

# **Operating systems**

## **- An overview -**

**+**

# **UNIX/Linux Introduction**

**Prof. Răzvan Zota**

# Operating systems

## Grading rules

- 30% - seminar activity (10% - test1 + 10% - test2 + 10% - project)
  - 70% - result from the final examen (test using online.ase.ro)

**Conditions to pass the exam:** you must achieve at least 50% from the total points at the seminar and at the final exam

# Operating systems

## Introduction

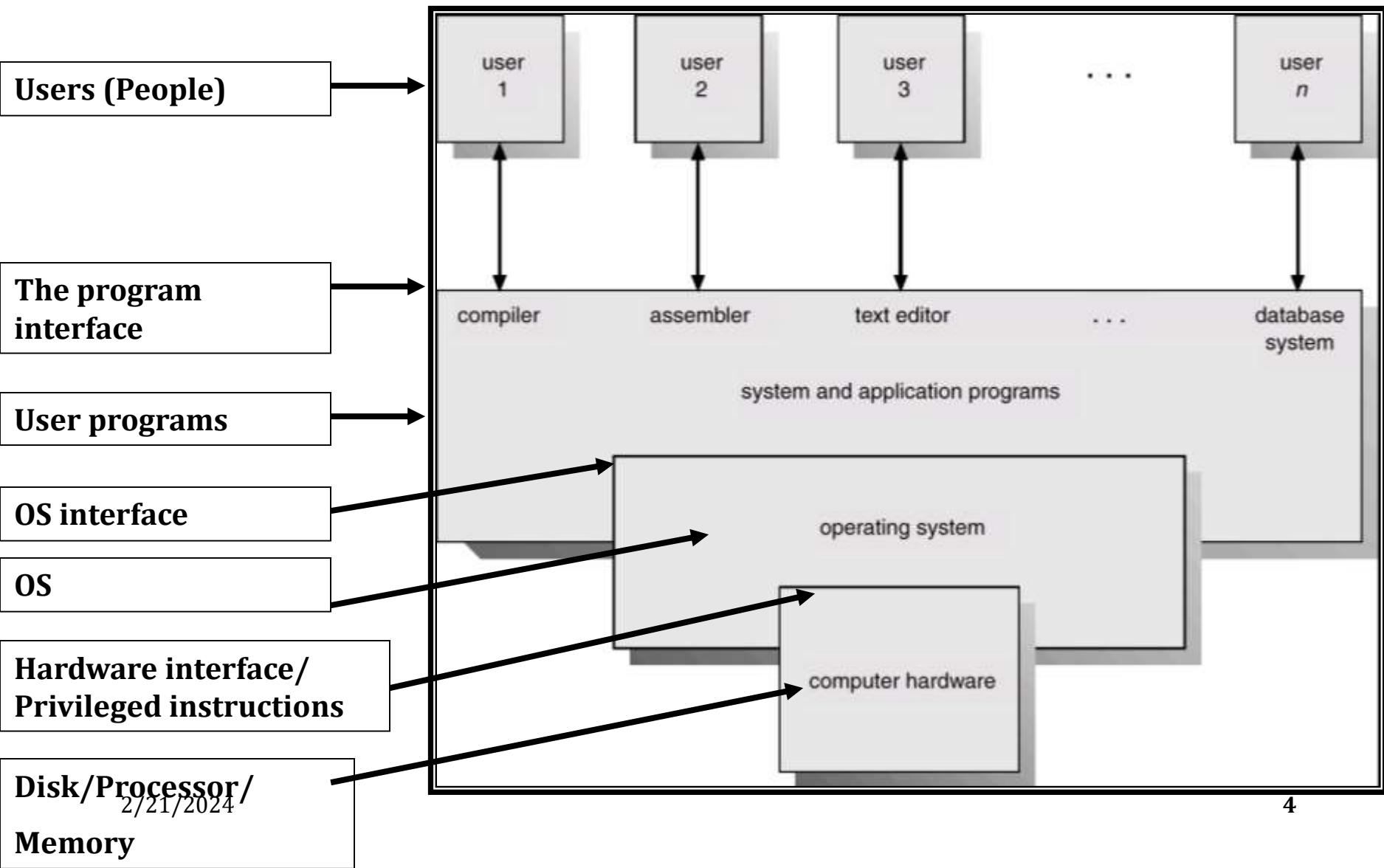
### What is an operating system?

- A set of programs for managing the computer's resources
- An interface between users and hardware – a medium "architecture"
- Enables convenient data storage, hiding details from the user
- Enables the efficient usage of the system, parallel execution of multiple activities, not wasting the clock cycles
- Enables the information protection
- Giving each user a “slice” from the total resources of the system
- It actions like a control program.

# Operating systems

## An overview

OS position



# Operating systems

## An overview

### Components

As a big picture, an OS can be seen as:

- A mechanism used for jobs' and processes' scheduling. The scheduling can be as simple as running the next process from a waiting queue or can be more complex by using more complicated rules for choosing next process to run
- A way for simultaneous execution for multiple CPUs and I/O management.

# Operating systems

## An overview

### Components

CPU activity is wasted if a job waits for an I/O operation. This lead to:

- **Multiprogramming.** While a job is waiting for a resource to be free the CPU can run another job. This means that multiple jobs are ready (in the same time) to run and are waiting for the CPU to continue.

**CPU scheduling** is an important part in the OS study.

All these are leading also to:

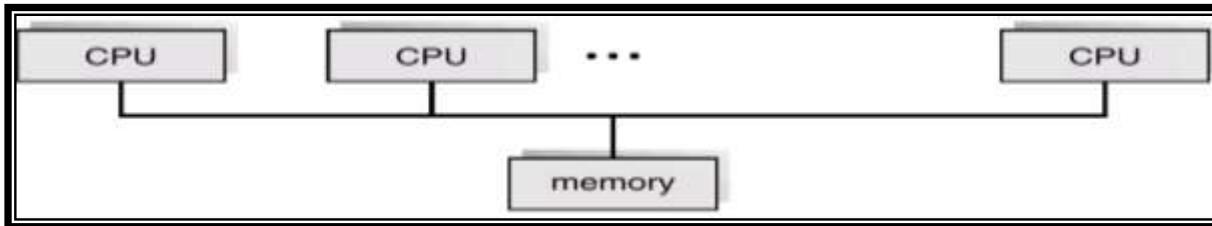
- **memory and process management**
- **resource planning**
- **deadlock protection**

# Operating systems

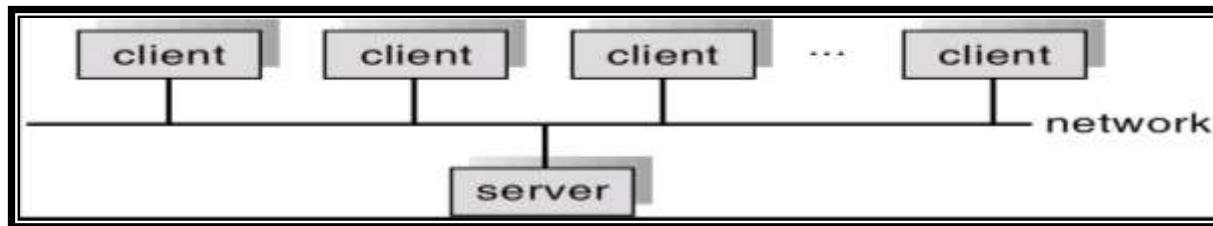
## An overview

### Characteristics

- **Time sharing** – the multiprogramming medium is, also, interactive
- **Multiprocessing** - shared memory collaboration and communication. Used especially for improving speed by putting to work more processors in the same time.



- **Distributed systems** – Long distance connected systems communicating by message transfer. Advantages: resource sharing, more speed, reliability, communication.



- **Real time systems** – The main characteristic is the quick answer. Used for controlling applications where a quick answer is essential.

# OS types

- **Smart card OS-s** – have basic functions like: secure access to card storage, authentication and cryptography(the most usual are JavaCard and MULTOS).
- **Embedded OS** – there are OS-s incorporated in mobile phones, TVs, etc. Examples: Android, iOS, Windows Phone.
- **Real time OS-s (RTOS – Real Time OS)** – used in scientific applications (space ships, etc.), industry (auto – car infotainment systems, robots, etc.), medicine (medical equipment). Examples: RTLinux, QNX. The main characteristic is the response time. Two categories: **hard RT** – where the time constraints are essential and **soft RT** – where these constraints are not so important (for example, in case of dedicated multimedia systems).
- **Desktop OS-s** – the Windows family (7,8,10), Linux (several distros), Mac OSX (El Capitan), macOS High Sierra.
- **Server OS-s** - Unix/Linux (RHEL, SLES – Suse Linux Enterprise Server), Windows Server 2016.
- **Mainframe OS-s** – IBM z /OS (z10), Linux, OpenSolaris.

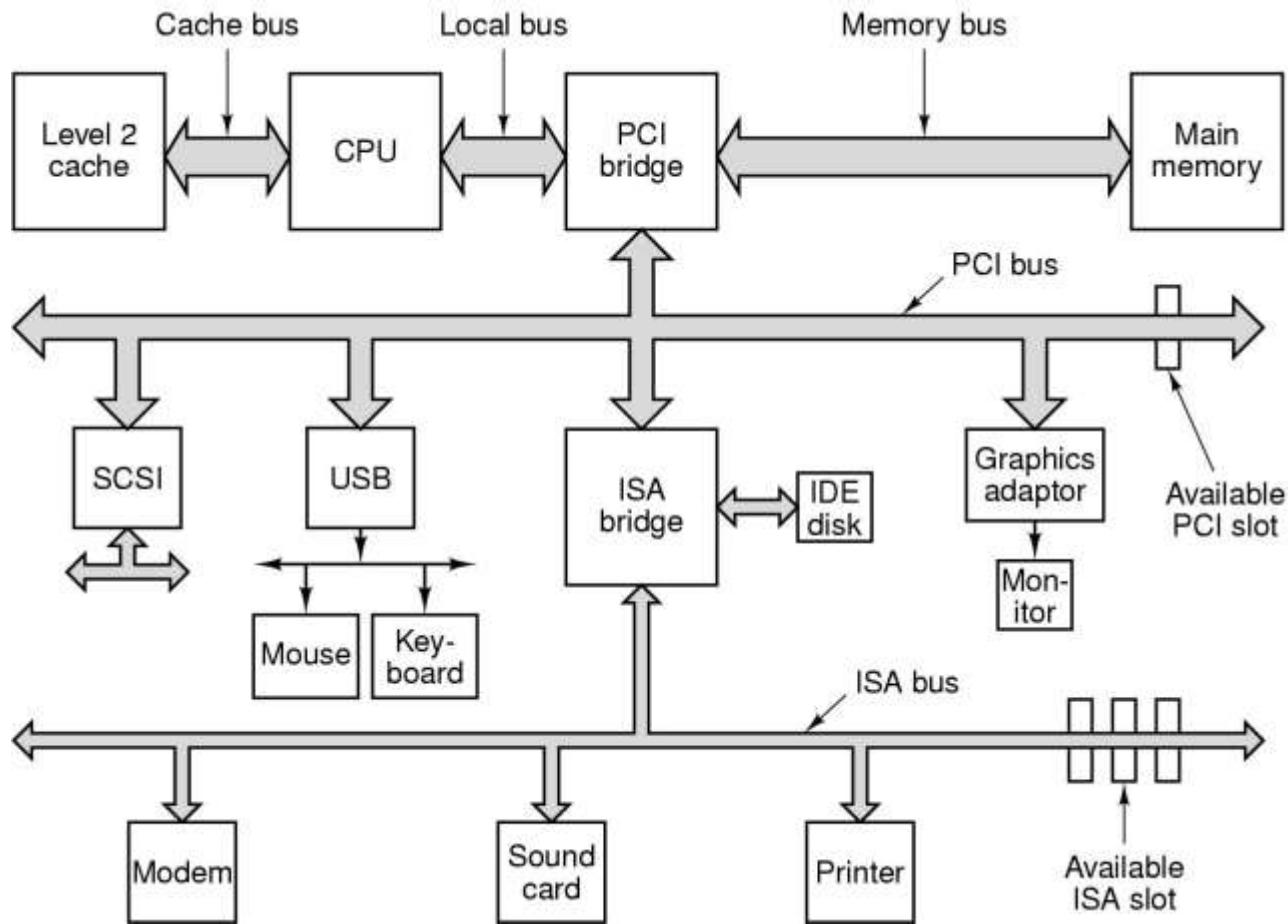
# Operating systems

## An overview

Hardware support

One system is made by several devices

These devices may generate an electrical signal (called interrupt) which catch the attention of the CPU.



# Operating systems

## An overview

Hardware  
support

**Interrupts** - a device is sending “a disturbing” signal to the CPU to get a service. Different from the action of **polling**.

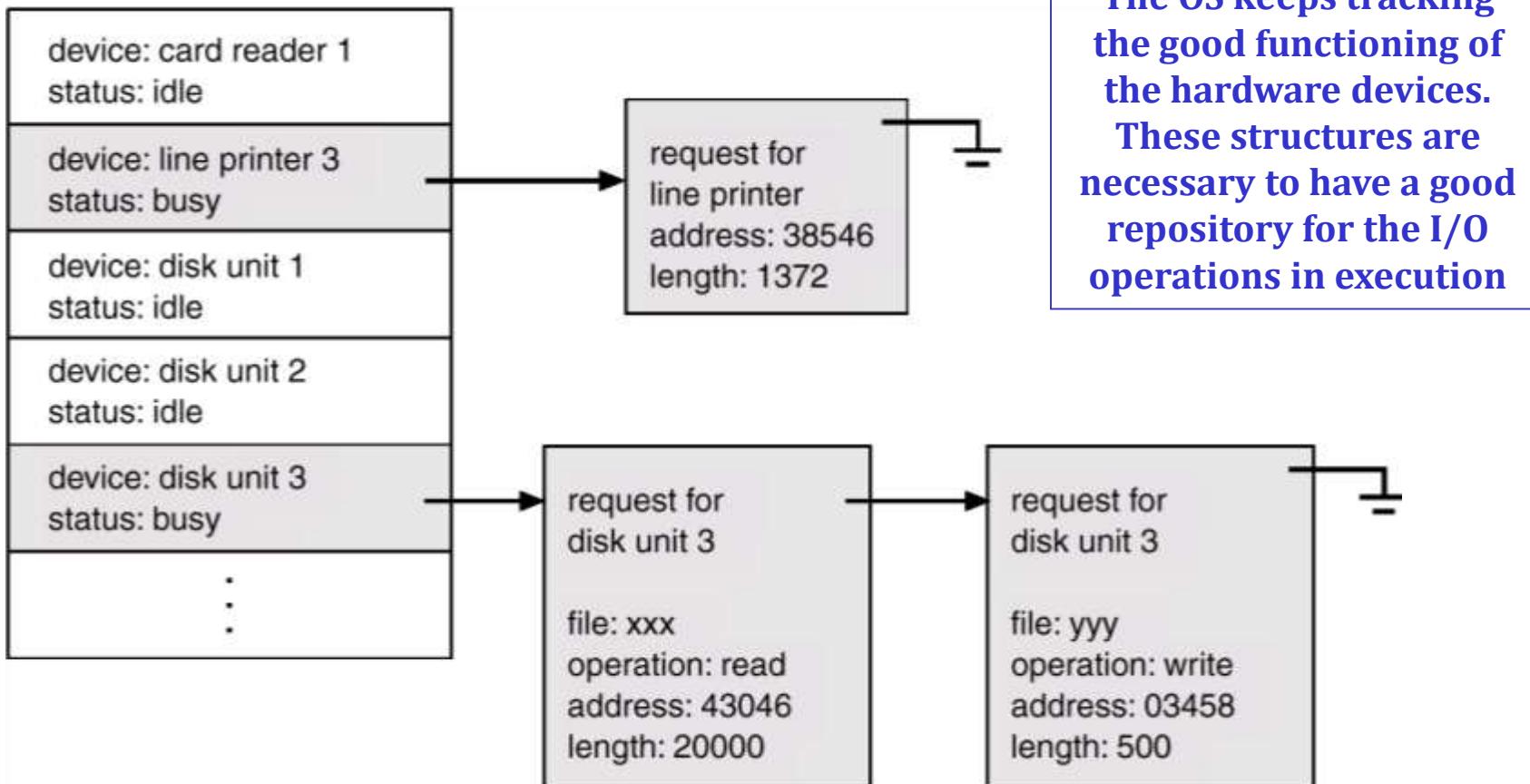
- Depending by the interrupt it is determined the next step to be done.
- There are *soft interrupts* and *hard interrupts*.
- The interrupt manager choose the code to be run on each device.

**DMA I/O Controllers** (Direct Memory Access) have direct access to the memory, without “asking” CPU.

# Operating systems

## An overview

### Hardware support



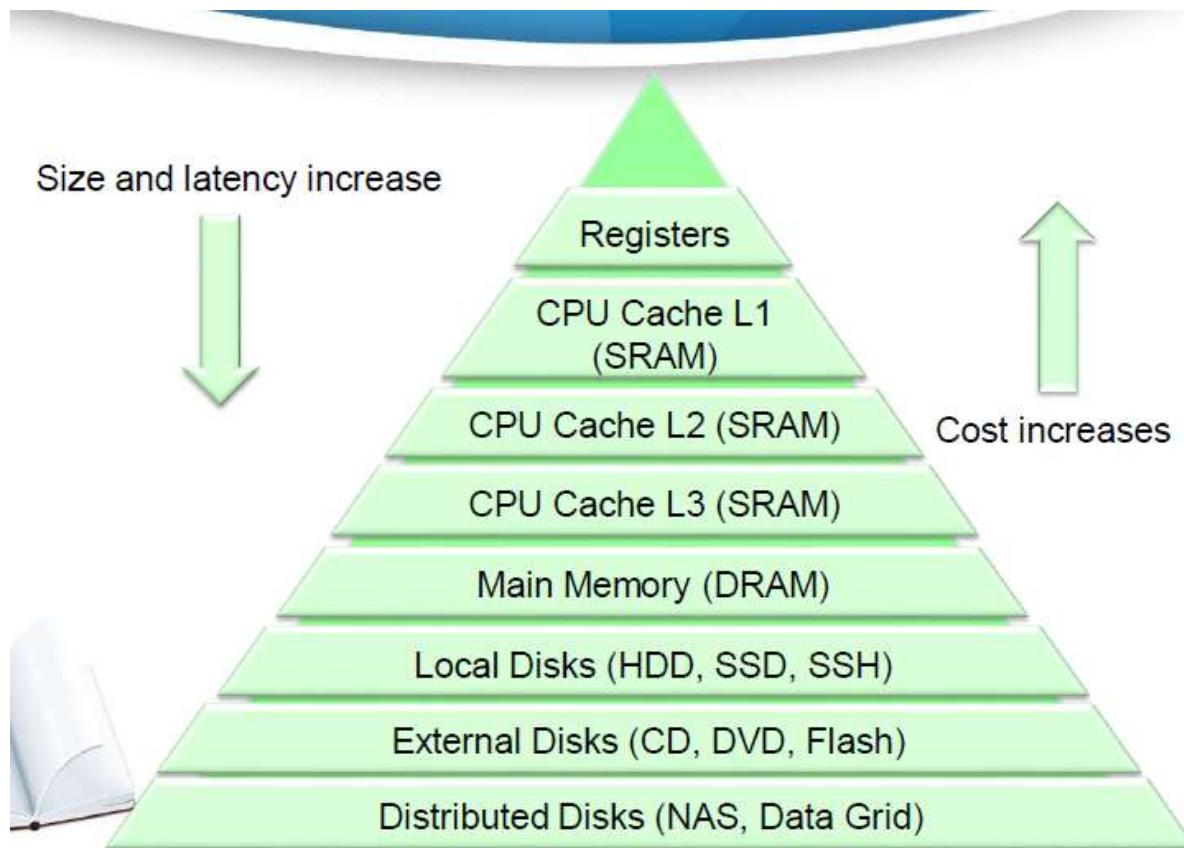
# Operating systems

## An overview

### Memory hierarchy

Ultra-fast memory is expensive.

OS manages the memory hierarchy in order to gain a better resource utilization.



# Operating systems

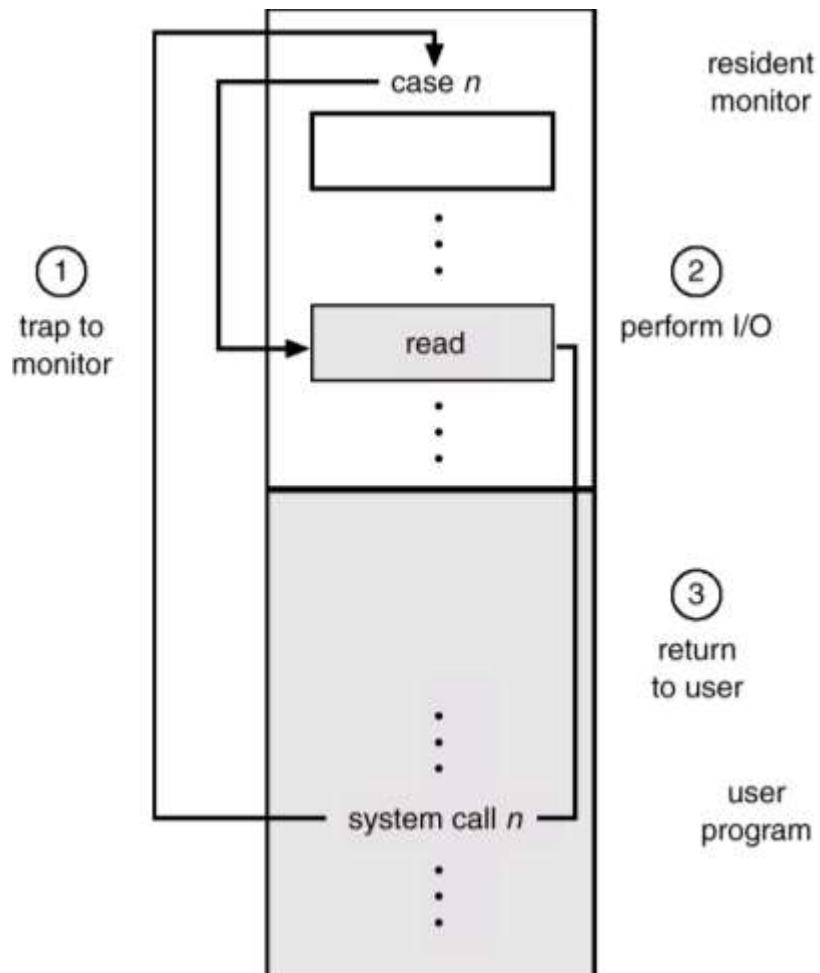
## An overview

### Protection

The main goal is to protect the OS and devices from malicious /ignorant users.

There are special instructions in **user mode (normal instructions)** or **supervisor mode (privileged instructions)**.

Concurrent threads may interfere one with another. This leads to the necessity for resource protection by using **user/supervisor modes**. For example, the I/O instructions are privileged, it can be run only in supervisor mode. The system calls are making the transfer from user mode into supervisor mode.

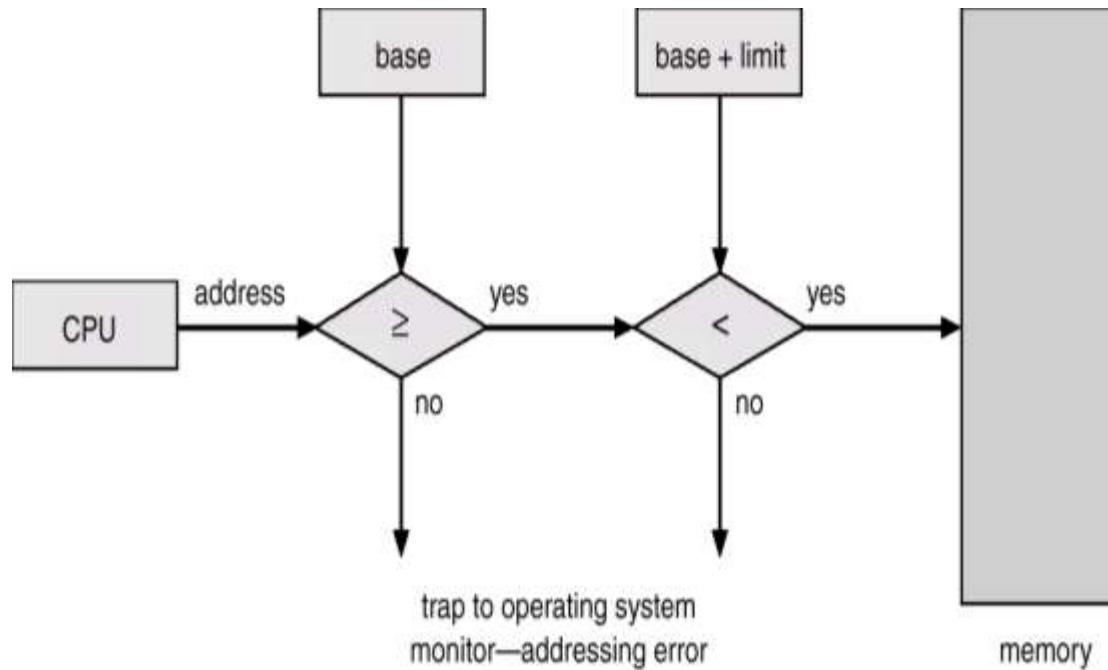


# Operating systems

## An overview

### Protection

**Memory** A user program may access only its logical memory. For example, it cannot modify supervisor code. It depends by a translation address schema.



# Operating systems

## An overview

### Protection and security

**CPU** The system clock is preventing programs to use all the CPU time. This clock generates an interrupt that enables the OS to take control from an user program.

For multiprocessor and distributed systems the protection must include :

- Resource sharing
- Multiprocessor architectures
- Cluster systems

The systems are known as “distributed operating systems”.

# UNIX – A (very) short history

**UNIX origins: AT&T Bell Laboratories + GE + MIT – SO Multics**

Ken Thompson

Dennis Ritchie (+ Brian Kernighan)

They wrote together a travel space game running on a DEC PDP-7 machine. In 1969 they have decided to write an OS for PDP-7, named UNICS (Uniplexed Information and Computing Service).

1971 – C

1973 – UNIX written in C -> **portable system**

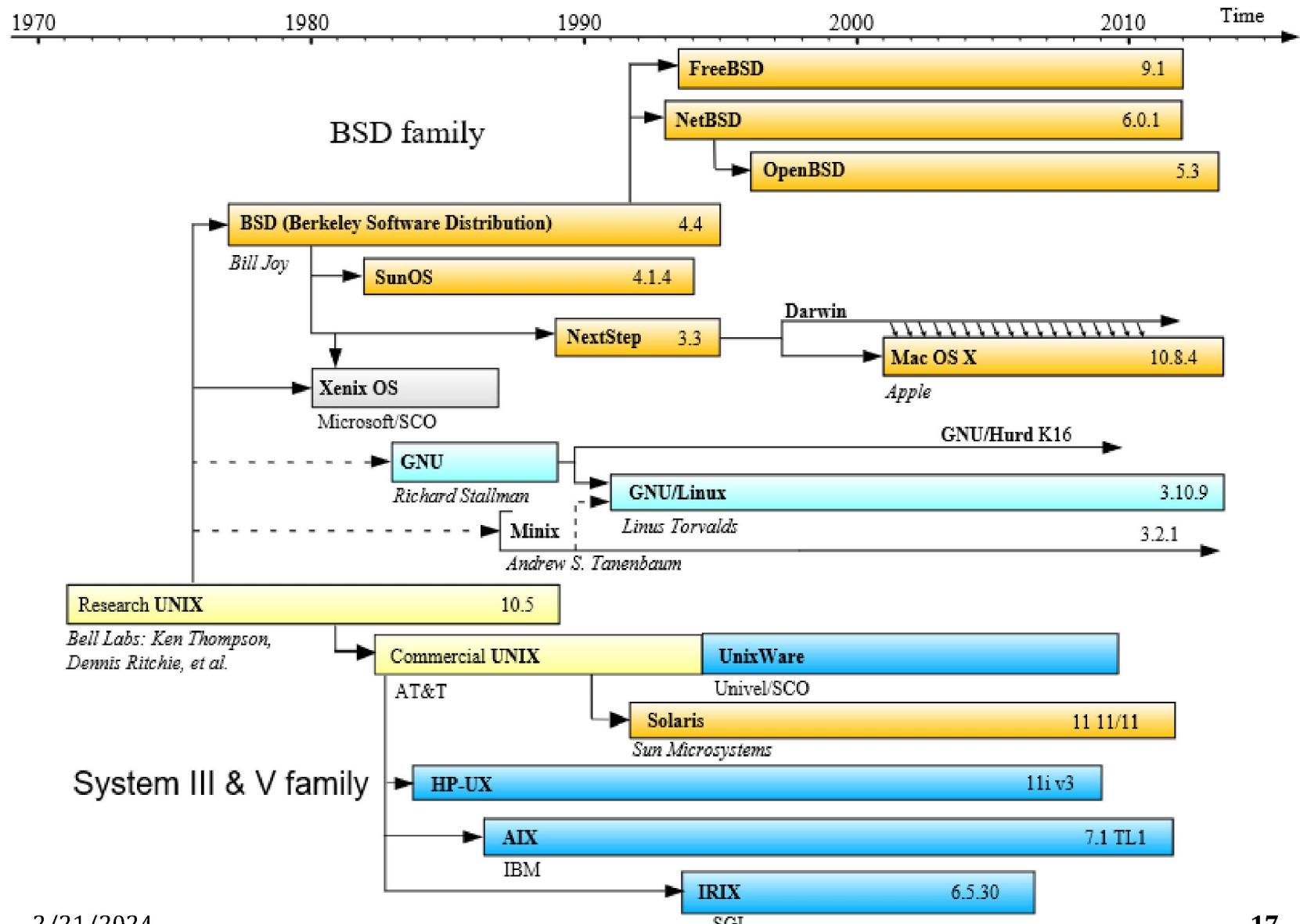
BSD UNIX – adding networking software

SunOS – BSD UNIX Version 4.2

AT&T System V (five)

1988 SunOS, AT&T System V and XENIX -> System V Release 4 (SVR4)

# Unix “families” evolution



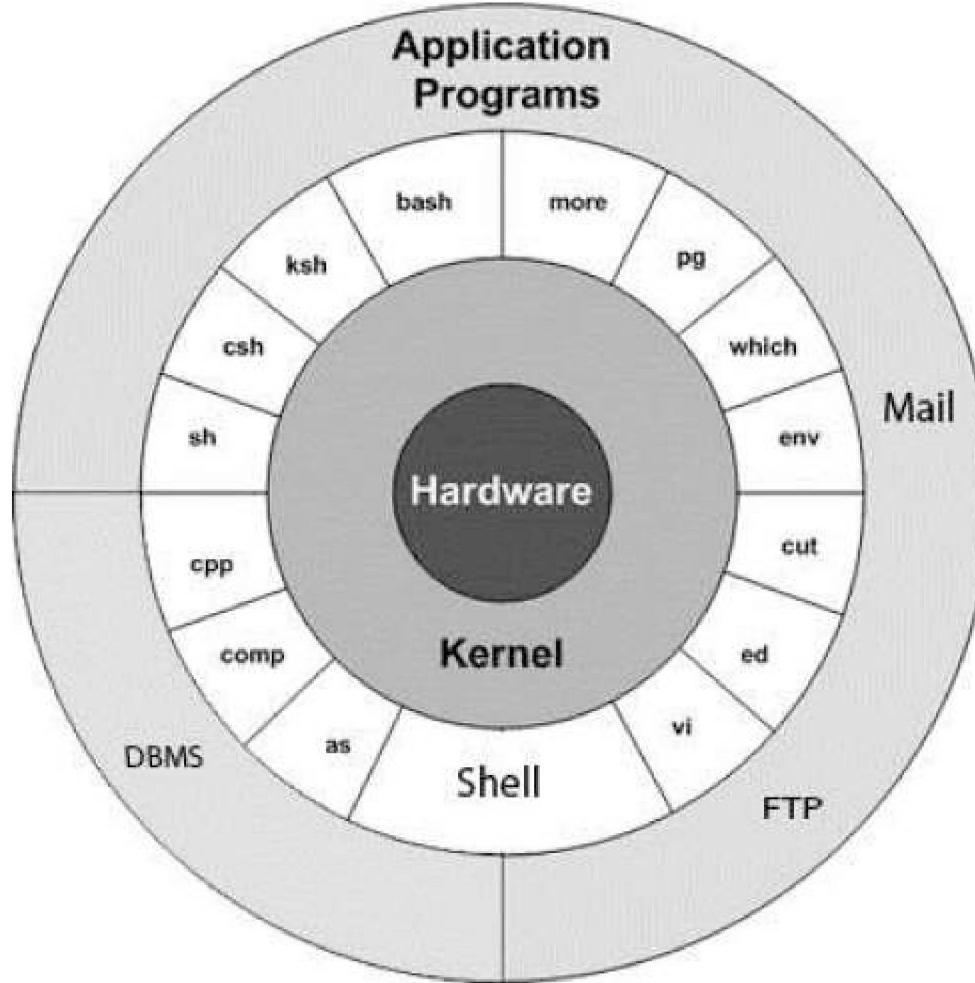
# UNIX

## **UNIX strong points:**

- Standards based
- Powerful, flexible, scalable, secured
- Good support from the hardware manufacturers
- Mature and stable OS
- Good integration with the TCP/IP network protocol stack
- Used on a large scale for critical applications

## **Key components:**

- Kernel
- Shell
- File system
- Commands



# UNIX

- **Kernel:** The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, task scheduling and file management.
- **Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Kom Shell are most famous shells which are available with most of the Unix variants.
- **Commands and Utilities:** There are various command and utilities which you would use in your day to day activities. `cp`, `mv`, `cat` and `grep` etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.
- **Files and Directories:** All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

# UNIX

## The kernel:

- The center of the OS- provides the capabilities for normal functioning of the computer
- It is an executable file loaded into memory when the computer is booting and it is called *unix* (System V) or *vmunix* (BSD) or ... *linux* !
- After loading in memory, the kernel has the following main functions:
  - Managing devices, memory, processes
  - Controlling the information transmission between system programs and the hardware

# UNIX

## The kernel:

-Managing functions about:

**-The *swap space*** – very important for Unix, reserved for virtual memory space

**-*Daemons*** – there are programs (processes) with a specific function or they monitor the programs execution or device functioning. Daemons are special processes loaded into memory with the kernel and waiting for an event to happen. These processes help for a good functioning of the OS, offering various services. Daemons can be started or stopped anytime it is necessary. Equivalent with the services from Windows and NLM – Novell Netware.

**-*File systems*** – files' hierarchy, directories and sub-directories for structuring and managing the information on the HD.

# UNIX

**Shells:** Bourne, Korn, C, Bash, TC (the *ps* command or *echo \$SHELL*).

- Bourne /bin/sh – Stephen Bourne AT&T System V.2 UNIX (prompter: \$)
- Korn /bin/ksh – Bell Labs (prompter: \$)
- C Shell /bin/csh - Bill Joy from University of California at Berkeley (prompter: %)
- Bash – GNU ([www.gnu.org](http://www.gnu.org))

## File system

**/bin** – UNIX commands

**/usr/bin** – commands, system administration utilities, library routines

**/usr/ucb** – original commands for the BSD variant

**/opt** – optional applications or from another producers

**/etc** – system administration files (see /etc/passwd)

# UNIX

**/dev** – files pointing to names of equipment

**(Solaris)/kernel** – contains the basic files of the OS

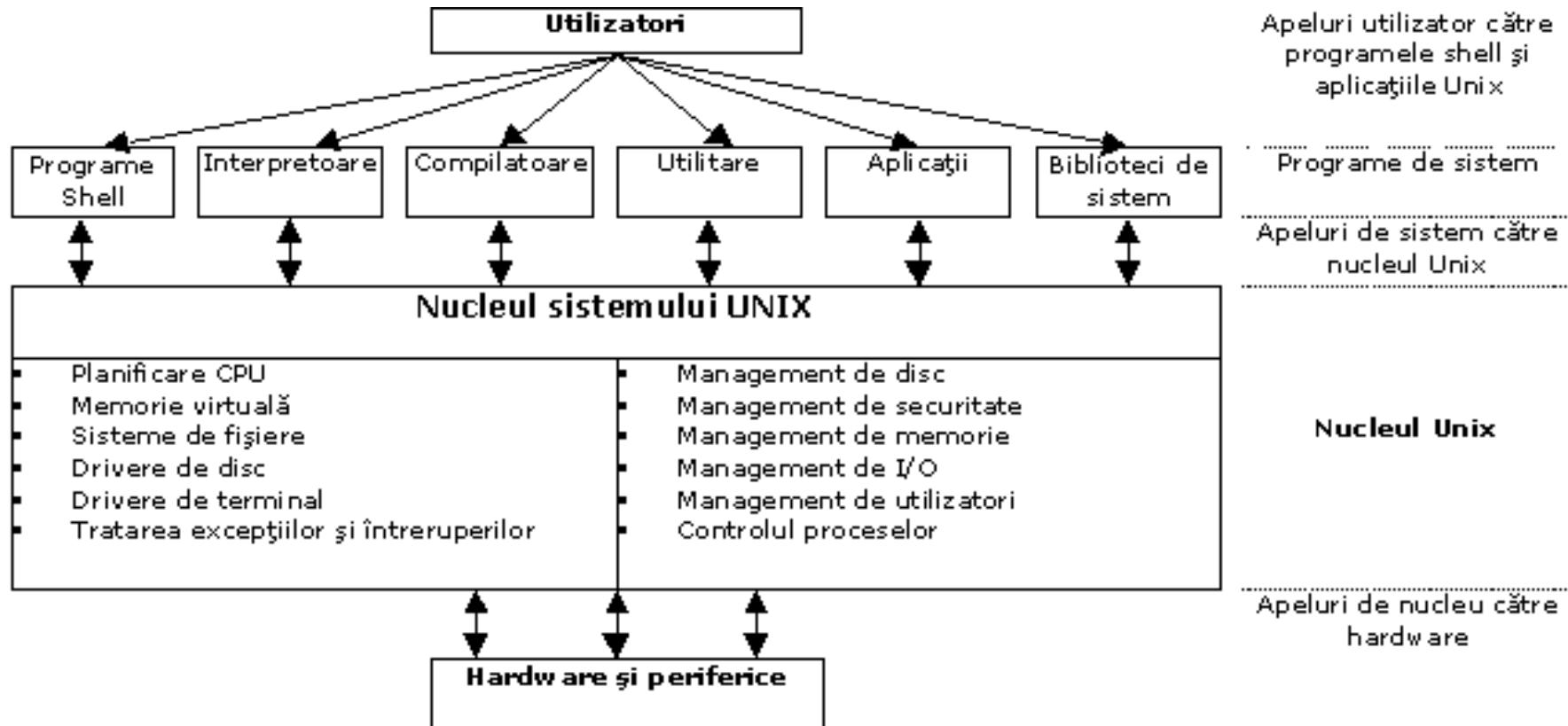
**/sbin** – the basic executables used for booting and recovery + administration programs

**/tmp** – users' temporary files

**/var** – location for printers' jobs (print spooling) and error messages for mail system.

**Commands** – around 350-400 commands and utilities

# UNIX General Architecture



# Bibliography

- Sisteme de operare - R. Zota, A. Vasilescu, Ed. ASE, 2015
- Sistemul de operare Unix – Utilizare si programare shell, R. Zota, Ed. ASE, 2003
- Unix – R. Zota, Ed. ASE, 2004
- Other materials posted on <http://zota.ase.ro/os>

# **Introducere în Sisteme de operare**

1.1 – Bazele SO

1.2 – Microsoft Windows

1.3 – Unix și Linux pe Desktop

1.4 – Bazele NOS

# Comenzi Unix/Linux introductive

- ▶ who am I (whoami)
- ▶ echo \$LOGNAME
- ▶ id -un
- ▶ uname -a
- ▶ hostname

Comenzi de tip “help”:

- ▶ man
- ▶ whatis
- ▶ apropos

# Comenzi Unix/Linux introductive (seminar)

- ▶ cd (change directory)
- ▶ mkdir (make directory)
- ▶ pwd (print working directory)
- ▶ rmdir (remove directory)
- ▶ ls (list) -list files&directories from a directory

Caractere speciale:

~ “home directory”

/ “root directory” – directorul rădăcină al sist de fișiere Unix/Linux

Orice **cale absolută** pornește din rădăcină (root):

/home/ubuntu/dir1/file01

O cale care **nu** pornește din rădăcină se numește **cale relativă**:  
dir1/dir2/file02

# Comenzi Unix/Linux introductive (seminar)

Pentru crearea unui shell script:

- folosim editorul pico astfel:

**\$ pico program01** (semnul \$ este prompterul Linux)

- după ce scriem liniile de program salvăm fisierul (Ctrl+o) și ieșim din editor (Ctrl+x)
- acordăm drept de execuție fișierului program01 cu comanda:

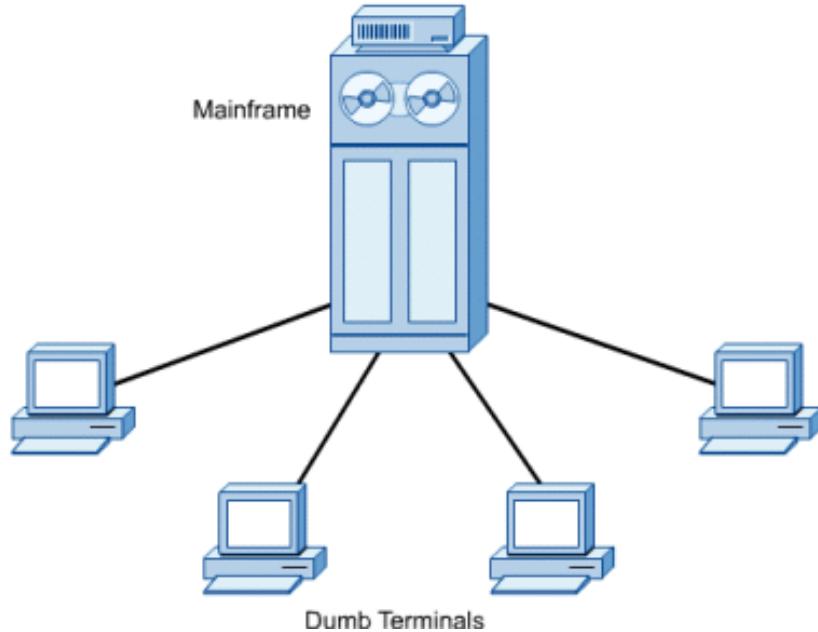
**\$ chmod +x program01**

- rulăm programul creat folosind comanda:

**\$ ./program01**

Cine este **\$PATH**?

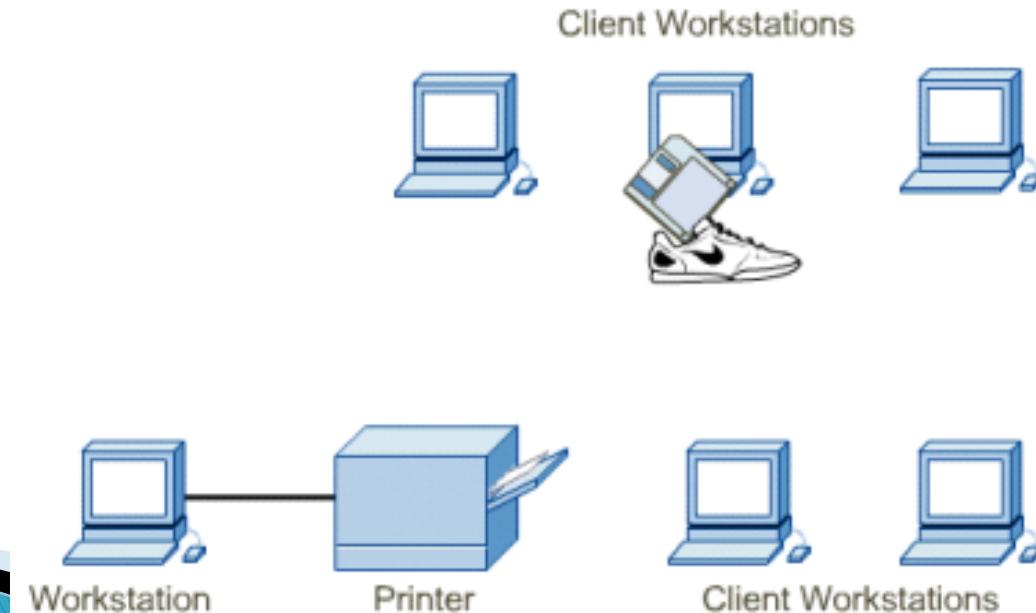
# O privire de ansamblu a SO pentru PC-uri



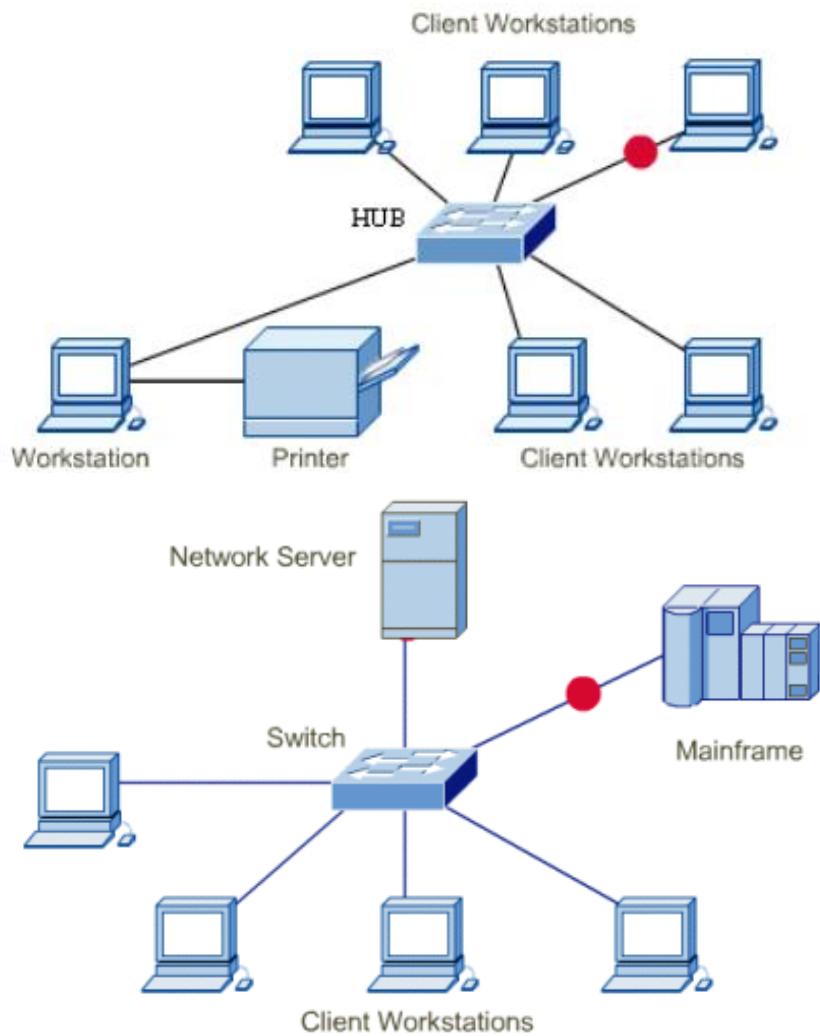
- ▶ Calculatoarele de tip desktop (microcomputere) au devenit populare la începutul anilor '80.
- ▶ Utilizatorii acestor prime PC-uri foloseau sistemele pentru îndeplinirea unor funcții diverse, precum procesare de texte, contabilitate sau jocuri pe calculator.
- ▶ Productivitatea era însă limitată de imposibilitatea acestora să partajeze informații cu alte sisteme.

