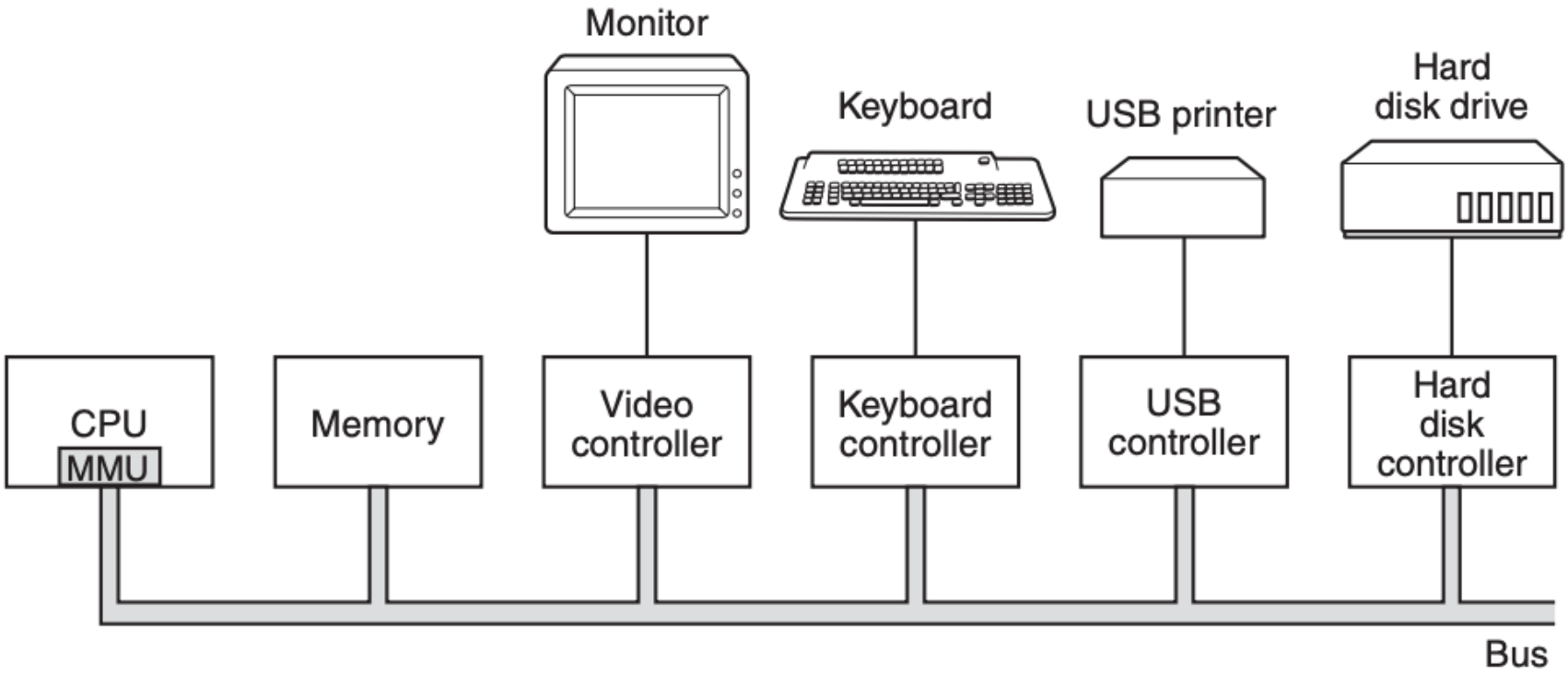
- evoluția SO a fost strâns legată de evoluția hardware-ului
  - hardware mai complex $\rightarrow$ SO mai complex
- anii '60-'70 Ken Thompson și Dennis Ritchie inventează Unix la Bell Labs
- anii '80: MS-DOS, Windows 1.0, Mac OS
- 1988: standardul POSIX
- 1991: Linus Torvalds lansează Linux
- 1995: Windows 95
- 2001: Windows XP (bazat pe Windows NT)
- 2007: iOS (bazat pe macOS)
- 2008: Android (bazat pe Linux)
- 2021: Fuchsia (proiectat de la 0)

Tipuri de sisteme de operare

- SO pentru mainframe-uri: *OS/390*, mai nou înlocuite de *Unix* / *Linux*
- SO pentru servere: *Unix*, *Linux*, *Windows Server*,
- SO pentru PC-uri: *Windows 10/11*, *OSX*, distribuții *Linux* (*Ubuntu*)
- SO mobile: *Android*, *iOS*
- SO pentru sisteme embedded: *Embedded Linux*, *QNX*, *VxWorks*
- SO real-time: *eCos*
- SO pentru smart cards: un card bancar sau o cartelă telefonică pot rula un SO (și chiar Java)

Cuprins

Organizarea componentelor hardware



figură preluată din [MOS]

Procesorul (CPU)

- Funcționalitate: **execută instrucțiuni**
  - *fetch* $\rightarrow$ *decode* $\rightarrow$ *execute*
- arhitecturi diverse (pipeline, superscalare)
  - mai multe detalii la *Arhitectura Calculatoarelor*

Procesorul (CPU)

- Funcționalitate: **execută instrucțiuni**
  - *fetch* $\rightarrow$ *decode* $\rightarrow$ *execute*
- arhitecturi diverse (pipeline, superscalare)
  - mai multe detalii la *Arhitectura Calculatoarelor*
- setul de instrucțiuni
  - fiecare tip de procesor (x86, ARM, MIPS, ...) are un set diferit de instrucțiuni
  - instrucțiuni pentru transfer de date / aritmetico-logice / controlul fluxului de execuție / intrare-ieșire

Procesorul (CPU)

- Funcționalitate: **execută instrucțiuni**
  - *fetch* → *decode* → *execute*
- arhitecturi diverse (pipeline, superscalare)
  - mai multe detalii la *Arhitectura Calculatoarelor*
- setul de instrucțiuni
  - fiecare tip de procesor (x86, ARM, MIPS, ...) are un set diferit de instrucțiuni
  - instrucțiuni pentru transfer de date / aritmetico-logice / controlul fluxului de execuție / intrare-ieșire
- regiștri
  - regiștri de uz general (x86: `EAX`, `BL`, `CH`, `ESI`, ...)
  - Program Counter (x86: `EIP`)
  - Stack Pointer (x86: `ESP`)
  - Program Status Word (x86: `EFLAGS`)

Moduri de execuție ale procesorului

Kernel mode – modul privilegiat

- acces la setul complet de instrucțiuni
- nucleul sistemului de operare rulează în acest mod

User mode – modul neprivilegiat

- acces la un subset limitat de instrucțiuni
- nu avem acces la instrucțiuni de intrare-ieșire
- nu putem încălca protecția memoriei
- aplicațiile utilizator și o parte din SO rulează în acest mod

Schimbarea între moduri de execuție

De unde știe procesorul că execută cod din SO sau din aplicații?

Schimbarea între moduri de execuție

De unde știe procesorul că execută cod din SO sau din aplicații?

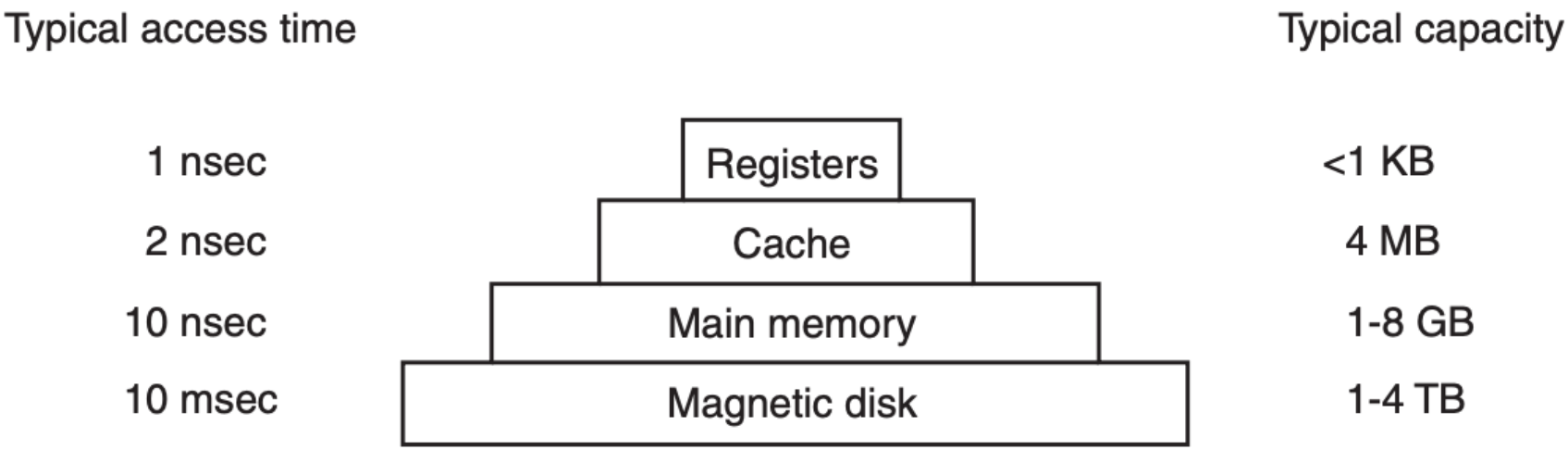
- un flag din *Program Status Word* indică modul de execuție
- când SO cedează controlul procesorului unei aplicații, trece în *user mode*
- se trece înapoi în *kernel mode* când
  - se face un apel de sistem (aplicația face o cerere explicită către sistemul de operare)
  - aplicația produce o excepție (împărțire la 0, instrucțiune nepermisă, acces la memorie neaccesibilă, ...)
  - întreruperi hardware (ex. *timer interrupts*;)

Memoria

- Funcționalitate: **stochează cod și date**
- În mod ideal trebuie să fie foarte rapidă, mare și ieftină