# PC-urile și rețelele de calculatoare

- ▶ Pe măsură ce tehnologia computerelor a evoluat, companiile au început să-și instaleze rețele locale (LAN) pentru a permite interconectarea PC-urilor desktop în scopul partajării datelor și a perifericelor (imprimante, spre exemplu).
- ▶ Un sistem de operare de rețea (NOS) necesită mai multă putere de procesare decât versiunea desktop.
- ▶ A apărut astfel o nouă categorie de PC-uri: serverele de rețea.
- ▶ Aceste calculatoare rulează un NOS și au devenit punctul central al rețelelor locale de PC-uri.

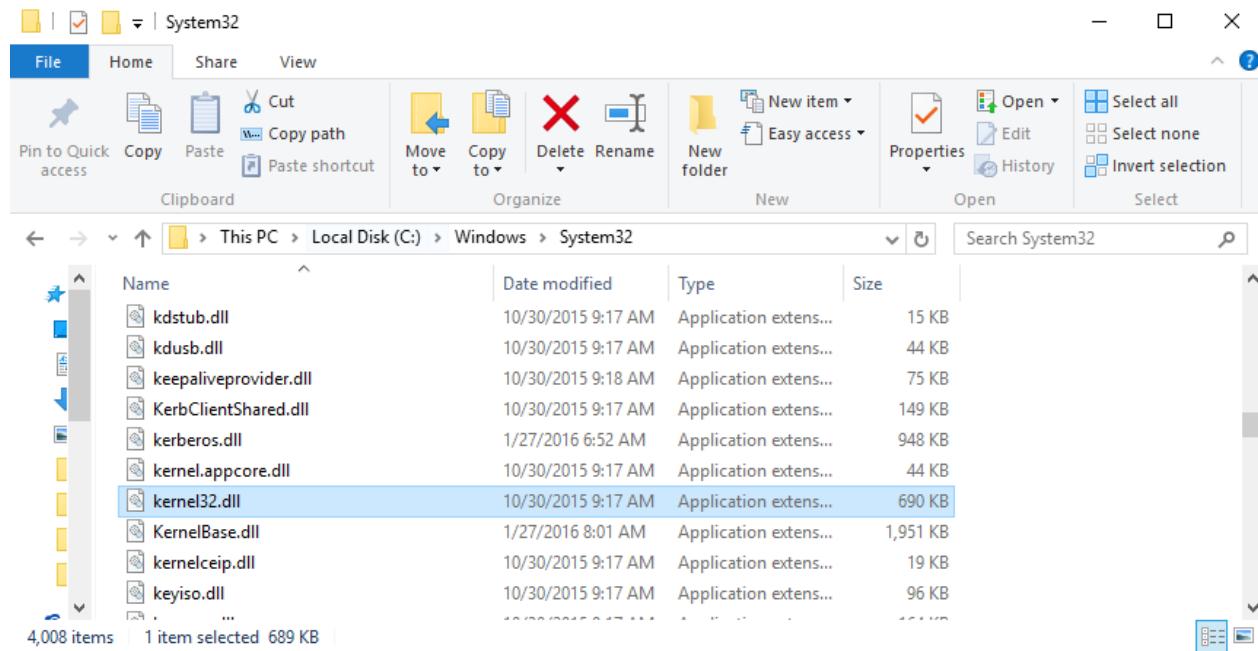


# PC-urile și rețelele de calculatoare



- ▶ Navigarea Web, electronic mail (e-mail) și alte aplicații bazate pe Internet sunt acum în centrul utilizării calculatoarelor.
- ▶ Pentru a oferi acces la aceste tehnologii Internet, marile companii de software (Microsoft și altele) și-au refacut sistemele de operare de tip desktop.
- ▶ SO de tip desktop includ astăzi multe dintre facilitățile și serviciile disponibile odinioară doar pentru sistemele de operare de rețea (NOS).

# Kernel-ul



- ▶ Kernel-ul este cel mai folosit termen pentru descrierea nucleului (centrului) sistemului de operare.
- ▶ Aceasta reprezintă o mică parte de cod (software) ce este încărcată în memorie atunci când computerul pornește.
- ▶ Acest cod conține instrucțiuni ce îi permit acestuia să administreze echipamente hardware, alocarea memoriei, procesele sistem și alte programe.
- ▶ Pentru Linux, kernelul se găsește în /boot.

Mai multe info: <https://learning.lpi.org/en/learning-materials/010-160/4/4.5/4.3.01/>

# Interfața utilizator



```
C:\WINNT\System32\cmd.exe
(C) Copyright 1985-2000 Microsoft Corp.

C:>dir
Volume in drive C has no label.
Volume Serial Number is 1C76-10F5

Directory of C:\

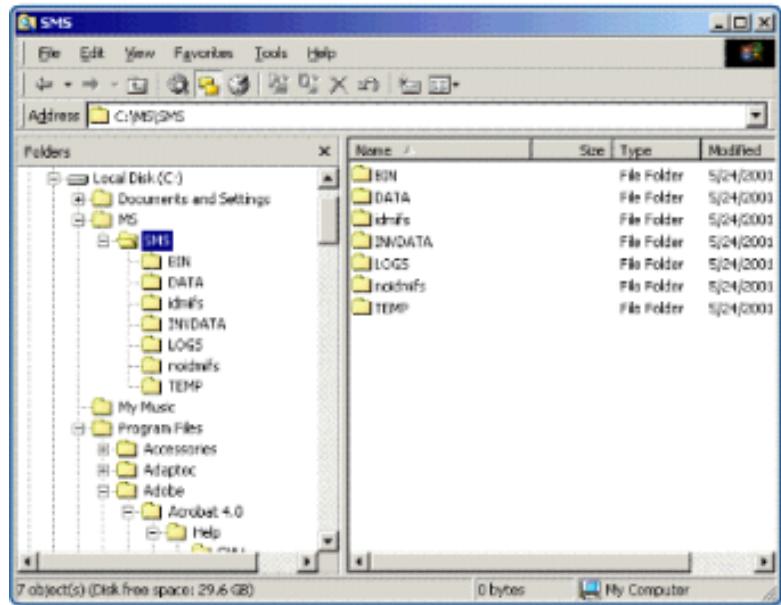
05/24/2001 10:27a <DIR>      WINNT
05/24/2001 10:29a <DIR>      Documents and Settings
05/24/2001 10:31a <DIR>      Program Files
05/24/2001 12:37p <DIR>      MS
05/24/2001 12:58p <DIR>      temp
01/11/2002 03:33p          65 HPJ\PP.dat
02/12/2002 03:30p          0 AdobeWeb.log
12/20/2001 11:31a <DIR>      My Music
06/26/2001 10:19a <DIR>      Ray
06/28/2001 12:08p        784,931 comreads.dbg
06/28/2001 12:08p        721,462 comused.dbg
08/02/1998 11:02a          10 windows
09/26/2001 10:50a <DIR>      Windows Update Setup Files
                           5 File(s)   1,506,468 bytes
                           8 Dir(s)  31,803,326,464 bytes free

C:>
```



- ▶ IU este componenta de interacțiune dintre SO și utilizator.
- ▶ IU este asemenea unui interpreter ce interpretează apăsarea unei taste a tastaturii, un click de mouse sau alt input pentru programele respective.
- ▶ O interfață utilizator grafică (GUI) permite utilizatorului să folosească software-ul folosind obiecte vizuale precum ferestre, meniuri de tip "pull-down", pointeri și simboluri grafice.

# Sistemul de fișiere



- ▶ Într-un sistem de fișiere ierarhic, fișierele se află în containere logice aranjate într-o structură arborezentă.
- ▶ Sistemul de fișiere începe cu rădăcina arborelui.
- ▶ UNIX și Linux numesc aceste containere "directoare" și "subdirectoare".
- ▶ Windows și Macintosh folosesc termenii de "folder" și "subfolder".

# Sistemul de fișiere

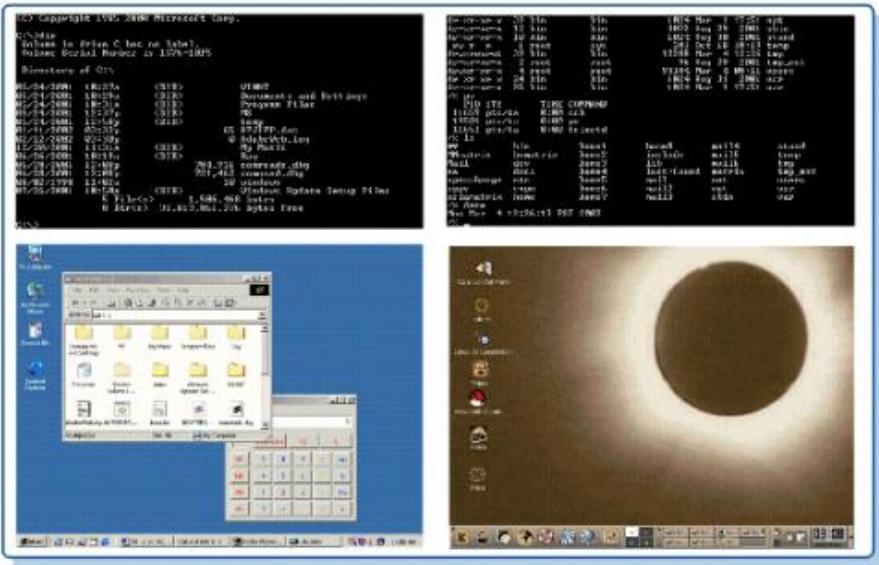
- ▶ Un tip de sistem de fișiere foarte răspândit este File Allocation Table (FAT).
- ▶ Sistemele FAT sunt administrate pe disc de către sistemul de operare.
- ▶ Tabela conține o hartă a fișierelor și locul unde sunt stocate acestea pe disc.
- ▶ Tabela FAT face referire la clusterele hard-discului, ce reprezintă unitatea logica de bază a stocării pe disc.
- ▶ Un anumit fișier poate fi stocat pe mai multe clustere, dar un cluster poate conține date dintr-un singur fișier.
- ▶ Sistemul de operare folosește tabela FAT pentru a găsi toate clusterele de pe disc unde sunt stocate fișierele.

# Sistemul de fișiere

- ▶ Există trei tipuri de sisteme FAT:
  - FAT12
  - FAT16
  - FAT32
- ▶ FAT16 și FAT32 reprezintă versiuni îmbunătățite ale sistemului original FAT.

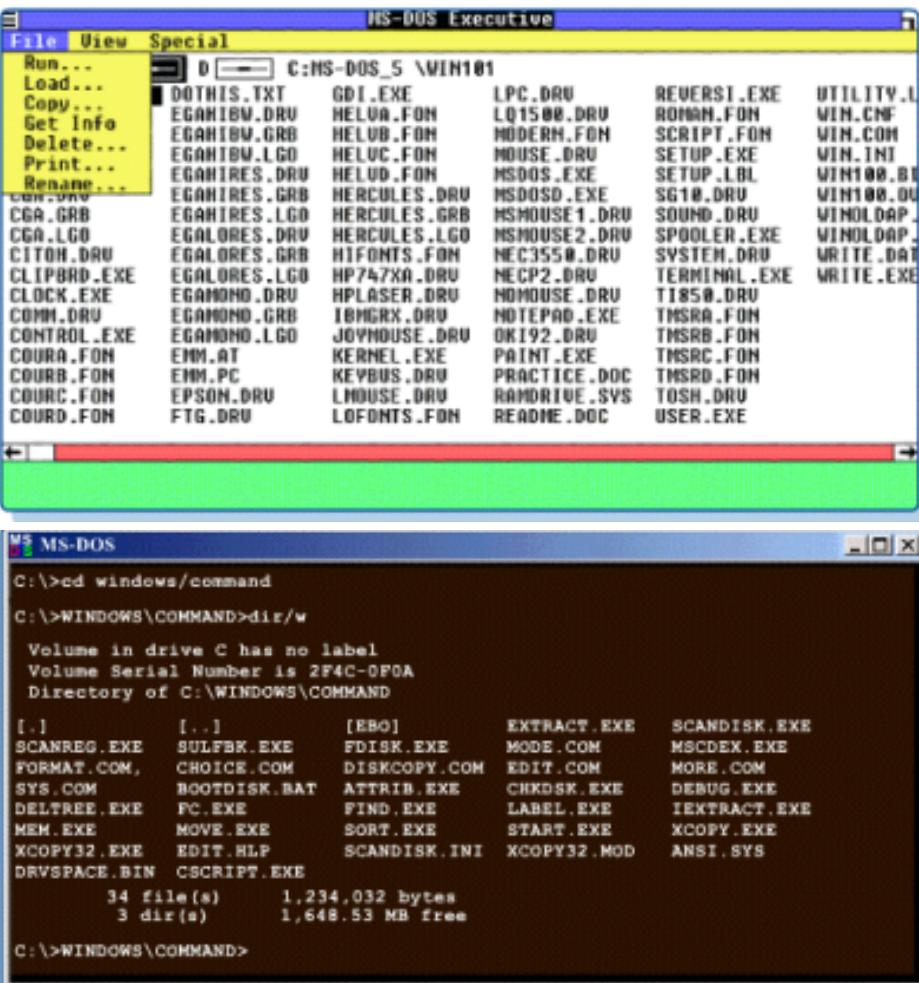
Operating System	Supported File System(s)
Windows 3.x	FAT16
Windows95, 98, ME	FAT16, FAT32
Windows NT, 2000	FAT16, FAT32, NTFS
Windows XP	FAT32, NTFS
IBM OS/2	HPFS (High Performance File System)
Linux	Ext2, JFS (Journaling File System)

# SO de tip Desktop



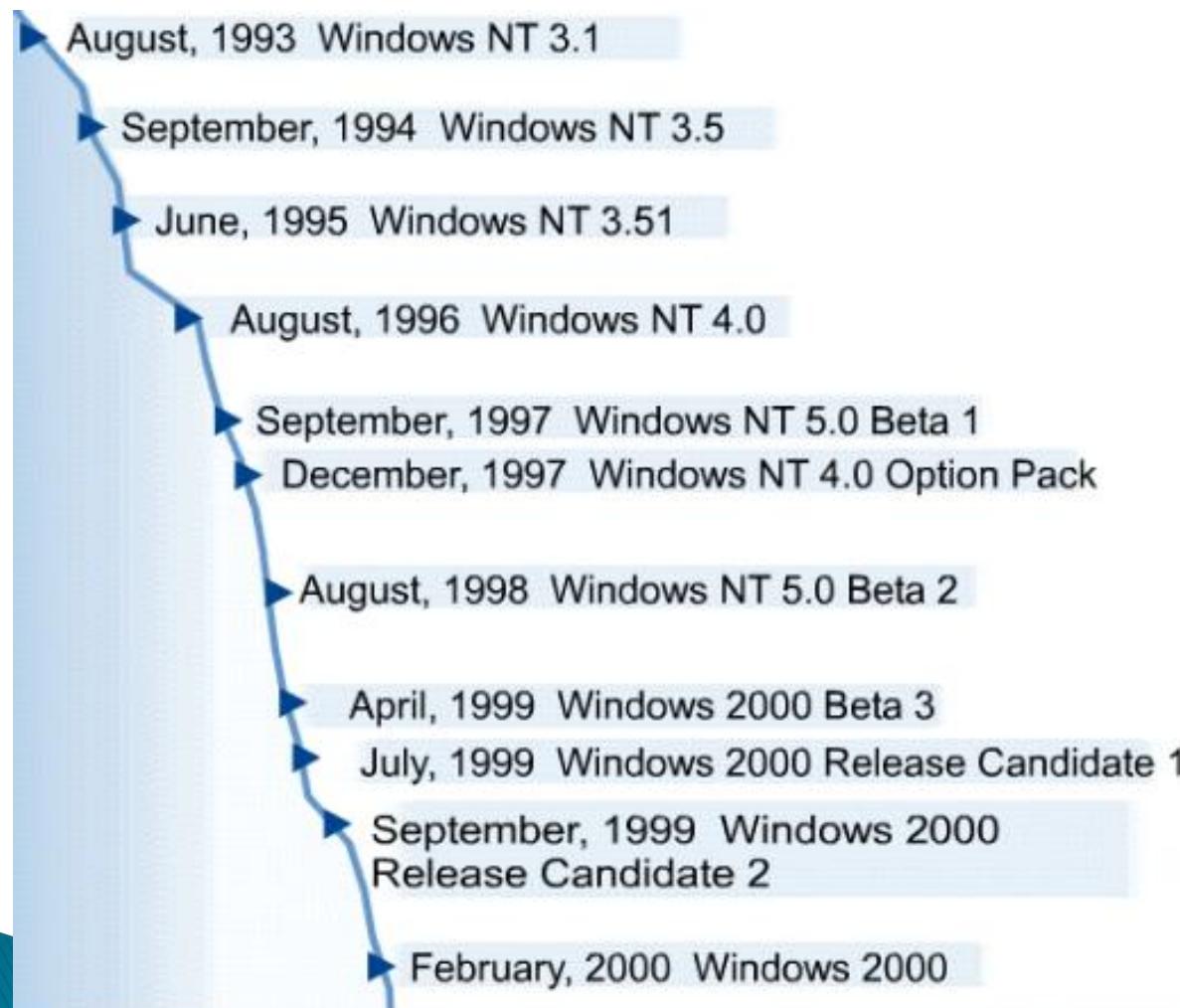
- ▶ Microsoft Disk Operating System (MS-DOS) este unul dintre primele SO desktop ce se mai întâlnește uneori pentru a oferi suport unor aplicații vechi.
- ▶ Microsoft Windows include Windows 95, 98, ME, NT, 2000, XP, Vista, Windows 7, Windows 8.
- ▶ Apple Macintosh OS (Mac OS) include OS 8, OS 9, and OS X (OS 10).
- ▶ Linux include diverse distribuții din partea mai multor companii precum Red Hat, Caldera, Santa Cruz Operation (SCO), SuSE și altele.
- ▶ UNIX include distribuții ale unor mari companii de software, precum HP-UX (HP), Sun Solaris (Sun Microsystems), AIX (IBM), și altele.

# MS-DOS



- ▶ Microsoft a scos pe piata primul produs Windows - Windows 1.0, in 1985.
- ▶ Versiunea Microsoft a SO DOS (MS-DOS) a fost construit pe un SO denumit 86-DOS sau QDOS (Quick and Dirty Operating System).
- ▶ Firma Seattle Computer Products a scris QDOS-ul pentru a rula pe procesorul Intel 8086.
- ▶ IBM a utilizat procesorul 8088 (o versiune mai ieftină) în noua linie de PC-uri.
- ▶ Microsoft a cumpărat drepturile de autor pentru QDOS și a scos pe piață MS-DOS în 1981.

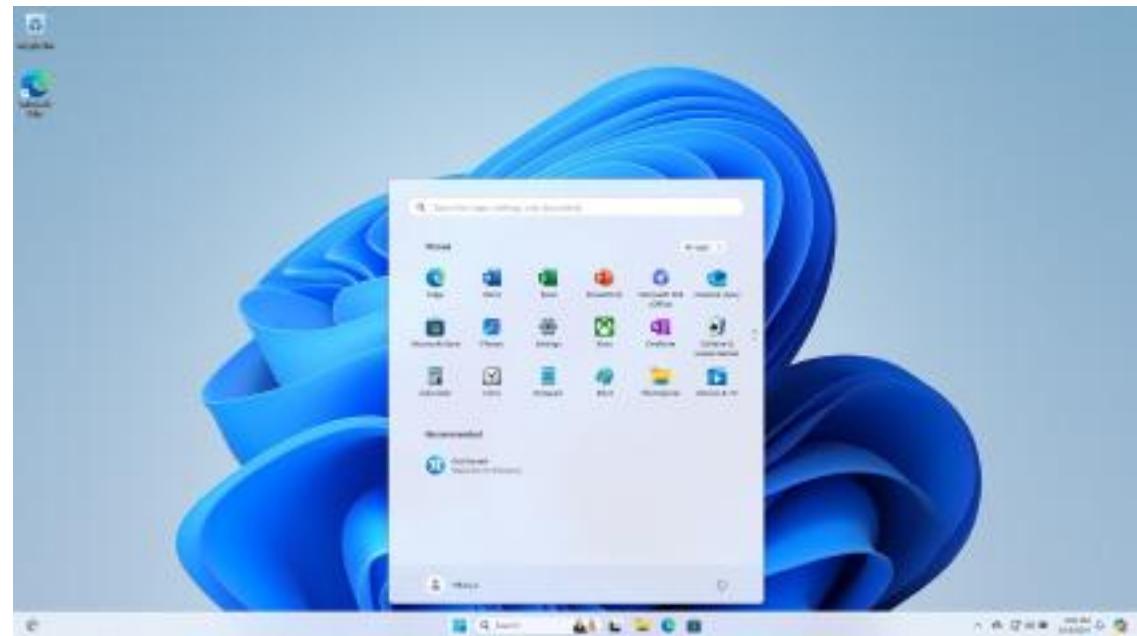
# Windows NT si Windows 2000



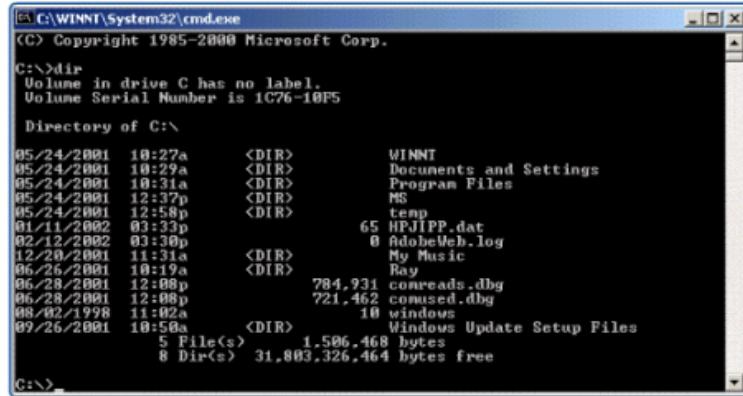
▶ Evolutia familiei de sisteme de operare Windows de la NT 3.1 până la apariția Windows 2000.

# Windows 11

- ▶ Ultima versiune a SO Windows este 11 este aceea din 5 Oct. 2021, cu ultimul update din feb 2025 - 24H2.
- ▶ Versiunea server - Windows Server 2025 a apărut în 2024.
- ▶ Compania Microsoft oferă actualizări majore o dată la fiecare 6 luni.



# Windows CLI



C:\>dir  
Volume in drive C has no label.  
Volume Serial Number is 1C76-10F5  
  
Directory of C:\  
05/24/2001 10:27a <DIR> WINNT  
05/24/2001 10:29a <DIR> Documents and Settings  
05/24/2001 10:31a <DIR> Program Files  
05/24/2001 12:37p <DIR> MS  
05/24/2001 12:58p <DIR> temp  
01/11/2002 03:33p 65 HPJIPP.dat  
02/12/2002 03:30p 0 AdobeWeb.log  
12/28/2001 11:31a <DIR> My Music  
06/26/2001 10:19a <DIR> Ray  
06/28/2001 12:08p 784,931 conreads.dbg  
06/28/2001 12:08p 721,462 conused.dbg  
08/02/1998 11:02a 18 windows  
09/26/2001 10:50a <DIR> Windows Update Setup Files  
5 File(s) 1,586,468 bytes  
8 Dir(s) 31,803,326,464 bytes free  
  
C:\>

Commands	Result
dir	Lists the files in the current directory
cd <i>directory name</i>	Changes to a different directory
time	Displays or sets the system time
date	Displays or sets the date
copy	Copies files to another location
diskcopy [source][destination]	Copies the contents of one floppy disk to another
Attrib	Displays or changes file attributes
find <i>text string</i>	Searches for a text string in a file
help	Displays a list of other available commands and their functions

- ▶ Toate versiunile de Windows include un mediu de lucru la linia de comandă ce permite utilizatorului să folosească comenzile MS-DOS uzuale.
- ▶ Pentru a accesa linia de comandă în Windows, selectați **Run** din meniul Start și tasteazăți **cmd**, în caseta de dialog Run (sau scrieți **cmd** după ce ați apăsat pe lupa de căutare din stânga jos).

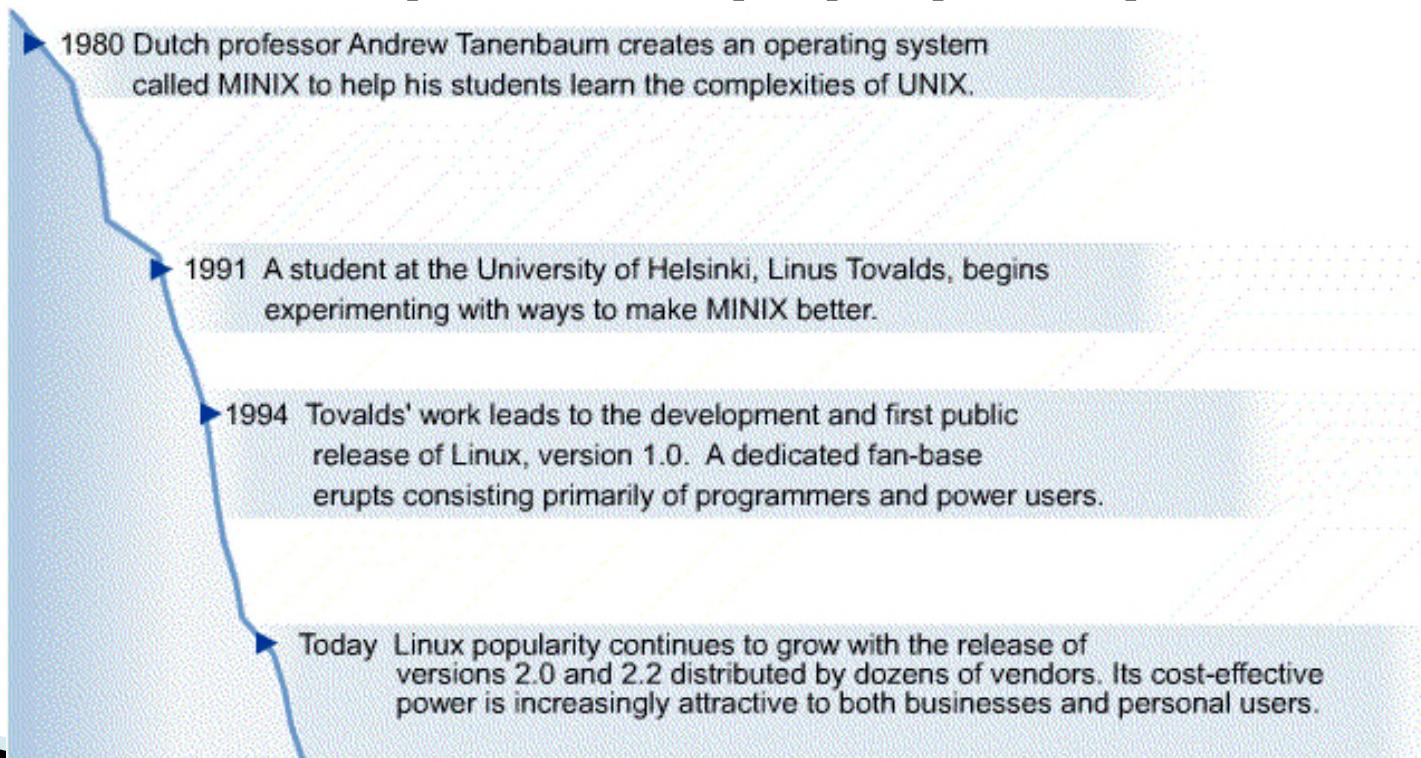
# Unix si Linux pe Desktop

- ▶ Există zeci de variante diferite de UNIX/Linux.
- ▶ O mare parte a rețelei Internet rulează pe sisteme puternice UNIX/Linux.
- ▶ Cu toate că SO UNIX este în general asociat cu hardware scump și este considerat neprietenos cu utilizatorul, ultimile dezvoltări (incluzând aici și crearea Linux-ului) au schimbat această imagine.

Operating System	Popular Uses
UNIX	Web servers FTP servers DNS servers Firewalls Large file servers
Windows	Client workstations Corporate file servers Low-scale web servers

# Originile Linux-ului

- ▶ Începând cu sfârșitul anilor '90, Linux-ul a devenit o alternativă viabilă pentru UNIX pe servere și pentru Windows pe desktop.
- ▶ Popularitatea SO Linux pe desktop a contribuit la interesul folosirii unor distribuții de UNIX pe desktop, precum FreeBSD și Sun Solaris.
- ▶ Versiunile de Linux pot rula acum pe aproape orice procesor.

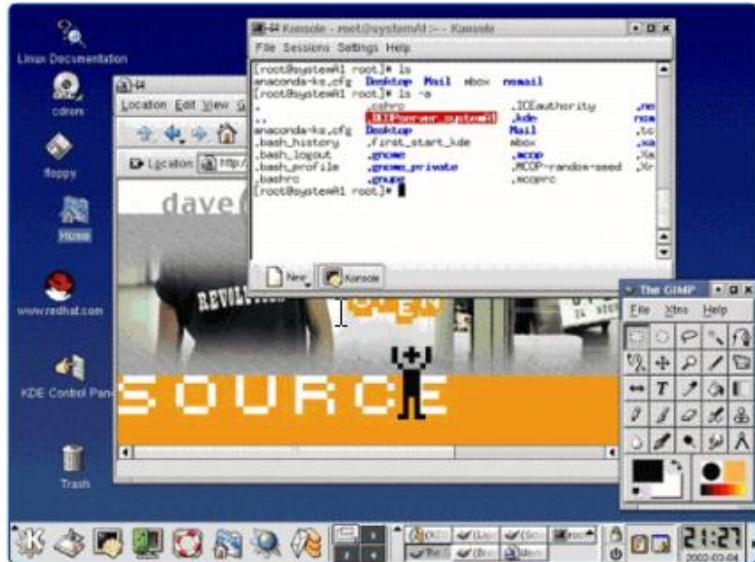


# Linux/UNIX GUI



- ▶ Atât UNIX-ul cât și Linux-ul sunt capabile să ruleze interfețe grafice de tip GUI.
- ▶ Din cauza faptului că există atât de multe versiuni diferite de UNIX și Linux, există o serie întreagă de interfețe grafice dintre care putem alege.
- ▶ UNIX/Linux se bazează pe sistemul X Window pentru a afișa interfața grafică.
- ▶ GNOME (GNU Network Object Model Environment) este un GUI și un set de aplicații dezvoltate pentru Linux.

# Linux/UNIX GUI

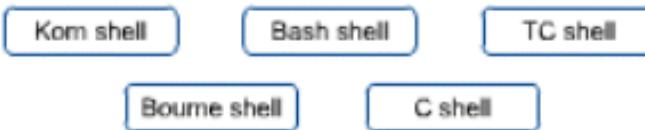


Window Button	Function
[Minimize]	Minimizes a window. A minimized window can be restored by clicking its title on the task bar (the taskbar is part of the panel).
[Maximize]	Maximizes a window.
[Close]	Closes a window. If the window contains an application running in the foreground, this option will terminate the application.

- ▶ Există mai multe medii grafice desktop pentru Linux, precum KDE (Kool Desktop Environment) și GNOME. Dintre acestea, GNOME a devenit treptat un GUI "standard" UNIX și Linux.
- ▶ SO Linux oferă suport pentru zeci de "window managers", astfel încât fiecare poate fi setat în funcție de dorințe - nu există un anume standard de design pe care trebuie să îl respecte o fereastră.

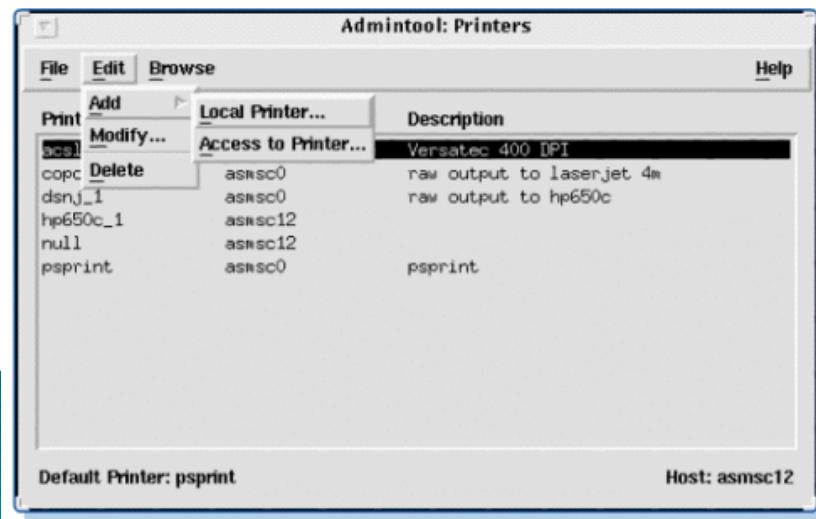
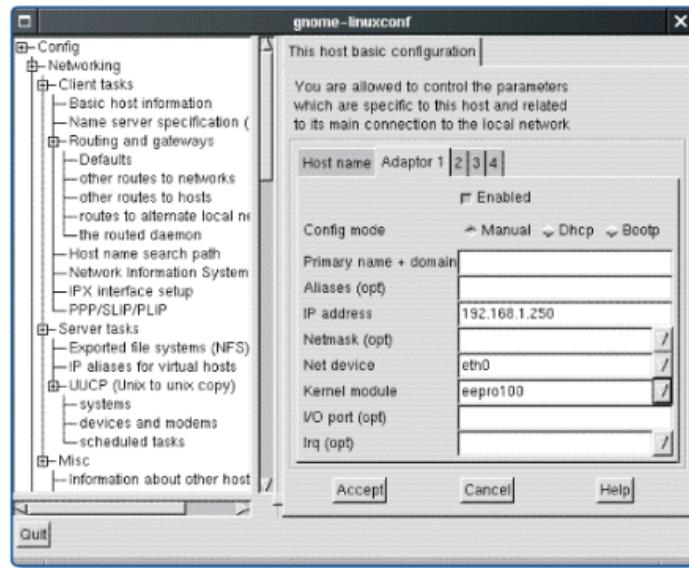
# Originile SO UNIX

ksh	
bash	
tcsh	
csh	
sh	



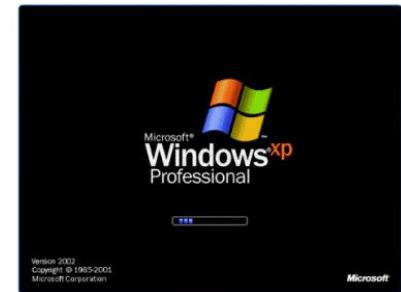
- ▶ SO UNIX și Linux au fost proiectate pentru a fi flexibile și personalizabile.
- ▶ SO UNIX și Linux oferă suport pentru diverse interfețe utilizator.
- ▶ Cele mai cunoscute interfețe bazate pe text se numesc shell-uri.
- ▶ Utilizatorii tastează comenziile, care sunt apoi interpretate de către shell.

# Instrumente de configurare a SO Linux și UNIX



- ▶ Diversele versiuni de UNIX și Linux oferă o mulțime de instrumente de configurare asemănătoare cu Control Panel-ul din Windows.
- ▶ Există astfel atât instrumente la linia de comandă pentru medii CLI, cât și instrumente pentru medii grafice (de ex. *linuxconf* pentru Linux, sau *admintool* pentru Solaris).

# SO de rețea



- ▶ Limitările primelor SO de tip desktop au condus la dezvoltări software mai puternice: SO de rețea (astăzi, practic toate sistemele de operare sunt So de rețea)
- ▶ SO de rețea au incluse componente și servicii de conectare la rețea, rulare multiuser și tehnologii sofisticate de securitate și partajare de fișiere.
- ▶ Principalele SO de rețea folosite astăzi:
  - Microsoft Windows
  - Linux
  - Unix



**debian**

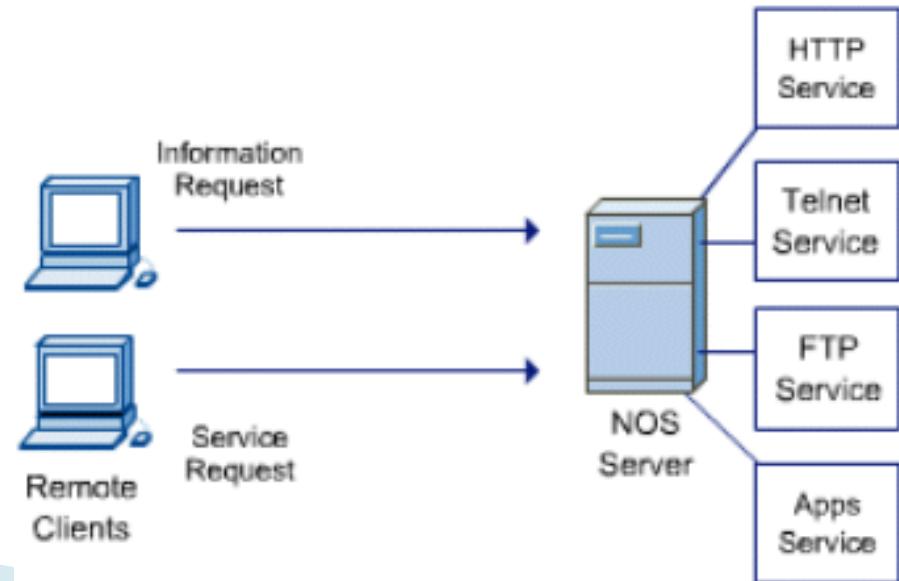


# Scurtă comparație între SO Windows și Linux

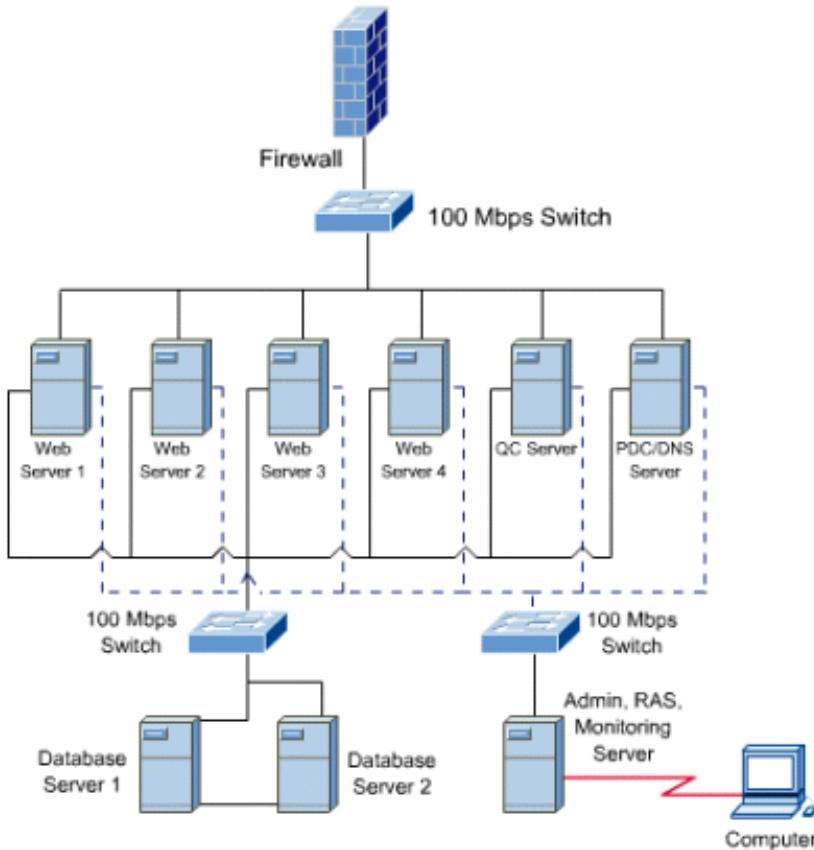
- ▶ Windows a fost introdus pe piață pentru a fi “user-friendly”, cu o interfață grafică (GUI), inițial ca SO desktop.
  - ▶ Rădăcinile Linux-ului încep cu UNIX și cu proiectarea modulară ce a făcut popular Linux-ul printre administratorii de sisteme.
- 
- ▶ Interfață în mod text/grafic (ambele)
  - ▶ Costuri (diferențe)
  - ▶ Modalitatea de obținere/instalare a SO
  - ▶ Abilitatea de a rula direct de pe CD
  - ▶ Disponibilitatea aplicațiilor și modul de obținere a software-ului
  - ▶ Vulnerabilitatea față de viruși
  - ▶ Caracteristici de securitate
  - ▶ Suport multi-utilizator

# Modelul Client-Server

- ▶ Majoritatea aplicațiilor de rețea ce includ aplicațiile Internet - precum World Wide Web (WWW) și e-mail, sunt construite pe baza unei relații client/server.
- ▶ Un server oferă servicii de rețea (spre exemplu, e-mail) altor programe denumite clienți.
- ▶ Odată pornit, un program server așteaptă să primească cereri din partea programelor client. Dacă este recepționată o cerere corectă, serverul răspunde printr-un mesaj ce conține informația respectivă către client.



# Modelul Client-Server



- ▶ Orice computer poate acționa ca server atât timp cât este conectat la rețea și configurat corespunzător.
- ▶ Majoritatea companiilor se bazează pe folosirea serviciilor cheie de rețea prin instalarea unor computere high-end (servere) ce rulează SO de rețea optimizate pentru asigurarea serviciilor clientilor la distanță.

# Sisteme de operare

Curs 3

Caracteristicile unui SO modern

# Comenzi de bază Unix/Linux

- **cd** (change directory) – comandă folosită pentru a naviga prin sistemul de fișiere
- cd ..
- cd
- cd ~
- cd \$HOME
- cd /etc/opt

Atenție la căile relative/căile absolute

# Comenzi de bază Unix/Linux

- **pwd** (print working directory) – comandă folosită pentru afișarea directorului curent de lucru
- **ls** (list) – comandă folosită pentru listarea conținutului unui director
- ls
- ls -lS
- ls -lSr (r=reverse)
- ls -la
- ls -R (R=recursiv)
- ls -lR > lista.txt

# Comenzi de bază Unix/Linux

- **mv** (move) – comandă folosită pentru redenumirea unui fișier
  - mv file1 file2
  - mv file1 dir1
- **cp** (copy) – comandă folosită pentru copierea unui fișier
  - cp file1 file2
  - cp -r dir1 dir2

# Comenzi de bază Unix/Linux

- **rm** (remove) – comandă folosită pentru ? unui fișier
- rm file2
- **rmdir** (remove directory) – comandă folosită pentru ? unui director, dacă acesta este gol
- rmdir dir\_gol
- rm -r dir\_care\_nu\_este\_gol

# Test 1

- ls -l | pause
- cat files > pause
- cat files | more
- ls -l | more

Care dintre următoarele comenzi va face listarea conținutului directorului curent ecran cu ecran ?

# Test 2

- -W
- -i
- -r
- -F

Dacă dorim să copiem sau să mutăm un fișier în altă locație și să fim avertizați pentru a nu șterge accidental un fișier existent, ce opțiune trebuie să folosim ?

# Test 3

- copy test /dir3
- cp test ../dir3
- copy test ../dir3
- cp /notes dir3

Directorul curent este **/home/stud1010/dir2**. Dacă dorim să copiem fișierul **test** din directorul curent în directorul **/home/stud1010/dir3**, ce comandă putem utiliza?

# Test 4

- whoisloggedon | sort
- who | sort
- whois | sort
- id | sort

Pentru a vedea cine este conectat la sistem, iar rezultatul să fie sortat după **user id**, ce comandă putem folosi ?

# Test 5

- chmod 742 fisier
- chmod 754 fisier
- chmod 764 fisier
- chmod 731 fisier

Care dintre comenziile anterioare va acorda drept de **citire**, **scriere**, **execuție** pentru utilizator, drept de **citire și scriere** pentru grup și drept de **citire** pentru ceilalți ?

# Test 6

- chmod g+x fisier
- chmod u-e fisier
- chmod g+e fisier
- chmod o+x fisier

Care dintre comenziile anterioare va acorda permisiune de **execuție** pentru grup ?