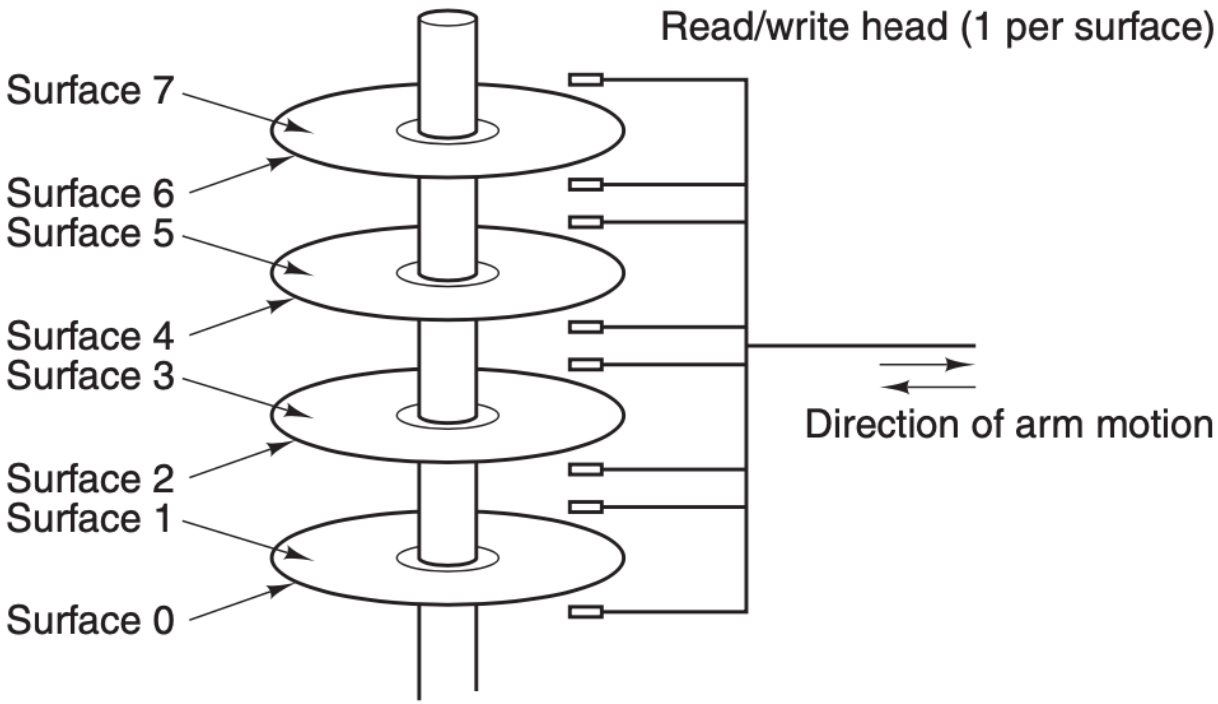
Memoria

- Funcționalitate: **stochează cod și date**
- În mod ideal trebuie să fie foarte rapidă, mare și ieftină



figură preluată din [MOS] - valorile numerice sunt aproximări

Hard disk-ul



figură preluată din [MOS]

- discurile magnetice au componente mecanice
  $\Rightarrow$ accesul aleator este mult mai lent decât accesul secvențial
- SSD-urile sunt adresate electronic, similar cu memoriile

Realizarea intrărilor și ieșirilor (1)

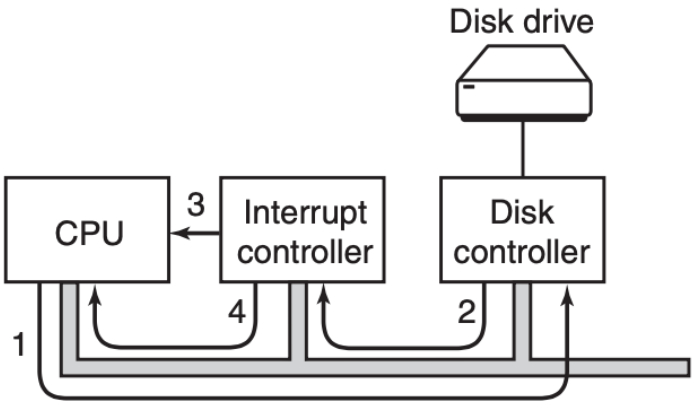
Actori

- dispozitivul propriu-zis (ex. hard disk, tastatură, cameră web)
- controller-ul
  - controlează direct dispozitivul fizic
  - primește comenzi de la SO
- driver-ul
  - de obicei rulează în kernel mode, fiind inserat în SO
  - generic sau furnizat de producător

- accesul la date pentru un dispozitiv periferic (ex. hard disk) poate fi cu câteva ordine de mărime mai lent decât accesul la datele din memorie

Realizarea intrărilor și ieșirilor (2)

- busy waiting
  - driver-ul transmite comanda de I/O, apoi verifică în buclă dacă s-a efectuat
  - dezavantaj: ține procesorul ocupat
- utilizarea întreruperilor (cu/fără DMA)



figură preluată din [MOS]

1. driver-ul transmite comanda de I/O controller-ului
2. după ce transferul de date s-a încheiat, controller-ul de disc anunță controller-ul de întreruperi
3. procesorul este "întrerupt" (dacă acceptă întreruperi)
4. controller-ul de întreruperi transmite detalii despre care device a terminat

Cuprins

Procese

Un **proces** este un program în execuție.