Altă variantă de stabilire a permisiunilor:

**chmod g=r-x**

De reținut: **u g o**

# Caractere de control în UNIX

- Așa cumitele caractere de control sunt utilizate pentru a îndeplini anumite funcții, precum oprirea sau continuarea afișării pe ecran, terminarea execuției unui program, etc.
- Majoritatea tastaturilor de PC au două taste de control (inscripționate cu **Ctrl**), în stânga jos, respectiv dreapta jos. Atunci când este afișată pe ecran, tasta Ctrl este reprezentată de semnul ^

Exemple de caractere de control:

- Ctrl-s** – oprește afișarea textului pe ecran
- Ctrl-q** – reia afișarea textului pe ecran (oprîtă cu Ctrl-s)
- Ctrl-c** – întrerupe activitatea curentă și se folosește de regulă pentru a opri procese sau afișări pe ecran.
- Ctrl-d** – semnifică sfârșitul fișierului sau ieșire, fiind utilizat pentru a ieși din unele utilitare Unix, ieșirea unei ferestre terminal sau pentru **logout**.
- Ctrl-u** – șterge întreaga linie de comandă, fiind o modalitate rapidă de a șterge o linie de comandă pe care ne-am decis să nu o mai executăm.
- Ctrl-w** – șterge ultimul cuvânt introdus la linia de comandă
- Ctrl-h** – șterge ultimul caracter introdus la linia de comandă, fiind folosit atunci când tasta <BACKSPACE> nu funcționează

# Determinarea tipului de fișier

- Comanda **file**

Sintaxa generală:

**file** *nume\_fisier*

Rezultatul comenzi anterioare poate fi: text,  
executabil, date.

```
-bash-3.00$ file I*
ICR.pdf:          Adobe Portable Document Format (PDF) v1.4
```

# Afișarea conținutului unui fișier ASCII (text)

- Comanda **cat**

Sintaxa generală:

**cat** *nume\_fisier*

- Comanda **more**

Sintaxa generală:

**more** *nume\_fisier*

- Comanda **head**

Sintaxa generală:

**head** [-n] *nume\_fisier*

- Comanda **tail**

Sintaxa generală:

**tail** [-n] *nume\_fisier*

# Alte comenzi pentru lucrul cu fișiere

- Comanda **wc** (**word count**) –utilizată pentru a număra linii, cuvinte, octeți sau caractere într-un fișier

Sintaxa generală:

**wc [optiune] nume\_fisier**

unde opțiunile sunt:

- l linii
- w cuvinte
- c octeți
- m caractere

**wc pico.save**

4            13            64 pico.save  
4 lini, 13 cuvinte, 64 de octeți

# Alte comenzi pentru lucrul cu fișiere

- Comanda **diff (difference)** –utilizată pentru a compara două fișiere text și a afla diferențele dintre ele

Sintaxa generală:

**diff [optiune] fisier\_1 fisier\_2**

Rezultatul comenuzii afișează diferențele linie cu linie dintre cele două fișiere text

Putem folosi două opțiuni:

- i ignoră diferențele între litere mari și mici
- c oferă o comparare detaliată: întâi sunt afișate datele referitoare la creare pentru cele două fișiere, apoi liniile din fisier\_1, cu semnul – în fața celor diferite față de cele din fisier\_2. La fel pentru fisier\_2, cu semnul + pentru liniile diferite față de fisier\_1

# Comenzi Unix/Linux

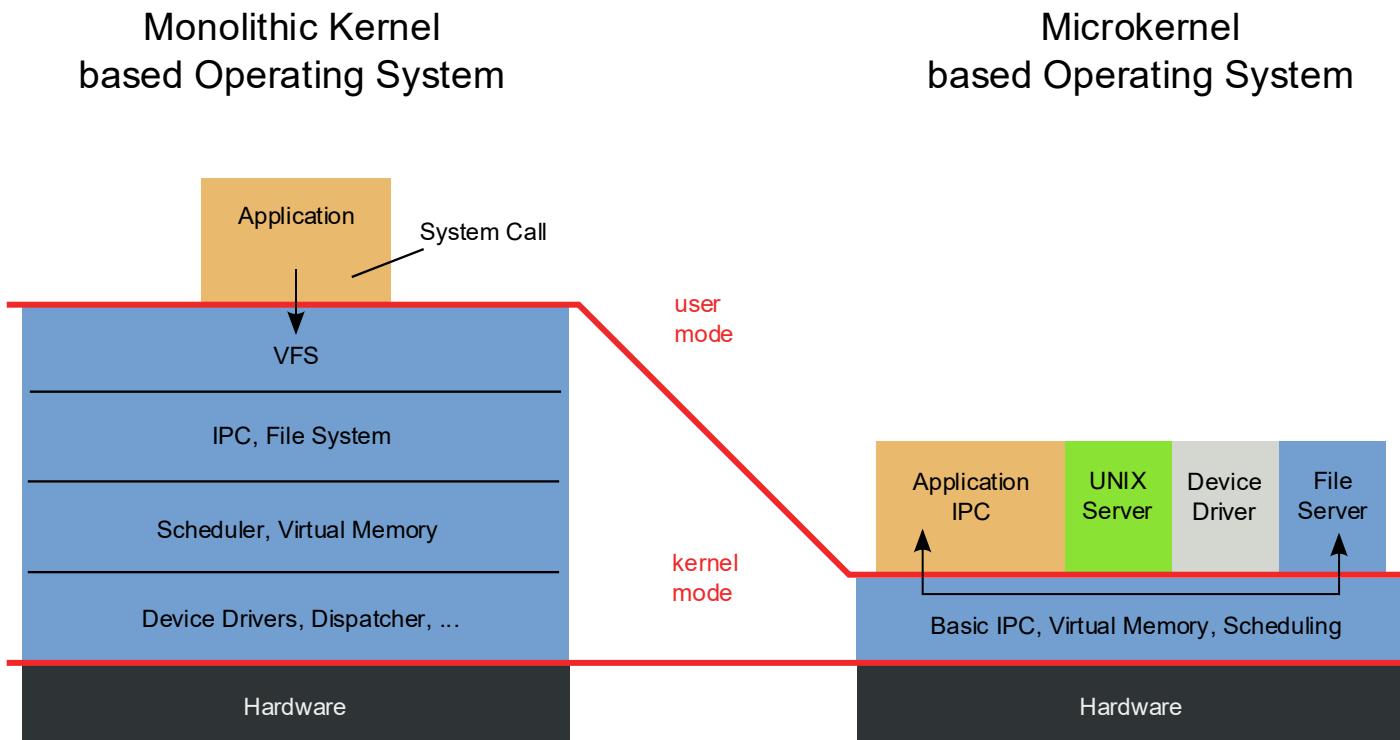
- <http://www.computerhope.com/unix.htm>

# Caracteristicile unui SO modern

- Arhitectură Microkernel
- Multithreading
- Multiprocesare simetrică
- Sisteme de operare distribuite
- Proiectare orientată obiect

# Caracteristicile unui SO modern

- Arhitectură **Microkernel**



# Caracteristicile unui SO modern

- **Multithreading**
- Un sistem de operare multithreading este un sistem de operare care susține și gestionează multithreading-ul într-un sistem de calcul. Multithreading-ul este un model de programare și execuție care permite mai multor fire (unități mai mici de execuție ale unui proces) să ruleze concurent în cadrul unui singur proces. Fiecare fir reprezintă o secvență de instrucțiuni, iar mai multe fire în cadrul unui proces pot rula independent și împărții resursele.
- Într-un sistem de operare cu multithreading, kernel-ul sistemului de operare este capabil să gestioneze și să programeze simultan mai multe fire de execuție. Acest lucru permite utilizarea mai eficientă a resurselor CPU și îmbunătățește reactivitatea în gestionarea sarcinilor, deoarece diferite fire se pot executa simultan.

# Caracteristicile unui SO modern

- **Multithreading**
- Multithreading-ul poate aduce beneficii precum îmbunătățirea paralelismului, creșterea reactivității și o utilizare mai bună a resurselor sistemului.
- Cu toate acestea, necesită sincronizare și coordonare atentă pentru a gestiona resursele partajate și pentru a evita probleme potențiale, cum ar fi accesul concurențial la date și blocajele.

# Caracteristicile unui SO modern

- **Multiprocesare simetrică**
- Multiprocesarea simetrică (SMP) se referă la un tip de arhitectură de multiprocesare în care două sau mai multe procesoare identice sunt conectate la o singură memorie principală partajată și sunt controlate de o singură instanță a sistemului de operare.
- Într-un sistem SMP, fiecare procesor are acces egal la resursele sistemului, inclusiv la memorie și dispozitive de intrare/ieșire (I/O). Această arhitectură permite mai multor procesoare să lucreze simultan la diferite sarcini, îmbunătățind performanța generală a sistemului.

# Caracteristicile unui SO modern

- **SO distribuite**
- Un sistem de operare distribuit este un sistem de operare care rulează pe mai multe calculatoare și le permite să lucreze împreună ca un mediu de calcul unitar. Într-un sistem de operare distribuit, resursele precum puterea de procesare, memoria și stocarea sunt distribuite pe întreaga rețea, iar sistemul oferă o vedere transparentă și integrată utilizatorilor și aplicațiilor.
- Exemple de sisteme de operare distribuite includ Chrome OS de la Google, care utilizează o arhitectură distribuită pentru serviciile bazate pe cloud, și diverse variante de Linux create pentru medii de calcul distribuit.
- În concluzie, un sistem de operare distribuit permite mai multor calculatoare să colaboreze și să partajeze resurse pentru a oferi un mediu de calcul eficient și lipsit de discontinuități.

# Introducere

## Prezentare generală a caracteristicilor unui sistem de operare de rețea

- **Funcția de bază a unui sistem de operare este aceea de a controla hardware-ul calculatorului, mediul de execuție al programelor și interfața utilizator.**
- Sistemul de operare asigură îndeplinirea acestor funcții pentru un singur utilizator sau pentru mulți utilizatori ce partajează calculatorul într-o manieră mai degrabă secvențială decât concurrentă.

# Introducere

## Prezentare generală a caracteristicilor unui sistem de operare de rețea

- Spre deosebire, sistemele de operare de rețea asigură o distribuție a funcțiunilor de-a lungul unor calculatoare conectate în rețea. Un sistem de operare de rețea depinde de sistemul de operare existent pe fiecare calculator în parte. Apoi adaugă funcții ce permit accesul la resursele partajate. Figura 1 ne prezintă relațiile de tip **pereche, client-server și mainframe**.
- Calculatoarele cu sisteme de operare de rețea au roluri specializate pentru a îndeplini accesul partajat la resurse. Sistemele client posedă software specializat ce permite ca acestea să acceseze resursele partajate ce sunt controlate de către sisteme server ce oferă răspunsuri cererilor client.
  - Figura 2 ne prezintă conceptul prin care datele sunt stocate pe servere și sunt disponibile cererilor client.

Figura 1

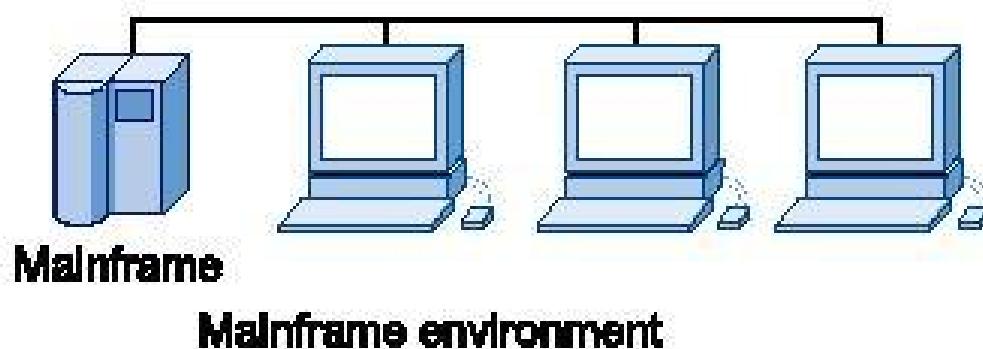
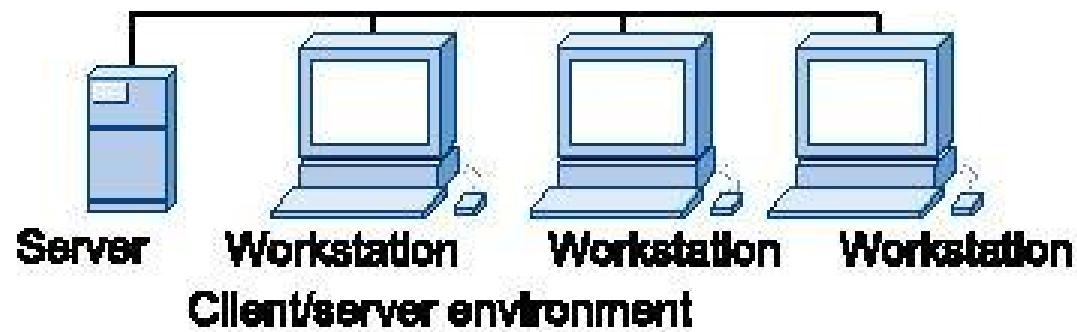
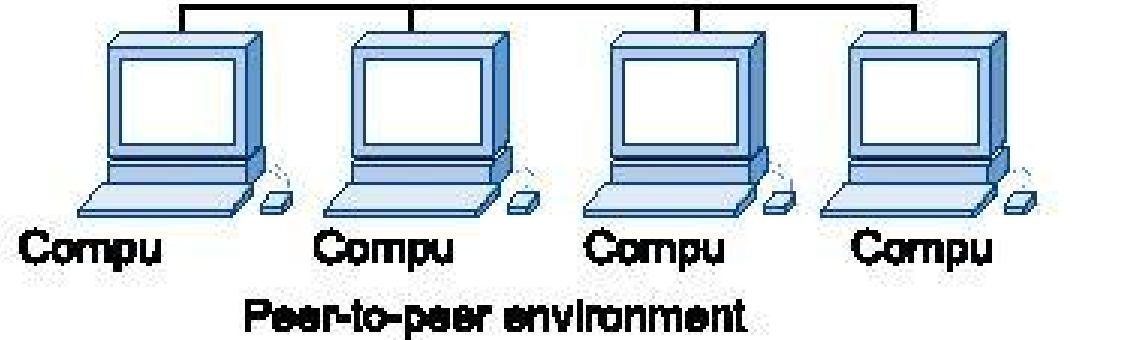
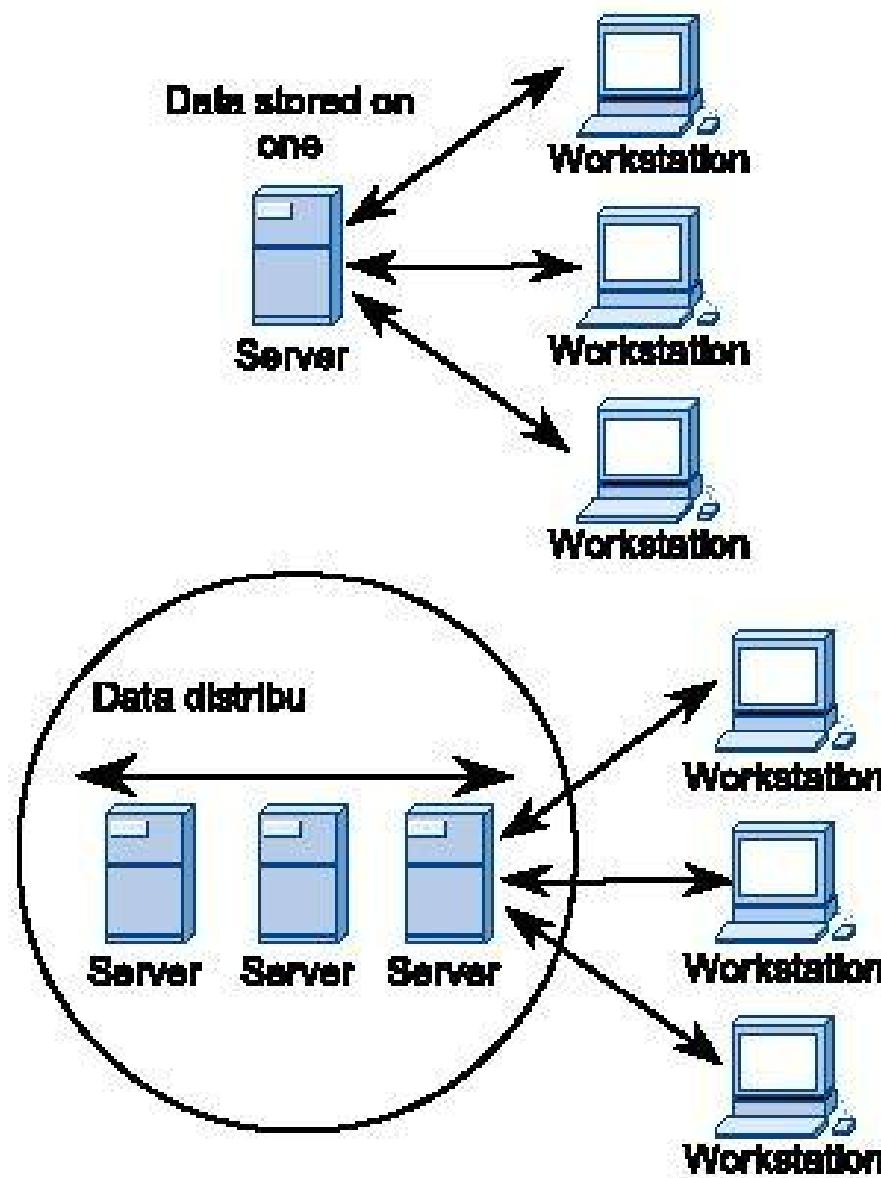


Figura 2



## Sisteme multiutilizator

Pentru a putea oferi suport pentru mai mulți utilizatori simultan și pentru a putea accesa resursele partajate ale rețelei (servicii, echipamente, diverse resurse), serverele de rețea trebuie să ruleze sisteme de operare cu caracteristici extinse față de sistemele de operare clasice (de tip client).

Dintre sisteme de operare cele mai cunoscute ce oferă servicii de rețea enumerăm: **Unix/Linux, Windows NT/2000/XP/2003/Vista/7/8/10.**

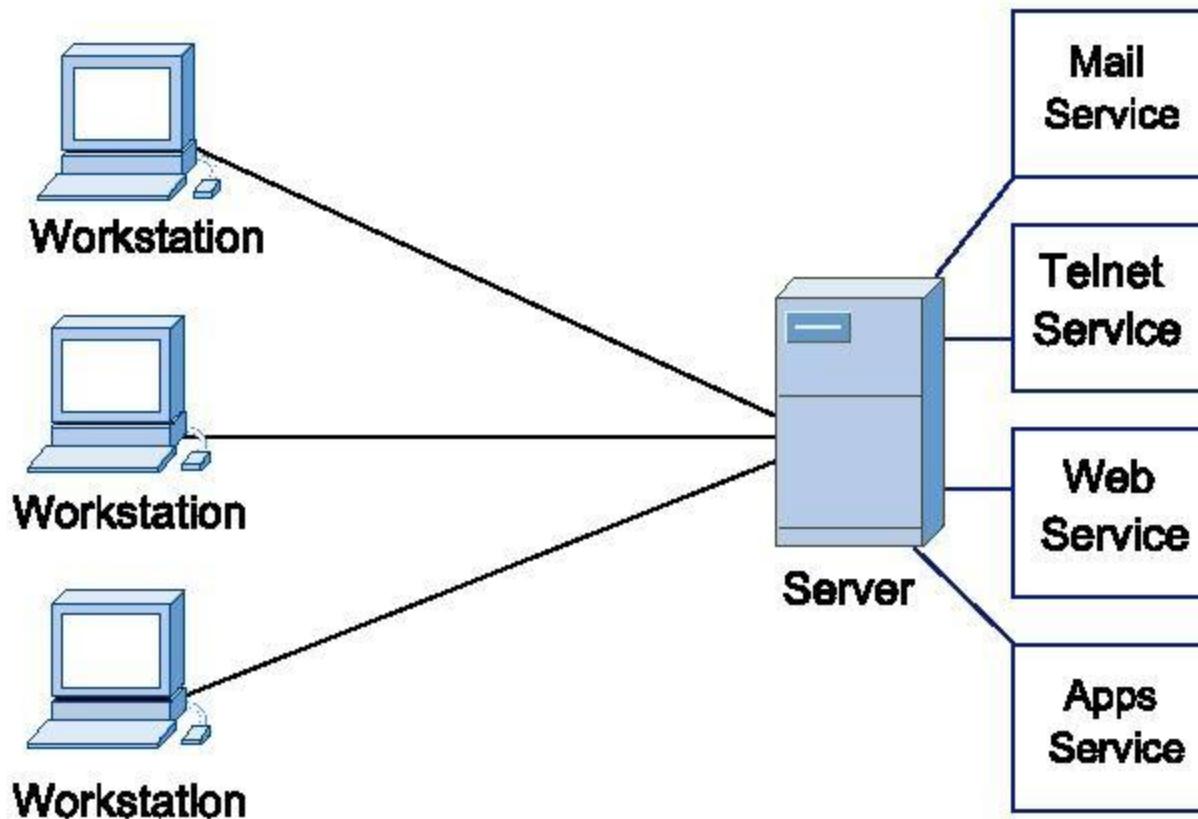
Un sistem capabil să funcționeze ca server NOS trebuie să ofere suport pentru mai mulți utilizatori simultan. Administratorul de rețea crează câte un cont pentru fiecare utilizator în parte, fapt ce permite ca fiecare utilizator să se conecteze pe sistemul server.

Un cont utilizator permite serverului să autentifice utilizatorul și să aloce resursele la care acel utilizator are acces. Sistemele ce oferă această funcționalitate se numesc **sisteme multiutilizator**. UNIX, Linux și Windows NT/2000/XP/2003/Vista/7/8/10/11 sunt exemple de astfel de sisteme.

# Sisteme multitasking

- De asemenea, un sistem de operare de rețea este un **sistem multitasking**. Acest lucru semnifică faptul că, intern, sistemul de operare este capabil să execute mai multe sarcini (*tasks*) sau procese în același timp. Sistemele de operare server realizează acest lucru printr-un cod software de planificare ce este integrat în mediul de execuție. Acest planificator are rolul de a **aloca timpul procesorului, memoria și alte elemente ale sistemului** pentru mai multe sarcini în aşa fel încât această alocare să permită **partajarea resurselor sistemului**.
- Fiecare utilizator de pe un sistem multiutilizator are ca suport un task sau un proces separat pe server. Aceste sarcini interne sunt create în mod dinamic pe măsură ce utilizatorii se conectează la sistem sau, dimpotrivă, sunt șterse atunci când utilizatorii se deconectează de la server.

# Multitasking



# Multitasking și multithreading

Figura anterioară ilustrează un server multitasking tipic, ce rulează mai multe instanțe de servicii de rețea ce sunt accesate de către mai multe sisteme client. Serverele de acest tip se regăsesc, de regulă, sub denumirea de servere de întreprindere (*enterprise servers*) datorită capacitații mari de a administra date și servicii complexe.

Servele de tip enterprise sunt capabile să ruleze copii concurente ale unei comenzi. Acest fapt permite execuția mai multor instanțe ale aceluiași serviciu sau fir de execuție al unui program. Termenul “fir de execuție” (**thread**) descrie un program ce are capacitatea de a se executa independent de altele. Sistemele de operare ce suportă “multithreading” permit programatorilor să proiecteze programe ale căror părți divizate în fire de execuție să fie **executate concurențial**.

# Multitasking cooperativ vs. preemptiv

**Multitasking-ul cooperativ** reprezintă un mediu în care programele partajează adrese de memorie și pot schimba informații între ele.

Într-un mediu multitasking, aplicațiile partajează utilizarea procesorului prin metoda “time-slicing”.

Programele sunt scrise astfel încât să renunțe la utilizarea procesorului după un anumit timp pentru a permite altor programe să folosească procesorul.

Dacă un program este prost scris, poate monopoliza întreaga activitate a procesorului; de asemenea, dacă un program se blochează, poate duce la blocarea altor programe.

Multitasking-ul cooperativ ducea de multe ori, pentru versiuni ale SO Windows, la apariția BSOD

([https://ro.wikipedia.org/wiki/Blue\\_Screen\\_of\\_Death](https://ro.wikipedia.org/wiki/Blue_Screen_of_Death)).

## Multitasking cooperativ vs. preemptiv

O formă mult mai eficientă de multitasking este implementată începând cu Windows 9x și se numește **multitasking preemptiv**. În acest caz, SO controlează alocarea timpului procesorului, iar programele pe 32/64 de biți rulează în spații separate de memorie. În cazul multitasking-ului preemptiv, un program ce nu respectă regula nu poate monopoliza sistemul, iar dacă se blochează, nu va afecta alte programe.

În Windows Task Manager (începând cu Win 2000/XP) utilizatorii pot vedea toate procesele și programele ce rulează pe sistem precum și identificatorii de proces (PID) pe care SO le folosește pentru a face distincție între procesele ce rulează pe sistem.

# Partea 1: Pipes, redirectare și REGEX

\$ ls /fake > output.txt

ls: cannot access /fake: No such file or directory

**Redirectarea fișierului standard de eroare (STDERR) cu 2>**

\$ ls /fake 2> error.txt

\$ more error.txt

ls: cannot access /fake: No such file or directory

Fie comanda următoare:

\$ echo 2 \* 3 > 4 este o inegalitate validă.

Ce credeți că se va afișa pe ecran?

Obs. \* este un caracter special în Unix/Linux

# Pipes, redirection and REGEX

Redirectarea atât a **STDERR** cât și a **STDOUT** cu **&>**

```
$ ls /fake /etc/ppp &> all.txt
```

```
$ cat all.txt
```

```
ls: cannot access /fake: No such file or directory /etc/ppp:  
chap-secrets
```

```
ip-down
```

```
ip-down.ipv6to4
```

```
ip-up ip-up.ipv6to4
```

```
ipv6-down
```

```
ipv6-up
```

```
options
```

```
pap-secrets
```

```
peers
```

# Pipes, redirection and REGEX

## Comanda **tr** (translate)

```
$ tr 'a-z' 'A-Z'
```

watch how this works

WATCH HOW THIS WORKS

```
$ tr 'a-z' 'A-Z' < sample.txt
```

Obs. "<" este folosit pentru redirectarea input-ului

```
$ tr 'a-z' 'A-Z' <<< 'Transformare in majuscule'
```

# Comenzi noi Unix/Linux

**find** – folosită pentru a găsi fișiere după un criteriu de căutare

**find** *starting\_dir matching\_criteria [options]*

find [starting directory] [search option] [search criteria] [result option]

Examples:

find /usr -name startx

find /etc -name hosts 2> errors.txt

find /etc -name hosts 2> /dev/null

find /etc -name hosts -ls 2> /dev/null

find /usr -name '\*tif'

find . -name dir05 -type d (d – directory, f - file)

## *Comanda **find***

Căutare după dimensiune:

`find /etc -size +300` (*size more than 300 blocks – one block is 512 bytes*)

When you specify a file size, you can give the size in bytes (c), kilobytes (k), megabytes (M) or gigabytes (G)

**sysadmin@localhost:~\$** `find /etc -size 10c -ls 2>/dev/null`  
(finds files that are exactly 10 bytes large)

`find . -mtime +5`

(*matches files modified more than 5 days ago*)

`find ~ -perm 777`

`find ~ -user stud03 -ls > listastud03`

# Mai multe opțiuni pentru comanda *find*

Dacă se folosesc mai multe opțiuni, acestea acționează ca un „și”, ceea ce înseamnă că, pentru a se potrivi, toate criteriile trebuie să fie îndeplinite, nu doar unul.

De exemplu, următoarea comandă va afișa toate fișierele din structura directorului **/etc** care au o dimensiune de 10 octeți și sunt fișiere obișnuite:

```
sysadmin@localhost:~$ find /etc -size 10c -type f -ls 2>/dev/null
432 4 -rw-r--r-- 1 root root 10 Jan 28 2015 /etc/adjtime
73468 4 -rw-r--r-- 1 root root 10 Nov 16 20:42 /etc/hostname
```

# Comanda ***cut*** folosită pentru a filtra conținutul fișierelor

Comanda ***cut*** poate extrage coloane de text dintr-un fișier sau din fișierul standard de intrare. Principala utilizarea a comenții ***cut*** este de a lucra cu fișiere baze de date ce folosesc delimitatori.

```
sysadmin@localhost:~$ cut -d: -f1,5-7 mypasswd
root:root:/root:/bin/bash
daemon:daemon:/usr/sbin:/bin/sh
bin:bin:/bin:/bin/sh
sys:sys:/dev:/bin/sh
sync:sync:/bin:/bin/sync
```

# Comanda *cut* folosită pentru a filtra conținutul fișierelor

Folosind comanda *cut* se pot extrage coloane de text pe baza poziției unui caracter folosind opțiunea **-c**. Acest lucru poate fi util atunci când dorim să extragem câmpuri din fișiere baze de date. Sper exemplu, comanda următoare va afișa tipul de fișier (primul caracter din listing), permisiunile (caracterele #2-10) și numele fișierului (caracterele #47+) din output-ul comenzi **ls -l** :

```
sysadmin@localhost:~$ ls -l | cut -c1-11,47-
```

```
total 12
```

```
drwxr-xr-x Desktop
```

```
drwxr-xr-x Documents
```

```
drwxr-xr-x Downloads
```

```
drwxr-xr-x Music
```

```
drwxr-xr-x Pictures
```

**Comanda grep** (este folosită pentru a căuta într-un fișier sau în output-ul unei comenzi)

**Sintaxa generală:**

**grep [options] what\_to\_search file\_name**

**Exemplu:**

\$ grep root /etc/passwd

\$ grep test ./\*

\$ ls -la | grep -i 'mar 14'

(-i ignore case)

# Expresii regulate

- O *expresie regulată* este o colecție de caractere ‘normale’ și ‘speciale’ ce sunt folosite să se potrivească cu diverse şabloane, mai simple sau mai complexe. Caracterele ‘normale’ sunt caracterele alfanumerice ce corespund valorii lor. Spre exemplu, litera *a* va corespunde unui *a*.

Regular Expression	Matches
.	Any single character
[ ]	A list or range of characters to match one character, unless the first character is the caret ^, and then it means any character not in the list
*	Previous character repeated zero or more times
^	Following text must appear at beginning of line
\$	Preceding text must appear at the end of the line

# Expresii regulate - exemplu

```
$ echo 'abcd' > example.txt
```

```
$ cat example.txt
```

```
abcd
```

```
$ grep 'a..' example.txt
```

```
abcd
```

```
$ grep 'a..c' example.txt
```

# Expresii regulate - exemple

Spre exemplu, următoarea comandă se va potrivi cu două caractere , primul fie un **a** sau un **b**, în timp ce al doilea poate fi un **a, b, c** sau **d**:

```
$ grep '[ab][a-d]' example.txt
```

```
abcd
```

Observație: putem lista toate valorile dorite precum [abcd] sau putem folosi un interval [a-d], atâtă timp cât acesta este în ordinea corectă. Spre exemplu, [d-a] nu este un interval valid:

```
$ grep '[d-a]' example.txt
```

```
grep: Invalid range end
```

# Expresii regulate - exemple

Pentru a indica faptul că vrem să specificăm faptul că un caracter nu este unul din caracterele listate, vom folosi la începutul sevenței ([ ]) semnul special **^**. Spre exemplu, următoarea comandă va căuta şablonul ce include un caracter ce nu este nici **a**, nici **b**, nici **c**, urmat de caracterul **d**:

```
$ grep '[^abc]d' example.txt  
abcd
```

Caracterul special **\*** poate fi utilizat să corespundă "zero sau mai multe caractere egale cu caracterul anterior". Spre exemplu, comanda următoare se va potrivi căutării a 'zero sau mai multe caractere **d**':

```
$ grep 'd*' example.txt  
abcd
```

# Expresii regulate - exemple

Când facem o potrivire de şablon, potrivirea poate avea loc oriunde pe linie. Putem face căutarea potrivirii la începutul sau la sfârşitul liniei. Pentru a se potrivi la începutul liniei, şablonul va începe cu simbolul ^ .

În exemplul următor, se adaugă o altă linie la fișierul example.txt pentru a demonstra utilizarea simbolului ^ :

```
$ echo "xyzabc" >> example.txt
```

```
$ cat example.txt
```

```
abcd
```

```
xyzabc
```

```
$ grep "a" example.txt
```

```
abcd
```

```
xyzabc
```

```
$ grep "^a" example.txt
```

```
abcd
```

# Expresii regulate - exemple

Pentru a căuta potrivirea la sfârșitul liniei, va trebui să încheiem şablonul cu caracterul special **\$**.

Spre exemplu, pentru a găsi doar liniile ce se termină cu litera **c** vom folosi comanda:

```
$ grep "c$" example.txt
```

xyzabc

```
$ grep "cd*" example.txt
```

abcd~~dd~~

xyzabc

abcd\*

Dacă dorim să căutăm chiar caracterul **\***, vom folosi caracterul de evitare **backslash (\)** înaintea sa:

```
$ grep "cd\*" example.txt
```

abcd\*

# **Partea a 2-a**

# **Structura și componentele unui SO**

**<https://zota.ase.ro/so>**

# Structura și componentele unui SO

- Componente
- Apeluri de sistem
- Integrarea componentelor
- Mașini virtuale

# Structura SO

## Componente de sistem

Managementul proceselor

Managementul memoriei principale

Managementul fișierelor

Managementul sistemului I/O

Managementul memoriei secundare

Conecțarea la rețea

Sistemul de protecție

Sistemul de interpretare al comenziilor

# Structura SO

## Managementul proceselor

Un proces este o instanță a unui program în execuție (un program este pasiv, un proces este activ).

Un proces are diverse resurse (timp UCP alocat, fișiere) și atribute ce trebuie administrate.

Managementul proceselor include:

- Planificarea proceselor (stabilirea priorităților, managementul timpului, etc.)
- Crearea/terminarea
- Blocarea/Deblocarea (suspendarea/reluarea)
- Sincronizarea
- Comunicarea
- Administrarea blocajelor
- Depanarea

## Structura SO

### Managementul memoriei principale

- Alocarea/de-alocarea pentru procese, fișiere, I/O.
- Administrarea mai multor procese în același timp
- Se ține cont de cine utilizează memoria
- Deplasarea memoriei proceselor către/de la memoria secundară.

### Managementul fișierelor

*Un fișier reprezintă o colecție de date/informații definit de creatorul său.* În mod normal, fișierele pot reprezenta programe (atât programe sursă cât și programe obiect) sau date.

SO este responsabil cu următoarele activități în legătură cu managementul fișierelor:

- Crearea și ștergerea fișierelor
- Crearea și ștergerea directoarelor
- Oferirea de suport pentru manipularea fișierelor și directoarelor
- Deplasarea fișierelor în memoria secundară
- Realizarea de copii de siguranță pentru fișiere pe medii de stocare non-volatile

# **Structura SO**

## **Managementul I/O**

- Sistemul “buffer caching”
- Cod generic pentru drivere de echipamente
- Drivere pentru fiecare dispozitiv - translatează cererile de citire/scriere în comenzi de poziționare pe disc

## **Managementul memoriei secundare**

- Discuri, benzi magnetice, optice, etc.
- Administrarea spațiului liber (paginare/swapping )
- Alocarea spațiului pe disc (ce date sunt scrise și unde pe disc)
- Planificări de citire/scriere de pe/pe disc

# Componente de sistem

## Structura SO

### Conecțarea la rețea

- Sistem de comunicație între procesoare distribuite
- Obținerea de informații despre fișiere/procese, etc. pe o mașină aflată la distanță
- Poate utiliza fie un model “message passing” sau un model de memorie partajată

### Protecție

- A fișierelor, memoriei, UCP, etc.
- = Controlul accesului
- Depinde de atributele fișierului și ale utilizatorului

Cum interacționează toate aceste componente?

În principal, toate *oferă servicii* unele altora.

### Programe de sistem

- Compilatoare/link-editoare/asamblătoare, etc.
- Comunicații (ftp, telnet, ssh, etc.)
- Interpretoare de comenzi – programe ce preiau securitatea de control (comenzi) (shell, interfață grafică)

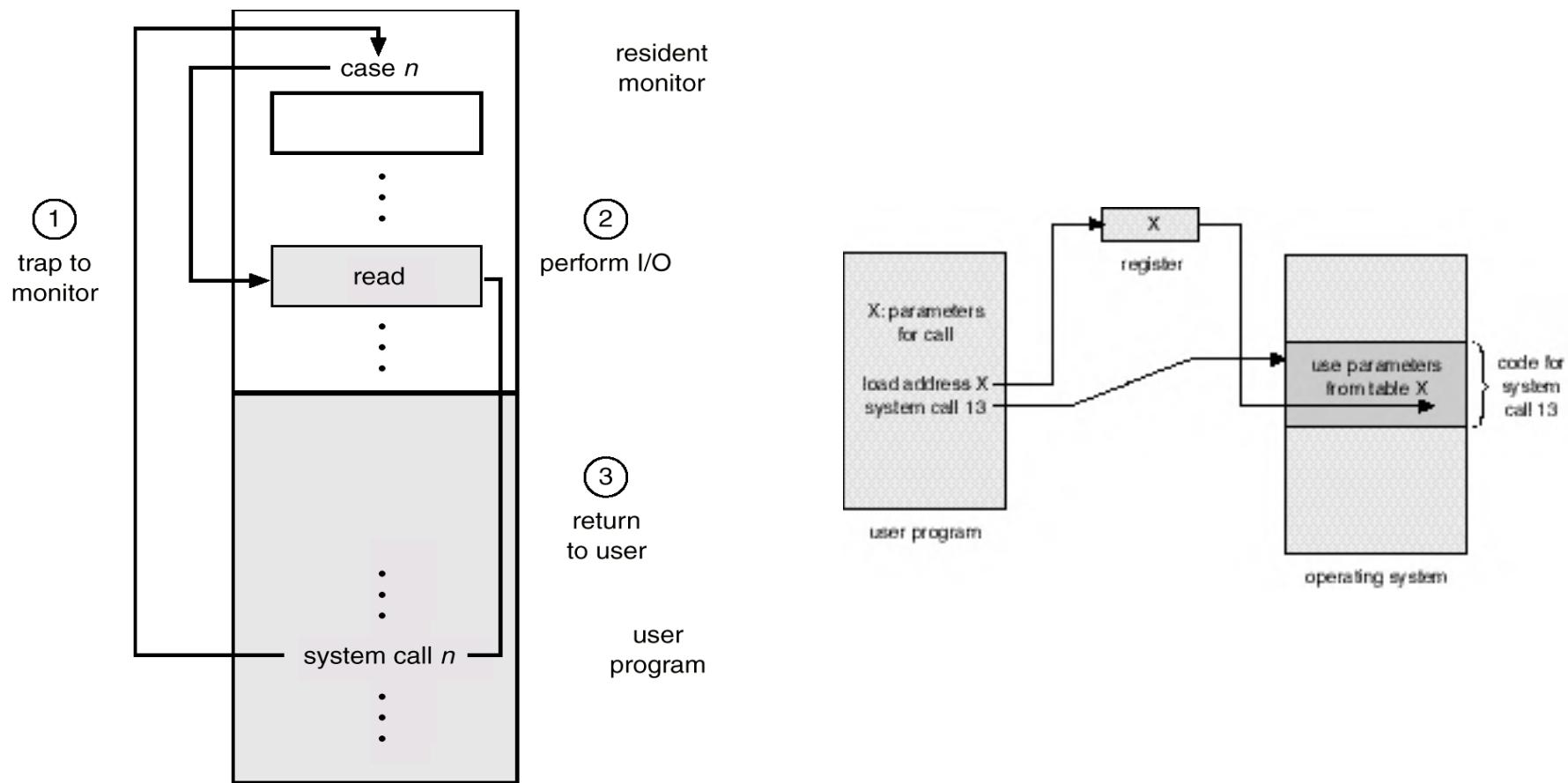
## Structura SO

- În mod normal un SO poate oferi suport pentru multe dispozitive posibile, dar fiecare instalare necesită doar o parte din aceste variante posibile.
- Facilitatea **Plug and play** permite detecția echipamentelor și includerea automată a codului (driverelor) necesare pentru ca aceste echipamente să funcționeze.
- Un **sysgen** reprezintă o legătură de mai multe rutine/module ale SO pentru a produce un executabil ce conține codul necesar pentru rularea driverelor.

# Componente de sistem

## Structura SO

Un apel de sistem reprezintă principala modalitate prin care un program utilizator interacționează cu SO.

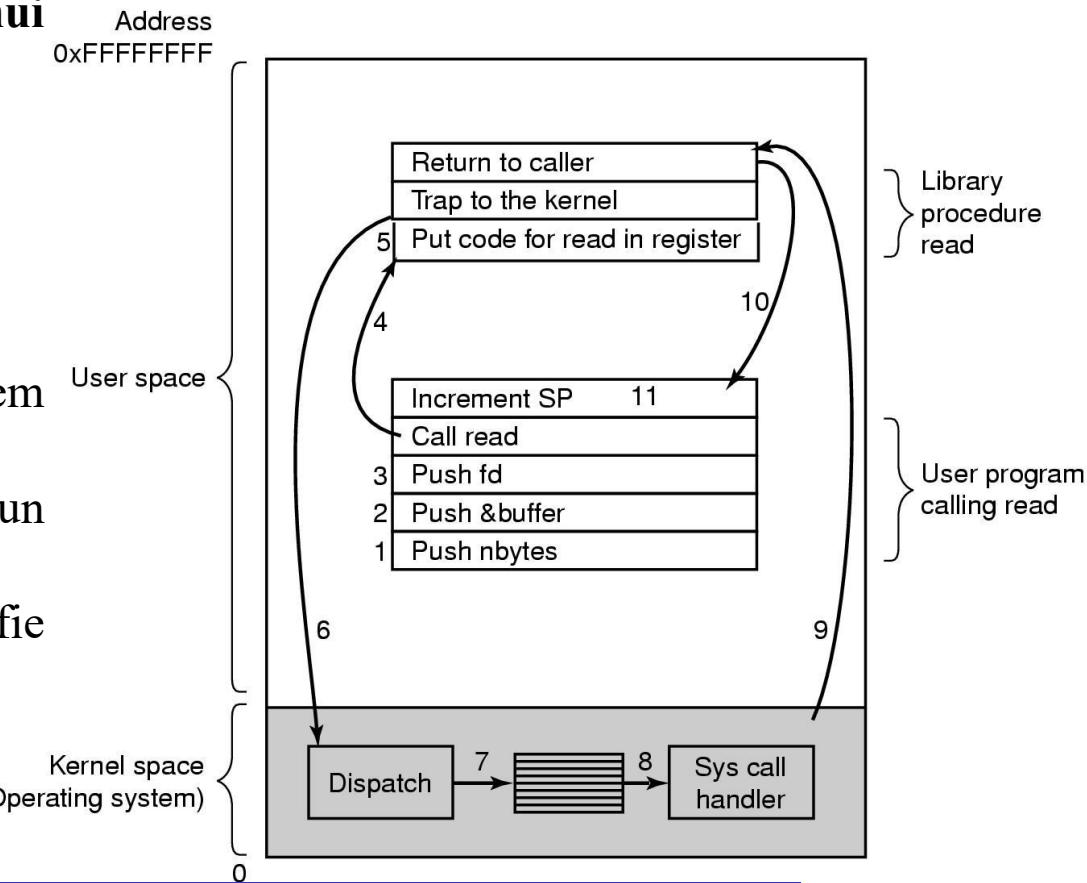


# Structura SO

## Componente de sistem

### Modalitatea de funcționare a unui apel de sistem

- Obține acces la spațiul sistem
- Face validarea parametrilor
- Face apel la resursele de sistem
- Interoghează un echipament/sistem pentru un anumit element
- Suspendă așteptarea pentru un echipament
- Întreruperea face ca acest thread să fie gata de execuție
- Mascare
- Întoarcere la utilizator

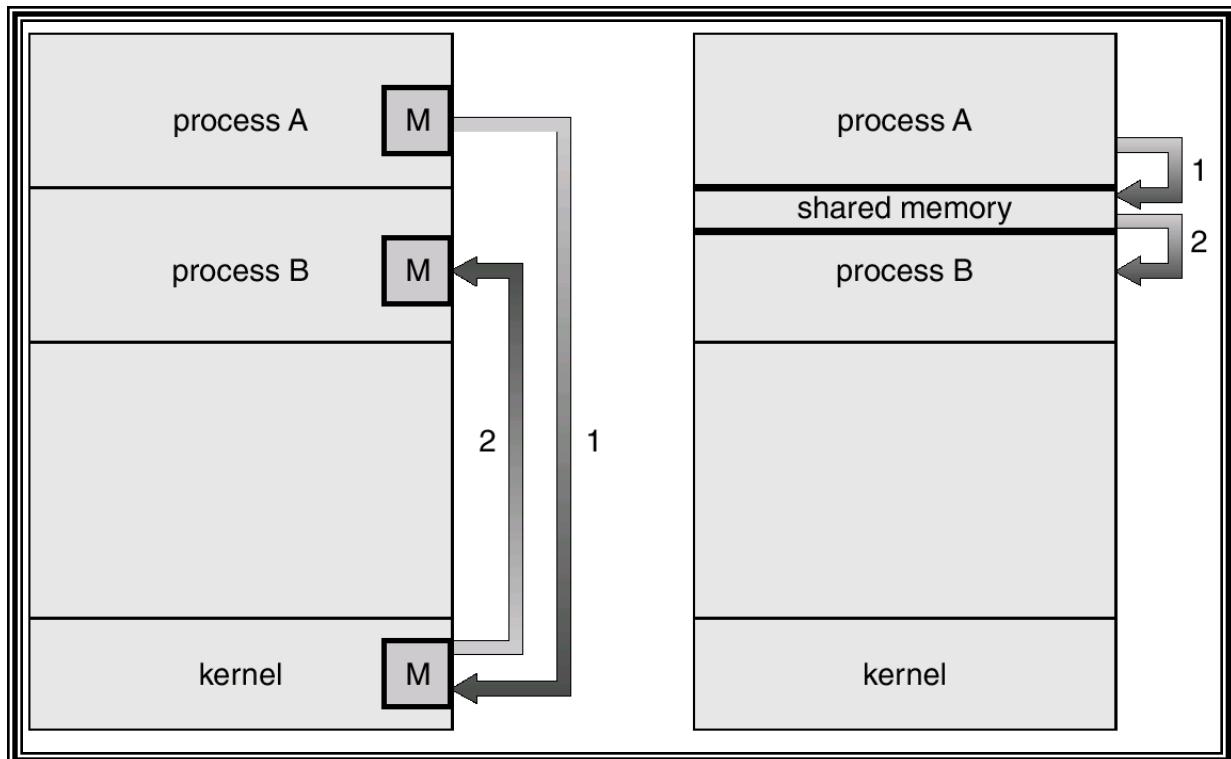


De regulă există 11 (sau mai mulți) pași la un apel de sistem  
**read (fd, buffer, nbytes)**

# Structura SO

Componente de sistem

Există două metode de transfer al datelor între programe:



“Message Passing”

Memorie partajată

# Structura SO

## Componente de sistem

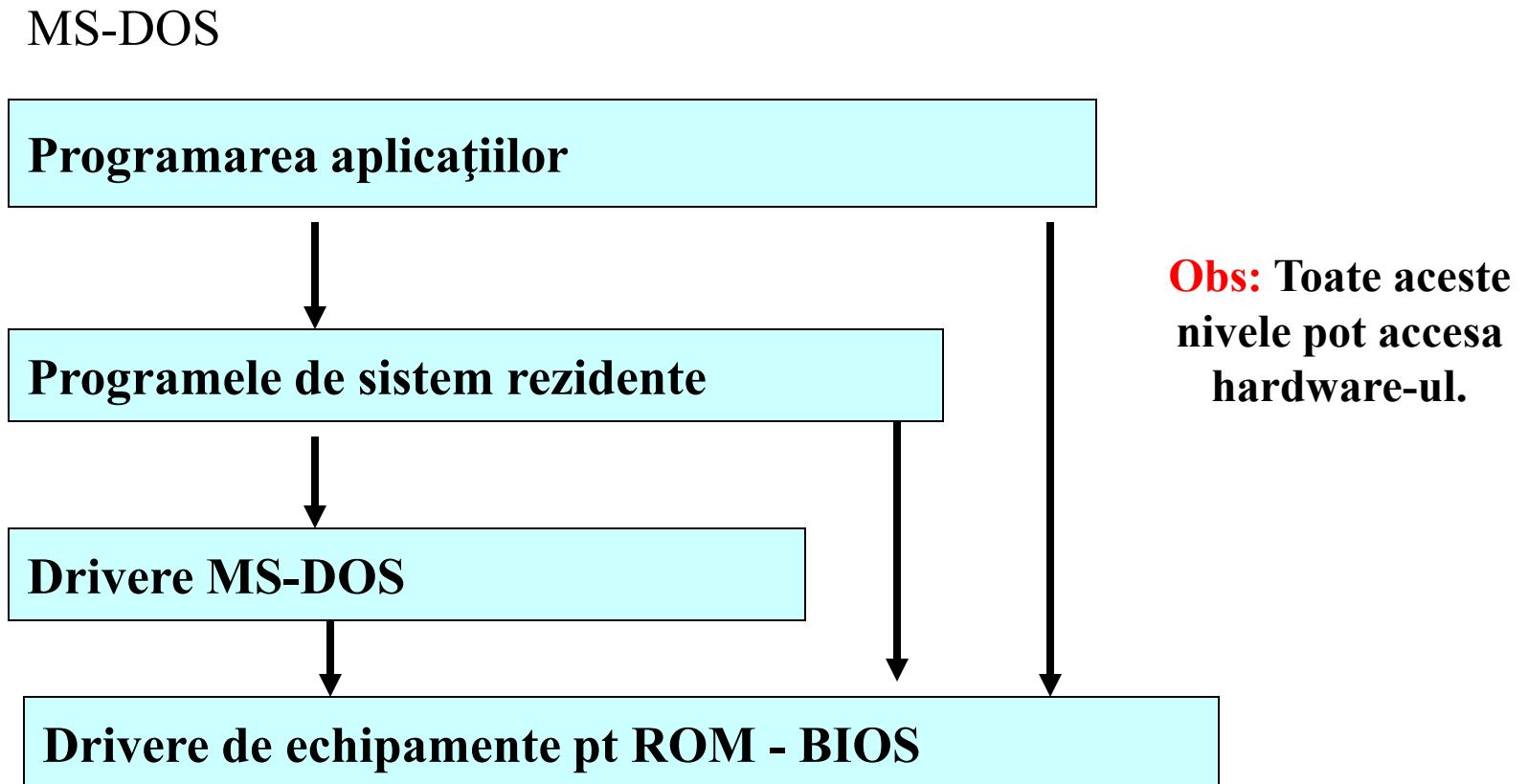
Exemplu de  
apeluri de sistem:

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

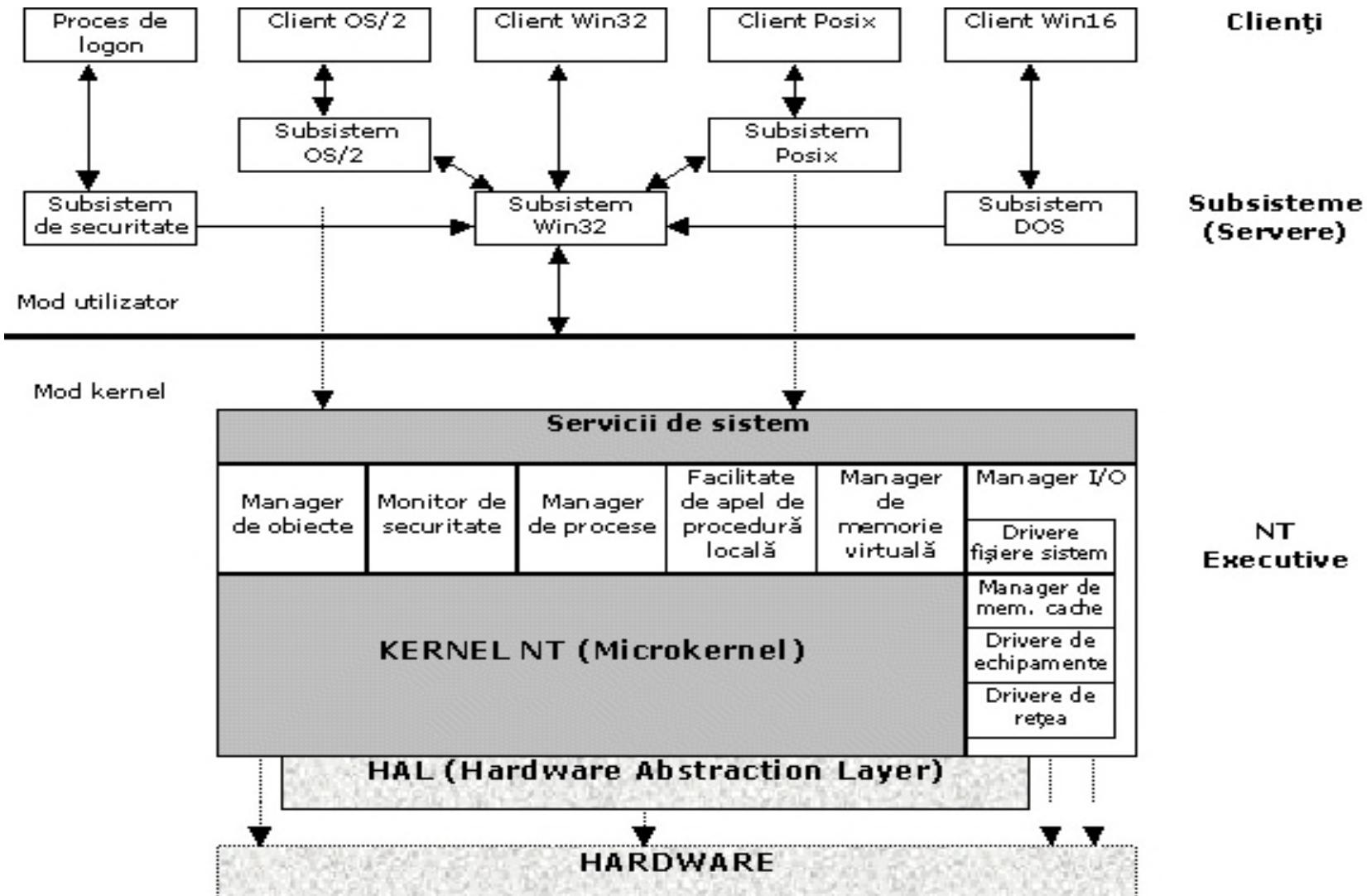
# Structura SO

“Asamblarea”  
componentelor  
SO

O structură simplă:



# Structura SO



# Structura SO

## Nucleul (kernelul)

**Nucleul** se ocupă cu întregul trafic de mesaje ce se desfășoară în cadrul sistemului de operare și rulează peste HAL. Nucleul este ocupat în principal cu **manipularea intreruperilor și excepțiilor** pentru comunicația între subsisteme și resursele hardware ale sistemului de operare.