Un proces are

- un spațiu de adrese: locații de memorie pe care le poate accesa
  - cod executabil
  - date
- resurse: regiștri, fișiere deschise

Procese

Un **proces** este un program în execuție.

Un proces are

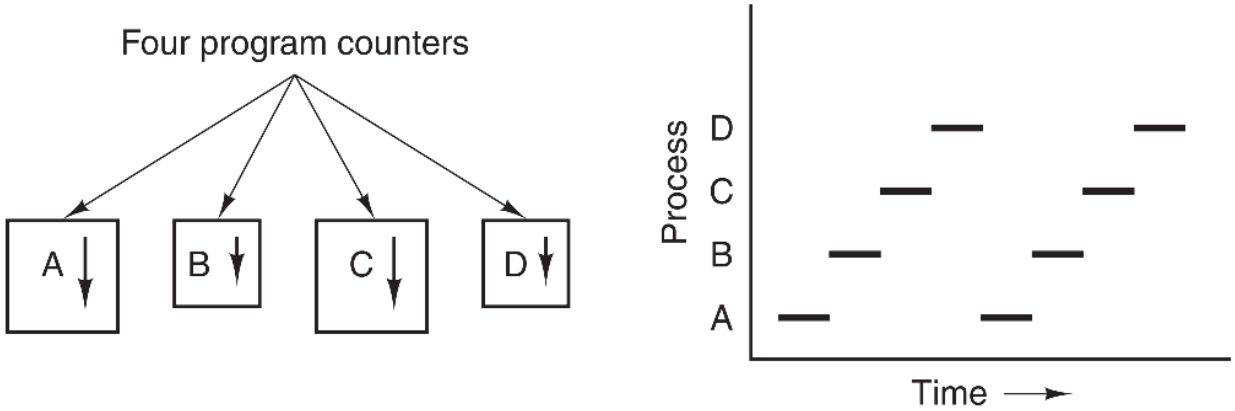
- un spațiu de adrese: locații de memorie pe care le poate accesa
  - cod executabil
  - date
- resurse: regiștri, fișiere deschise

Concepte înrudite:

- *multiprogramming*: mai multe programe pot rula simultan
- *Inter-Process Communication (IPC)*

Schimbarea de context

- de obicei rulează mai multe procese în paralel
  - paralelism real, dacă avem mai multe procesoare / core-uri
  - **pseudoparalelism** - multiplexarea procesorului în timp

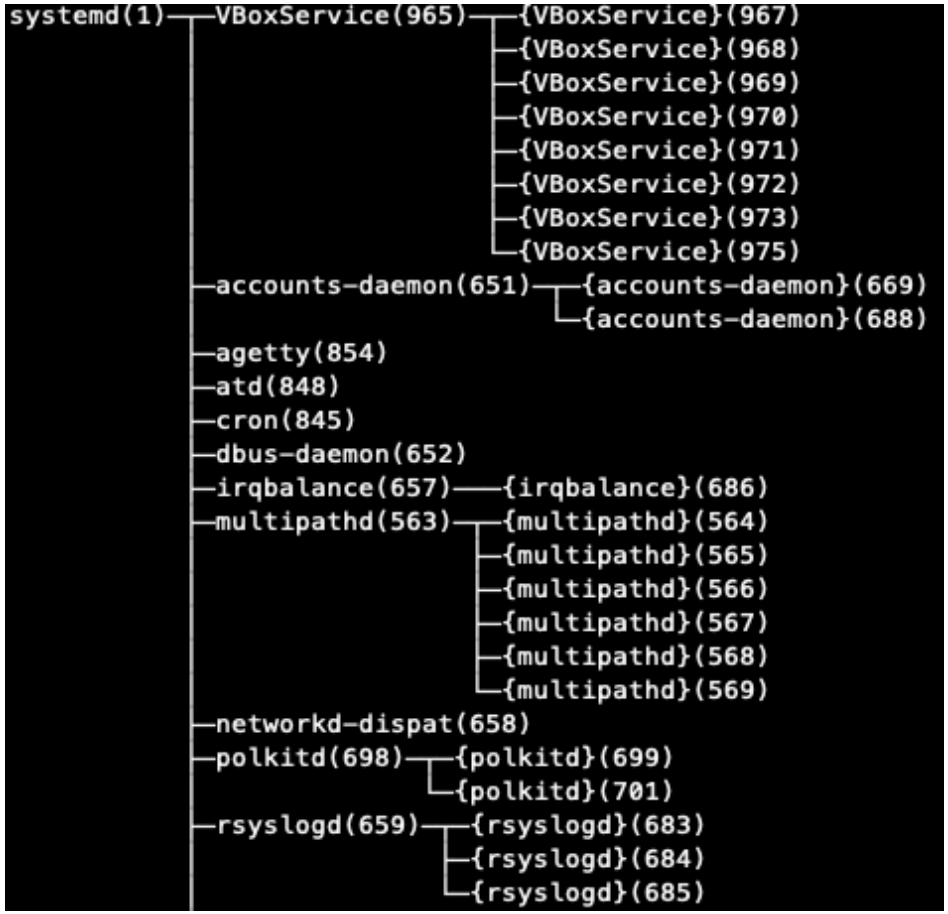


figură preluată din [MOS]

- fiecare proces primește cuante de timp (de ordinul ms)
- când se schimbă procesul activ, cel curent e suspendat și i se salvează starea (ex. valorile regiștrilor)
- la reluarea execuției, continuă în mod transparent

Ierarhia de procese

- în general, procesele au o durată limitată
  - sunt create
  - își încheie execuția (voluntar sau nu)
- un proces poate crea alte procese
- apare o relație părinte-copil



Utilizator

- utilizator ca persoană
- utilizator ca entitate (*user account*)

Utilizator

- utilizator ca persoană
- utilizator ca entitate (*user account*)

Sisteme multi-utilizator

- fiecare utilizator are un id unic (`UID`)
- utilizatorii pot face parte din grupuri
- utilizatorii trebuie să se autentifice
- ideea de *ownership*: procesele și fișierele aparțin utilizatorilor
- mecanism de protecție în SO: permisiuni
  - fișierul F aparține utilizatorului X
  - alți utilizatori au dreptul să îl citească dar nu îl pot modifica
- utilizator privilegiat: *superuser* / *root* / *administrator*

Spațiul de adrese

- mai multe procese pot fi simultan în memorie (multiplexare în spațiu)
- SO trebuie să ofere protecție
  - dacă P_1 conține un bug și scrie din greșeală în memoria lui P_2?

Spațiul de adrese

- mai multe procese pot fi simultan în memorie (multiplexare în spațiu)
- SO trebuie să ofere protecție
  - dacă P_1 conține un bug și scrie din greșeală în memoria lui P_2?
- fiecare proces are un **spațiu de adrese** (virtuale) pe care le poate accesa
- SO (cu suport hardware) asociază adresele virtuale cu adrese în memoria fizică

Sistemul de fișiere

- **fișierul**
  - elementul de bază în stocarea persistentă a datelor
  - concept abstract: SO asociază fișierul cu blocurile reale de pe disc unde se stochează datele
- **directorul** (*folder*)
  - un tip special de fișier, util pentru a organiza sistemul de fișiere
  - directoarele dau naștere unei ierarhii / unui arbore (graf)
  - căi (*path*-uri) obținute prin concatenarea numelor de directoare, folosind un separator (Windows: '\', Linux: '/')
- alte tipuri de fișiere speciale
  - legături simbolice (*symlink*)
  - pipe-uri
  - fișiere de tip bloc / caracter

Apeluri de sistem

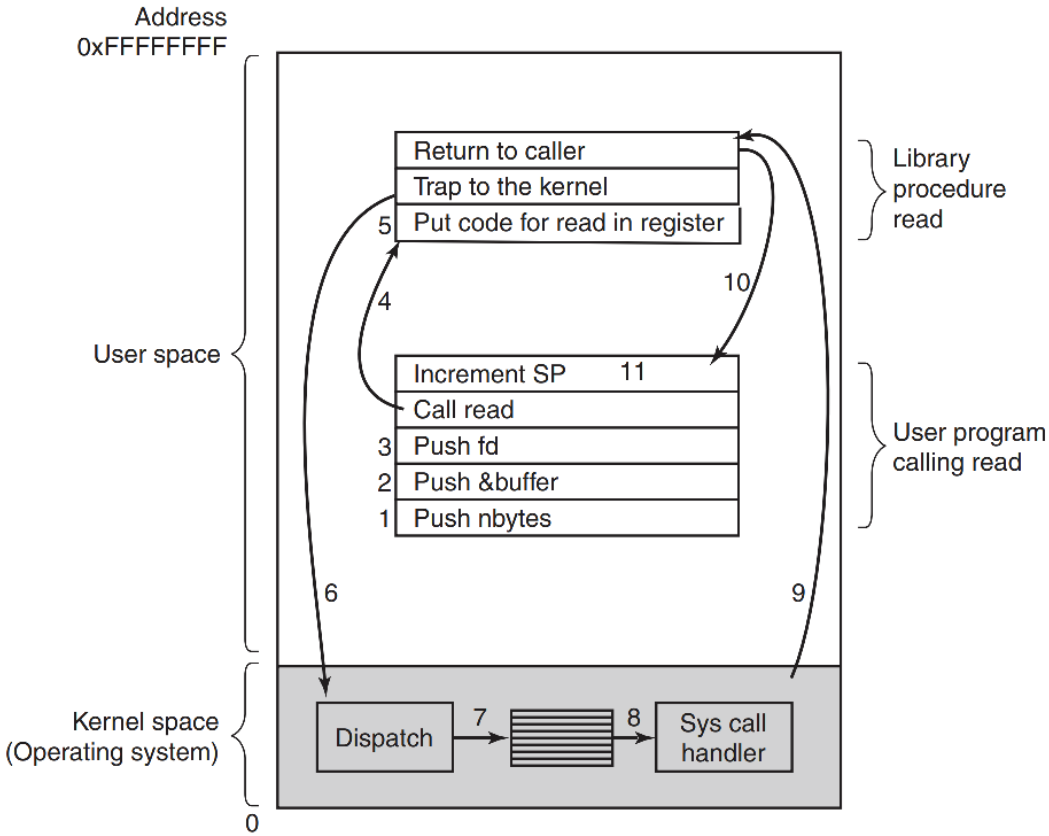
- interfața prin care aplicațiile comunică cu SO
  - apelurile de sistem pot fi considerate un API
- practic, sunt niște funcții (C)
  - nu apelează cod dintr-o bibliotecă obișnuită, ci din SO
  - se face trecerea în *kernel mode*
- avem nevoie de apeluri de sistem pentru
  - operații cu fișiere
  - management-ul proceselor și al thread-urilor
  - management-ul și protecția memoriei
  - comunicarea între procese