Parte integrantă a managementului tuturor comunicațiilor dintre subsisteme, nucleul este responsabil și cu verificarea constantă cu subsistemul de securitate a administratorului NT pentru a se asigura faptul că cererile pentru servicii au fost autorizate în mod corespunzător.

# Structura SO

Nucleul NT

Nucleul NT este responsabil cu:

- sincronizarea mai multor procesoare atunci când Windows NT rulează pe un calculator ce suportă multiprocesare simetrică (SMP);
- manipularea întreruperilor și a excepțiilor;
- refacerea sistemului în caz de cădere;
- verificarea securității și respectării restricțiilor;
- programarea firelor de execuție în mediul NT *multi-threading* (multiple fire de execuție ale proceselor).

**Manipularea întreruperilor** ocupă cel mai mult din timpul nucleului NT, o întrerupere NT fiind generată pentru fiecare interacțiune a subsistemelor administratorului NT.

**Nucleul NT rulează în mod privilegiat** și de aceea nu poate fi niciodată expulzat din memorie.

# Structura SO

## Administratorul NT

Administratorul NT (*NT Executive*) este compus din nucleul NT la care se adaugă o varietate de subsisteme cunoscute împreună sub numele de **servicii sistem**. Printre aceste servicii se află:

- managerul intrărilor și ieșirilor (managerul I/O);
- managerul apelului de procedură locală;
- managerul de obiecte;
- managerul de procese;
- managerul memoriei virtuale;
- monitorul de securitate.

# Structura SO

## Managerul I/O

### *Managerul I/O*

Acesta are în sarcină administrarea tuturor intrărilor și ieșirilor pentru sistemul de operare Windows.

Managerul I/O este în mod special preocupat cu administrarea comunicațiilor dintre driverele de echipament, driverele de rețea, managerul memoriei cache și driverele sistemelor de fișiere.

*Driverele de echipament* (sau drivere de echipament hardware) sunt scrise în special pentru a suporta anumite dispozitive periferice cum ar fi imprimanta, tastatura sau mouse-ul. Windows furnizează un mediu standardizat cuprins în managerul I/O în care aceste drivere pot rula.

# Structura SO

Managerul I/O

Datorită acestui mediu standardizat driverele dispozitivelor periferice pot rula pe orice platformă care suportă Windows. Aceste drivere sunt scrise în C și pot fi ușor modificate sau adăugate.

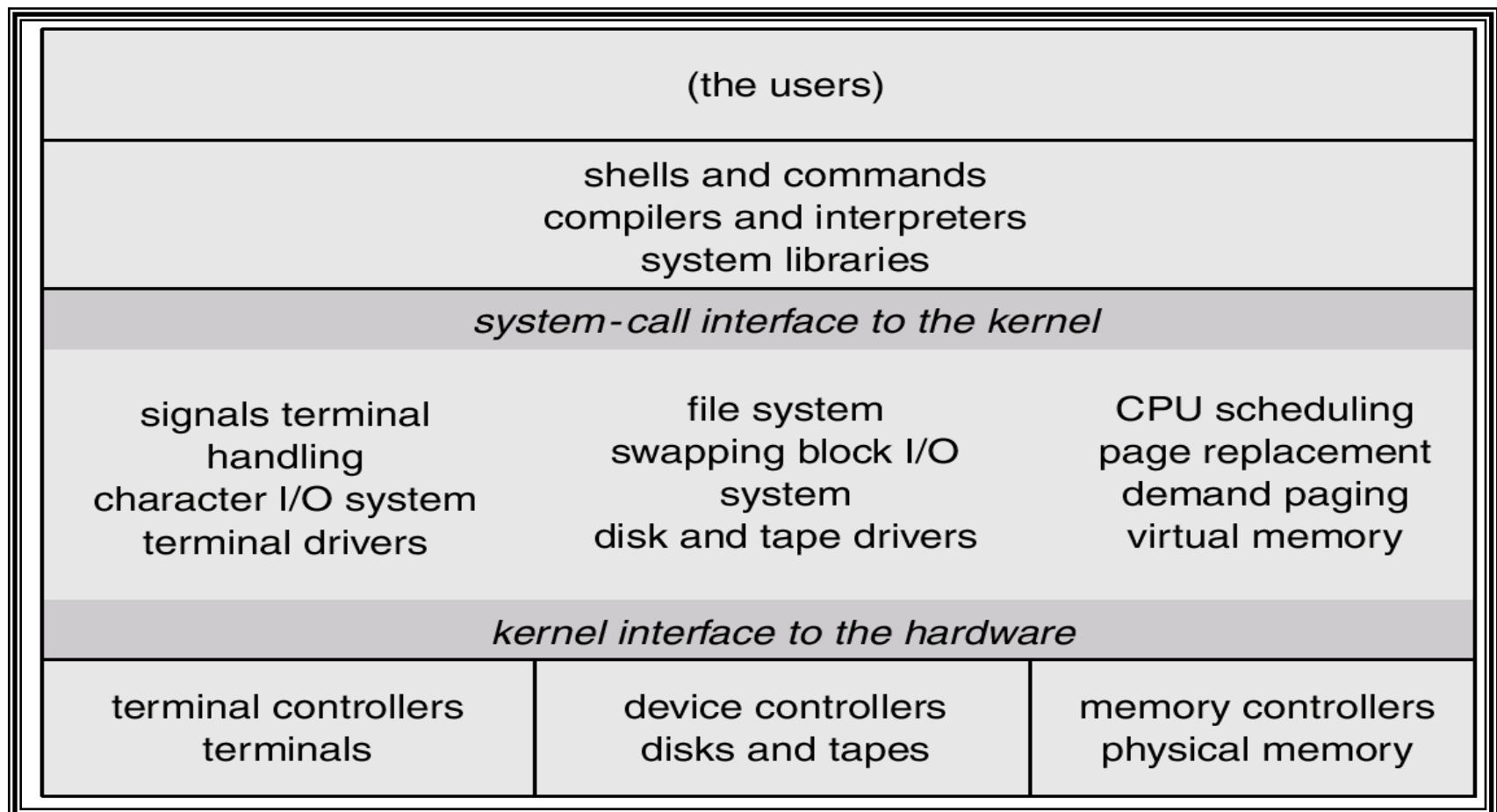
Printre *driverele de rețea* existente în Windows se află următoarele: NetBIOS, redirector și interfața server SMB cu aplicațiile și sistemul de fișiere.

Protocole de comunicație ca TCP/IP, NetBEUI, IPX/SPX furnizând servicii transport.

# Structura SO

## Componente de sistem UNIX

### Structura pe nivale UNIX:



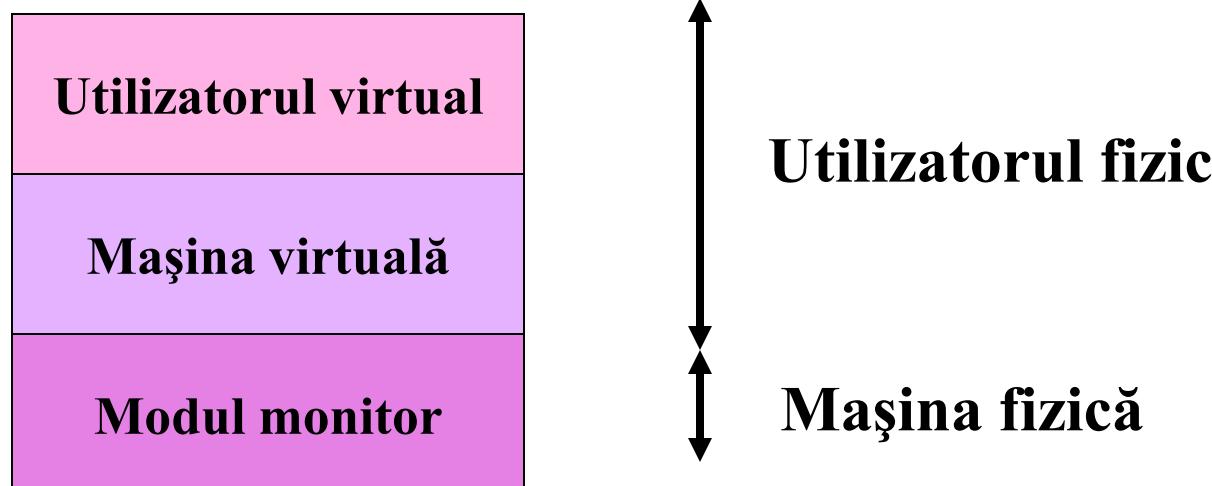
# Structura SO

**Mașini virtuale**

Într-o “mașină virtuală” fiecare proces pare să se execute pe propriul procesor și cu propria memorie, echipamente, etc.

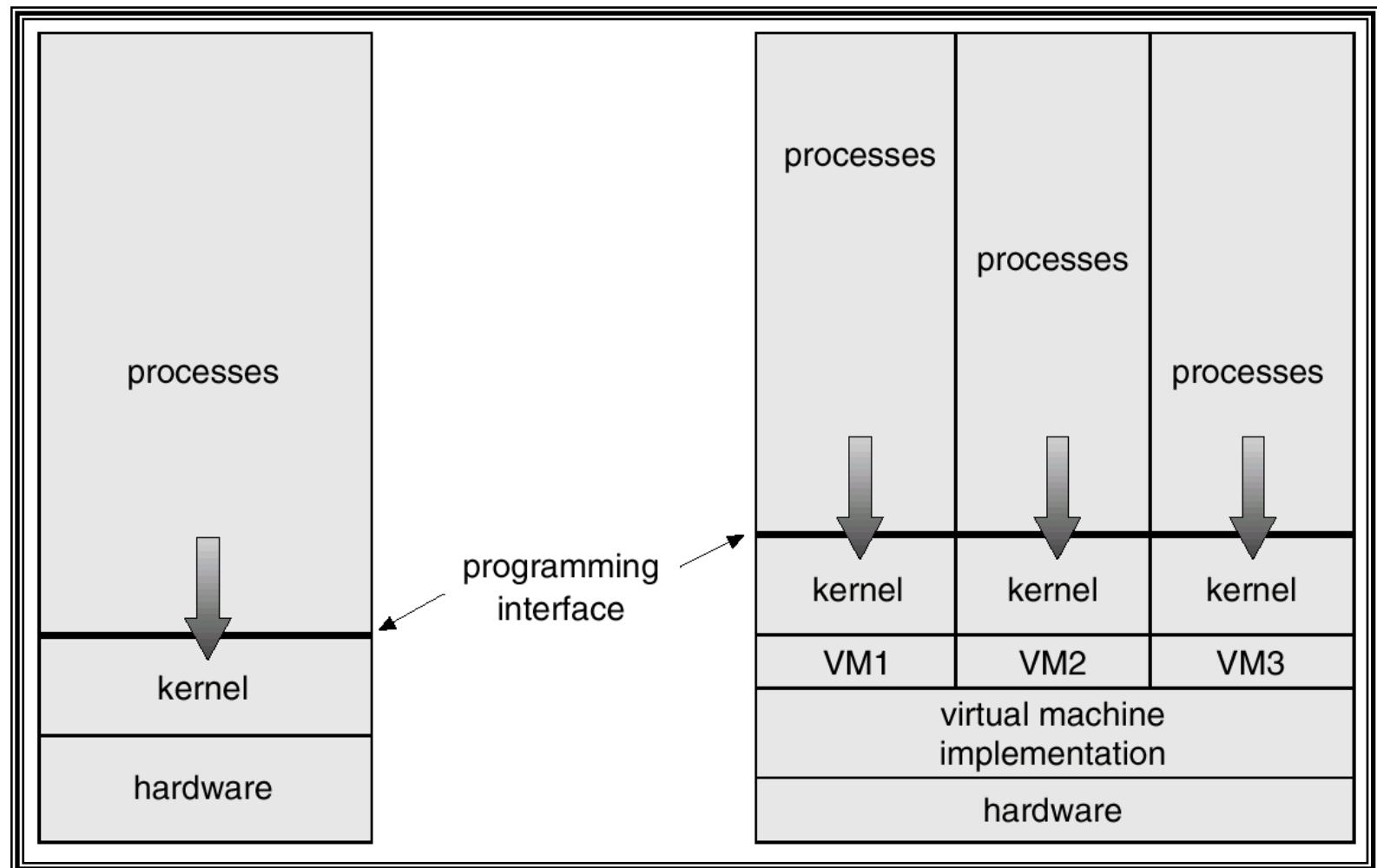
Resursele mașinii fizice sunt partajate. Echipamentele virtuale sunt “desprinse” din cele fizice. Discurile virtuale reprezintă submulțimi ale celor fizice.

- Util în cazul rulării mai multor SO simultan pe aceeași mașină.



# Structura SO

Mașina virtuală



# **Aplicații de mașini virtuale**

**Exemple (free): VirtualBox (Win/Mac/Linux),  
VMware (Win/Linux), QEMU (Linux)**

**VirtualBox (Sun/Oracle) -**

**<http://www.virtualbox.org/wiki/VirtualBox>**

**VMware -**

**<http://www.vmware.com/products/player/faqs.html>**

**QEMU – [wiki.qemu.org](http://wiki.qemu.org)**

*Sisteme de operare*  
*Curs #6*  
*Sisteme de fișiere*

---

Răzvan Daniel ZOTA

Facultatea de Cibernetică, Statistică și Informatică  
Economică

[zota@ase.ro](mailto:zota@ase.ro)

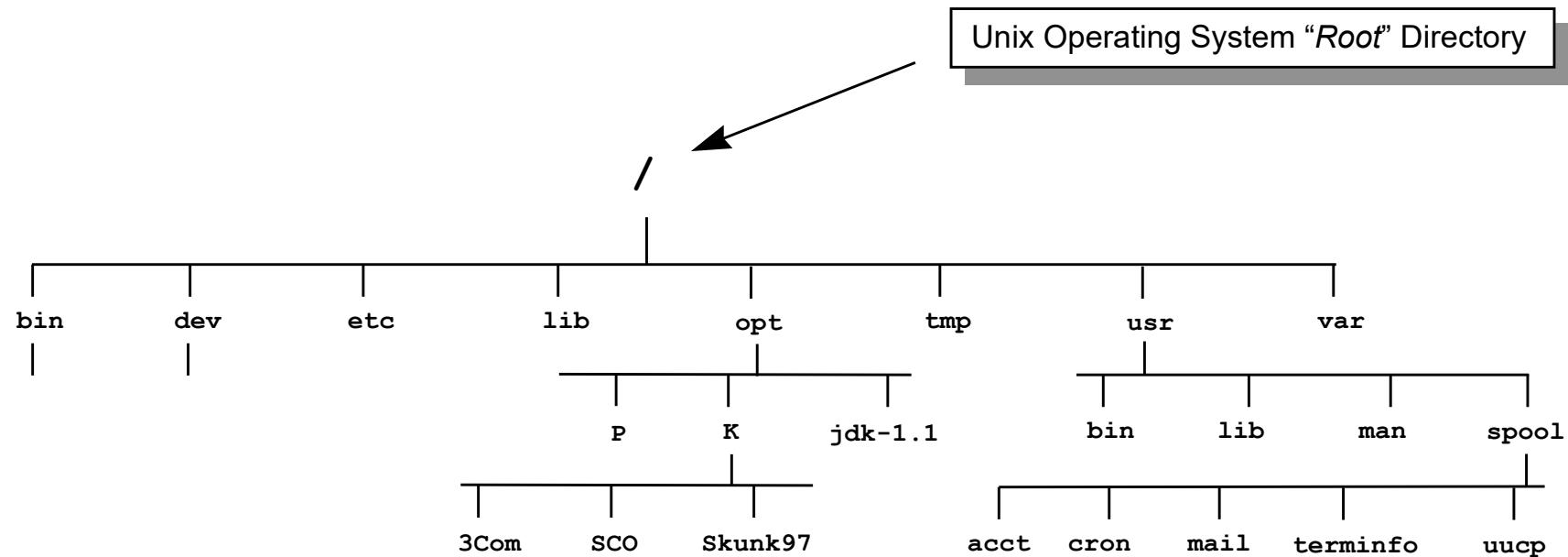
<https://zota.ase.ro/so>

# *Ce este un sistem de fișiere ?*

*Un sistem de fișiere este o parte integrantă distinctă a sistemului de operare, ce constă din fișiere, directoare, precum și informațiile necesare pentru accesarea, localizarea (și eventual refacerea) acestora.*

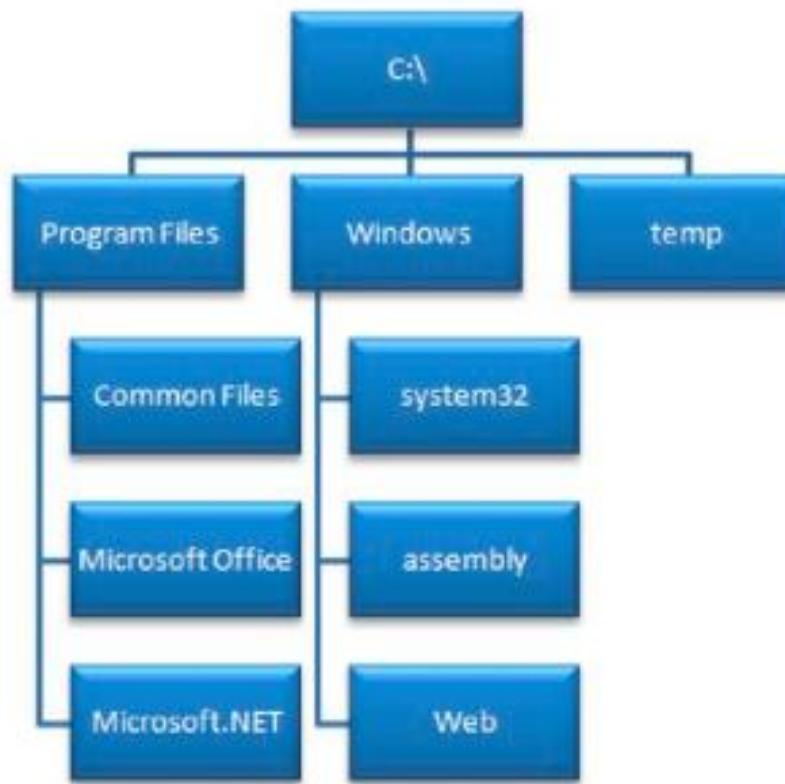
# *Structura arborescentă a unui sistem de fișiere*

## File System Structure



# *Structura arborescentă a unui sistem de fișiere*

Exemplu Windows:



# ***Conținutul celor mai importante directoare UNIX***

/bin	Comenzi UNIX
/dev	Director pentru dispozitive
/etc	Programe adiționale și fișiere de date
/lib	Biblioteca de programe C
/mnt	Directorul “mount”; rezervat pentru “montarea” sistemelor de fișiere
/kernel	Conține kernel-ul și driverele pentru acesta
/opt	Fișiere și pachete de programe instalate local
/tmp	Director temporar
/usr	Rutine utilizator
/sbin	Fișiere necesare pentru pornirea sistemului și script-uri ce controlează procesul de boot

# *Cele mai importante directoare Linux*

/bin	Fișiere executabile – programele de bază ale sistemului
/boot	Directorul de boot. Kernelul, legăturile între module, harta sistemului și managerul de boot se află aici
/dev	Directorul echipamentelor
/etc	Script-uri de configurare
/proc	Directorul proceselor. Conține diverse informații și statistici despre procesele ce rulează și despre parametrii kernelului.
/sys	Directorul echipamentelor de sistem. Conține informații și statistici despre echipamente/nume de echipamente
/tmp	Director temporar
/usr/bin	Alte programe binare (executabile)
/usr/local/bin	Diverse programe binare locale
/usr/share/doc	Documentația legată de software-ul instalat pe sistem

# *SO și sisteme de fișiere oferite*

OS	Filesystems
Windows 7/8/10	NTFS, FAT16, FAT32
Mac OS	HFS+ (Hierarchical File System Plus)
Linux	Ext 2, Ext 3, Ext 4

- NTFS (New Technology File System)- introdus în Windows NT și astăzi prezent pe majoritatea mașinilor Windows. Sistem de fișiere implicit pentru partiții peste 32GB. Fiecare fișier din NTFS este stocat printr-un descriptor de fișier în Master File Table. MFT conține informații despre dimensiune, alocare, nume, etc.
- FAT12 a fost utilizat pentru diskete (floppy disks). FAT16 (sau simplu FAT) și FAT32 sunt folosite pentru carduri de memorie flash și stick-uri USB (tipuri de echipamente ce folosesc FAT: smartphone-uri, camere digitale și alte dispozitive portabile).

# ***SO și sisteme de fișiere oferite***

- Sistemul de fișiere HFS+ se aplică produselor desktop de la Apple, inclusiv computere Mac, iPhone, iPod, precum și produse Apple X Server. Mai multe informații:  
[https://ro.wikipedia.org/wiki/HFS\\_Plus](https://ro.wikipedia.org/wiki/HFS_Plus))
- Produse superioare de tip server folosesc, de asemenea, sistemul de fișiere Apple Xsan, un sistem clusterizat de fișiere derivat din sistemele de fișiere StorNext /CentraVision.
- Ext2, Ext3, Ext4 – sistem de fișiere “nativ” Linux. Acest sistem de fișiere este în permanentă dezvoltare și îmbunătățire. Ext3 este doar o extensie a lui Ext2, ce folosește operații tranzacționale de scriere a fișierelor într-un “jurnal”. Ext4 este o dezvoltare a lui Ext3, oferind suport pentru alocare optimizată a fișierelor și atribută extinse de fișiere. Acest sistem de fișiere este utilizat frecvent de majoritatea instalărilor Linux.

# **CDFS și UDF**

- CDFS (CD-ROM File System) reprezintă un format relativ simplu definit în 1988 ca fiind formatul standard pentru CD-ROM. Windows implementează standardul compatibil ISO 9660 în \Windows\System32\Drivers\Cdfs.sys. Restricțiile formatului CDFS:
  - Numele (de fișiere și directoare) < 32 caractere
  - Structura arborescentă a (sub)directoarelor <= 8 nivele
- UDF (Universal Disk Format) – standard compatibil cu ISO 13346, oferă suport pentru versiunile 1.02 și 1.5 OSTA (Optical Storage Technology Association) definite în 1995 ca format de înlocuire a CDFS pentru mediile de stocare magneto-optice, în special DVD-ROM.
  - Numele (de fișiere și directoare) <= 255 caractere
  - Maximum pentru lungimea unei căi = 1023 caractere
  - Numele de fișiere pot fi lower/upper case

# **FAT, FAT16 și FAT32**

- Mult timp în Windows cel mai popular sistem de fișiere a fost FAT (File Allocation Table). Există 3 tipuri de sisteme FAT.
- Primul este sistemul original FAT (FAT12), apoi au apărut FAT16 și FAT32, versiuni îmbunătățite ale sistemului original FAT. Sistemul original FAT a fost utilizat pe primele versiuni de DOS, fără a avea posibilitatea de a fi utilizat pe discuri de dimensiuni mai mari sau pe sisteme de operare precum Windows 3.1, Windows 95 și Windows 98.
- Acest sistem FAT era limitat din mai multe puncte de vedere, fiind capabil să recunoască nume de fișiere până la 8 caractere în lungime. Alte limitări erau legate de imposibilitatea de a utiliza discuri de dimensiuni mari și sisteme de operare avansate la acea vreme.

# **FAT, FAT16 și FAT32**

- De asemenea, spațiul de pe discurile cu capacitate mari era utilizat ineficient, aceeași problemă existând și pentru FAT16. FAT16 a fost conceput pentru a fi utilizat pentru partiții până la 4 GB.
- Chiar dacă discuri de dimensiuni mari pot fi formatare folosind FAT16, această manieră este ineficientă deoarece spațiul de pe disc este utilizat ineficient. Spre exemplu, la o partiție de 512 MB, dimensiunea clusterelor este de 8 KB. Aceasta înseamnă că fișierele de 1 KB vor ocupa tot 8 KB de spațiu pe disc, deoarece nu se pot stoca mai multe fișiere într-un cluster. De aici rezultă 7 KB pierduți.
- Pentru a rezolva această problemă a apărut FAT 32, sistem ce folosește dimensiuni mai mici pentru clustere și oferă suport pentru partiții până la 2 TB.

- De regulă se consideră că pentru selecția dimensiunii clusterului, regula este “cu cât mai mic, cu atât mai bine”. Deoarece partițiile FAT16 iroseau o grămadă de spațiu pe disc, trecerea la FAT32 pentru a reduce dimensiunea clusterilor a fost imediată. Cu toate acestea, FAT32 are și el limitările sale.
- Să considerăm o partiție de sub 2.048 MB (2 GB), cea mai mare pe care o poate oferi FAT16. Dacă această partiție este formatată folosind FAT16, va rezulta o tabelă de alocare a fișierelor cu 65,526 clustere în el, fiecare cluster consumând 32 KB.
- Dimensiunea mare a clusterului va avea ca rezultat o irosire a spațiului pe disc. Astfel, este recomandat ca această partiție să utilizeze FAT32, ce va conduce la o dimensiune a clusterilor de doar 4 KB. Acest lucru va avea ca efect o diminuare cu până la 90% a irosirii spațiului de pe disc.

## FAT 32

- Cu toate acestea, există și un preț plătit pentru asta. Deoarece fiecare cluster este mai mic, trebuie să fie mai multe pentru a acoperi aceeași capacitate a discului. Deci, în loc de 65.526 clustere, vom avea 524.208 ( $65.526 * 8$ ) clustere.
- Mai mult, intrările FAT din tabela FAT32 sunt de dimensiune 32 de biți, față de intrările de 16 biți din FAT16. Rezultatul final este acela că dimensiunea tabelei FAT este de 16 ori mai mare în cazul FAT32 decât la FAT16! Tabelul următor ilustrează aceste observații:

# *Caracteristici FAT16 și FAT32*

Tipul de FAT	FAT16	FAT32
Dimensiunea clusterului	32 KB	4 KB
Numărul de intrări FAT	65,526	524,208
Dimensiunea tabelei FAT	~ 128 KB	~ 2 MB

## *Caracteristici FAT16 și FAT32*

- Dacă mărimea volumului FAT32 crește de la 2 GB la 8 GB, dimensiunea tabelei FAT crește de la 2 MB la 8 MB. În acest caz nu contează că volumul FAT32 va irosi câțiva MB de spațiu de pe disc pentru a memora tabela FAT.
- Problema este că tabela FAT este foarte des referită în momentul utilizării normale a discului deoarece conține toate referirile către clustere pentru fiecare fișier din acest volum; astfel, dimensiunea mare a tabelei FAT poate influența negativ viteza de lucru.

# *Caracteristici FAT16 și FAT32*

- Practic fiecare sistem implementează un mecanism de disk caching pentru a ține în memorie structurile discului frecvent accesate precum tabela FAT. Operațiunea de cache a discului implică utilizarea memoriei principale pentru a stoca informații referitoare la disc ce sunt necesare în mod regulat, pentru a evita citirea în permanență de pe disc (foarte lent în comparație cu memoria).
- Atunci când tabela FAT este mică (cum ar fi 128 KB pentru FAT16) întreaga tabelă FAT poate fi stocată mai ușor în memorie și de fiecare dată când se caută ceva în FAT se ia din memorie. Atunci când tabela crește însă la 8 MB, sistemul este forțat să aleagă între două alternative: fie să folosească o cantitate considerabilă de memorie pentru FAT, fie să nu folosească deloc.

## *Caracteristici FAT16 și FAT32*

- Din această cauză trebuie redusă limita dimensiunii talei FAT la o mărime rezonabilă. În cele mai multe cazuri este o problemă de alegere a unei balanțe între dimensiunea clusterului și dimensiunea FAT (cel mai bun exemplu fiind acela pentru FAT32).
- Din moment ce FAT32 poate avea (maxim) aproximativ 268 milioane de clustere, dimensiunea de 4 KB pentru un cluster este capabilă din punct de vedere tehnic să ofere suport un disc de 1 TB. Singura problemă este că, în acest caz, dimensiunea talei FAT va avea peste 1 GB! (268 milioane înmulțit cu 4 bytes pentru fiecare intrare în tabelă).
- Astfel, FAT32 folosește clustere de 4 KB pentru volume până la 8 GB în dimensiune, apoi folosește clustere de dimensiuni mai mari, ca în tabelul următor:

# *Dimensiunea clusterelor la FAT 32*

Dimensiunea clusterului	Dimensiunea “minimă” a partiției	Dimensiunea “maximă” a partiției
<b>4 KB</b>	0.5 GB	8 GB
<b>8 KB</b>	8 GB	16 GB
<b>16 KB</b>	16 GB	32 GB
<b>32 KB</b>	32 GB	64 GB

## *Caracteristici FAT32*

- După cum se poate remarca, cu toate că FAT32 “promisese” că va rezolva problemele discurilor de dimensiuni mari, în realitate nu a fost chiar aşa. În momentul în care se folosesc partiții mai mari de 32 GB, suntem înapoi în cazul folosirii clusterilor de 32 KB, atât de hulită în cazul FAT16. Desigur, 32 GB înseamnă mult mai mult decât 1 GB, dar FAT32 este totuși o soluție temporară aleasă până la implementarea unui sistem de fișier cu adevărat performant (NTFS).
- În tabelul următor se pot vedea dimensiunile talelei FAT (în MB) în funcție de dimensiunea partiției, pentru diferite mărimi ale clusterilor. Putem vedea că FAT32 nu utilizează clustere de 4 KB decât până la partiții de 8 GB, altfel s-ar consuma cantități considerabile de memorie pentru a stoca tabela FAT. Intrările marcate cu bold din tabelă ne arată ce va alege FAT32 pentru o partiție de acea dimensiune (Microsoft menține dimensiunea talelei FAT la 8 MB).

# *Dimensiunile talelei FAT32 în funcție de mărimea partiției și mărimea clusterilor*

Dimensiunea partiției	4 KB clusteri	8 KB clusteri	16 KB clusteri	32 KB clusteri
<b>8 GB</b>	<b>8 MB</b>	4 MB	2 MB	1 MB
<b>16 GB</b>	16 MB	<b>8 MB</b>	4 MB	2 MB
<b>32 GB</b>	32 MB	16 MB	<b>8 MB</b>	4 MB
<b>64 GB</b>	64 MB	32 MB	16 MB	<b>8 MB</b>
<b>2 TB (2,048 GB)</b>	--	<i>1,024 MB</i>	<i>512 MB</i>	<i>256 MB</i>

## ***exFAT (Extensible File Allocation Table)***

- ***exFAT*** este ultimul sistem de fișiere introdus de către Microsoft în 2006, optimizat pentru a fi folosit în cazul memoriilor de tip flash precum stick-uri USB sau carduri SD (Secure Digital) .
- Standard proprietar până în 2019, când Microsoft a făcut publice specificațiile acestuia.
- exFAT rezolvă problema FAT32 care nu poate să stocheze fișiere mai mari de 4GB, fiind sistemul de fișiere implicit (*SD association*) pentru carduri SDXC cu capacitați mai mari de 32 GB.

# *NTFS – New Technology File System*

Începând cu Windows 2000, NTFS este sistemul de fișiere **nativ**.  
**NTFS** folosește indexuri de cluster pe 64 de biți.  
Această capacitate oferă abilitatea de a adresa volume până la 16 exabytes.

Multiples of <u>bytes</u>		
<u>SI decimal prefixes</u>		<u>Binary usage</u>
Name (Symbol)	Value	
<u>kilobyte</u> (kB)	$10^3$	$2^{10}$
<u>megabyte</u> (MB)	$10^6$	$2^{20}$
<u>gigabyte</u> (GB)	$10^9$	$2^{30}$
<u>terabyte</u> (TB)	$10^{12}$	$2^{40}$
<u>petabyte</u> (PB)	$10^{15}$	$2^{50}$
<b>exabyte</b> (EB)	$10^{18}$	$2^{60}$
<u>zettabyte</u> (ZB)	$10^{21}$	$2^{70}$
<u>yottabyte</u> (YB)	$10^{24}$	$2^{80}$

# *Caracteristici NTFS*

Caracteristici NTFS	Importanță
<b>Controlul accesului</b>	Stabilirea drepturilor de acces se poate face atât pentru fișiere individuale cât și pentru directoare.
<b>MFT (Master File Table)</b>	Conține înregistrări pentru fiecare fișier și director din NTFS; Înregistrările cu privire la organizarea NTFS și MFT sunt redundante în cazul în care prima înregistrare devine coruptă; Fișierele de dimensiune mică (sub 1500 octeți) sunt stocate în întregime în MFT pentru acces mai rapid.
<b>Atributele fișierelor NTFS</b>	Atributele fișierelor sunt conținute în înregistrarea MFT a fișierului. Lista atributelor fișierelor poate fi particularizată pentru anumite medii (Mac, UNIX) și adăugată pentru a extinde funcționalitatea NTFS.
<b>Numele fișierelor</b>	NTFS permite nume de fișiere până la 255 de caractere dar poate genera și nume 8+3 pentru compatibilitatea cu FAT/DOS

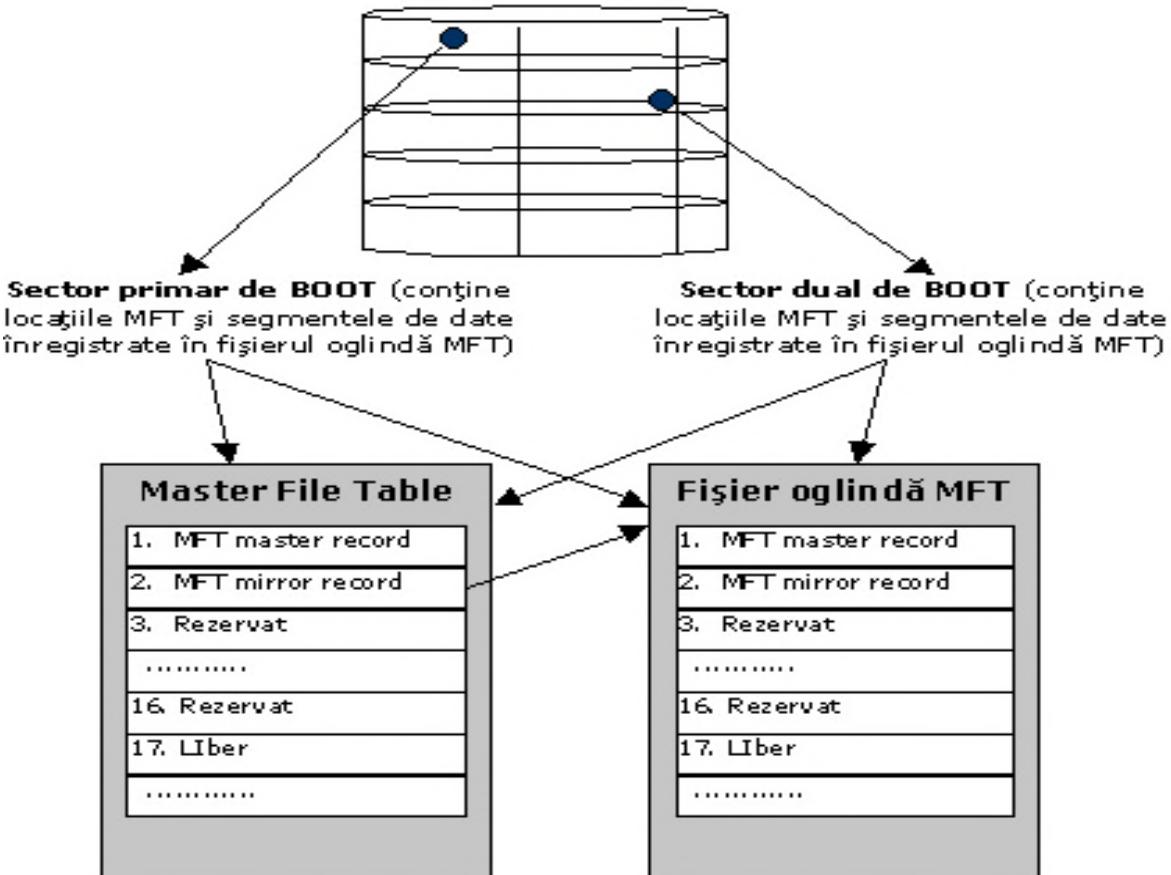
# *Caracteristici NTFS*

<b>Conformitate POSIX</b>	Conformitatea cu standardul POSIX permite aplicațiilor UNIX accesul la fișiere stocate în NTFS sub Windows. Pentru a face asta, NTFS are nevoie de câteva atrbute de fișier unice specifice POSIX cum ar fi: <ul style="list-style-type: none"><li>- Nume de fișiere sensibile la majuscule;</li><li>- Legături hard (hard-link) ce permit unui fișier să fie accesat de mai multe nume de fișiere;</li><li>- Atribute suplimentare "time stamp" care să identifice momentul în care un fișier a fost accesat/modificat ultima oară.</li></ul>
<b>Suport Macintosh</b>	Serviciile Windows pentru Macintosh permit fișierelor să fie accesate atât de utilizatorii Macintosh cât și de clienții Windows. Pentru utilizatorii Mac serverul NT arată ca un server AppleShare. NTFS suportă atrbute de fișiere Mac (resource și data forks) precum și Finder. Sunt suportate, de asemenea, drepturi Macintosh de control al accesului.

# *Caracteristici NTFS*

<b>Hot Fixing</b>	Dacă NTFS găsește un sector stricat pe un disc SCSI va muta automat fișierele afectate și îl va marca "bad" fără a fi nevoie de intervenția utilizatorului.
<b>Refacerea sistemului de fișiere</b>	NTFS utilizează managerul memoriei cache pentru scrierile buffer pe disc în cadrul unui proces denumit "lazy-write". De asemenea, rulează un proces de monitorizare a scriierilor pe disc ceea ce îi permite să refacă sistemul de fișiere în urma unui căderi.

# *Master File Table - NTFS*



**Fiabilitatea încorporată prin proiectarea MFT**

- Master File Table (Tabela Master a Fișierelor) a fost proiectată pentru a garanta accesul rapid și sigur la fișiere. Cele două obiective de cele mai multe ori contradictorii ale sale sunt: **performanță superioară la regăsirea fișierelor pe disc** (rapiditate în special pentru fișierele mici și directoare) pe de o parte, și o deosebită **fiabilitate**, (datorată multiplelor caracteristici redundante) pe de altă parte.
- MFT poate îndeplini ambele obiective foarte bine. În primul rând, definiția înregistrărilor din MFT permite fișierelor mici și directoarelor să fie incluse în aceste înregistrări astfel încât să nu impiedice căutarea ulterioară sau accesul la disc. Pentru fișierele mari NTFS utilizează o structură arborescentă binară ierarhică pentru a asigura rapiditatea la căutare în directoare mari la fel de bine.
- Fiabilitatea este asigurată prin legătura dintre următoarele caracteristici redundante:
  - înregistrarea master redundantă - înregistrarea oglindă (copia) a MFT;
  - fișiere și segmente de date MFT redundante - fișierul oglindă MFT;
  - sectoare de boot redundante (existența sectorului primar și a celui dual - copia sa - de boot).

# *Tabel comparativ NTFS, FAT16 și FAT32*

- [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc766145\(v=ws.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc766145(v=ws.10)?redirectedfrom=MSDN)

## *Alte sisteme de fișiere*

- Sistemul de fișiere implicit pe Linux este ***ext3*** (third extended filesystem) sau ***ext4***

Informații despre aceste sisteme de fișiere:

- <http://en.wikipedia.org/wiki/Ext3>
- <http://en.wikipedia.org/wiki/Ext4>
- **JFS (Journaling FileSystem)** este un sistem de fișiere creat de IBM. A fost folosit pe versiunea de Unix de la IBM, AIX, precum și pe versiuni de Linux.
- **OCFS2 (Oracle Cluster File System)** – sistem de fișiere creat de Oracle pentru clustere Linux.

<https://oss.oracle.com/projects/ocfs2/>

# *Sisteme de fișiere pentru Linux*

Ce sistem de fișiere să alegem pentru o mașină Linux?

- <https://youtu.be/AhjL1SHku7M>

# *Comenzi Unix legate de HD și partiții*

- ***df (disk free)*** – utilizată pentru a determina mărimea spațiului liber de pe disc
- df -h (h = human readable)
- ***du (disk usage)*** – utilizată pentru a determina spațiul ocupat de un director în număr de blocuri de 512 octeți
- du -k (spațiul ocupat în blocuri de câte 1 KB)
- du -k | tail -1

# *Comenzi de backup și arhivare de fișiere-Linux*

- **Arhivarea și compresia fișierelor** reprezintă operațiile utilizate atunci când unul sau mai multe fișiere trebuie transmise sau stocate în mod eficient. Există două aspecte cu privire la această operație:
- **Arhivarea** – Reprezintă combinarea mai multor fișiere într-unul singur, ceea ce facilitează transmiterea acestora
- **Compresia** – Reprezintă reducerea dimensiunilor fișierelor prin eliminarea informației redundante
- Putem arhiva mai multe fișiere într-o singură arhivă și apoi o putem comprima, sau putem comprima un fișier individual. Prima operație se numește **arhivare**, iar cea de-a doua se numește **comprimare**.
- Operația de preluare a unei arhive, decomprimare și extragere a unuia sau a mai multor fișiere poartă numele de **dezarchivare**.

# *Comprimarea fișierelor*

- **Comprimarea fișierelor** le face să aibă dimensiuni mai mici prin eliminarea informațiilor duplicate, ele putând fi restaurate atunci când este necesar.

Există două feluri de comprimare:

- **“Lossless”**: Comprimare fără pierderea niciunei informații din fișierul original. Comprimarea și decomprimarea fișierului lasă fișierul original intact.
- **“Lossy”**: În acest caz comprimarea se face prin eliminarea unor informații din fișierul original astfel încât operația de decompresie va genera un fișier puțin diferit față de cel original. Spre exemplu, un fișier imagine cu două nuanțe apropiate de roșu poate fi făcut mai mic prin considerarea acestor două nuanțe la fel. De cele mai multe ori, ochiul uman nu poate face diferență între aceste două nuanțe.

# *Compresia fișierelor*

- În concluzie, compresia de tip “*lossy*” este utilizată în cazul fișierelor media deoarece rezultatul este sub forma unor fișiere de dimensiuni reduse și, de regulă, oamenii nu pot face diferență între original versiunea modificată (comprimată).
- Pentru fișiere ce trebuie să rămână nemodificate (documente, fișiere de tip log sau software) trebuie să folosim compresia de tip “*lossless*”.
- Majoritatea formatelor de imagini, precum GIF, PNG sau JPEG implementează algoritmi de compresie de tip “*lossy*”.