Exemplu: `read`

- aplicația vrea să citească

```
count = read(fd, buffer, nbytes);
```

- la revenirea din apel, în `buffer` se vor găsi `count` octeți citiți



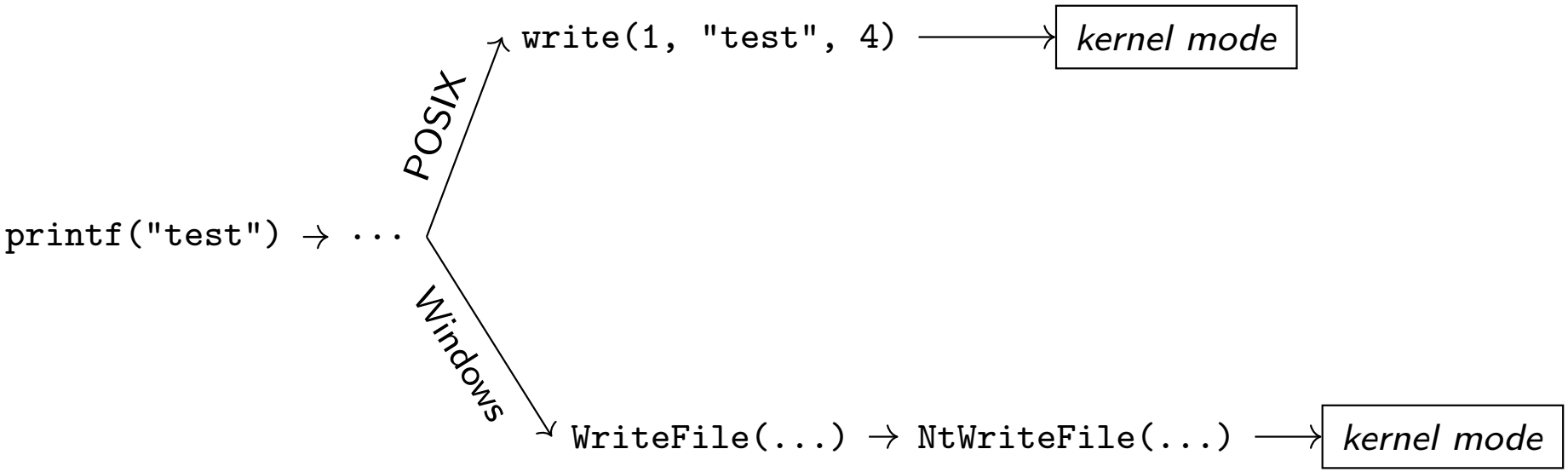
figură preluată din [MOS]

Apeluri de sistem prin funcții de nivel înalt

- cum am "trăit" până acum fără apeluri de sistem?

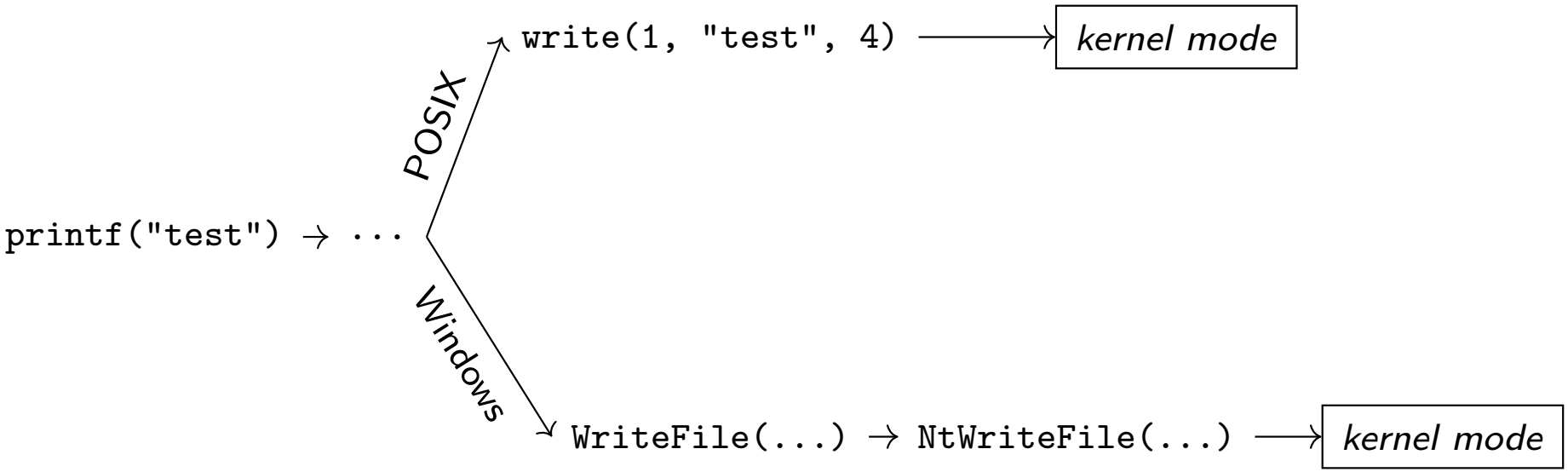
Apeluri de sistem prin funcții de nivel înalt

- cum am "trăit" până acum fără apeluri de sistem?