# *Comenzi de backup - UNIX/Linux*

- **tar** (**tape archive**) –standard pentru toate versiunile de Unix  
Sintaxa generală:
  - **tar** funcție [modificator] fisier\_destinatie fisier(e) | directoare
  - Funcții tar:
    - **c (create)** pentru crearea unei arhive
    - **t (table of contents)** – pentru vizualizarea tabelei conținut a fișierului tar
    - **x (extract)** folosit pentru extragerea fișierelor din arhivă
  - modificatori:
    - f (filename) – fișierul **tar** va fi creat; altfel se alege dispozitivul specificat de variabila de mediu TAPE, dacă este setată; dacă nu, se folosește valoarea implicită din **/etc/default/tar**
    - v (verbose) – împreună cu funcția **t** oferă informații suplimentare despre intrările din fișierul **tar**.

# *Comenzi de compresie - UNIX*

Exemple pentru tar:

- tar -cvf dir2backup.tar dir2 (creează arhiva **tar** pentru directorul **dir2**)
- tar -cvf ex.tar f1 f2 f3 (creează arhiva tar cu fișierele f1, f2, f3)
- tar -tvf ex.tar (pentru vizualizarea continutului arhivei)
- tar -xvf myfile.tar (extragerea din fișierul tar)
- tar -zcvf myfile.tar.gz \*.pdf (opțiunea **-z** face și comprimarea fișierelor cu algoritmul **gzip** în fișierul myfile.tar.gz)
- tar -rvf ex.tar director/\* (opțiunea **-r** este folosită pentru a adăuga fișiere la o arhivă existentă)

Combinarea backup-ului și a compresiei se poate face și cu comanda **jar** (Java archive):

**jar cvf home.jar \*** (se archivează și comprimă în fișierul **home.jar** toate fișierele din directorul curent)

# *Comenzi de compresie - UNIX*

## Program de compresie GNU: gzip

Comanda **gzip** creează un fișier mai mic, cu extensia **.gz** (fișierul inițial)

**gunzip** este folosit pentru decompresie (**echivalentă cu gzip -d**)

Obs. Există și comenziile **zip** și **unzip**, similare cu cele din Windows, care pot lucra cu fișiere comprimate prin metoda **zip** pe sisteme Windows.

Spre exemplu, comanda **zip home.zip \*** va crea o arhivă denumită **home.zip** ce conține toate fișierele din directorul curent de lucru

# *Comenzi de compresie - UNIX*

**Alte comenzi de compresie sunt: bzip2 și xz**

Comanda **bzip2** folosește algoritmul de compresie **Burrows-Wheeler**, mai bun în unele cazuri față de algoritmul folosit de gzip. Rezultatul este un fișier cu extensia **.bz2** în loc de extensia **.gz**

Comanda **xz** : similară cu gzip, folosind algoritmul **Lempel-Ziv-Markov (LZMA)**. Poate oferi o rată de compresie mai bună decât oferă **bzip2**.

Fișierele comprimate cu comanda **xz** au extensia **.xz** .  
Pentru decomprimare se poate folosi comanda **unxz**.

---

# **Sisteme de operare**

## **Curs #7**

## **Procese**

Răzvan Daniel ZOTA  
Facultatea de Cibernetică, Statistică și Informatică Economică  
[zota@ase.ro](mailto:zota@ase.ro)

<https://zota.ase.ro/so>

# Sistemul de operare și procesele

---

- ✖ Definiție
- ✖ Cu ce se ocupă?
- ✖ Planificarea proceselor
- ✖ Comunicația între procese

Un **program** reprezintă ceva pasiv, static; **procesul** este activ (proces = *instanță a unui program în execuție*). Exemple de atribută asociate cu un proces: starea, memoria alocată, resurse UCP, progresul înregistrat.

### De ce este nevoie de procese ?

- Pentru partajarea resurselor logice (fișiere) și fizice (echipamente hardware).
- Pentru îmbunătățirea vitezei de calcul, prin implementarea multiprogramării.
- Asigurarea modularității pentru protecție.

# INTRODUCERE

---

- ✖ Principala funcție a unui procesor este aceea de a executa *instrucțiuni mașină* ce se află în memoria principală.
- ✖ Aceste instrucțiuni apar sub formă de programe. Din motive ce țin de eficiență și ușurință în exploatare, un procesor poate *intercala* porțiuni provenite de la mai multe programe (procese) în timpul execuției.
- ✖ Pentru ca un program să fie executat, SO creează un proces (sau *task*) asociat aceluia program.

# INTRODUCERE

- Din punct de vedere al procesorului, acesta execută instrucțiuni din “repertoriu” în ordinea dictată de către registrul contor program (PC- Program Counter)- cunoscut și ca IP (Instruction Pointer).
- În decursul execuției, PC se poate referi la codul diferitelor programe ce reprezintă părți ale unor procese diferite.
- Din punct de vedere al unui program individual, execuția sa implică o serie de instrucțiuni din cadrul acelui program.

## “URMA” UNUI PROCES

---

- Putem caracteriza comportamentul unui proces individual prin listarea secvenței de instrucțiuni ce se execută pentru acel proces.
- O astfel de listă se numește *urma (trace) unui proces*. Putem caracteriza comportamentul procesorului prin prezentarea urmelor diferitelor procese și modalitatea prin care acestea sunt intercalate.

# “URMA” UNUI PROCES (CONT.)

- În exemplul simplu din figura 1 este prezentată o hartă a memoriei pentru trei procese.
- Pentru simplificare, putem presupune că nu se utilizează memoria virtuală; astfel toate cele trei procese sunt reprezentate prin programe ce sunt complet încărcate în memoria principală.

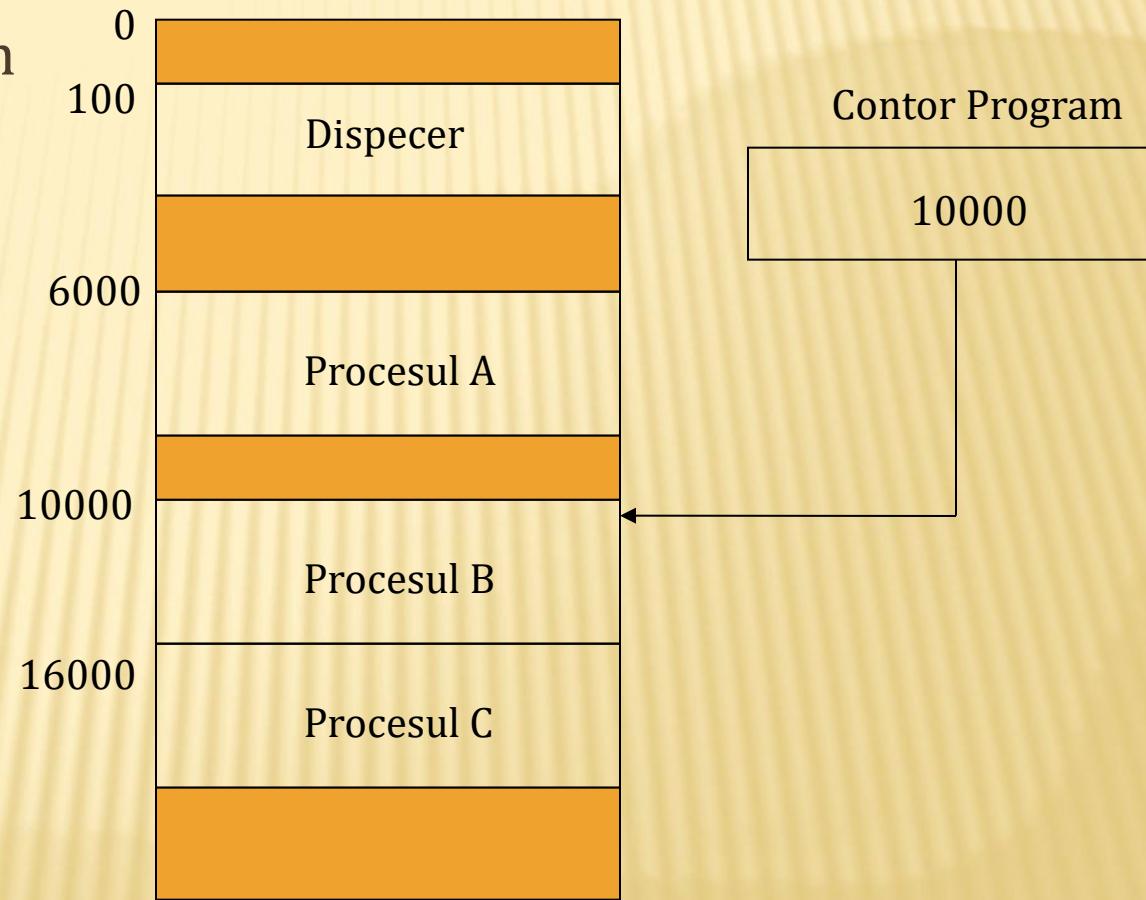


Figura 1. Exemplu de execuție a trei procese – harta memoriei

# “URMA” UNUI PROCES (CONT.)

- În plus, există un mic program sistem numit ***dispecer*** ce comută execuția procesorului de la un proces la altul.
- În figura 2 sunt prezentate “urmele” celor trei procese la începutul execuției lor.
- Sunt prezentate astfel primele 16 instrucțiuni ale proceselor A și C.
- Procesul B execută 6 instrucțiuni și presupunem că a 6-a instrucțiune apelează o operație de I/O pentru terminarea căreia trebuie să se aștepte.

6000	10000	16000
6001	10001	16001
6002	10002	16002
6003	10003	16003
6004	10004	16004
6005	10005	16005
6006		16006
6007	2) Urma procesului B	16007
6008		16008
6009		16009
6010		16010
6011		16011
6012		16012
6013		16013
6014		16014
6015		16015

1) Urma procesului A                                    3) Urma procesului C

**Figura 2. Urmele proceselor din figura 1**

## “URMA” UNUI PROCES (CONT.)

- Să considerăm în cele ce urmează aceste urme din punct de vedere al procesorului. Figura 3 ne prezintă **urmele intercalate** ce rezultă din execuția primelor 70 de cicluri de instrucțiuni. Presupunem că SO permite ca un proces să își continue execuția maxim 8 cicluri instrucțiune, după care procesul este întrerupt ( acest lucru previne monopolizarea procesorului de către un anumit proces).
- După cum apare în figura 3, sunt executate primele 8 instrucțiuni ale procesului A, urmate de un “time-out” și de execuția unei porțiuni de cod ale dispecerului, care execută 8 instrucțiuni înainte de a oferi controlul procesului B. După ce sunt executate primele 6 instrucțiuni ale procesului B, acesta cere o operație de I/O al cărei rezultat trebuie așteptat.

## “URMA” UNUI PROCES (CONT.)

---

- Astfel, procesorul oprește execuția procesului B și trece la execuția procesului C (după execuția codului dispecer).
- După un alt moment de “time-out”, procesorul se mută din nou la procesul A. În acest timp, procesul B așteaptă pentru terminarea operației I/O, iar dispecerul comută din nou la procesul C.

# “URMA” UNUI PROCES (CONT.)

1	6000	17	10000	31	16000	47	6008
2	6001	18	10001	32	16001	48	6009
3	6002	19	10002	33	16002	49	6010
4	6003	20	10003	34	16003	50	6011
5	6004	21	10004	35	16004	51	6012
6	6005	22	10005	36	16005	52	6013
7	6006	Cerere I/O		37	16006	53	6014
8	6007	23	100	38	16007	54	6015
<hr/>							
time-out		24	101	time-out		time-out	
9	100	25	102	39	100	55	100
10	101	26	103	40	101	56	101
11	102	27	104	41	102	57	102
12	103	28	105	42	103	58	103
13	104	29	106	43	104	59	104
14	105	30	107	44	105	60	105
15	106			45	106	61	106
16	107			46	107	62	107

Figura 3. Urmele intercalate ale proceselor din figura 1

## “URMA” UNUI PROCES (CONT.)

63	16008
64	16009
65	16010
66	16011
67	16012
68	16013
69	16014
70	16015
time-out	

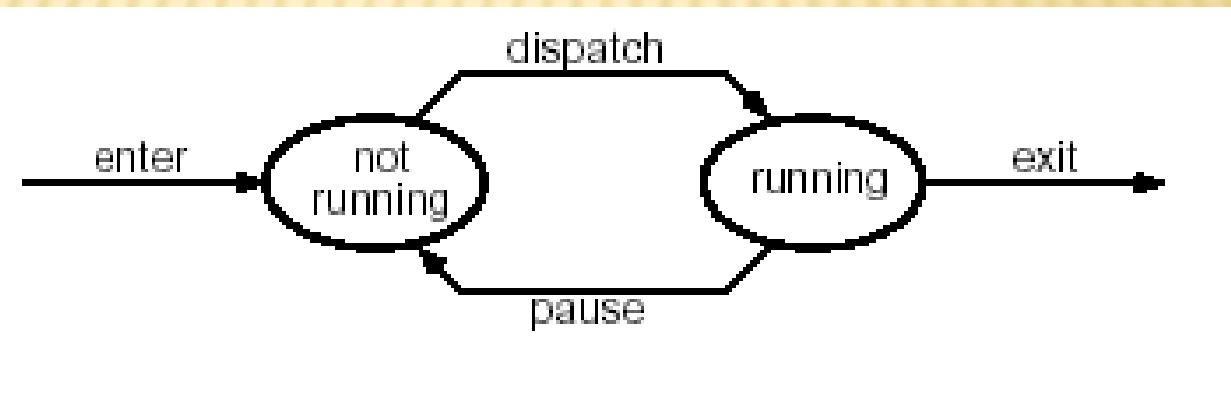
**Figura 3. Urmele intercalate ale proceselor din figura 1 (cont.)**

# MODELUL DE PROCES CU DOUĂ STĂRI

Planificarea proceselor folosește un mecanism de cozi de planificare. Într-un astfel de mecanism, atunci când un proces intră în sistem, este plasat într-o coadă a job-urilor. Pentru a se stabili starea unui proces la un moment dat, sunt folosite diverse modele.

Cel mai simplu dintre acestea este *modelul cu două stări*:

- Starea *în execuție*
- Starea *non-execuție*



# Model de proces cu 5 stări

- **Nou** - Procesul de-abia a fost creat.
- **Gata de execuție** - Procesul are toate resursele necesare, așteptând să intre în execuție.
- **În așteptare** - Așteaptă apariția unui eveniment (echipament hardware, intervenția utilizatorului sau a altui proces).
- **În execuție** - În acest caz instrucțiunile sunt executate. Procesul ce se află în execuție face uz de UCP.
- **Suspendat/Terminat** - Un alt proces a “ordonat” ca acest proces să intre în starea suspendat (“go to sleep”). Procesul aflat în starea suspendat va fi “trezit” de către alt proces / Procesul și-a încheiat execuția.

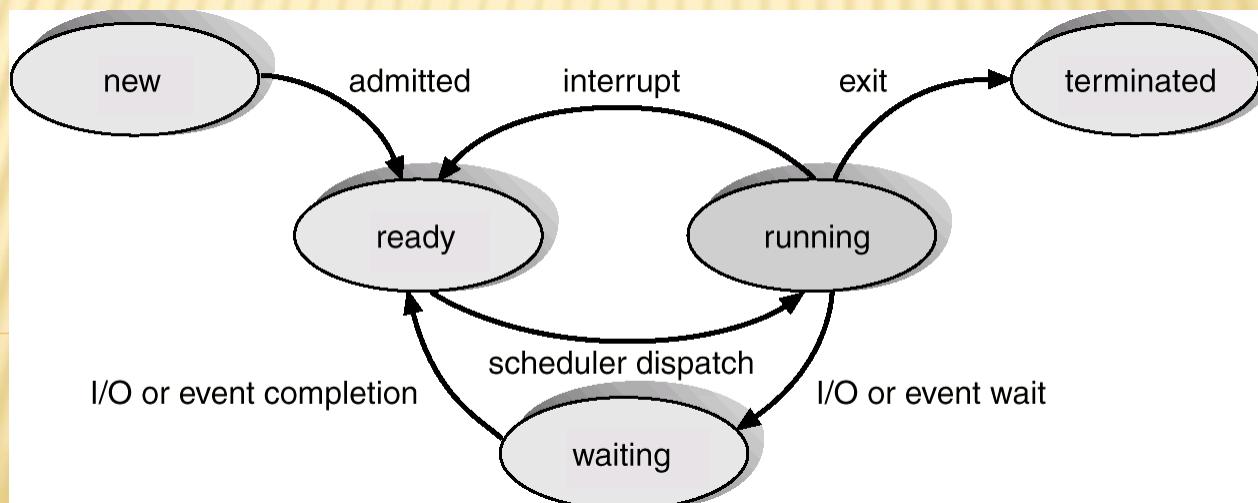


Figura 4. Model de proces cu 5 stări

# Stările proceselor

## Blocul de control al proceselor (BCP):

Conține informații asociate fiecărui proces.

Are următoarea structură:

- PC (Program Counter), regiștri UCP
- Informații despre managementul memoriei
- Informații de contabilizare (timpul utilizat, identificatorul de proces, etc.)
- Starea I/O (resursele alocate unui fișier, etc.)
- Date referitoare la planificare (priorități, etc.)
- Starea procesului (în execuție, suspendat, etc.)

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

Figura 5. Blocul de control al proceselor

# Planificarea proceselor

Planificarea proceselor reprezintă, de fapt, modificarea BCP referit de către UCP – se mai numește **comutator de context (context switch)**.

Un comutator de context poate fi considerat un comutator de procese.

Comutatorul face trecerea de la memoria alocată unui proces la memoria alocată altui proces ( fiecare proces are propria lui “viziune” despre memorie). **(Figura 6)**

## Cozi de planificare:

(un proces este determinat de anumite evenimente ce apar în urma necesităților și disponibilităților)

- coada “gata de execuție” = formată din toate procesele gata de execuție.
- coada I/O (starea de așteptare) = formată din procesele ce așteaptă încheierea unui proces de I/O.

Implementarea cozilor se poate face prin înlănțuire simplă sau dublă.

# Planificarea proceselor

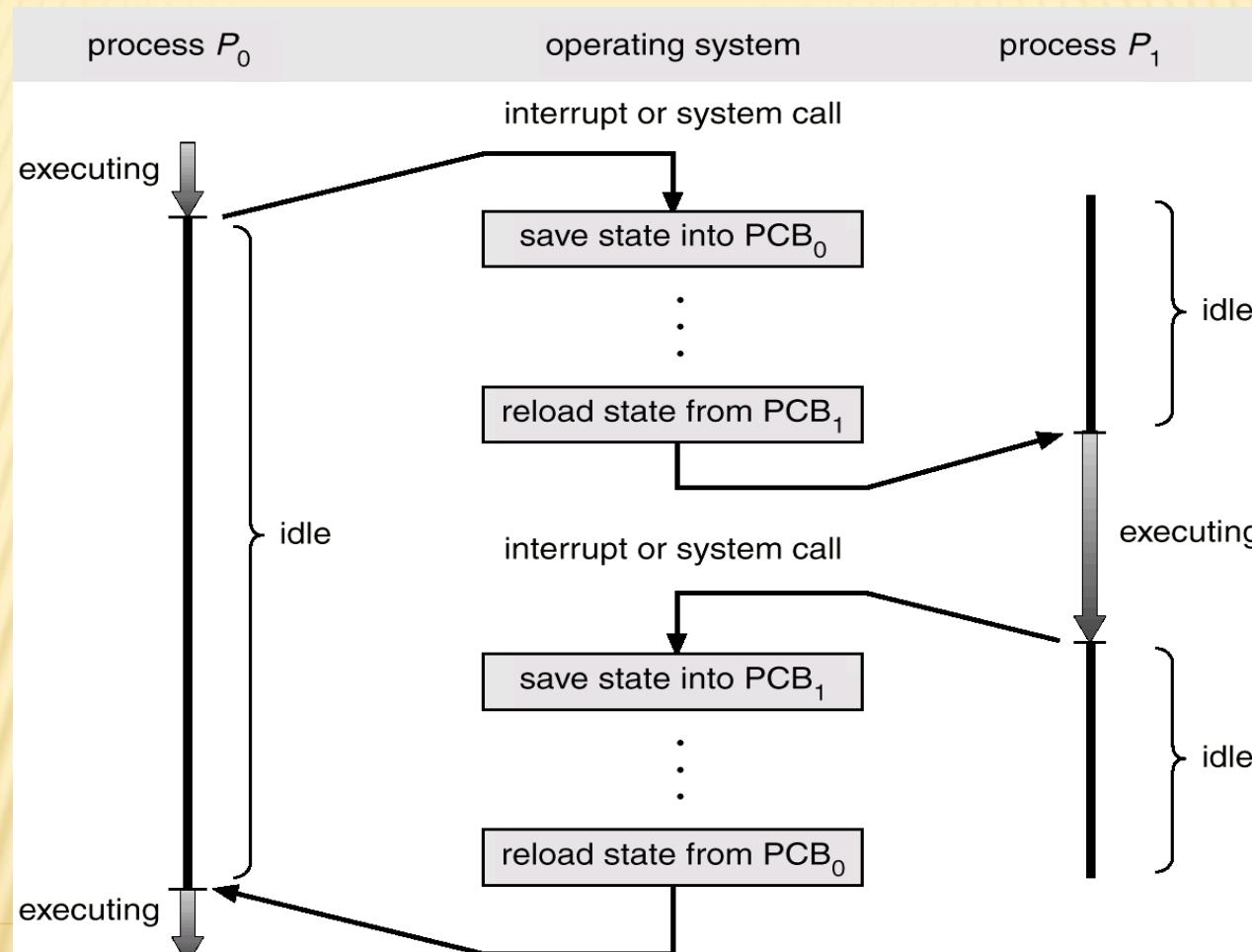
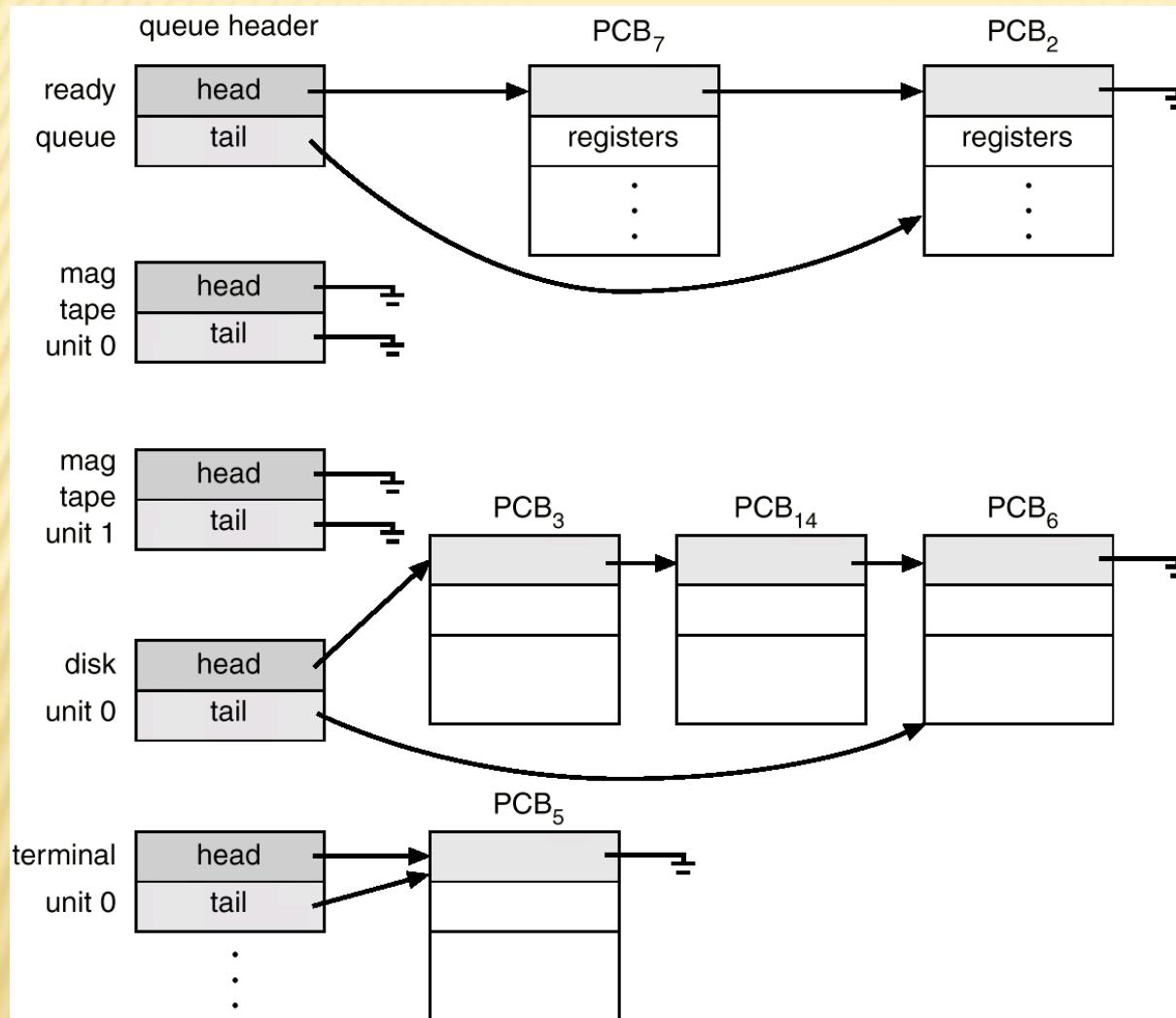


Figura 6. Exemplu de *context switch*

# Planificarea proceselor



*Coada Gata de execuție*

*Coada de I/O*

Figura 7. Exemple de cozi

## Tipuri de planificatori: Planificatorul pe termen lung

Decide ce joburi/procese trebuie admise în coada gata de execuție. În momentul în care un program trebuie să se execute, admiterea acestuia în cadrul proceselor ce se vor executa este fie autorizat, fie amânat de către planificatorul pe termen lung.

În acest mod, planificatorul hotărăște ce procese vor rula pe sistem precum și gradul de concurență suportat la un moment dat (un număr mai mare sau mai mic de procese sunt executate concurrent și modul de împărțire a acestora în *procese I/O intensive* și *CPU intensive*).

# Planificatorul pe termen lung

Pentru un sistem desktop nu există practic un astfel de planificator, iar procesele sunt admise în sistem în mod automat. Acest planificator este utilizat în cazul sistemelor ***real-time*** (sau în cazul *supercomputerelor*), caz în care capacitatea sistemului de a satisface termenele limită ale proceselor poate fi compromisă în cazul în care sunt admise mai mult procese decât este cazul.

Caracteristici principale:

- Rulează rareori (atunci când job-ul vine din memorie)
- Controlează gradul de multiprogramare

# Planificatorul pe termen scurt

În cazul sistemelor desktop nu avem nevoie de un planificator pe termen lung, astfel că se folosește un aşa numit **planificator pe termen scurt** (*dispatcher*). Acesta este invocat ori de câte ori apare un eveniment, astfel încât acest lucru poate conduce la întreruperea procesului curent aflat în execuție.

Planificatorul pe termen scurt alege din coada gata de execuție procesele ce vor rula. Deoarece este apelat intens, trebuie să fie simplu, scurt și eficient.

**Planificatorul pe termen mediu** se ocupă cu acțiunea de *swapping*. Atunci când se eliberează memorie, SO analizează lista proceselor suspendate și decide care dintre acestea se va încărca în memorie de pe disc.

# Handler-ul de întreruperi

Pentru ca un dispozitiv să funcționeze trebuie ca procesele asociate să fie gata de execuție și să se poată muta dintr-o stare în alta.

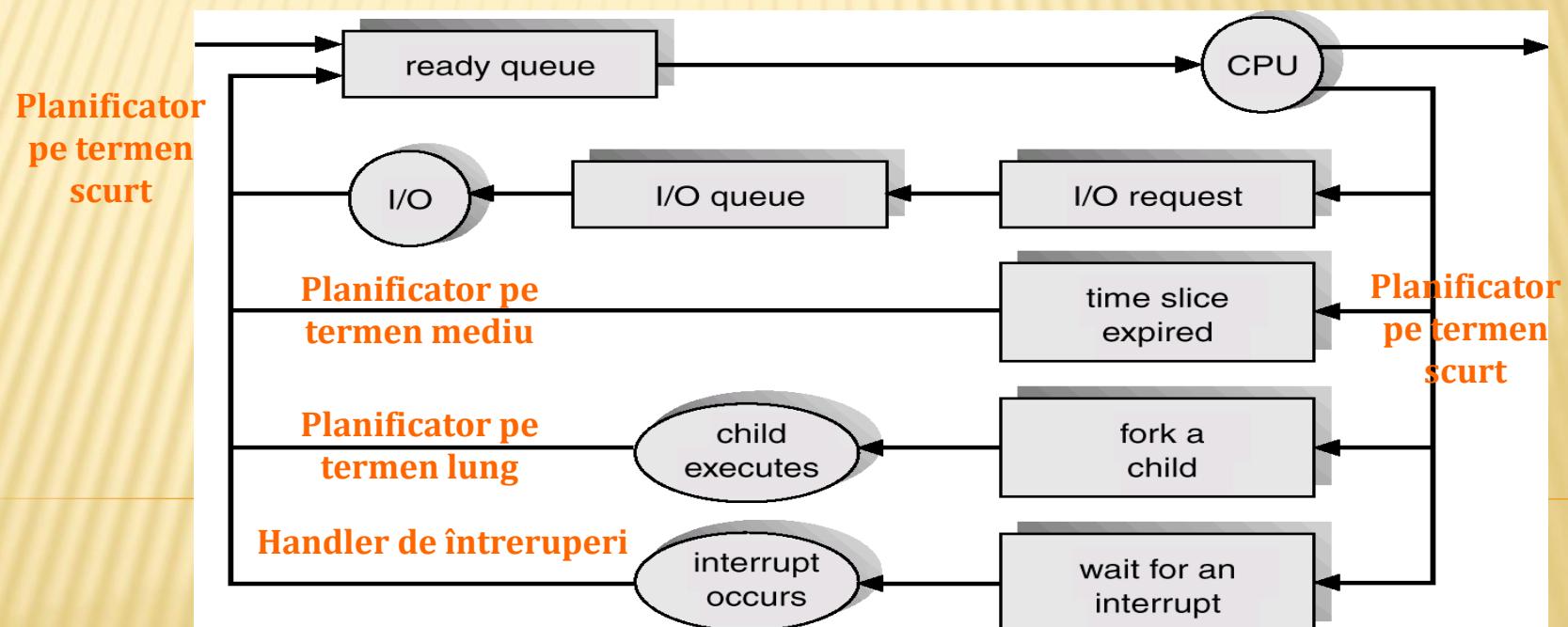


Figura 8. Modul de “îmbinare” a acestor componente

# Relații între procese

- Relațiile dintre procese sunt arborescente. Orice proces poate da naștere altui proces (numit *proces copil*).
- Există schimburi între relațiile copil/părinte.
- Părintele poate rula concurent cu copilul sau poate aștepta terminarea execuției acestuia din urmă.
- Copilul poate partaja (fork/join) toate resursele părintelui sau doar o parte dintre acestea (UNIX).
- Părintele poate “omorî” copilul (în momentul în care nu mai este nevoie de el sau dacă și-a depășit resursele alocate).
- Moartea *procesului părinte* implică, de regulă, moartea *procesului copil*.

# Comunicația între procese

Există două metode de bază pentru comunicarea între procese:

- **Memorie partajată** – modalitate rapidă/fără transfer de date
- **Transfer de mesaje** – modalitate distribuită/o mai bună izolare.

## Funcționalitățile legăturilor de comunicație:

- Modul de formare a legăturilor
- Numărul de procese pentru fiecare legătură
- Numărul de legături pentru fiecare pereche de procese
- Capacitatea memoriei buffer - pentru ca mesajele să fie stocate într-o coadă de așteptare.
- Formatul și dimensiunea mesajelor
- Uni sau bi-directionale

## Metode de implementare:

- Directă sau indirectă – către proces sau în cutia poștală.
- Mecanism de buffering
- Trimiterea prin copiere sau stabilirea unei referințe?
- Mesaje de dimensiune fixă sau variabilă?

# Comunicația directă

În acest caz se cunosc *transmițatorul* și *receptorul*. Procedura de comunicatie este următoarea:

**send** (Process\_P, message) ;

**receive** (Process\_Q , message);

**receive** (id, message)            <-- din partea oricărui transmițător

Mecanismul standard utilizat în acest caz este cel denumit **Producător/Consumator** (un proces produce elemente ce sunt trimise către un consumator pentru a fi utilizate). Caracteristici: două mesaje implicate, bidirecțional.

**repeat**

produce element

trimite (consumator, nextp)

**until false**

**repeat**

receptioneaza(producer, nextp)

consumă element

**until false**

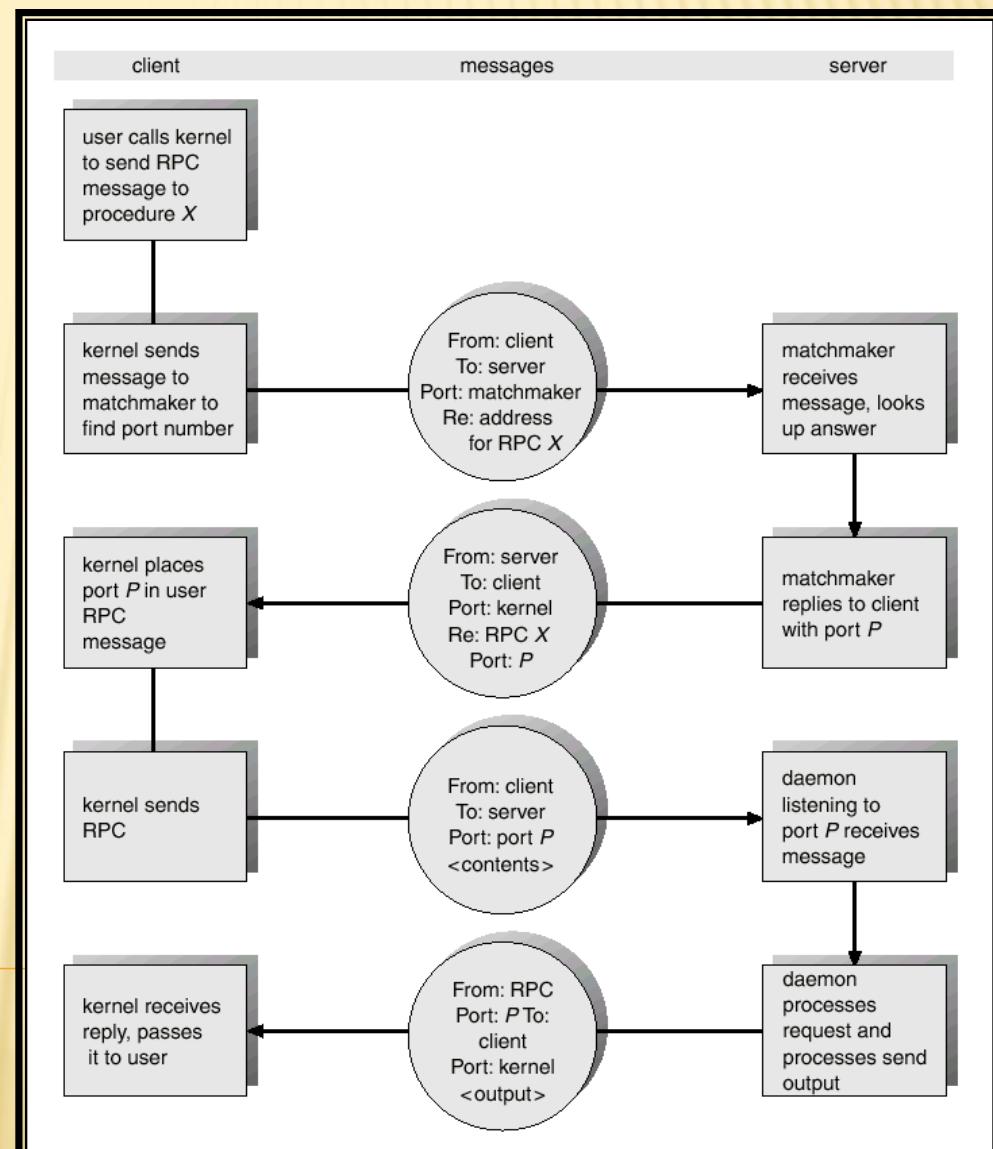
## Comunicația indirectă

- Procesele comunică printr-o căsuță poștală (mailbox). Procedura arată astfel:  
**open(mailbox\_name);**  
**send (mailbox\_name, message);**  
**receive (mailbox\_name, message);**
- Legătura se stabilește dacă procesele au o cutie poștală partajată (ce trebuie setată înaintea transmisiei/recepției).
- O cutie poștală poate fi accesată de două sau mai multe procese.
- În cazul mai multor receptori se poate crea confuzie: cine va primi mesajul?

# Comunicația între procese

Un alt exemplu de comunicație între procese îl reprezintă *RPC* (*Remote Procedure Call*).

Apelul de procedură la distanță (*RPC*) abstractizează apelurile de proceduri între procese în cadrul comunicațiilor din rețea.



# PROCESE LINUX

- În lumea Linux, un proces poate crea un nou proces (folosind funcțiile fork() și exec()). În acest caz, procesul care a fost creat se numește copil (child) iar procesul care l-a creat se numește părinte (parent). Un proces este identificat prin identificatorul de proces - **process ID (PID)** precum și prin identificatorul de proces al procesului părinte - **parent processes ID (PPID)**.
- **Procesele părinte** – sunt procesele ce creează alte procese în timpul execuției.
- **Procesele copil** – sunt procesele ce sunt create de către alte procese în timpul execuției acestora.
- Obs.: Putem folosi comanda *pidof* pentru a afla identificatorul unui proces (PID).
- Sintaxa: \$ pidof <process\_name>

# DOUĂ MARI TIPURI DE PROCESE LINUX

- Procese ***foreground*** (~asa zis interactive) – acestea sunt inițializate și controlate într-o sesiune terminal. Cu alte cuvinte, trebuie ca un utilizator să fie conectat la sistem pentru a porni astfel de procese; acestea nu pornesc automat ca parte a serviciilor sau funcțiilor de sistem.
- Procese ***background*** (~non-interactive sau automate) – reprezintă procesele ce nu sunt conectate cu aplicația terminal; nu așteaptă nici un input din partea utilizatorului.

# PROCESE SPECIALE LINUX

- **Daemons:** tip special de procese background ce pornesc la startup-ul sistemului și își mențin execuția permanent, pe post de servicii. Ele sunt pornite ca task-uri de sistem (rulează ca servicii) în mod spontan și pot fi controlate prin intermediul procesului de sistem *init*.
- Procesul *init* este procesul părinte al tuturor proceselor existente pe un sistem, fiind primul program executat după ce sistemul Linux bootează; el administrează toate celelalte procese ce rulează pe sistem. Este pornit de către kernel, deci nu are un proces părinte.
- Procesul *init* are întotdeauna un PID egal cu 1. El se manifestă ca un părinte “adoptiv” pentru toate procesele orfane.

# LINUX – STĂRILE PROCESELOR

- ✖ ***Running*** – în această stare procesul fie rulează, fie este gata de rulare (așteaptă să fie preluat de către UCP și rulat).
- ✖ ***Waiting*** – în această stare procesul așteaptă apariția unui eveniment sau eliberarea unei resurse de sistem. În plus, kernelul face distincție între două tipuri de procese aflate în stare de așteptare; procese ***“interruptible”*** – sunt cele care pot fi întrerupte de semnale și procese ***“uninterruptible”*** – ce așteaptă în mod direct mesaje/condiții hardware și nu pot fi întrerupte de alte evenimente sau semnale.

# LINUX – STĂRILE PROCESELOR (CONT.)

- **Stopped** – în această stare un proces a fost oprit, de regulă prin receptionarea unui semnal. Spre exemplu, un proces care este depanat.
- **Zombie**: reprezintă o altă stare specială a unui proces. Un proces **zombie** este un proces a cărui execuție s-a încheiat dar care încă are o intrare în tabela de procese.
- Putem vizualiza procesele active pe un sistem Linux folosind comenziile: **ps**, **top**, **htop**.

# LINUX – PROCESE BACKGROUND

- Pentru a porni un proces în background (non-interactiv), putem folosi simbolul special & . În acest caz, procesul nu poate receptiona input de la utilizator până când nu este mutat în foreground.
- Comenzi conexe: jobs, fg, bg.
- Pentru a trimite un proces din background în foreground, putem folosi comanda **fg** urmată de identificatorul de proces.
- Dacă avem de-a face cu o comandă suspendată (prin Ctrl+Z), putem continua rularea comenзii suspendate folosind comanda **bg** .

# Sisteme de operare

## Cursul #8

### Planificarea proceselor

Răzvan Daniel ZOTA  
Facultatea de Cibernetică, Statistică și Informatică Economică  
[zota@ase.ro](mailto:zota@ase.ro)  
<https://zota.ase.ro/so>

# **Planificarea proceselor**

- **Modalitatea prin care un proces este "atașat" procesorului**
- **Este centrată în jurul algoritmilor eficienți**
- **Proiectarea unui planificator se ocupă cu asigurarea faptului că toate procesele utilizator au acces în mod corect la resurse**

# Planificarea UCP

## Concepție (recap.)

### Multiprogramarea

Mai multe programe pot fi în memorie în același timp. Permite supra-alocarea UCP și a I/O.

### Job-uri

Programe (de tip *batch*) ce rulează fără a fi nevoie de intervenția utilizatorului.

### Programe *utilizator*

Sunt programe (ce rulează pe baza conceptului de *time sharing*) ce pot avea nevoie de intervenția utilizatorului.

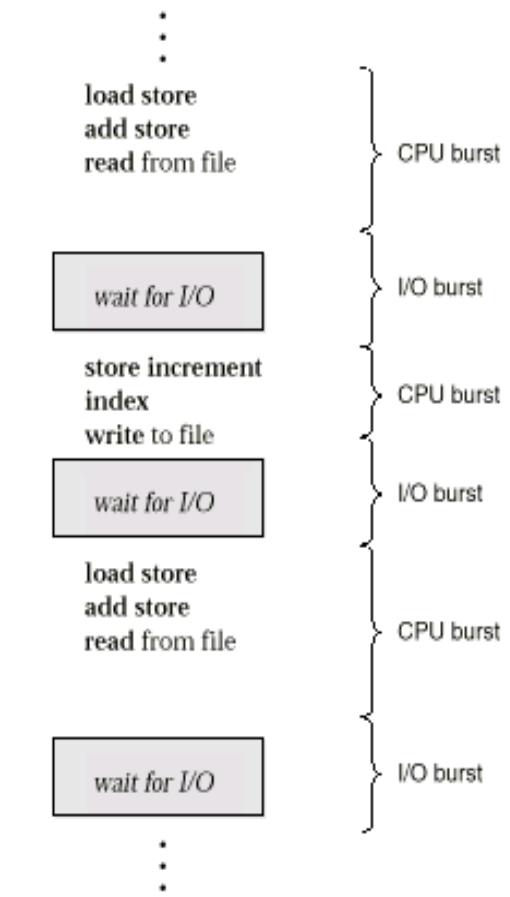
### Proces

### UCP - I/O burst cycle

ambele

Reprezintă execuția proceselor ce alternează între activitatea UCP și a I/O. Timpii asociați UCP sunt mult mai mici decât cei ai operațiilor I/O.

**Planificare preemptivă** În acest caz o intrerupere determină oprirea procesului curent ce ocupă UCP și are loc înlocuirea acestuia cu alt proces.



# Planificarea UCP

## Criterii utilizate pentru evaluarea performanțelor

### Gradul de utilizare

Fracțiunea de timp în care un dispozitiv este utilizat ( $\text{timp\_utilizare}/\text{timp\_total}$ )

### Throughput

Numărul de job-uri terminate într-o perioadă de timp (job-uri/secundă)

### Timpul de serviciu

Timpul pe care îl necesită un dispozitiv pentru a rezolva o cerere (în secunde)

### Timpul de așteptare în coadă

Timpul petrecut în coada de așteptare (în secunde)

# Planificarea UCP

## Criterii pentru evaluarea performanțelor (cont.)

**Timpul de rezidență** Timpul petrecut de către o cerere la un dispozitiv.

$\text{Timpul de rezidență} = \text{Timpul de serviciu} + \text{Timpul de așteptare în coadă.}$

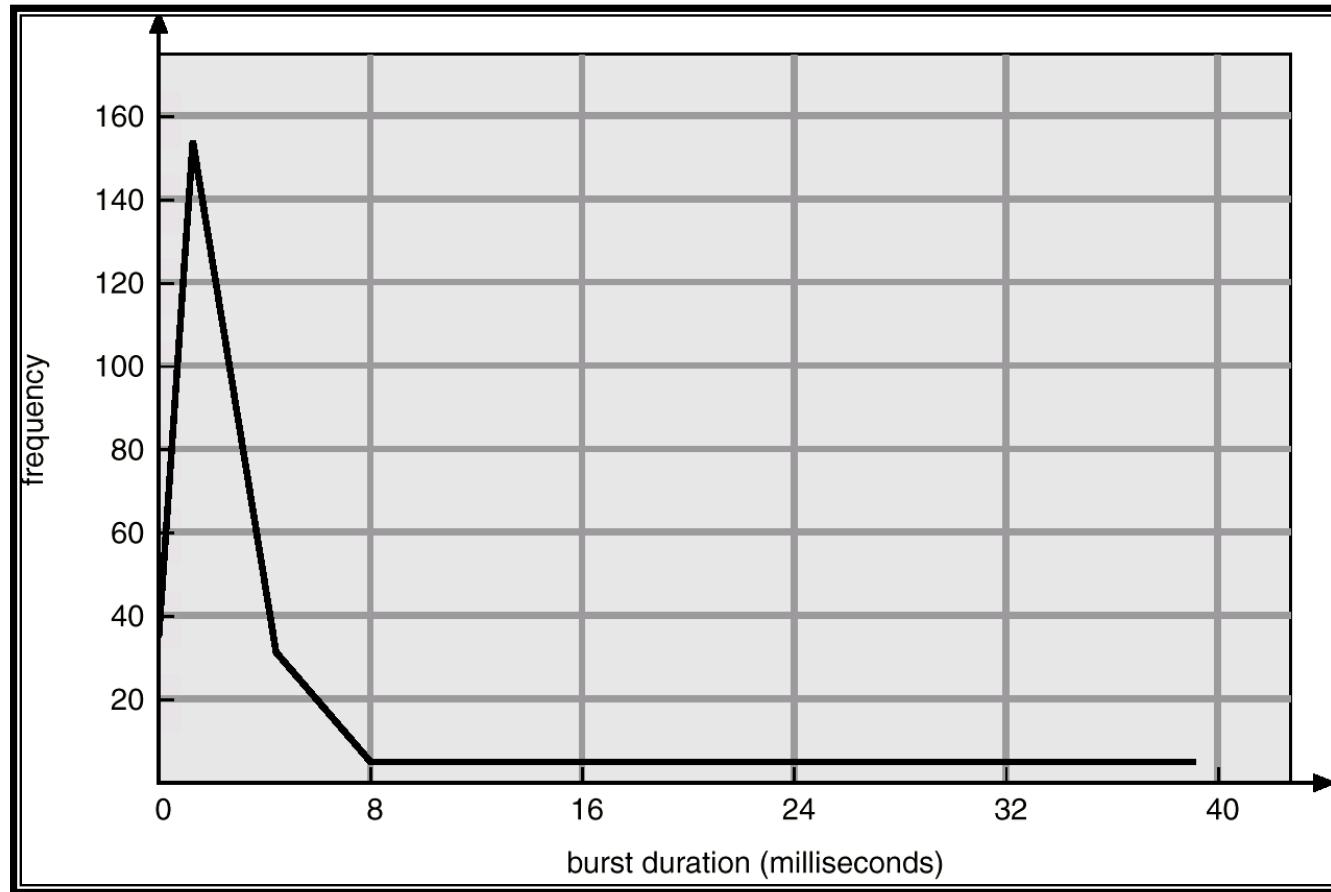
**Timpul de răspuns** Timpul utilizat de către sistem pentru a răspunde unui job utilizator (secunde).