Apeluri de sistem prin funcții de nivel înalt

- cum am "trăit" până acum fără apeluri de sistem?



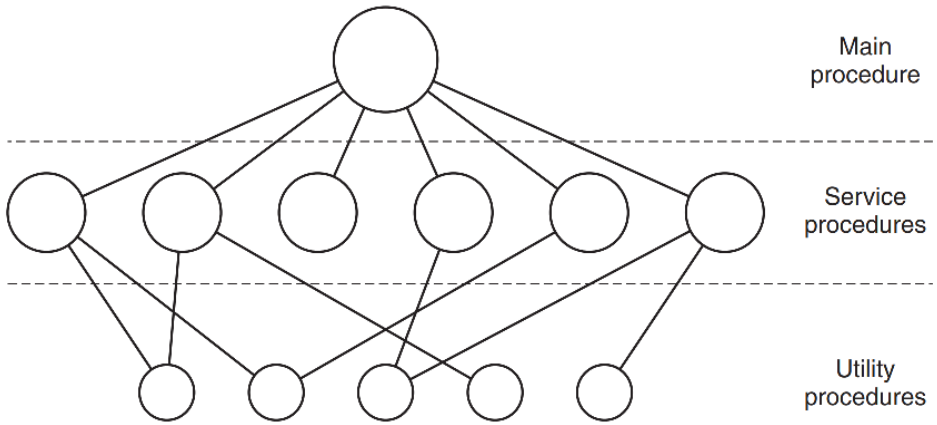
- orice funcție care face I/O (chiar și din alte limbaje) ajunge până la urmă să facă un apel de sistem

Cuprins

Sisteme monolitice

- fără o structură propriu-zisă (deși există o structură logică)
- kernel-ul e o colecție de proceduri care se pot apela unele pe altele

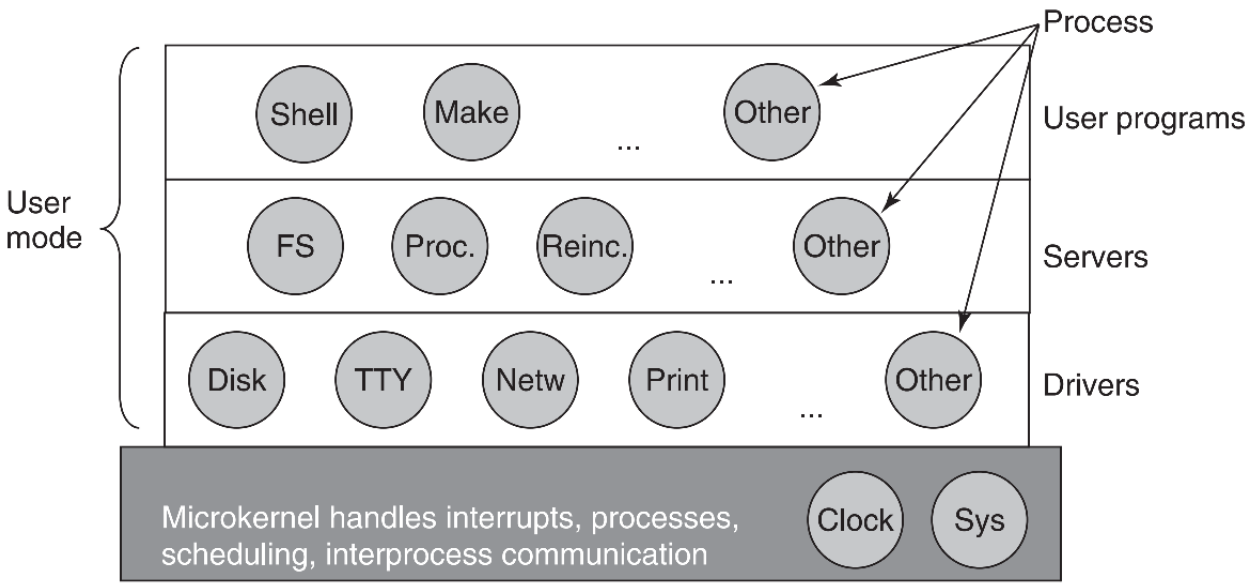
- avantaj: performanța
- dezavantaje
  - complexitate crescută
  - un bug poate crasha tot sistemul
- exemple: Linux, Windows



figură preluată din [MOS]

Sisteme microkernel

- doar strictul necesar rulează în kernel mode

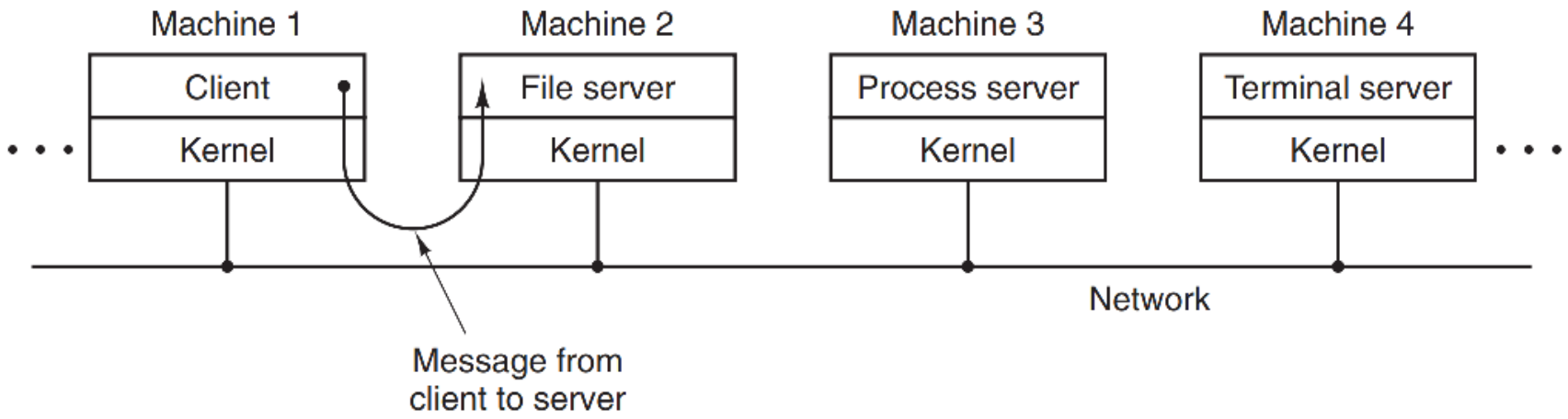


figură preluată din [MOS]

- avantaje: modularitate, robustețe
- dezavantaje: overhead adus de IPC
- exemple: MINIX3, Mac OS

Sisteme client-server (distribuite)

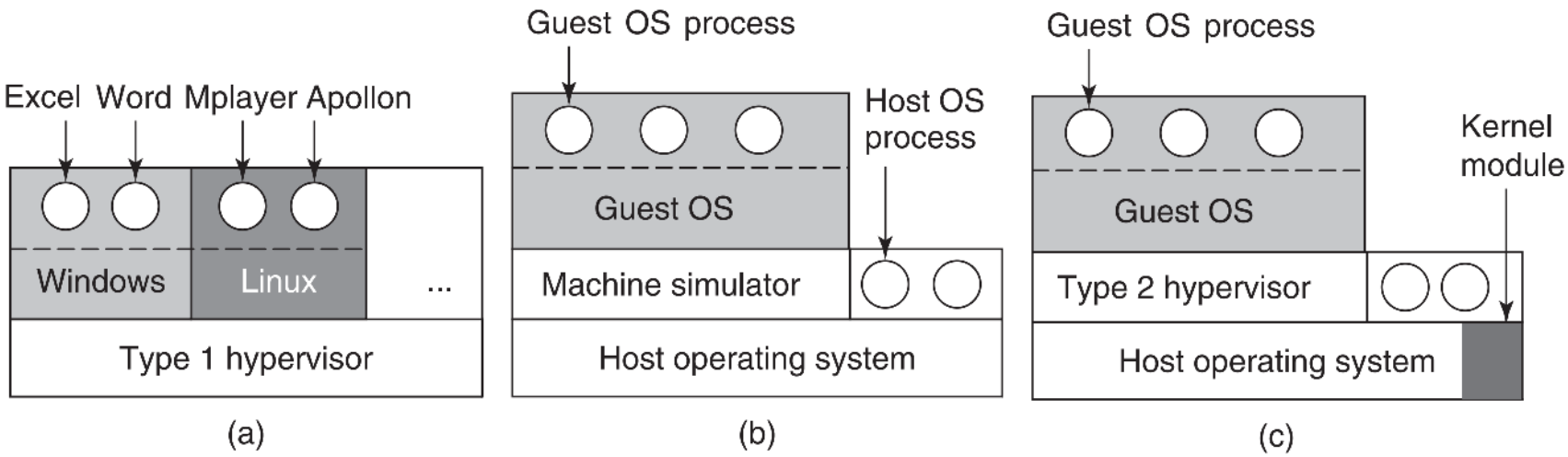
- variație a modelului microkernel
- componentele sistemului pot fi distribuite în rețea



figură preluată din [MOS]

Mașini virtuale și hipervizoare

- posibilitatea de a simula un sistem de calcul
- hipervizor de tipul 1: rulează direct peste hardware
- hipervizor de tipul 2: rulează peste un alt SO
  - în practică, are și un modul în kernel-ul gazdă



figură preluată din [MOS]

Bibliografie

[MOS] A. Tanenbaum și H. Bos, *Modern Operating Systems*, 4th Edition, Pearson Education, 2015, Capitolul 1