**Timpul de “gândire”** Timpul petrecut de utilizatorul unui sistem interactiv pentru a realiza următoarea cerere (secunde).

Scopul principal este acela de a optimiza media acestor timpi.

# Planificarea UCP

**Majoritatea proceselor nu își utilizează pe deplin timpul alocat!**



# Algoritmi de planificare - FIFO

## Primul venit, primul servit:

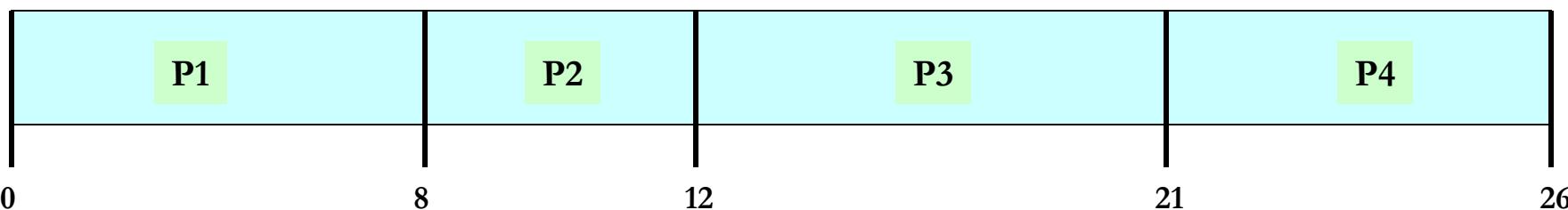
- FIFO
- Simplu, corect, dar cu performanțe slabe. Timpul mediu de așteptare în coadă poate fi destul de mare.

# Exemplu de planificare FIFO

Exemplu:

Procesul	Timpul sosirii	Timpul de serviciu
1	0	8
2	1	4
3	2	9
4	3	5

FIFO



$$\text{Timpul mediu de rezidență} = ((8-0) + (12-1) + (21-2) + (26-3)) / 4 = 61 / 4 = 15.25$$

# Algoritmi de planificare: SJF

## Cel mai scurt job primul (SJF – Shortest Job First):

- Optimal pentru minimizarea timpului de așteptare în coadă dar imposibil de implementat 100% în practică. Pentru a putea fi implementat, se încearcă să se prevadă următorul proces pe baza istoricului anterior.
- Predicția timpului pe care procesul îl va utiliza la următoarea planificare se poate face după formula:

$$t(n+1) = w * t(n) + (1 - w) * T(n)$$

unde:  $t(n+1)$       timpul următorului burst

$t(n)$       timpul actualului burst

$T(n)$       media burst-urilor anterioare

$w$       factor de ponderare ce reflectă burst-urile curente sau anterioare.

# Algoritmi preemptivi de planificare

## Algoritmi preemptivi:

- Se scoate procesul din execuție în momentul în care un alt proces cu prioritate mai mare este gata de execuție.
- Se poate aplica atât în cazul SJF cât și în cazul planificării pe bază de priorități.
- Se evită acapararea UCP de către un proces
- Pe mașinile de implementează *time sharing* este necesară această schemă deoarece UCP trebuie protejată de către procesele cu priorități mici.
- Dacă se acordă job-urilor scurte o prioritate mai mare–timpul de răspuns este mai bun.

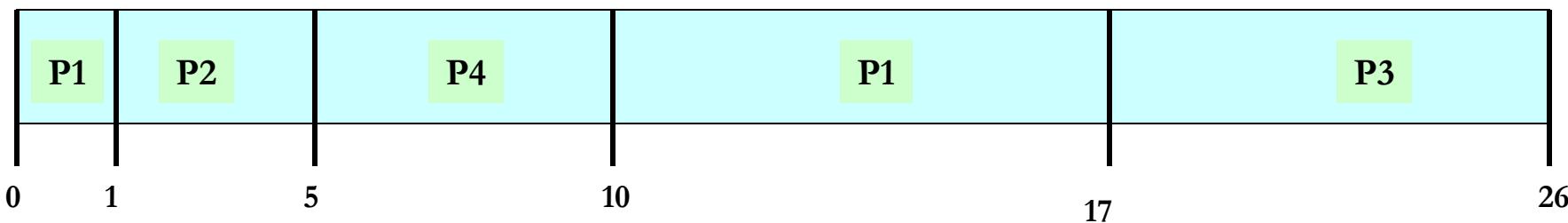
# Planificarea UCP

## Exemplu de planificare SJF

Exemplu:

Procesul	Timpul sosirii	Timpul de serviciu
1	0	8
2	1	4
3	2	9
4	3	5

SJF preemptiv



$$\text{Timpul mediu de rezidență} = ((5-1) + (10-3) + (17-0) + (26-2)) / 4 = 52 / 4 = 13.0$$

# Planificarea pe bază de priorități

## Planificarea pe bază de priorități:

- Se atribuie o prioritate fiecărui proces. Planificarea selectează primul proces cu prioritatea cea mai mare. Toate procesele cu aceeași prioritate sunt tratate conform algoritmului FIFO.
- Prioritatea poate fi atribuită de către utilizator sau prin intermediul unui mecanism implicit. Sistemul poate determina prioritatea pe baza necesităților de memorie, a limitelor de timp sau a altor resurse.
- ***Starvation*** – fenomenul ce apare atunci când un proces cu o prioritate scăzută nu apucă să se execute niciodată. Soluția: implementarea unei variabile ce va stoca “vârsta”.
- Se asigură un echilibru între acordarea răspunsului favorabil pentru joburile interactive, fără a apărea fenomenul de “starvation” pentru joburile batch.

# Planificarea UCP – Round Robin

## Coada circulară (ROUND ROBIN)

- Folosește un timer pentru generarea unei întreruperi după un timp prestabilit. Asigură multitasking-ul preemptiv dacă un task depășește cuantumul de timp alocat.
- În slide-ul 15 testăm exemplul anterior pentru un cuantum = 4 sec.
- Definiții:
  - **Comutare de context** - Modificarea stării de rulare a procesorului de la un proces către altul (modificarea memoriei).
  - **Partajarea procesorului** - Utilizarea unui cuantum a.î. fiecare proces rulează la o frecvență de  $1/n$ .
  - **Latență de replanificare** - Reprezintă timpul de așteptare din momentul în care un proces face o cerere de rulare și până în momentul în care obține controlul UCP (rulează).

# Planificarea UCP – Round Robin

## Coada circulară (ROUND ROBIN)

Se alege un quantum de timp

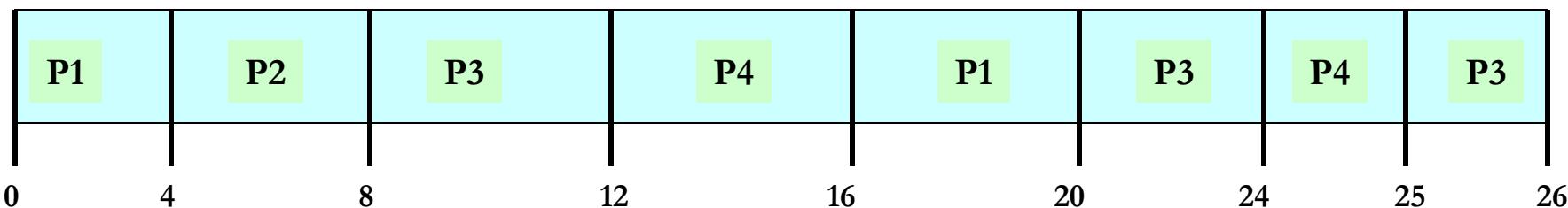
- Dacă acesta este prea scurt – se va pierde prea mult timp cu comutarea contextului
- Dacă acesta este prea lung – latența de replanificare este prea mare. Dacă multe procese doresc UCP, atunci se pierde prea mult timp ca acestea să acceseze UCP și se ajunge la cazul FIFO.
- De regulă, se ajustează a.î. majoritatea proceselor să nu își utilizeze timpul într-un singur quantum.

# Exemplu de planificare Round Robin

Exemplu:

Process	Timpul de sosire	Timpul de serviciu
1	0	8
2	1	4
3	2	9
4	3	5

Round Robin, cuantum = 4, fără priorități

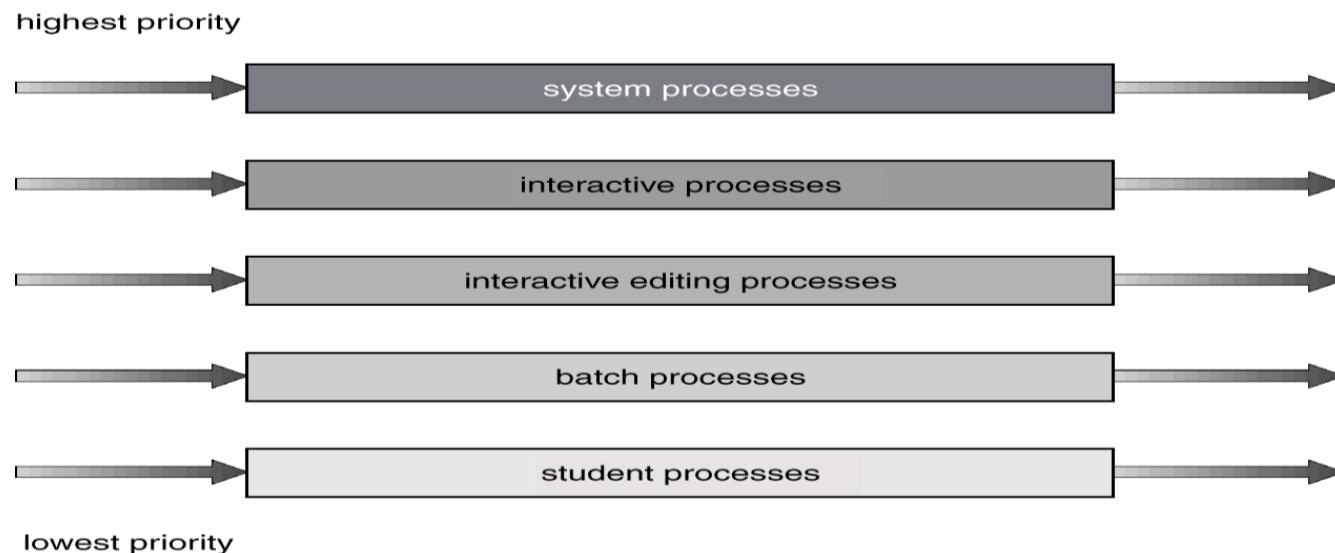


$$\text{Timpul mediu de rezidență} = \frac{(20-0) + (8-1) + (26-2) + (25-3)}{4} = \frac{74}{4} = 18.5$$

# Planificarea UCP – cozi de priorități

## Cozi multi nivel

- Fiecare coadă are propriul algoritm de planificare
- Alt algoritm (de regulă bazat pe priorități) asigură arbitrajul între cozi
- Se poate utiliza un feedback pentru a schimba coada
- Metodă complexă, dar flexibilă
- Astfel, se pot separa procesele de sistem, cele interactive, batch, favorizate sau nefavorizate



# Exemplu de priorități - Windows

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx)

## Clasa de priorități ale proceselor

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

## Nivelul de prioritate al thread-urilor

# Planificarea multi-procesor

## Planificarea în cazul multi-procesor:

- Reguli diferite pentru procesoare identice sau nu
- Se utilizează balansarea încărcării în distribuirea job-urilor, a.î. toate procesoarele să aibă aceeași cantitate de procesat
- Fiecare procesor face planificarea prin alegerea proceselor dintr-o coadă obișnuită de procese gata de execuție (mașini pereche) SAU se poate utiliza un model master/slave

# **Planificarea UCP – alte resurse informaționale**

## **Wikipedia despre planificarea UCP:**

- [http://en.wikipedia.org/wiki/Scheduling\\_\(computing\)](http://en.wikipedia.org/wiki/Scheduling_(computing))

## **Procese, thread-uri, job-uri și quantum în Windows:**

- <https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=7>

# **Administrarea proceselor în Linux**

**Despre crearea, monitorizarea și terminarea proceselor în Linux:**

**<https://developer.ibm.com/tutorials/l-lpic1-103-5/>**

**Priorități în execuția proceselor în Linux:**

**<https://developer.ibm.com/tutorials/l-lpic1-103-6/>**

# Sisteme de operare

Cursul #9

Bloaje (Deadlocks)

Răzvan Daniel ZOTA

Facultatea de Cibernetică, Statistică și Informatică Economică

[zota@ase.ro](mailto:zota@ase.ro)

<https://zota.ase.ro/so>

# Blocaje (Deadlocks)

- Ce este un blocaj?
- “Un blocaj reprezintă o stare în care, fiecare membru al unui grup de acțiuni așteaptă ca un alt membru să elibereze un blocaj”
- Sunt posibile două abordări:
  - În siguranță: prevenirea și evitarea blocajelor
  - În pericol: se permite apariția unui blocaj, apoi se detecteză și se rezolvă problema.

# Blocaje

## Exemple din lumea reală:

- “E nevoie de bani pentru a face bani”.
- Nu poți avea un job fără experiență; nu poți avea experiență fără a avea un job.

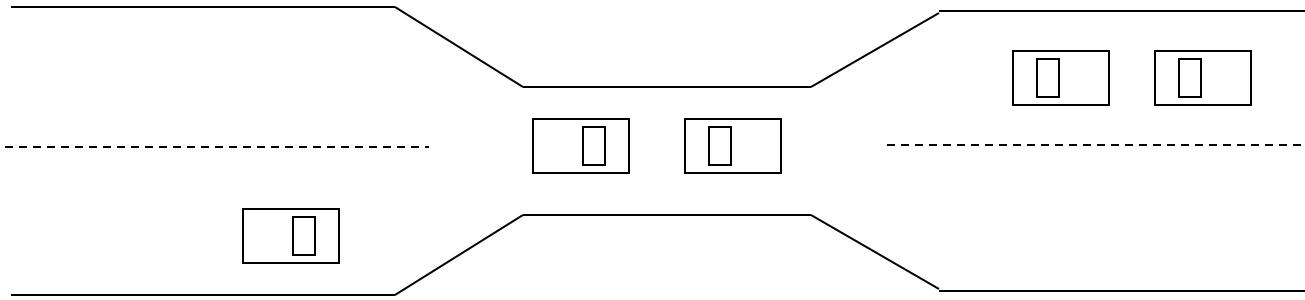
Cauza blocajelor: fiecare proces are nevoie de ce oferă alt proces.

Acest fapt are ca rezultat partajarea resurselor (memorie, echipamente, legături de comunicație, etc).

În cazul unei operări normale, alocarea resurselor se realizează în modul următor:

1. Are loc o cerere a resursei.
2. Se folosește resursa.
3. Se eliberează resursa.

# Exemplu - trecerea peste un pod



- Trafic doar într-o direcție.
- Fiecare secțiune a podului poate fi considerată ca o resursă.
- Dacă apare un blocaj poate fi rezolvat dacă una dintre mașini dă înapoi (preemptia resurselor).
- Mai multe mașini trebuie să se întoarcă dacă apare un blocaj.
- Poate apărea fenomenul de “starvation”.

A se vedea și problema celor 5 filozofi:

([http://en.wikipedia.org/wiki/Dining\\_philosophers\\_problem](http://en.wikipedia.org/wiki/Dining_philosophers_problem))

# Caracterizarea blocajelor

## Condiții necesare (Coffman - 1971)

Următoarele 4 condiții trebuie să se întâmple simultan pentru ca să apară un deadlock:

### “Excluderea mutuală”

Una sau mai multe resurse trebuie să fie deținute de către un proces într-un mod ne-partajabil (exclusiv).

### “Hold and Wait”

Un proces deține o resursă în timp ce așteaptă eliberarea altei resurse.

### Fără preemție

Nu există decât eliberarea voluntară a unei resurse – nimeni altcineva nu poate forța eliberarea resursei.

### Așteptare circulară

Procesul A așteaptă pentru procesul B care așteaptă pentru procesul C .... care așteaptă pentru procesul A.

# Graful alocării resurselor

O modalitate vizuală (matematică) de a determina apariția (sau posibilitatea apariției) unui blocaj.

$G = (N, M)$  noduri și muchii.

N Nodurile sunt procese = {P1, P2, P3, ...} și tipuri de resurse {R1, R2, ...}

M Muchiile sunt ( Pi, Rj ) sau ( Ri, Pj )

O săgeată de la un **proces** către o **resursă** indică faptul că procesul **are nevoie** de acea resursă.

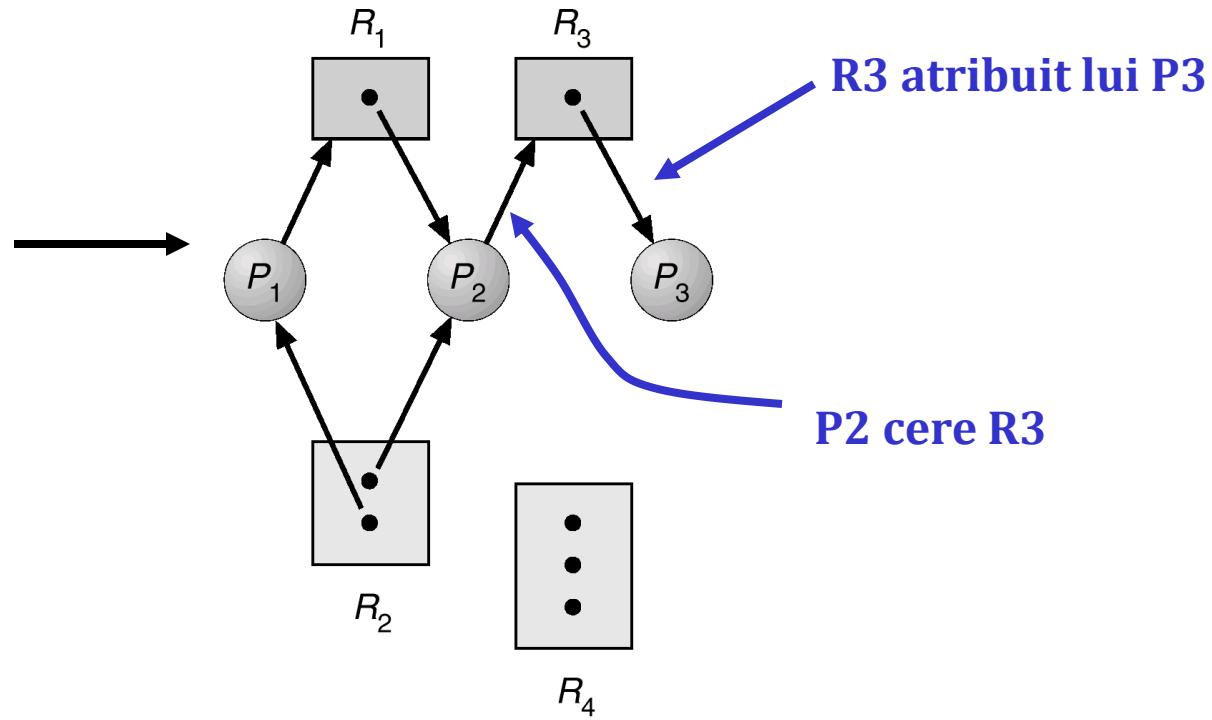
O săgeată de la o **resursă** către un **proces** indică faptul că o instanță a resursei a fost **alocată** aceluia proces.

Procesul este un cerc, tipul de resursă este un pătrat; punctele reprezintă numărul instanțelor resursei de un anumit tip.

# Graful alocării resurselor

- Dacă graful nu are cicluri, nici un proces nu este blocat.
- Dacă apare un ciclu, atunci:
  - a) Dacă tipurile de resurse au mai multe instanțe, atunci POATE exista un blocaj.
  - b) Dacă fiecare tip de resursă are o singură instanță, apare un blocaj.

Graful alocării resurselor



# Strategii generale de abordare a blocajelor

3 metode generale:

- **Ignorarea** blocajelor.
- Asigurarea faptului că blocajul nu poate surveni **niciodată** prin:

**Prevenire** Prevenirea apariției uneia dintre cele 4 condiții (**Excluderea mutuală**, **“Hold and Wait”**, **Fără preemție**, **Așteptare circulară**).

**Evitare** Se permit condițiile de deadlock dar se calculează ciclurile de apariție a acestora și se opresc operațiile periculoase (ce pot conduce la un deadlock).

- **Se permite** apariția blocajelor. În acest caz se folosesc:

**Detectia** Se știe momentul în care a apărut un blocaj.

**Refacerea** Se “refac” resursele.

# Prevenirea blocajelor

Nu se permite apariția niciuneia dintre cele 4 condiții.

## Excluderea mutuală:

- a) Are loc automat pentru imprimante și alte echipamente ne-partajabile.
- b) Entitățile de tipul fișierelor **read only** nu au nevoie de excludere mutuală, nefiind susceptibile la blocaje.

## “Hold and wait”:

- a) Se strâng toate resursele înaintea execuției.
- b) O anumită resursă poate fi cerută doar când nimeni nu o detine. O secvență de resurse este totdeauna cerută începând cu prima.
- c) Gradul de utilizare este scăzut, poate apărea fenomenul de “starvation”.

# Prevenirea blocajelor

Nu se permite apariția niciuneia dintre cele 4 condiții.

## Fără preempție:

- a) Se eliberează orice resursă care este deținută dacă procesul nu poate obține altă resursă.
- b) Dacă o resursă este deținută de alt proces care este în stare de așteptare pentru altă resursă, aceasta este "furată". În caz contrar, se așteaptă eliberarea resursei.

## Așteptarea circulară:

- a) Numerotarea resurselor și cererea lor în ordine crescătoare.
- b) Poate duce la o scădere a utilizării resurselor; nu este neapărat cea mai bună metodă de implementat.

# Evitarea blocajelor

Dacă putem cunoaște de dinainte modul de utilizare al resurselor este posibil să determinăm dacă poate apărea o stare “nesigură”.

Stări posibile:

**Deadlock** Nu se poate face nimic constructiv în continuare.

**Stare nesigură** O stare în care **poate apărea** un blocaj.

**Stare sigură** O stare se numește **sigură** dacă există o secvență de procese astfel încât există suficiente resurse pentru ca primul proces să se termine și fiecare proces se termină și eliberează resursele ce sunt necesare pentru ca următorul proces să se termine.

Regulă simplă: “dacă o cerere de alocare de resurse poate conduce la o stare nesigură, nu satisface acea cerere”.

**Obs. Toate blocajele provin din stări nesigure, dar nu toate stările nesigure sunt blocaje.**

# **Mecanisme de concurență a proceselor în UNIX/Linux**

În lumea UNIX/Linux există mai multe mecanisme pentru comunicarea și sincronizarea între proceze:

- **Pipes**
- **Mesaje**
- **Memorie partajată**
- **Semafoare**
- **Semnale**

# Mecanisme de concurență a proceselor în UNIX/Linux

**Pipes** – buffer circular ce permite comunicarea între două procese pe baza procesului “producător-consumator” – coadă de tip FIFO în care un proces trimite un mesaj spre alt proces. Atunci când este creat un *pipe*, acesta are o dimensiune fixă în octeți. În momentul în care un proces încearcă să scrie în *pipe*, cererea de scriere este executată imediat (dacă există suficient spațiu; în caz contrar, procesul este blocat).

În mod asemănător, un proces ce necesită o operație de citire este blocat în cazul în care încearcă să citească mai mulți octeți decât există în *pipe* – în caz contrar, cererea de citire este executată imediat.

SO aplică în acest caz excluziunea mutuală – doar un singur proces poate accesa pipe-ul la un moment dat.

# Mecanisme de concurență a proceselor în UNIX

**Mesaje** – Un mesaj este un bloc de text de un anumit tip. SO UNIX oferă apelurile de sistem ***msgsnd*** și ***msgrcv*** pentru transferul de mesaje. Fiecare proces are o coadă de mesaje care funcționează asemenea unei cutii poștale.

**Memorie partajată** – reprezintă cea mai rapidă formă de comunicare între procese oferită de UNIX. Există în acest sens un bloc comun de memorie virtuală partajată de mai multe procese. Procesele citesc sau scriu în spațiul comun de memorie folosind aceleași instrucțiuni cu care citesc/scriu alte porțiuni de memorie virtuală.

Permisiunile sunt *read-only* sau *read-write* pentru fiecare proces în parte. Excluderea mutuală este asigurată pentru procesele care folosesc memoria partajată.

# Mecanisme de concurență a proceselor în UNIX

**Semafoare** – reprezintă o generalizare a primitivelor de sistem *wait* și *signal*. Kernelul face toate operațiile una câte una și nici un proces nu poate accesa semaforul până când nu sunt îndeplinite toate operațiile.

**Semnale** – un semnal este un mecanism software ce informează procesul de apariția unui eveniment asincron. Un semnal este similar cu o întrerupere hardware dar nu implică priorități (toate semnalele sunt considerate “egale”). Semnalele care apar în același timp sunt prezentate unui proces pe rând, fără a se stabili o ordine bazată pe priorități.

Procesele pot trimite semnale unele altora sau acestea sunt trimise de către kernel. Un semnal este livrat prin actualizarea unui câmp în tabela de procese a procesului ce recepționează acel semnal.

# InformIT.com

Pe site-ul informIT.com găsiți informații din toate domeniile ce au legătură cu “zona” de IT

Aici aveți o *Introducere în blocajele SO*:

<https://www.informit.com/articles/article.aspx?p=25193>

# Operating Systems

## Course #10 Virtual Memory

Răzvan Daniel ZOTA

Faculty of Cybernetics, Statistics and Economic Informatics

[zota@ase.ro](mailto:zota@ase.ro)

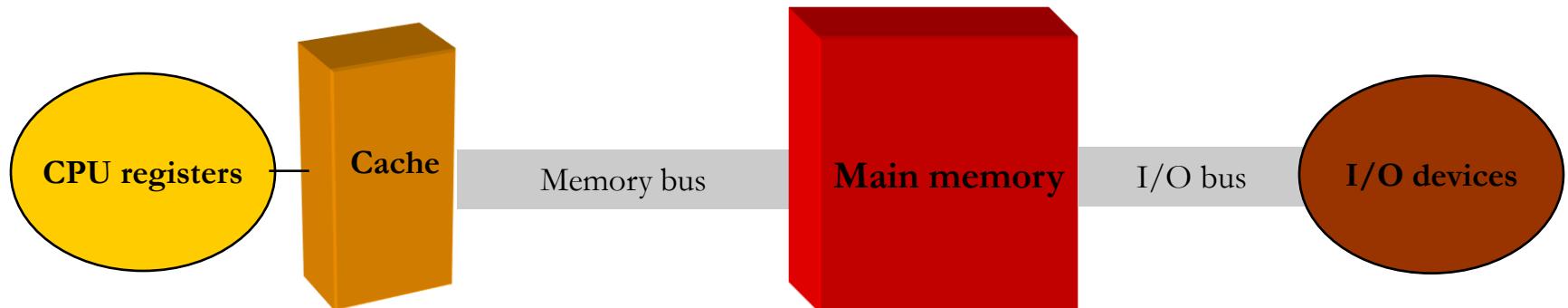
<https://zota.ase.ro/os>

# *Contents*

---

- Memory types & memory hierarchy
- Virtual memory (VM)
- Page replacement algorithms in case of VM

# *Memory hierarchy*

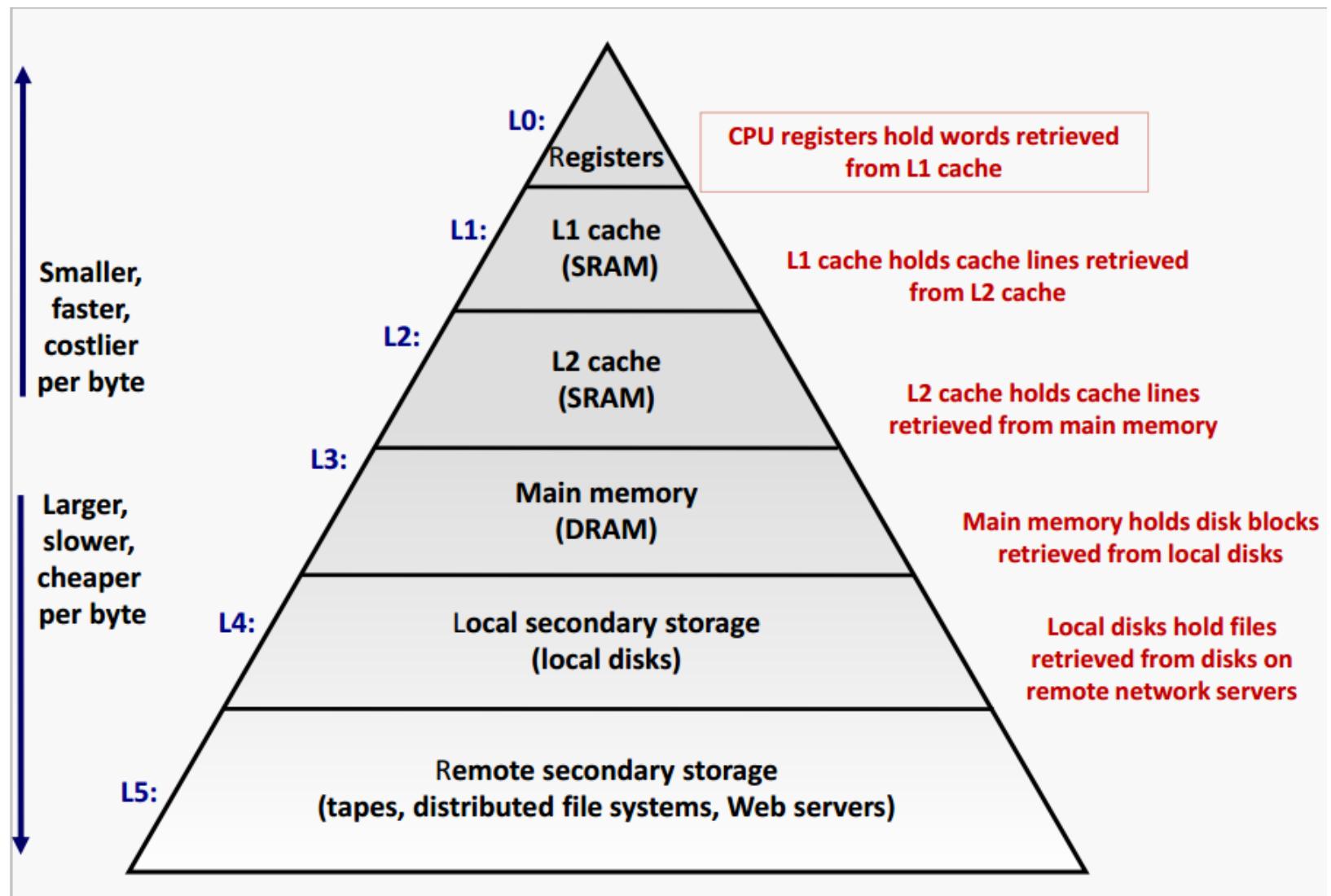


	CPU registers	Cache	Main memory	I/O devices
Dimension	200 B	>2 MB	>1 GB	>2 GB
Speed	1-5 ns	10-15 ns	60-100 ns	5 ms
Maximum dimensions	1 KB (CMOS or BiCMOS)	8 MB SRAM	32 GB DRAM	2-3 TB
Bandwidth (MB/sec)	4000-32000	800-5000	400-2000	4-32
Managed by	Compiler	Hardware	OS	OS/user
Backup	Cache	Main memory	Disk	Magnetic tape

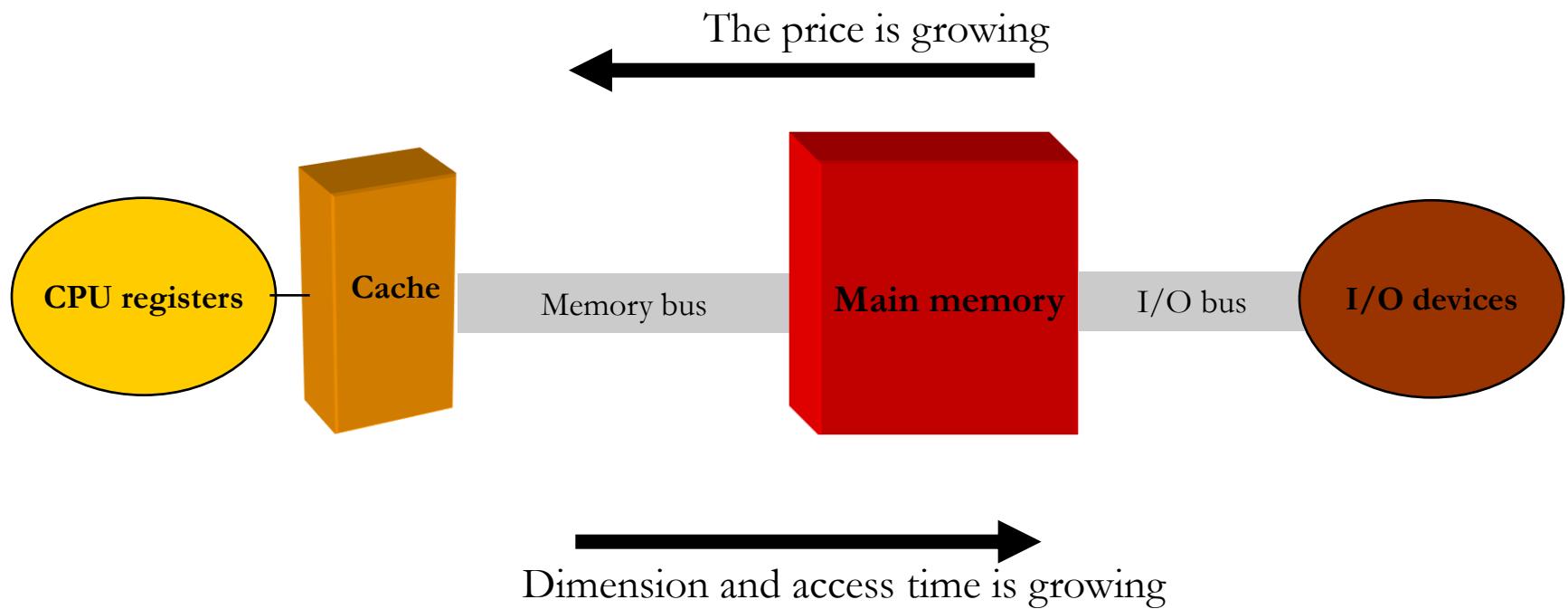
# *Different memory types - characteristics*

Memory type	Dimension	Speed	Maximum dimensions	Bandwidth (MB/sec)
CPU Registers	Octeți (bytes)	1 ns	Very small	~10,000+
Cache L1	16–128 KB	1–5 ns	~128 KB	~40,000+
Cache L2	128 KB – 1 MB	5–20 ns	~1 MB	~25,000
Cache L3	2–50 MB	10–50 ns	~50 MB	~10,000
RAM (DRAM)	4–128 GB	50–100 ns	~512 GB (server)	~25,600–51,200
SSD (NVMe)	256 GB – 4 TB	0.1–1 ms	~8 TB (desktop)	~3,500–7,000
HDD	500 GB – 20 TB	5–20 ms	~20 TB	~100–200
Magnetic tape (LTO - Linear Tape-Open)	1–30 TB (on band)	Slow (seconds)	Over 45 TB (for archives)	~100

# *Memory hierarchy*

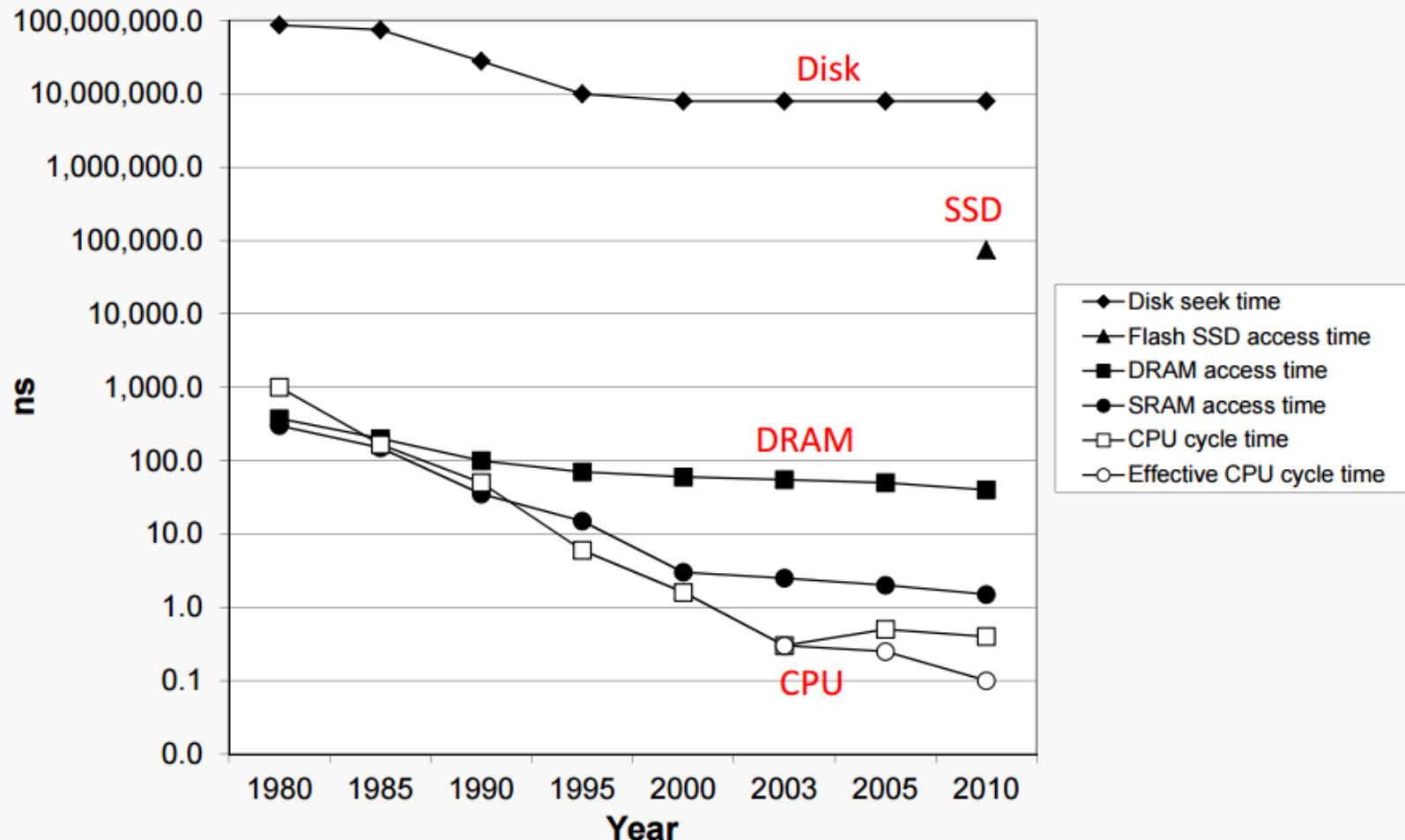


# *Memory types*



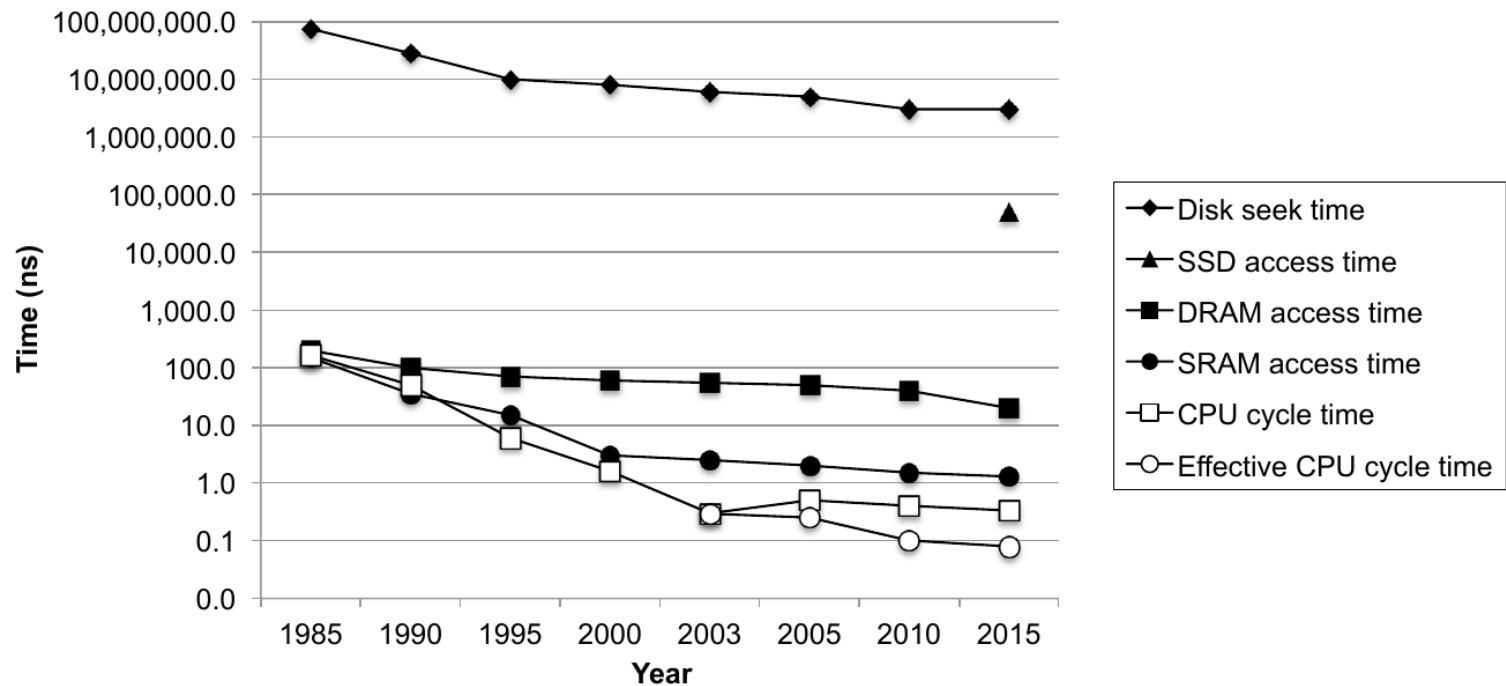
# *The gap between DRAM, SRAM, disk and CPU*

The gap widens between DRAM, disk, and CPU speeds.



# *The gap between DRAM, SRAM, disk and CPU (cont.)*

Similar evolution up to 2015 and present time:



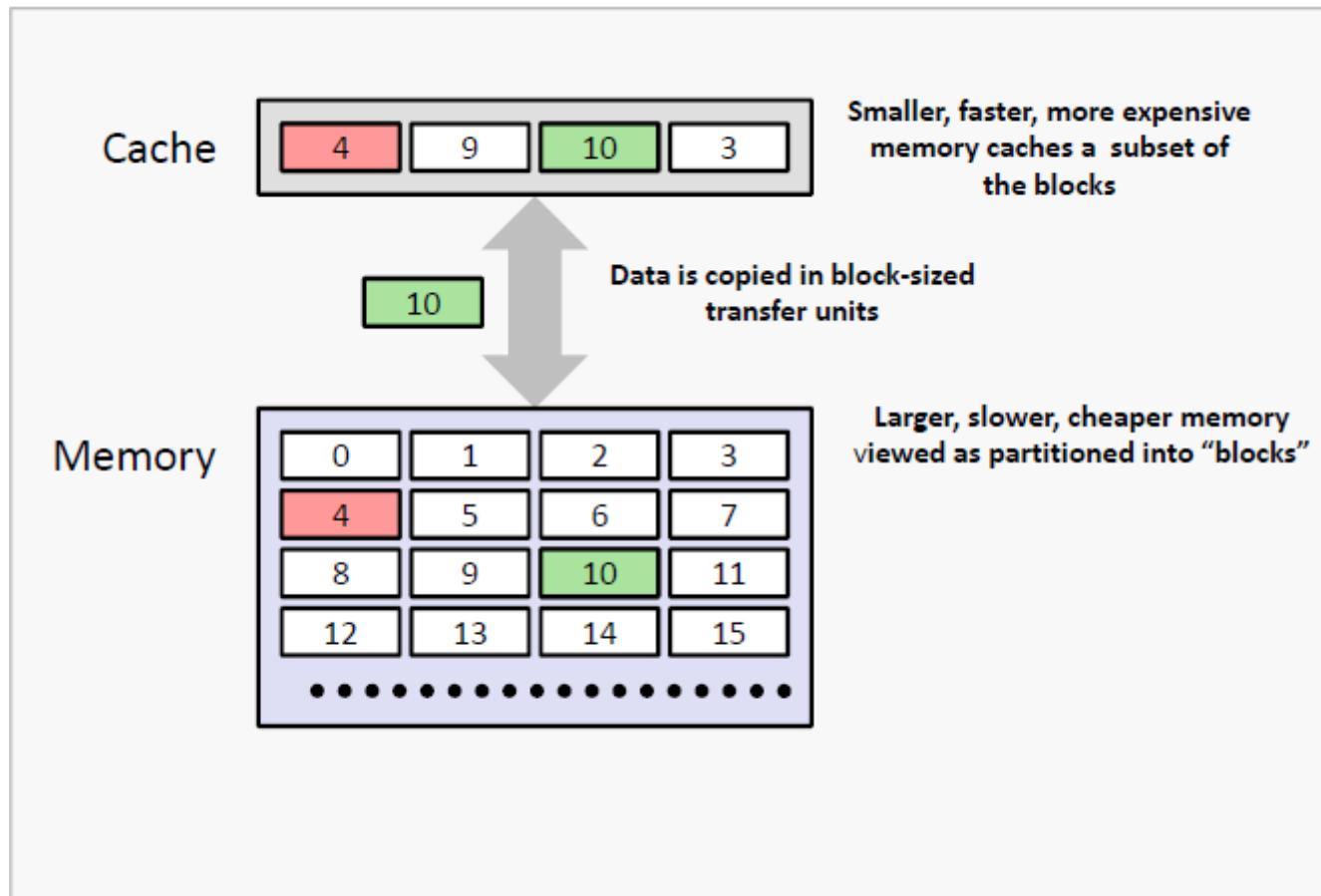
# *Cache memory*

- Small, ultra-fast memory, inside the CPU(or close to the CPU ), containing the most recently accessed data and/or instruction code
- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently
- Principle of “**temporal locality**” – Recently referenced items are likely to be referenced again in the near future, meaning that it is very probably to need the data soon, so it is **placed into cache**, where we can access quickly.
- Principle of “**spatial locality**” - Items with nearby addresses tend to be referenced close (or relatively close storage locations) together in time.

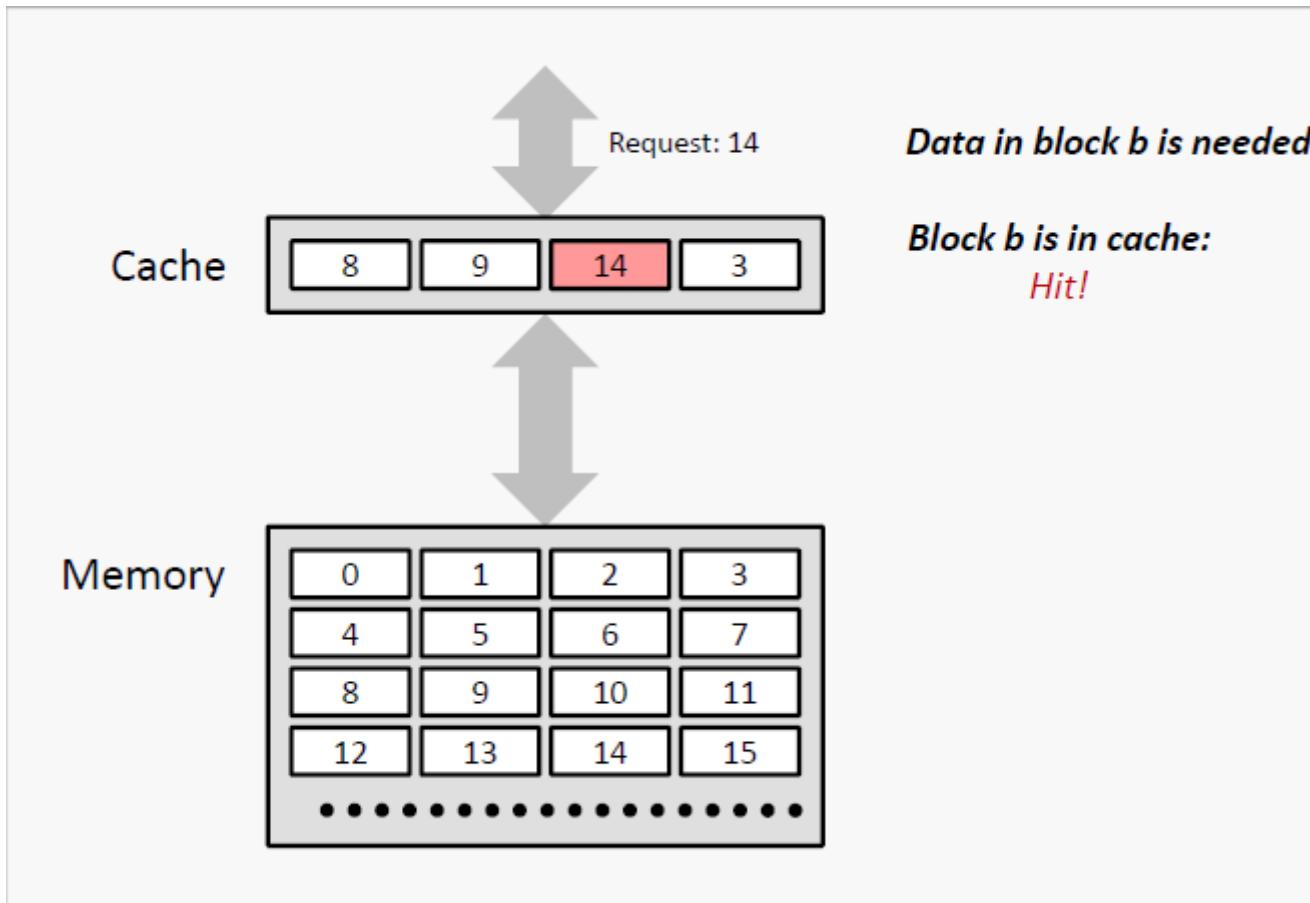
# *Cache memory*

- For these reasons, we may have:
  - A “Cache hit”, or
  - A “Cache miss” (a data block of fixed dimension containing the data necessary to be extracted from the main memory and inserted into the cache memory)
    - The time necessary for a “cache miss” depends of the memory latency and the bandwidth – and will determine the time needed to read the whole memory block. A “cache miss” managed by hardware will determine a break in CPU functioning – until the moment when the data are available.
- <https://sabercomlogica.com/en/a-case-study-intel-nehalem-i7-coherence-in-cache/>
- <http://csillustrated.berkeley.edu/PDFs/handouts/cache-3-associativity-handout.pdf>

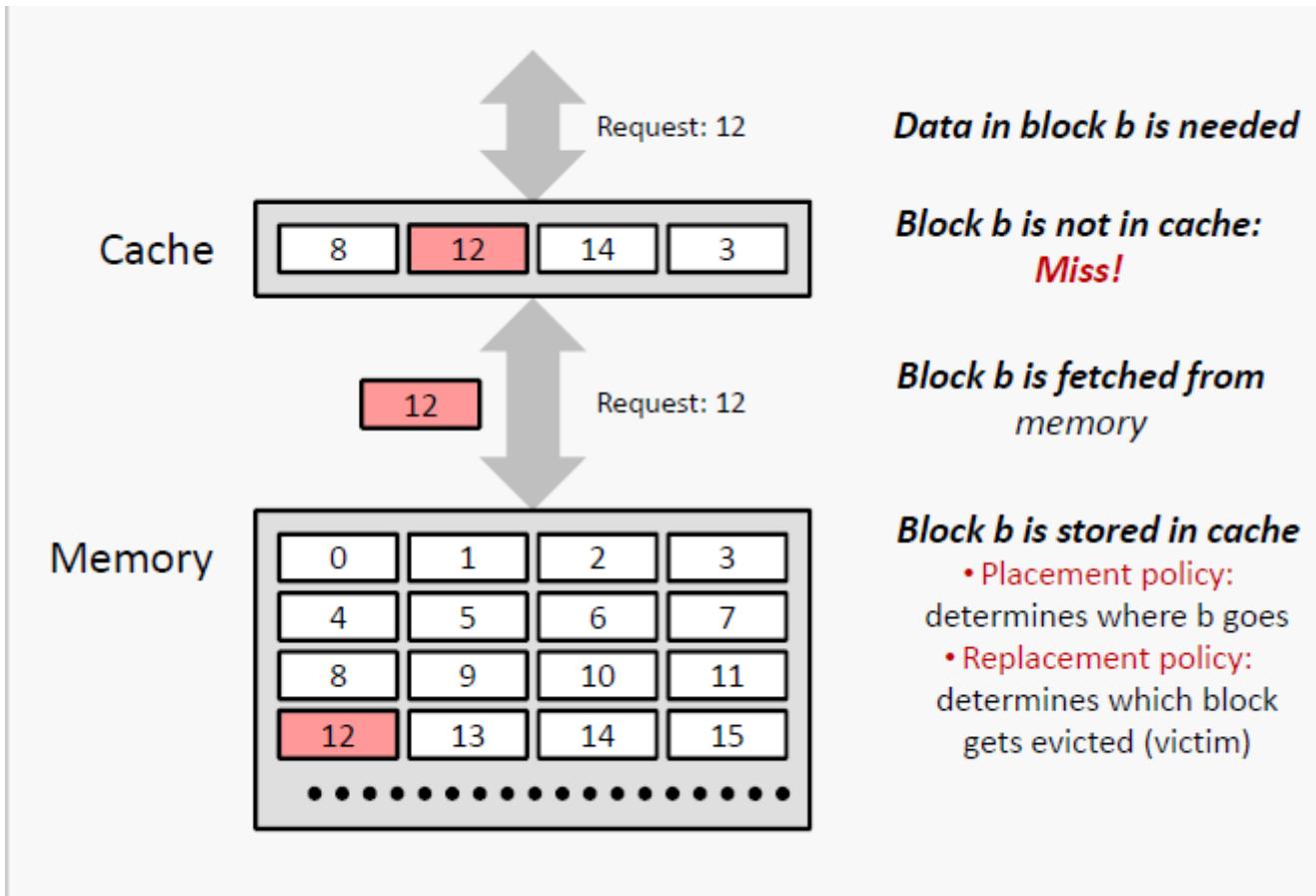
# *Cache memory*



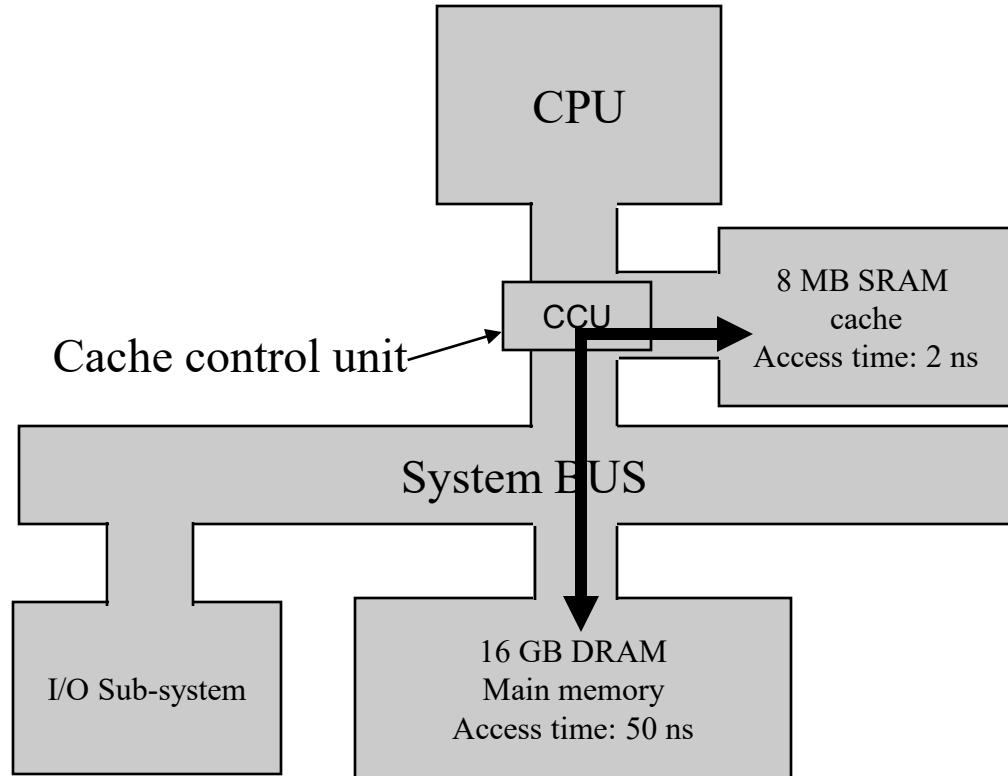
# *Cache hit*



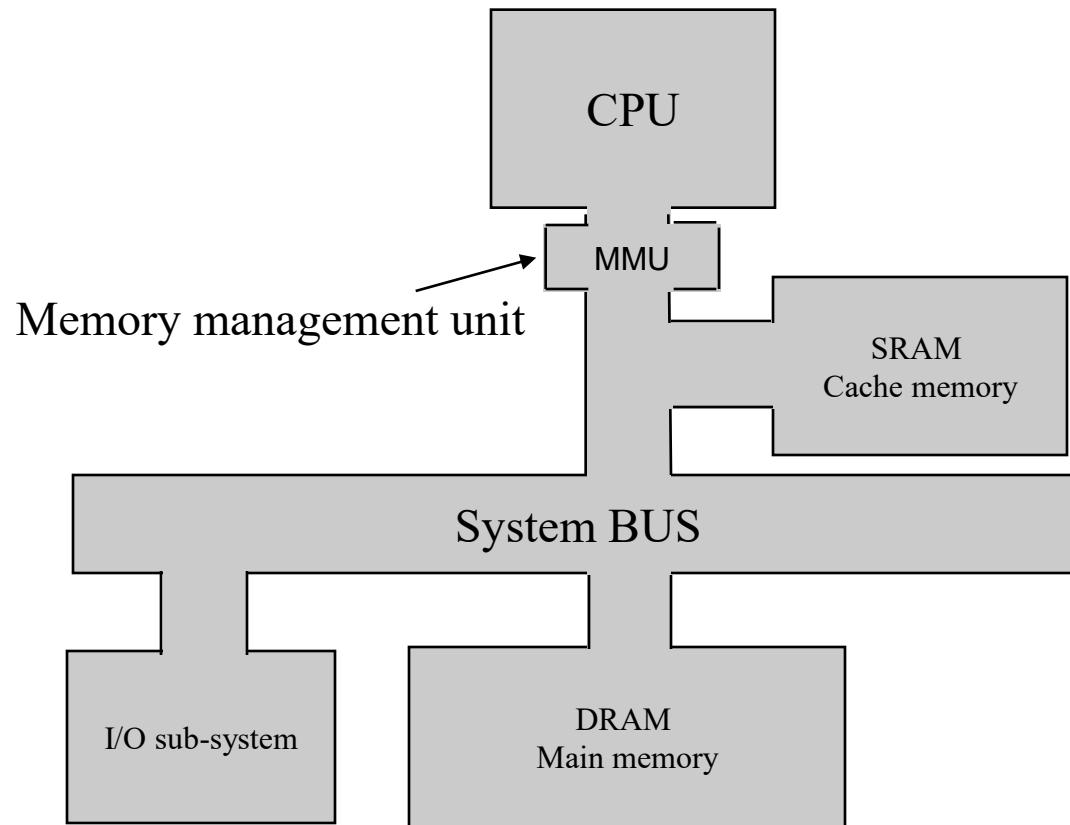
# Cache miss



# *A simplified CPU architecture: Cache memory and cache memory controller*



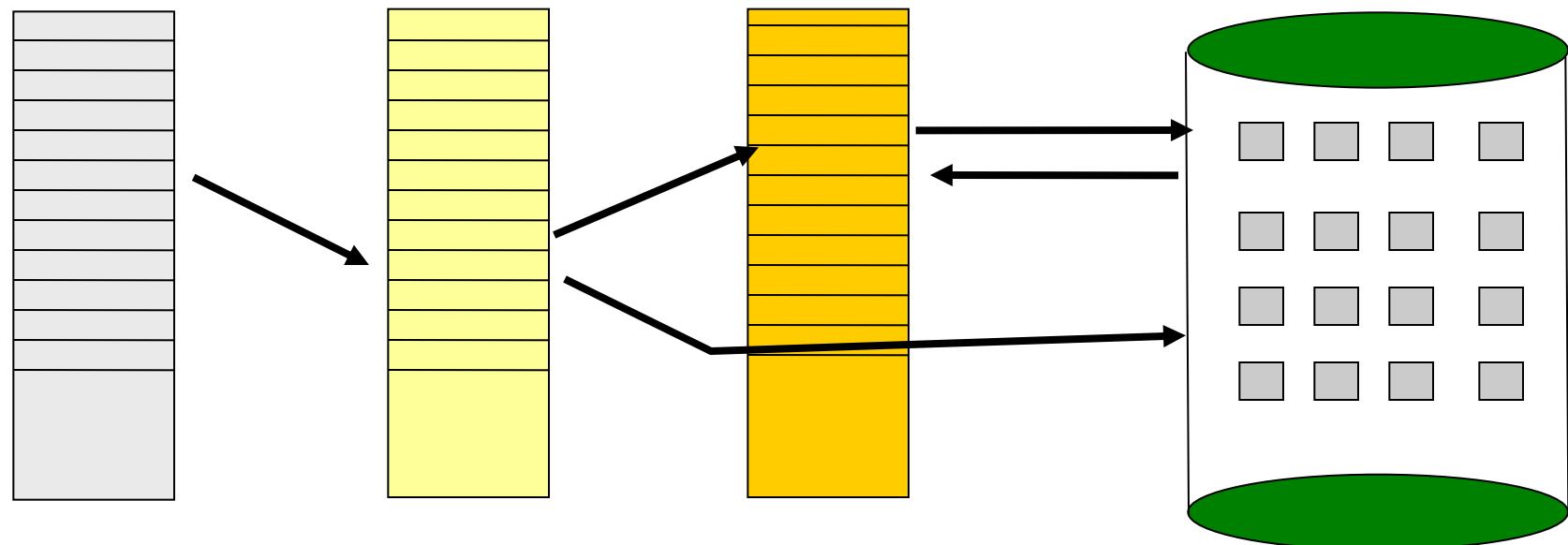
# *A simplified CPU architecture: Virtual memory management unit*



# *Virtual memory*

**Virtual memory** – It represents, in short terms, the use of the disk as an extension of the RAM memory, so that the effective dimension of the usable memory is increased. In this way, we can have a virtual memory of big dimensions, **larger** than the physical memory.

Memory pages



Virtual memory

Memory map

Physical memory (RAM)

Disk

# *Virtual memory*

- In case of using virtual memory not all the objects (instructions or data) are in the main memory at a specific time; some of them may be on disk.
- Addressing space is divided in fixed length blocks – called *pages*.
- At some moment in time, the pages are *in the main memory* or *on disk*.
- When the CPU tries to access an object that is not in cache, nor in the main memory, it appears a “page-fault” – in this moment the whole page is moved from the disk into main memory. These “page-faults” take a lot of time and involve *swapping*. *Swapping* is the mechanism in which a process can be swapped temporarily out of main memory to a backing storage (disk), and then brought back into memory to continue the execution.
- The *cache memory* and the *main memory* have the same relation between them as the one between the *main memory* and the *disk*.

# *Virtual memory*

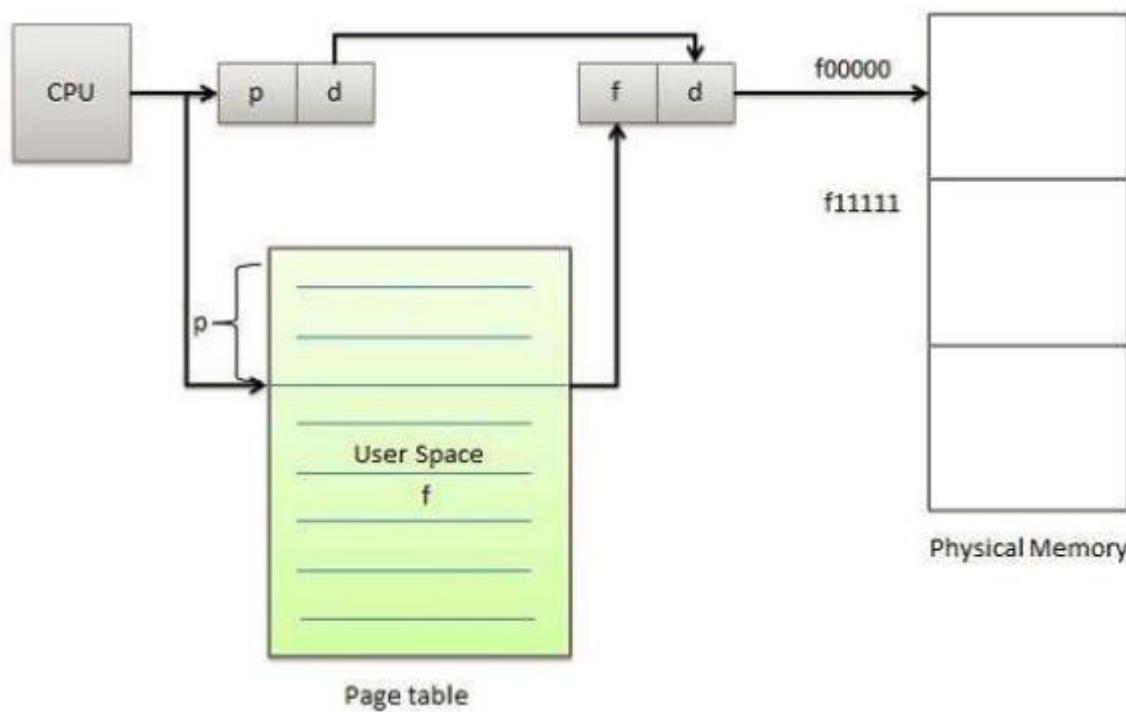
- At any moment in time, a computer runs several processes, each with its own memory address space. A part of the main memory is shared between multiple processes.
- The physical memory is divided into blocks allocated to several processes.
- In this case, a *protection scheme* is needed – this restrict the access of the processes to blocks belonging to other processes. The mechanism of virtual memory is reducing also the starting time of a program, because not all its code or data must be in main memory before starting the program.
- In the case of virtual memory, the memory blocks are called *pages* or *segments*. CPU uses virtual addresses translated into physical addresses in order to access main memory. This is called *memory mapping* or *address translation*.

# *Paging*

- *Paging* represents a technique in which physical memory is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes).
- When a process needs to be executed, its corresponding pages are loaded into any available memory frames. Logical address space of a process can be non-contiguous and a process is allocated physical memory whenever the free memory frame is available.
- Operating system keeps track of all free frames. Operating system needs  $n$  free frames to run a program of size  $n$  pages.
- Address generated by CPU is divided into:
  - A Page number ( $p$ ) -- this is used as an index into a page table which contains base address of each page from the physical memory.
  - A Page offset ( $d$ ) – the page offset is combined with base address to define the physical memory address.

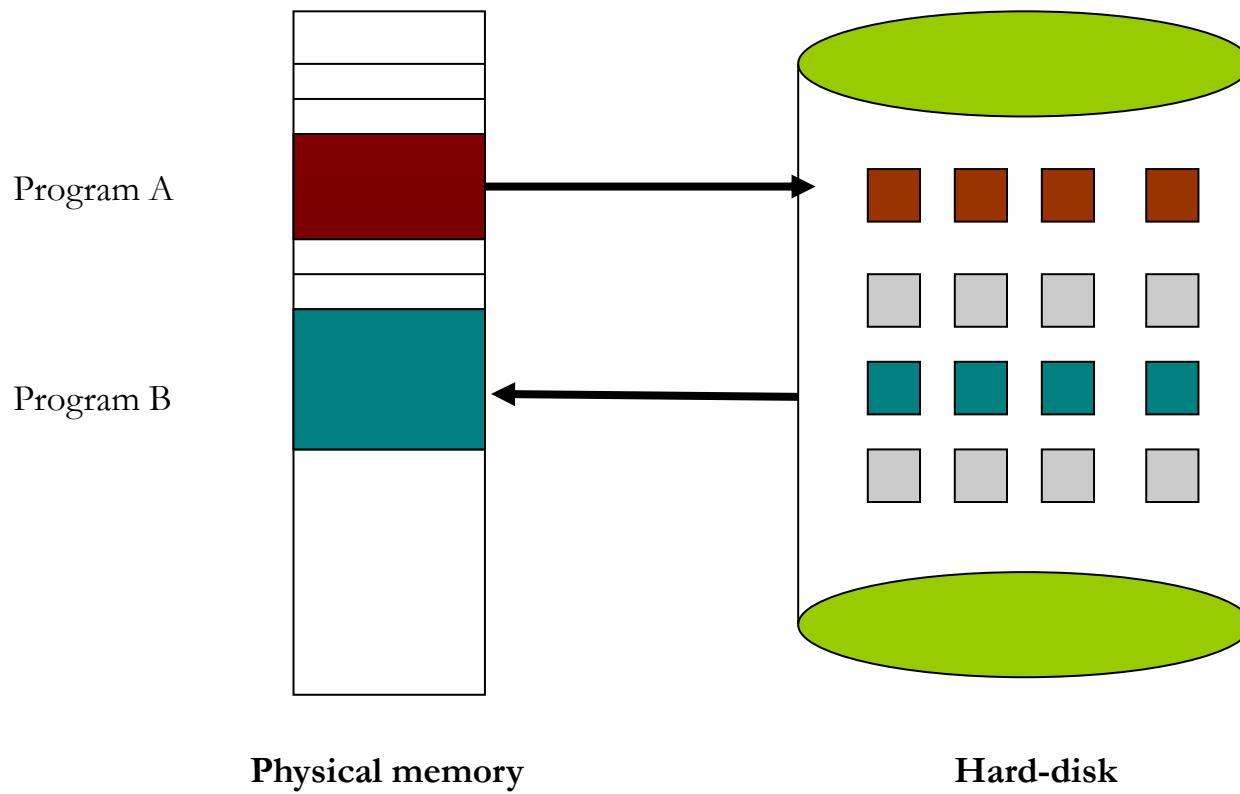
# Paging

- How paging may look:



# *Demand paging*

A demand paging system is a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, it is used a swapper (called pager) to swap a page.



**Virtual memory is about main memory and hard-disk**

# Demand paging

The “rule” says that *page faults* are *rare*.

The page table needs a “resident” bit to show us if the page is in memory or not. Sometimes it is used the “valid” term to indicate residence in memory.

An “invalid” page is a non-resident page or a page with an illegal address.

Another implementation uses two bits: one bit is indicating that the page **is valid** and the second one is showing if the page **it's in memory or not**.

Page no.	Valid/ invalid bit
	1
	1
	1
	1
	0
:	
	0

Page no.	Resident bit	Valid/ invalid bit
	1	1
	0	0

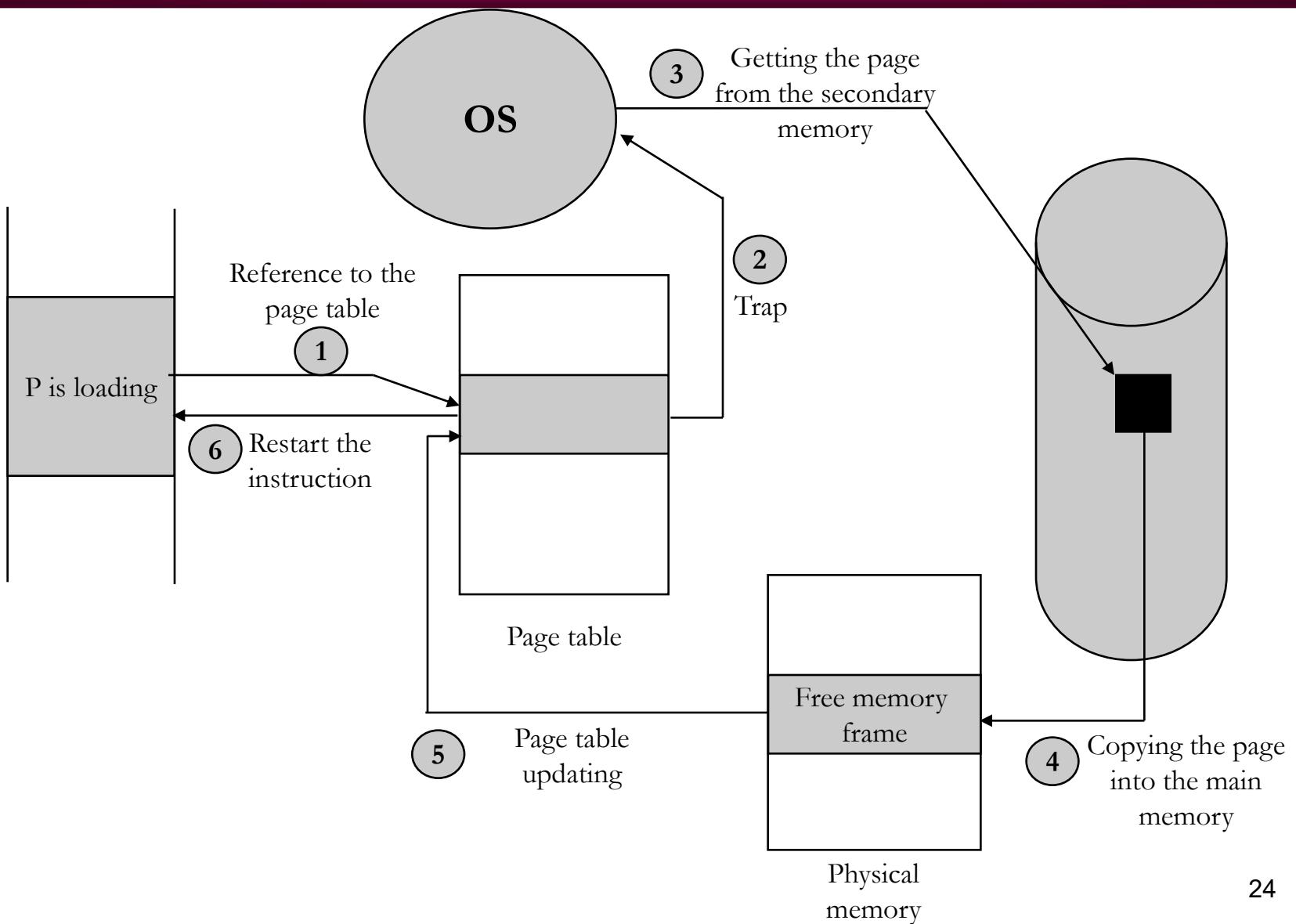
# *Demand paging*

Page fault means that a process needs a page that is not resident in memory.

The administration procedure implies the following steps (see next slide):

1. The page table is checked to see if the memory reference is valid or not.
2. If it's valid but not resident (in memory), it is tried to obtain it from the secondary memory.
3. A free frame is searched and allocated (a physical memory page unused till present – a memory page may be needed to be freed up).
4. A disk operation is scheduled to copy that page from the secondary memory in the new allocated frame.
5. After writing the page into memory the page table is modified – now the page is resident in memory.
6. The instruction that generated the *page fault* is restarted.

# *Page fault administration*



# *Page replacement*

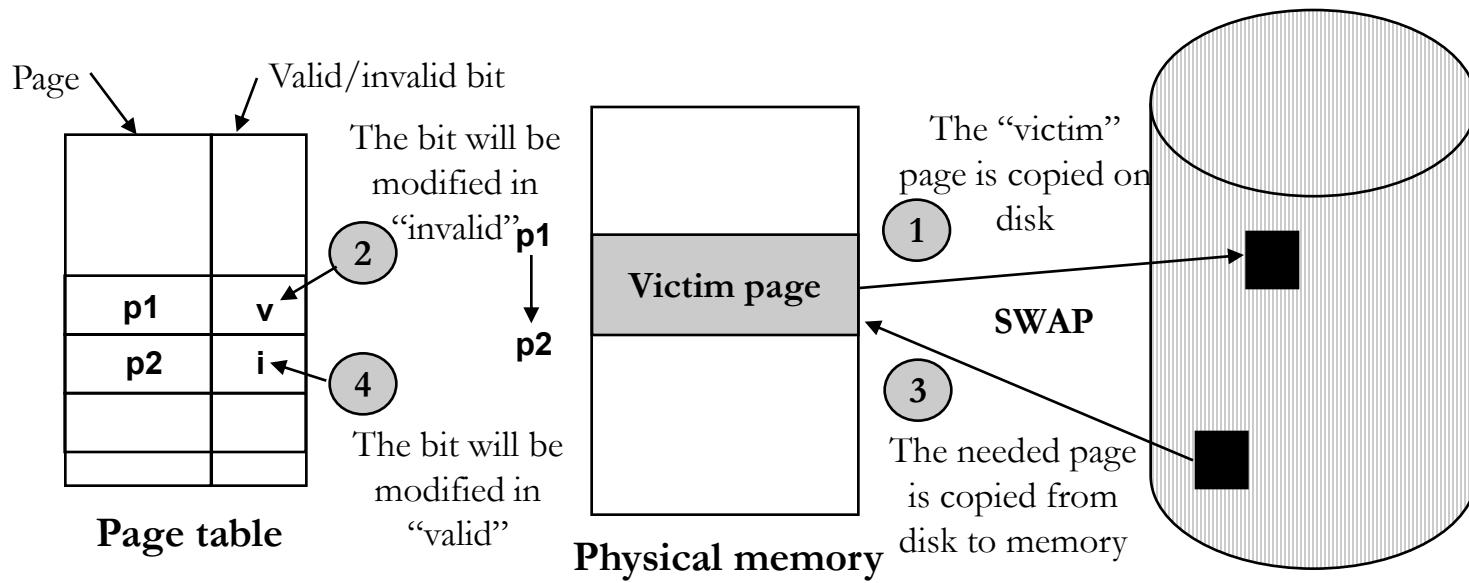
When the memory is over-allocated we must swap something already loaded into memory.

Over allocation appears when the programs need more memory pages than those physically existent.

**Approach:** If no physical page is free, one page that it is not used is swapped, using the following steps:

1. Find the page on disk.
2. Find a free frame:
  - a. If exists, use it
  - b. Otherwise, select a “victim page” and
  - c. Write the “victim page” on disk.
3. Read the new page in the freed frame. Update the page table.
4. Restart the process.

# *Page replacement*



The page replacement mechanism

## *Page replacement algorithms: FIFO*

Page replacement algorithms are the techniques used by the OS to decide which memory pages to swap out (write to disk) when a page of memory needs to be allocated.

What pages are to be replaced? – in order to minimize the number of *page faults*.

Suppose we consider the following reference string for memory pages to be replaced:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

### FIFO

Easy to implement. Oldest page in main memory is the one which will be selected for replacement.

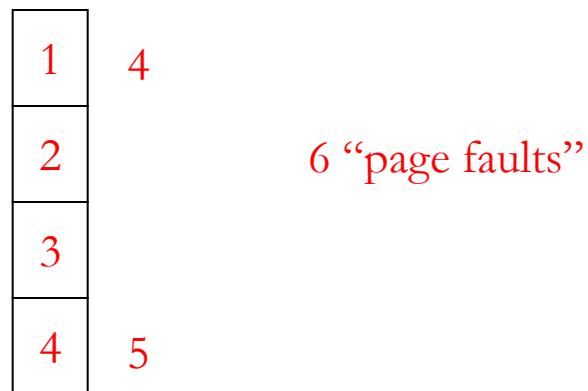
It can be used a “time-stamp” for pages, or an organization in a queue (the easiest method).

1	5	4
2	1	5
3	2	10 “page faults”
4	3	

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

## *Page replacement algorithms: Optimal replacement*

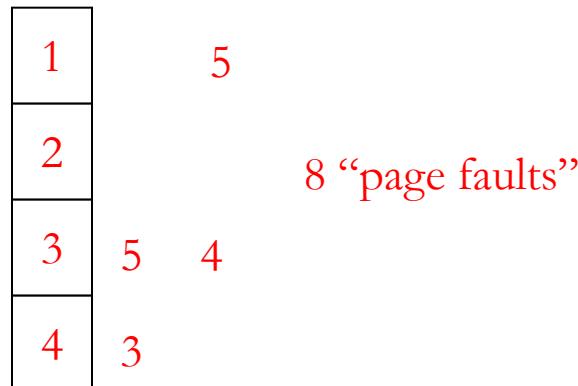
- Represents a page replacement policy for the pages that have the smallest “page fault” rate.
- The algorithm: we replace the page that will not be used for the longest period of time.
- Practically is impossible to implement it.



Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

## *The LRU(Least Recently Used) algorithm*

- It replaces the page that have not been used for the longest period of time.
- The results are good
- Implementation alternatives:
  - Using a "time stamp" for pages - record the last use.
  - Using a list – replace pages by looking back into time.



Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

## *Short introduction – VM in Linux*

---

- Virtual memory in Linux: <https://www.tldp.org/LDP/sag/html/vm-intro.html>

# Operating Systems

Course #11

OS Security

Răzvan Daniel ZOTA

Faculty of Cybernetics, Statistics and Economic Informatics

[zota@ase.ro](mailto:zota@ase.ro)

<https://zota.ase.ro/os>

# Today's cybercrimes

Malicious COVID-19 domains and special virus-themed sales on the dark web are two ways criminals are using the outbreak to ramp up business, said security provider Check Point.



The coronavirus is causing pain, anxiety, and stress for people and countries around the world. But for cybercriminals, the spread of COVID-19 is an area ripe for exploitation and profitability. The

# Cybercrimes from the (near) past

**WannaCry (or WannaCrypt,<sup>[3]</sup> Wana Crypt0r 2.0,<sup>[4][5]</sup> Wanna Decryptor<sup>[6]</sup>)** is a ransomware program targeting the Microsoft Windows operating system. On Friday, 12 May 2017, a large cyber-attack was launched using it, infecting more than 230,000 computers in 150 countries, demanding ransom payments in the cryptocurrency Bitcoin in 28 languages.<sup>[7]</sup> The attack has been described by Europol as unprecedented in scale.<sup>[8]</sup> (Wikipedia)



# Ransomware attacks

CNN tech

business

culture

gadgets

future

startups

CNN tech  
Cyber-Safe

## Massive ransomware at countries

**Analysis** Global ransomware at why Apple wouldn't hack terrori

WannaCry ransomware used in

**Telefónica hack: Ransomware attack on internal network forces computer shut do**

▶ 1:14 / 1:51

◀ ▶ 🔍

Ransomware 'WannaCry' attack explained

# Recent major security issues in 2024

<https://securityboulevard.com/2024/04/vulnerabilities-for-ai-and-ml-applications-are-skyrocketing/>

<https://securityboulevard.com/2025/05/genais-new-attack-surface-why-mcp-agents-demand-a-rethink-in-cybersecurity-strategy/>

# Introduction

Network/OS security represents a **hot topic** in the IT world

Security: **warranty/steps that must be taken to protect** a computer and the information stored on it

The need is for a **balance between accessibility and security**

The users prefer the first notion, administrators the second one!

# Introduction

The security level depends on multiple factors:

## **The business type of the company**

- Governmental, juridical institutions, etc.
- Educational institutions (the students information is stored – marks, etc.)
- Hospitals, other medical institutions
- Companies in the military domain/ other institutions involved in state's security
- Diverse companies/organizations which are gathering data under the confidentiality warranty, etc.

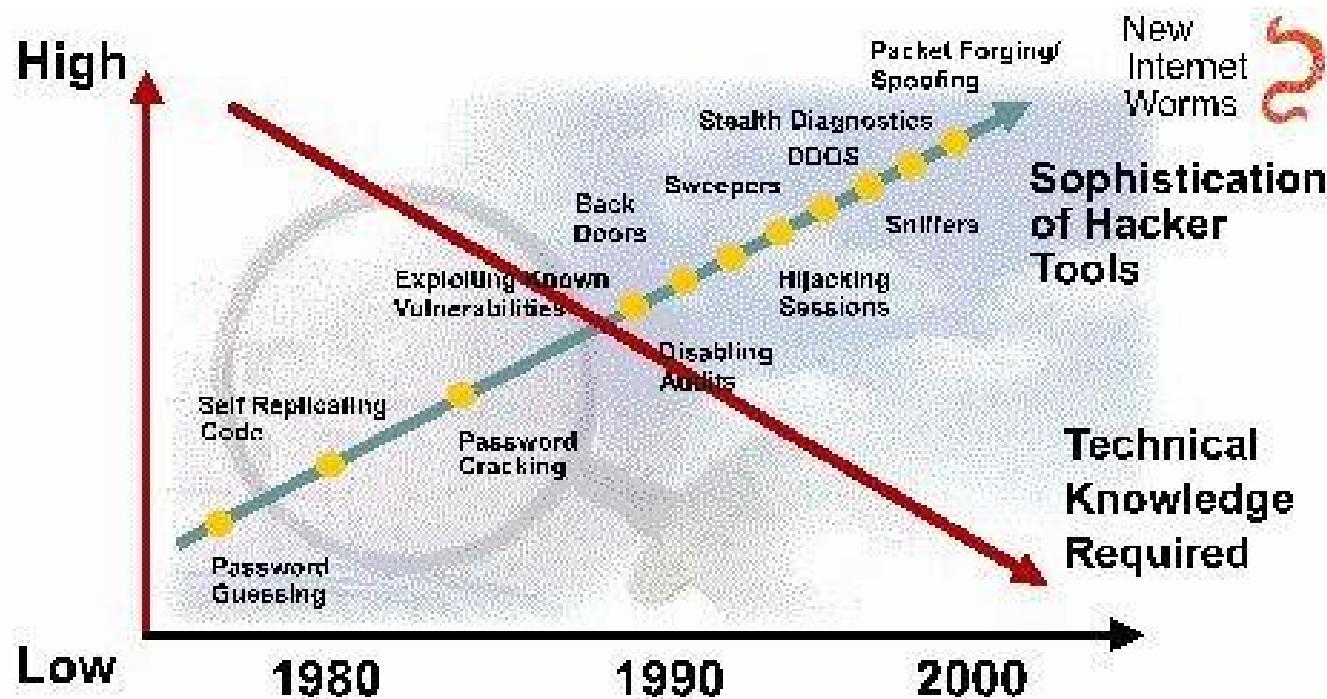
## **Types of data stored in the network/in the company's computers**

- Records about payment/personal information of the clients/customers
- Information about accountancy, taxes, etc.
- Commercial secrets (plans, drawings, sketches, recipes, business strategies, etc.)

## **Company's management philosophy**

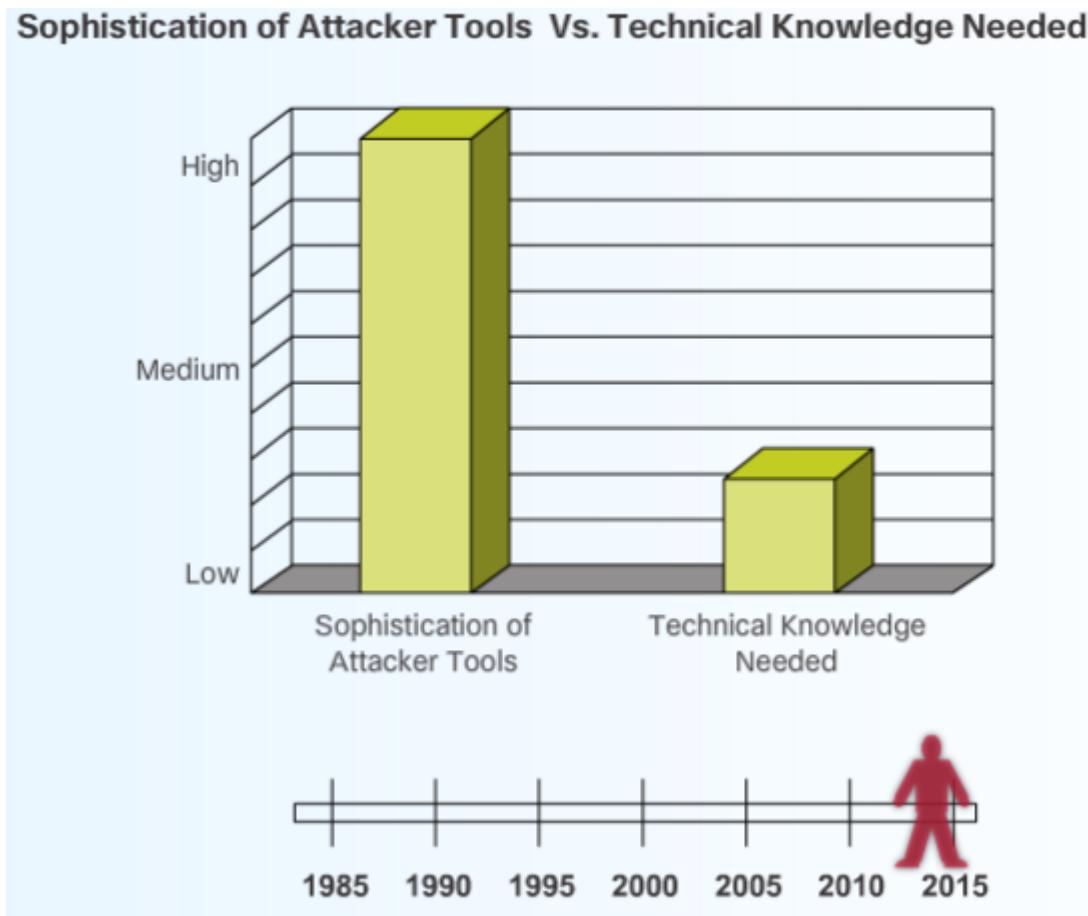
- The company is considered as a big, happy, family...
- Information accessible only when is needed, only for the ones interested

# Attacks' characteristic evolution



Threats continue to become more sophisticated as the technical knowledge required to implement attacks diminishes.

# Evolution of the attacker tools in the last years



# Usernames and passwords

Normally, the system administrator will define a convention to establish usernames in a network (to access the OS).

Password rules are important, the control level will be accordingly with the necessary protection level.

A good *security policy* will contain the following:

- Passwords must expire after a specified time period.
- Passwords must contain letters, digits and special characters in order to not be easily broken.
- The standard rule for passwords says that users must not write down the passwords on paper (or other support) and leave it with no supervision, or make it public.
- There must be defined rules concerning password expiry and account blocking (in the moment when an unsuccessful connection attempt was made).

# Other standard security measures

Protection against viruses

Protection against spam

(example - <http://policy.ucop.edu/doc/7000471/AntiSpam>)

# Evolution of Security Threats

“Necessity is the mother of invention.”

- As network security became an integral part of everyday operations, devices dedicated to particular network security functions emerged.
- One of the first network security tools was the intrusion detection system (IDS), first developed by SRI International in 1984.
- An IDS provides real-time detection of certain types of attacks while they are in progress. This detection allows network security professionals to more quickly mitigate the negative impact of these attacks on network devices and users. In the late 1990s, the intrusion prevention system (IPS) began to replace the IDS solution. IPS devices enable the detection of malicious activity and have the ability to automatically block the attack in real-time.

# Evolution of Security Threats

In addition to IDS and IPS solutions, firewalls were developed to prevent undesirable traffic from entering prescribed areas within a network, thereby providing perimeter security.

In 1988, Digital Equipment Corporation (DEC) created the first network firewall in the form of a packet filter.

These early firewalls inspected packets to see if they matched sets of predefined rules, with the option of forwarding or dropping the packets accordingly.

Packet filtering firewalls inspect each packet in isolation without examining whether a packet is part of an existing connection.

# Evolution of Security Threats

In 1989, AT&T Bell Laboratories developed the first stateful firewall.

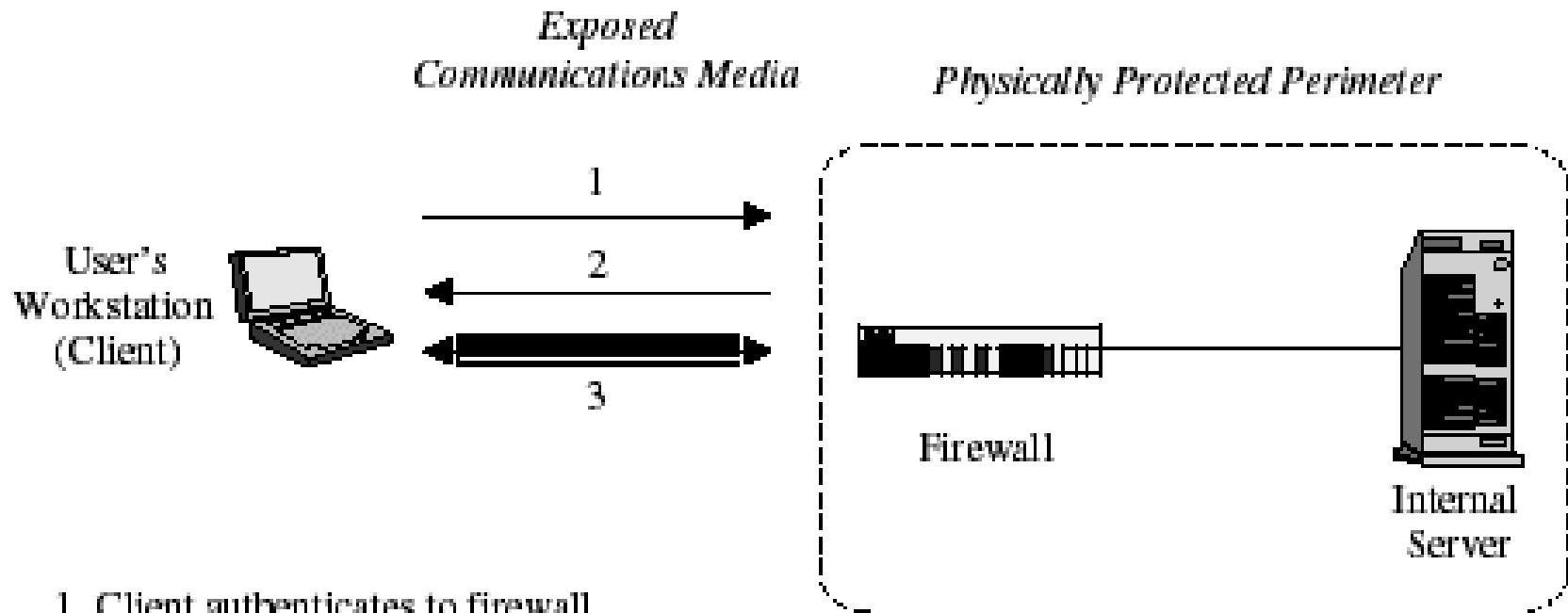
**Like packet filtering firewalls**, stateful firewalls use predefined rules for permitting or denying traffic.

**Unlike packet filtering firewalls**, stateful firewalls keep track of established connections and determine if a packet belongs to an existing flow of data, providing greater security and more rapid processing.

# Virtual Private Network (VPN)

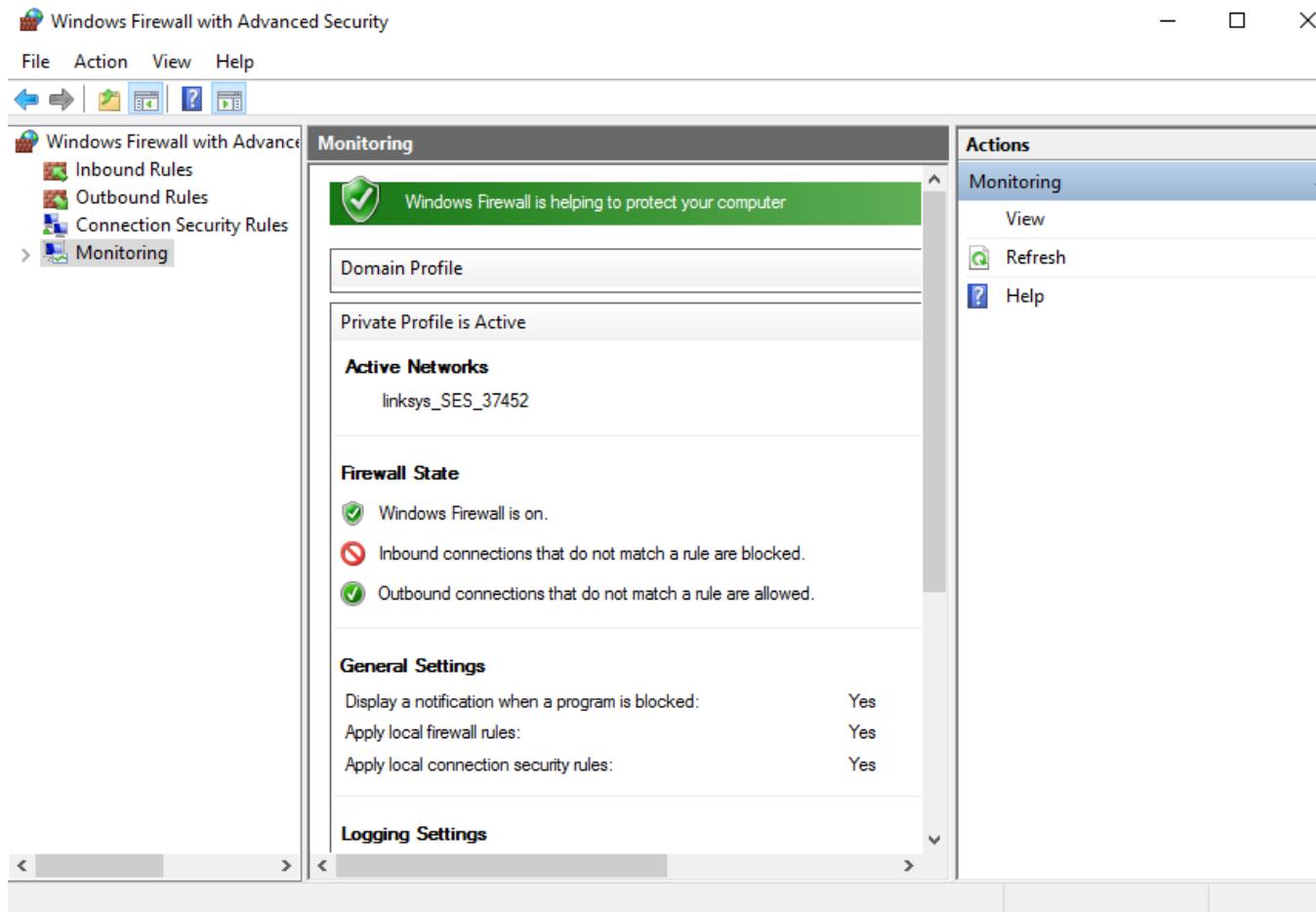
- Link encryption can be used in the sense that users have the “illusion” they are in a private network even when they actually use a public network. This approach is called ***virtual private network - VPN***.
- A firewall represents a device (hardware or software) for access control between two networks or network segments. The firewall is filtering all the traffic between the internal network (more protected) and the external network which is less protected.
- The data transmission mode in a VPN is called ***encrypted communication channel*** or, for short, ***tunnel***.

# VPN Basic architecture



1. Client authenticates to firewall
2. Firewall replies with encryption key
3. Client and server communicate via encrypted tunnel

# Windows 10 firewall



# Linux firewalls

There are several firewalls we may install on Linux. The most well-known firewall is IPtables. Several characteristics are implemented.

Some of the features of IPtables:

- It lists the contents of the packet filter ruleset.
- It's lightning fast because it inspects only the packet headers.
- You can Add/Remove/Modify rules according to your needs in the packet filter rulesets.
- Supports Backup and restoration with files.

IPcop is another Linux firewall, for home and SOHO users.  
More details at: <http://www.ipcop.org/>

# Conclusions

The OS security represents an important part of the network security. When we must implement the OS security we must take into consideration the following:

- Defining an acceptable security policy for the company's network; here is defined what is permitted and what is not permitted in the company's network.
- The security policies regarding passwords must be enforced and respected, including an expiration date, blocking rules and using a combination of letters, digits and special characters, imposing a minimum length for these.

# Conclusions (2)

- The security threats from the Internet are including hackers, crackers, viruses and worms. Although a **hacker** can produce damage, by definition a **cracker** is the one who enters a system to produce a damage or to steal information.
- A virus or a worm can also create substantial damages. A worm is not attaching to files but remains active in memory auto-replicating.
- Good security policies help minimizing threats. The employees and trust users have access to critical information about the network (including passwords) and can facilitate commercial espionage.

# Conclusions (3)

- System administrators must protect the computers against data theft and destruction and “denial of service” attacks. Moreover, the “Distributed Denial of Service (DDoS)” attacks, originating from multiple sources, can be extremely difficult to counter attack.
- In order to keep the OS updated, the security patches and upgrades must be installed permanently as soon as they are available.
- An Internet firewall represents a good protection against the exterior attacks.
- Other important security instruments are the VPNs, IPSs (IPS - Intrusion Prevention System) that can be used in combination with a firewall solution.

# Operating Systems

Course #12  
**Introduction to OS Networking**

Răzvan Daniel ZOTA  
**Faculty of Cybernetics, Statistics and Economic Informatics**  
[zota@ase.ro](mailto:zota@ase.ro)  
<https://zota.ase.ro/os>

# Network architecture model

## What is an “architecture model”?

An *architecture model* offers a general frame of reference for the problems connected to the network communications. Such a model is used not only to explain the communication protocols, but also to develop them.

An architecture model separates the functions assigned to the communication protocols in different layers (easier to manage). Each layer has specific role(s) in the communication process among the network.

## Definitions – basic concepts:

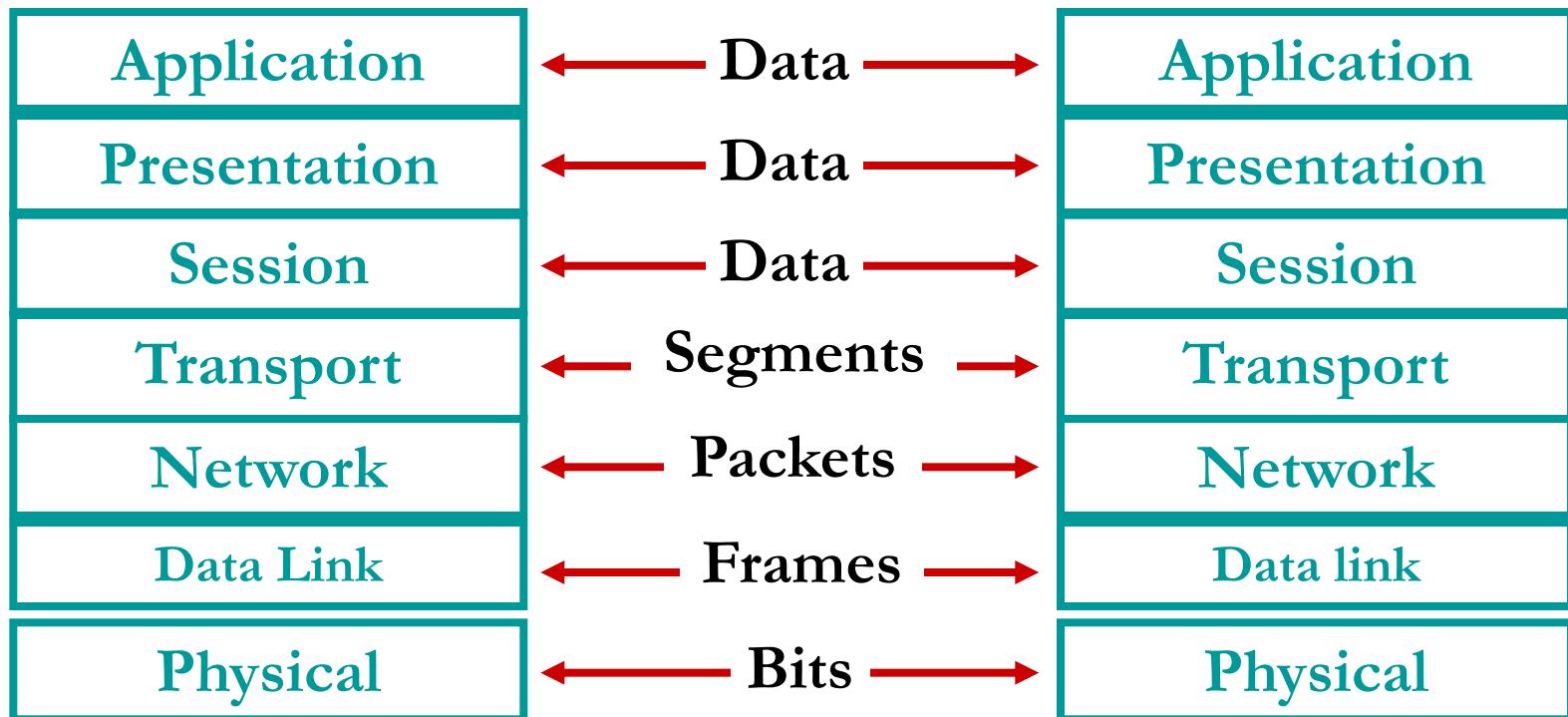
- Networking protocol
- Communication types / data transmissions
- Bandwidth / Throughput / Goodput

# The ISO-OSI Model

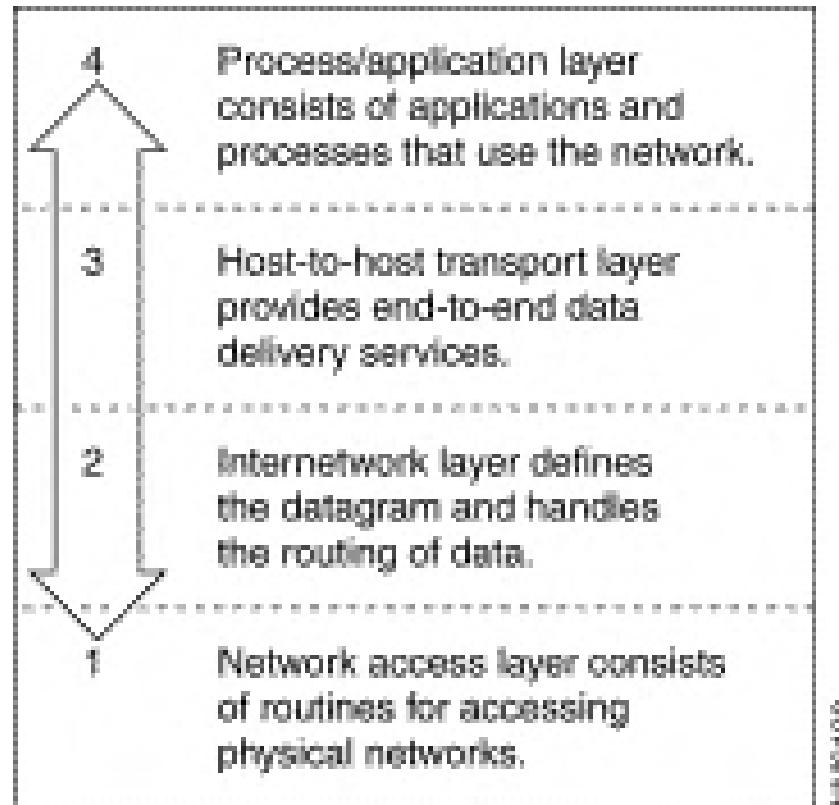
## Why a layered model?

- It reduces the complexity of the problem
- It standardizes the interfaces
- It facilitates modularity
- It assures interoperable technologies
- Accelerates the evolution
- It simplifies and helps the teaching/learning process

# ISO-OSI Model

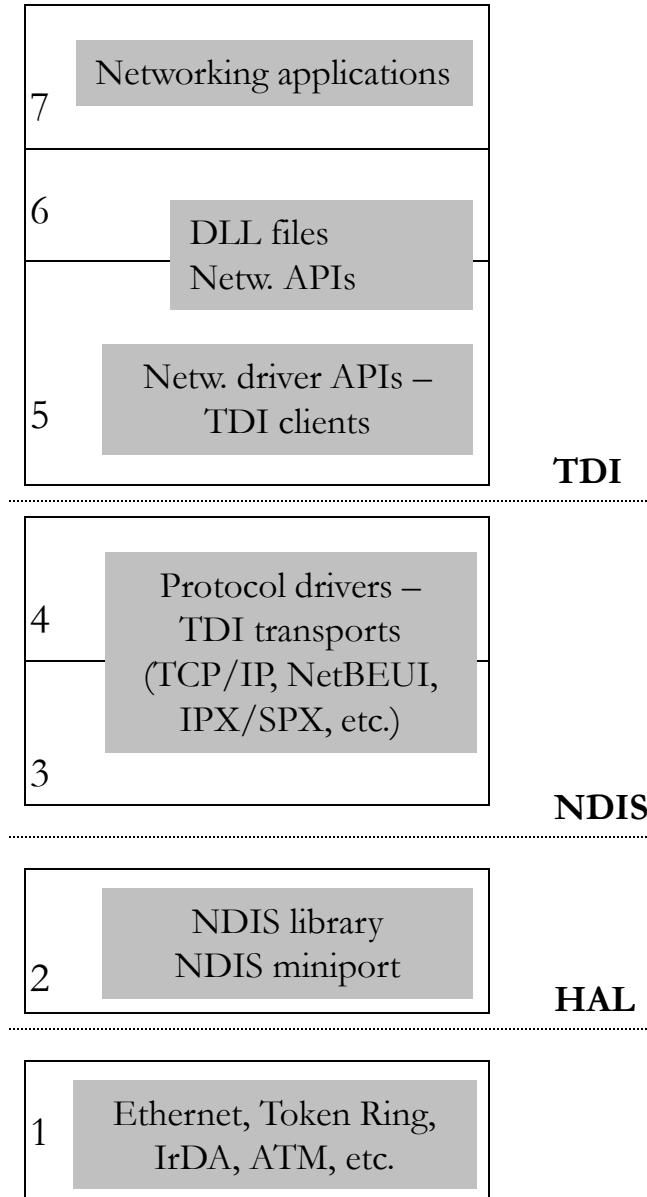


# TCP/IP Model



# OSI – TCP/IP Comparison

OSI	TCP / IP
Application (Layer 7)	Application
Presentation (Layer 6)	
Session (Layer 5)	
Transport (Layer 4)	Transport
Network (Layer 3)	Internet
Data Link (Layer 2)	
Physical (Layer 1)	Network access



Networking components in Windows - OSI model

# Networking components in Windows

- TDI (Transport Driver Interface) transports, protocol drivers  
NDIS (Network Driver Interface Specification)

There are drivers in kernel mode, accepting IRP (I/O Request Packets) packets from TDI clients and processing the requests of these packets.

The TDI transports assure the networking communication by executing message transmitting operations like segmenting and reassembling, sequencing, acknowledgment and retransmission.

- NDIS library (ndis.sys)

The NDIS library exports functions for the TDI transports and support functions for networking adapter drivers.

# Networking components in Windows

- Networking API DLL files

There is an independent way for communication in the network. The networking APIs can be implemented in user or kernel mode.

- TDI clients

There are device drivers in kernel mode implementing the networking API.

- NDIS miniport drivers

Kernel mode drivers responsible with assuring the interface between TDI transports and several network adapters.

The miniport NDIS drivers are communicating with the network adapters using NDIS library functions assuring the interface with HAL functions.

# Operating systems

## Partitioning

### Course #13

# Introduction

A well-thought-out partitioning scheme is vital to optimizing the use of system resources. While it is technically possible to make post-installation changes to partitions, this may be challenging and require system down time.

Therefore, it is important to consider how the system will be used before you begin creating partitions.

Partitioning is necessary in order to optimally use the space that hard drives (hard disks) provide. A hard drive is like a completely empty building, with no internal walls or floors. As it is, such a building wouldn't be useful, but populating the building with walls and floors would provide the structure needed to make use of the building.

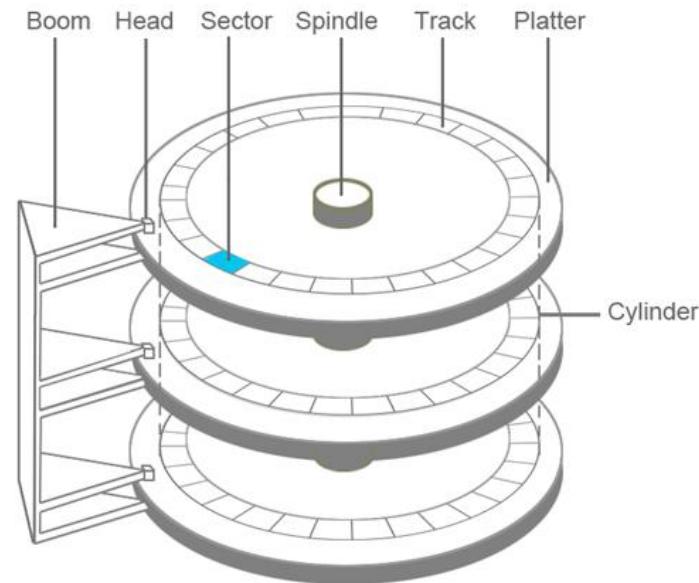
# Introduction

There are three steps in the process of making and using partitions:

1. Divide the hard drive into partitions
2. Format the hard drive with a filesystem
3. Mount the filesystem onto the directory tree

# Introduction

A typical hard disk drive (HDD) consists of multiple *platters*, which store data in magnetic form. *Heads* are able to read and write on both sides of each platter. The platters are connected to the *spindle* and spin at the same speed. The *boom* moves back and forth, allowing each head to read from or write to the same location on each platter simultaneously.

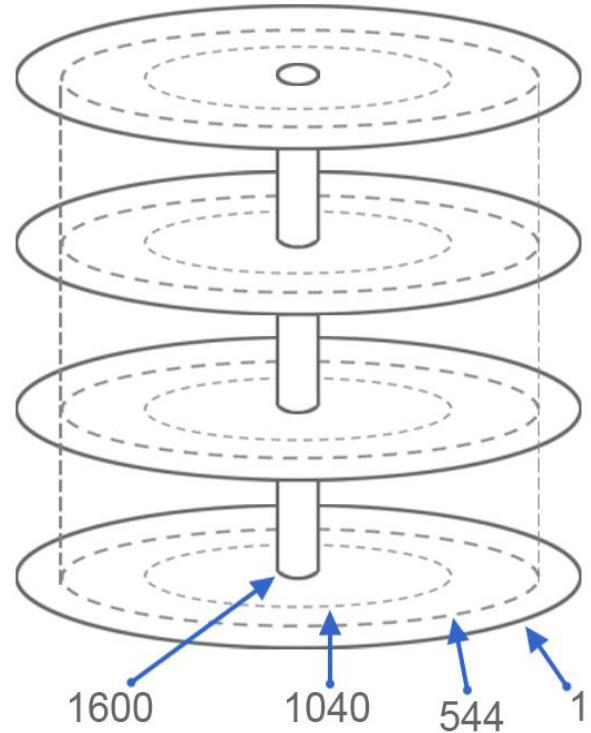


# Introduction

One complete rotation around one side of a platter is called a *track*. The same track on both sides of all of the platters forms a *cylinder*.

A typical hard drive may have thousands of cylinders. Multiple consecutive cylinders can be assigned to a partition.

In the example below, the first partition would be from cylinder 1 to cylinder 544. The second partition would be from cylinder 545 to cylinder 1040. The last partition would be from cylinder 1041 to 1600.



# Why Create Partitions?

As a rule, at least one partition needs to be created.

Regardless of the operating system, whether Linux, Microsoft Windows or Mac OSX, at least one single partition needs to be created because filesystems are stored within partitions, not entire hard disks.

## **Offering support for multiple operating systems**

Some systems may contain Linux as well as Microsoft Windows operating systems; these are called dual boot systems. Because the filesystems that are used with Microsoft Windows are different than Linux, this requires multiple partitions.

## **Home Directories**

A separate partition for the user home directories is common and typically provides many advantages, including:

- It is easier to backup or restore an entire filesystem than it is to manage each user's home directory. Recall that each partition has a separate filesystem.
- When home directories are separated from the root filesystem, upgrades of the operating system can be performed much more safely and efficiently.
- Filesystems can have features enabled on them, such as disk quotas. A disk quota allows the administrator to limit how much space a user uses in a filesystem. Being able to create a disk quota for the home directories that doesn't affect the rest of the operating system can be an advantage.

# Why Create Partitions?

## Common Writable Directories

Some directories, such as the ***/tmp*** and ***/var/tmp*** directories, are world writable. This means that any user can store files in these directories without any limitations by default. Unfortunately, that can cause problems if these directories are not placed on separate filesystems.

If the ***/tmp*** and ***/var/tmp*** directories are not mount points, then files placed in these directories will go on the same partition as the ***/*** filesystem. If a user creates a very large file, then he could end up filling up the entire ***/*** filesystem. This would make the operating system unusable by all other users (except the root user for whom some space is reserved).

Separating the ***/tmp*** and ***/var/tmp*** directories allows the administrator to enable disk quotas to limit how much space can be used by each user in those directories. Even if no quotas are put into effect, having directories on a separate partition means if a regular user fills up the partition(s) ***/tmp*** or ***/var/tmp*** are located on, the root filesystem or other critical filesystems are not affected.

# Why Create Partitions?

## Security

Using separate partitions may be done for security. Having files in a separate partition is safer than if they were a part of a larger root filesystem because there is less risk of corruption.

When using a separate filesystem, there will be separate *inodes* and data blocks, so there is less chance that they might be overwritten.

Sometimes a collection of files, perhaps old databases, need to be stored but shouldn't be modified. By placing them on a separate filesystem, that partition can be mounted as read-only to prevent any alteration.

## Heavily Active

If running out of space is a concern due to heavy activity within a directory, it may be desirable to have it mounted on a separate partition. This is often the case for directories like the **/var/log** directory where the system adds log file data on a regular basis.

When system processes (as opposed to regular users) write to the disk, quotas are not usually used to limit the space. Instead, these types of directories are mounted on separate filesystems so that if they fill up then it won't fill up the / filesystem. While this may be bad because log files aren't generated, it is better than filling up the / filesystem and making the operating system unusable for regular users.

# Partition Naming

In order to distinguish one partition from another, each partition is given a unique name. Recall that everything in Linux is treated as a file, so names of devices such as drives and partitions are stored in the /dev directory.

There are two different types of drive interfaces:

/dev/sd\* - Drives that start with sd are either SATA (Serial ATA), SCSI (Small Computer System Interface) or USB drives. The first sd drive is called /dev/sda, the second is called /dev/sdb, etc.

/dev/hd\* - Drives that start with hd are PATA (Parallel ATA), also known as IDE (Integrated Drive Electronics) drives. The first hd drive is called /dev/hda, the second is called /dev/hdb, etc.

How can I see if my disk is a SSD or a HD?

Use: cat /sys/block/sda/queue/rotational

1 – means HD

0 – means SSD

# Partition Naming

Partitions are then given names based on the drive they reside on. A number is added to the end of the drive name to distinguish one partition from another.

For example, the partitions located on *sda* would be named *sda1*, *sda2*, etc. Partitions located on drive *sdb* would be *sdb1*, *sdb2*, etc.

Test the following commands:

```
cd /dev ; ls sd*
```

```
df -kh
```

```
sudo fdisk -l | more
```

```
top
```

```
htop
```

# Partition Limitations

Historically, most PCs are limited by their compatibility with a Master Boot Record (MBR) as to the number of partitions that they can use. Recall that the MBR is usually contained within the first sector or 512 bytes of the disk and contains a boot loader program as well as the partition table. The partition table contains the definition of each partition on the hard drive, including which cylinder it starts and ends on and how big it is.

Traditional disks using MBR partitioning can have a maximum of four primary partitions . In the next slide, for example, on a traditional SATA drive, four partitions can be created with device names of /dev/sda1, /dev/sda2, /dev/sda3, and /dev/sda4:

# Partition Limitations



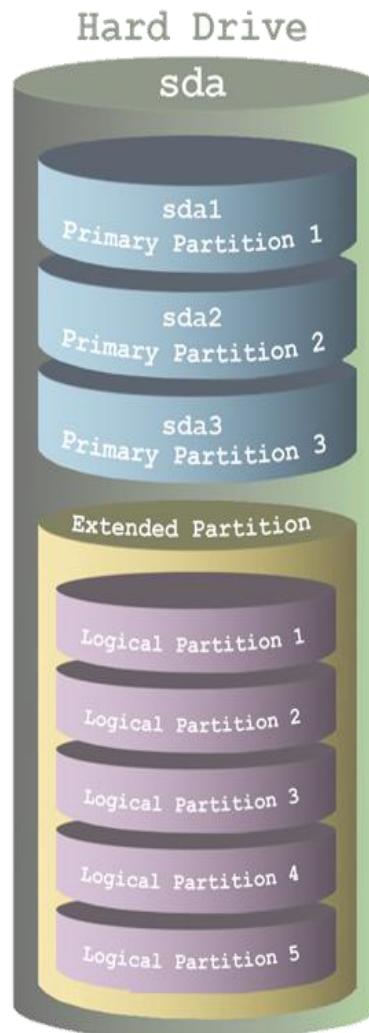
# Partition Limitations

Modern hardware allows the administrator to make one of the four partitions an extended partition.

An extended partition acts like a container for additional partitions called logical partitions.

The /dev/sda4 partition is an extended partition. The only thing that is placed in this extended partition are additional partitions, logical partitions.

Depending upon the type of hard disk and kernel configuration, Linux can access a maximum of either 15 or 63 total partitions when extended partitions are used.



# Partition Limitations

Recently, administrators have begun to use a different type of partitioning technology known as Globally Unique Identifier (GUID) designed to replace traditional MBR partitioning.

This new partitioning is available if the system supports the Unified Extensible Firmware (UEFI).

The GUID Partition Table (GPT) supports much larger disks with up to 9 ZB in size. Extended and logical partitions are not used with GPT; instead all partitions are the same and GPT supports a maximum of 128 partitions per disk.

In addition to MBR and GPT partition tables, there are other methods for dividing up the disk, including Berkeley Software Distribution (BSD) Unix Labels, Apple Partition Maps, and others.

# GUID/GPT (Globally Unique Identifier - GUID Partition Table)

## GPT vs. MBR

Characteristics	MBR	GPT
Maximum no of partitions	4 primary (or 3 + 1 extended)	128 partitions (on Windows)
Disk – max dimension	2 TB	Theoretically: 9.4 ZB (zettabytes)
Redundancy	No	Yes – it has backup copies of the table
Error checking	No	Yes – with CRC32
BIOS compatible	Yes	No (only UEFI compatible)
Needs special partition	No	Yes –EFI partition (FAT32)

# The filesystem

In order to place files and directories on a partition, a filesystem needs to be created. This is accomplished by formatting the partition.

A filesystem provides the operating system with a way to organize the files and directories, as well as store information about files, such as the permissions of the files, the owner of the files and the file type.

Each file will have an entry in the filesystem database, similar to a library catalog, although the database is often referred to as a table. Metadata about the file will be contained in this entry, which is everything about the file except for its contents.

The entry also includes a unique identification number for the file, called an *inode* number and pointers (or links) that inform the operating system where in the filesystem (where on the hard drive) the file's data is stored.

# Common filesystem types

- ***ext2: Second Extended Filesystem***

Advantages: Works well with small and solid-state disk filesystems.

Disadvantages: No journaling capability, making it susceptible to data loss in the event of power loss.

- ***ext3: Third Extended Filesystem***

Advantages: Can be upgraded from existing ext2 filesystem without data loss. This filesystem performs journaling, which allows for data recovery after a crash.

Disadvantages: Writes more to disk than ext2 because of journaling, making it slower. Does not support very large filesystems.

- ***ext4: Fourth Extended Filesystem***

Advantages: Support for very large disk volumes and file sizes. Can operate with or without a journal. Backwards compatible with ext3 and ext2.

Disadvantages: Not a huge improvement over ext3. No dynamic *inode* creation.

- ***reiserfs: The Reiser Filesystem***

Advantages: The first journaling filesystem for Linux. Works efficiently with small files.

Disadvantages: Development of this version has ceased with its successor<sup>7</sup>, Reiser4 proceeding slowly without help from the founder.

# Common filesystem types

- ***xfs: Extents Filesystem***

Advantages: Works very efficiently with large files. Compatible with the IRIX operating system from SGI. Announced to be the default filesystem for RHEL 7.

Disadvantages: The filesystem cannot be shrunk.

- ***vfat: File Allocation Table***

Advantages: Supported by almost all operating systems. Commonly used for removable media.

Disadvantages: Unable to support very large disks or files. Microsoft's intellectual property claims.

- ***iso: ISO 9660***

Advantages: The International Organization for Standardization standard for optical disc media that is supported by almost all operating systems.

Disadvantages: Multiple levels and extensions complicate compatibility. Not designed for rewritable media.

- ***udf: Universal Disc Format***

Advantages: Designed to replace ISO 9660 and adopted as the standard format for DVDs by the DVD Consortium.

Disadvantages: Write support is limited to support revision 2.01 of the standard.

# Summary of key terms

**Filesystem:** A database system used to organize files.

**Table:** The database where the filesystem stores its information.

**Metadata:** File attributes like ownership, timestamps, data block locations, etc.

**Inode:** A unique number given to each file on the filesystem.

# Summary of key terms

## *About filesystem journaling:*

As files are changed, metadata is not initially stored on the hard drive, but rather in memory. This speeds up the filesystem as writing these changes to the hard drive for every single file change would result in a lot of hard drive writes.

At specific intervals, this metadata is written to the hard drive in large chunks; this process is called ***syncing***. Normally this poses no problems, but if the system were to crash or lose power before a sync occurred, all of that metadata would be lost.

A filesystem recovery program, like the ***fsck*** command, can often resolve these problems, but this can take quite a bit of time for large filesystems.

A journal limits filesystem recovery. For every change made to a file, a journal entry is stored on the hard drive. While this does increase filesystem writes, a journal entry has much less impact on hard drive performance than writing the metadata directly to the hard drive for each file change.

In a nutshell, a journal filesystem aids in recovering corrupted filesystems while allowing for reduced hard drive writes.

# Linux Filesystem Components

The term "Linux filesystem" often refers to the ext2/ext3/ext4 family of filesystems. While there are differences between these filesystems, they are similar enough when it comes to core filesystem components:

- ***Superblock***: At the beginning of the filesystem is an area called the superblock. This area is used to store important information about the filesystem, including the size of the filesystem, the type of filesystem and which data blocks (where file data is stored) are available. The superblock is a key component of the filesystem; a corrupted superblock would make the filesystem inaccessible.
- ***Group Block***: The filesystem is divided into smaller sections called groups. The group block serves to hold data about the group. Each group block also contains a backup copy of the superblock.
- ***Inode Table***: Each file is assigned a unique inode number for the filesystem. This *inode* number is associated with a table that stores the file's metadata.

# Physical vs. Virtual Filesystems

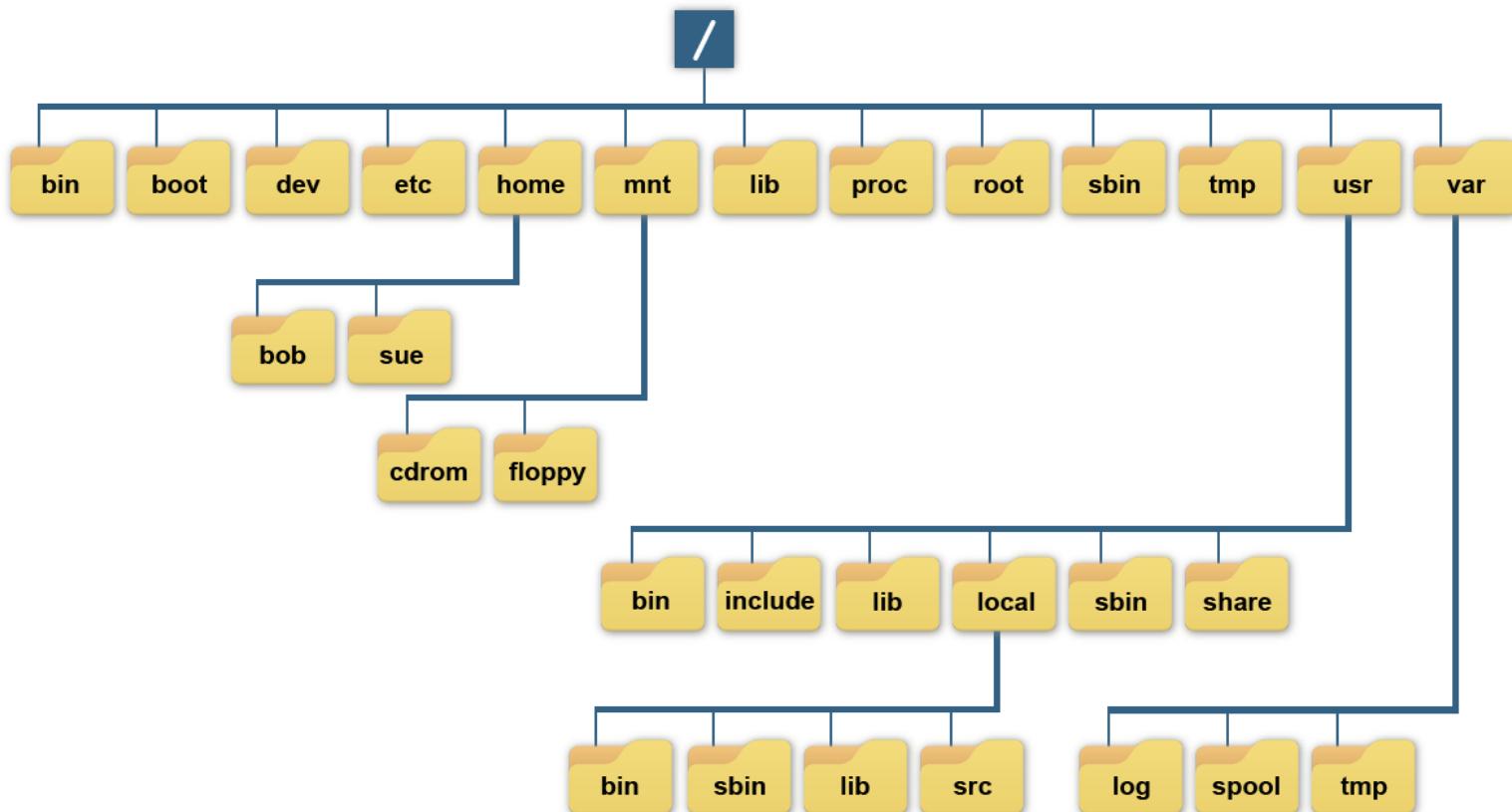
In case of Windows we use drive letters. (see disk management as an example)

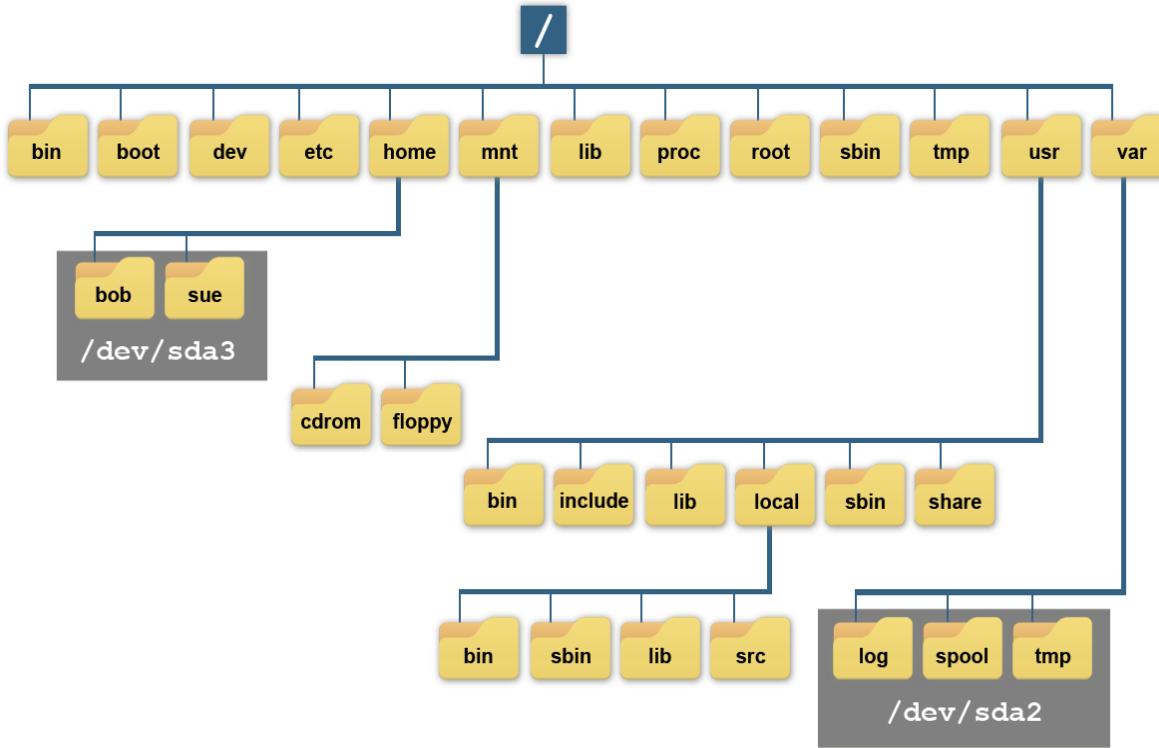
In Linux there are no drive letters. Instead, each partition is assigned a device file name, like:

- `/dev/sda1`, `/dev/sda2`, etc. for partitions on SATA, SCSI and USB devices
- `/dev/hda1`, `/dev/hda2`, etc. for partitions on PATA/IDE devices

Users do not access the files that are stored on these partitions by directly using these device file names. After all, it's not convenient to remember what files are located on `/dev/sda1` versus what files are stored on `/dev/sda2`. Instead, these device files are merged into the directory structure by mounting the partition's filesystem under a directory tree.

# Virtual Filesystems



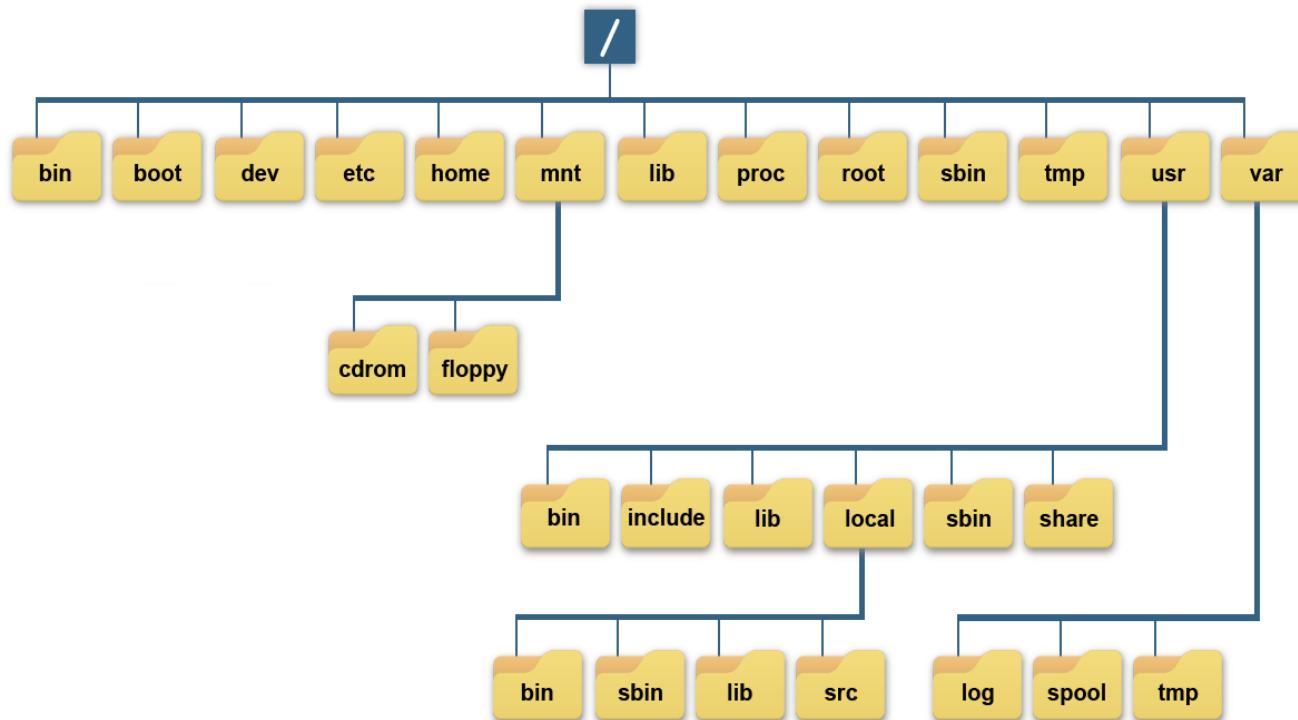


All of the files under the /home directory are really stored in the filesystem that is on the /dev/sda3 partition. All of the files under the /var directory are stored in the filesystem that is on the /dev/sda2 partition. This makes /home and /var mount point directories. The / directory is also a mount point directory, likely for the filesystem on the /dev/sda1 partition.

The process of placing a filesystem under a mount point is called *mounting*. This is accomplished either automatically at boot or manually with the *mount* command.

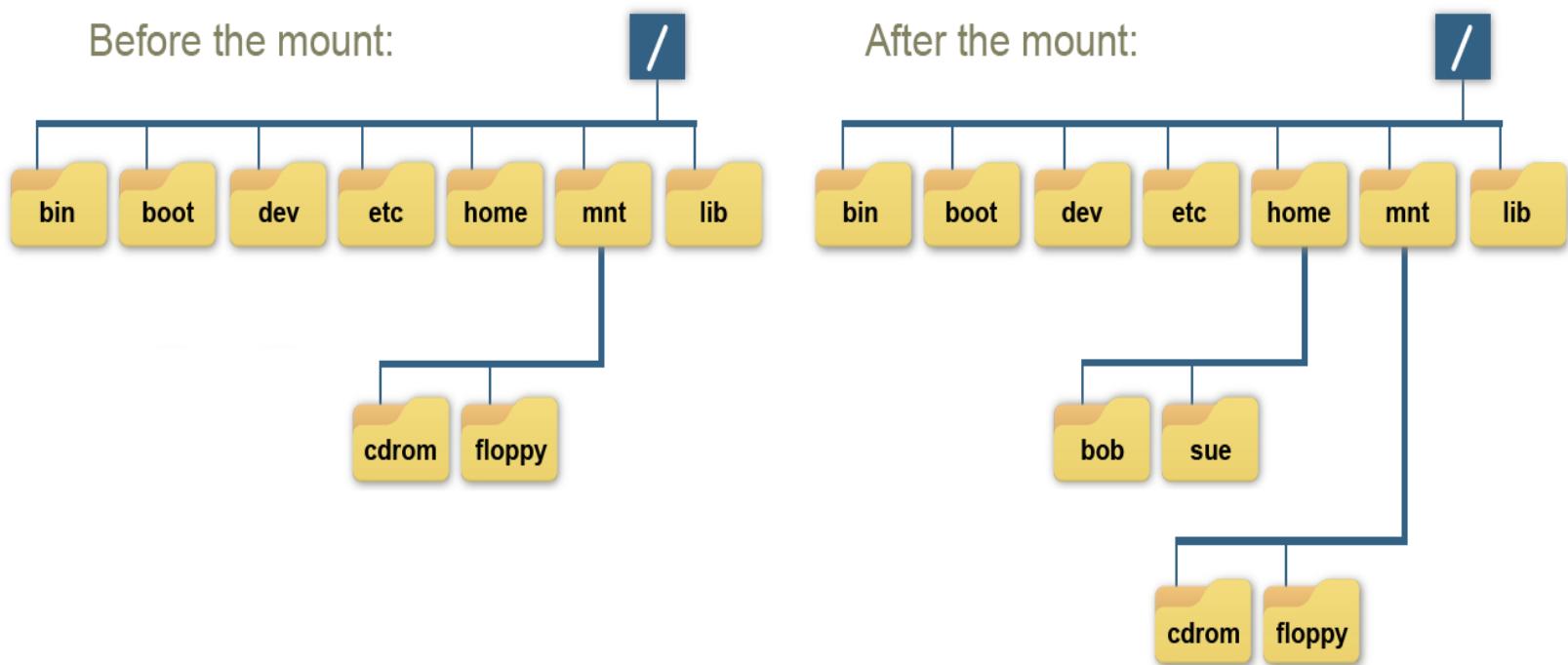
# Virtual filesystems

If the administrator were to ***umount*** the `/dev/sda3` filesystem from the previous example, then the virtual filesystem would look like the following:



# Virtual filesystems

Notice that now there is nothing under the /home directory. This is common because a mount point directory should be empty, as anything contained in a mount point directory will be inaccessible after the unmount. The /dev/sda3 filesystem is mounted under the /home directory, then the bob and sue directories would again be available.



# Key terms summary

## **/ (root) filesystem**

The filesystem that is contained on the same partition on which the root directory is located, and it is the filesystem on which all the other filesystems are mounted as the system is booted up.

## **/boot filesystem**

The directory containing everything required for the boot process except for configuration files not needed at boot time.

## **/home filesystem**

Typically the parent directory for individual user home directories. Each users personal files may be stored within the user subdirectories along with user specific configuration files.

## **/var filesystem**

The directory containing variable data like system logging files, mail and printer spool directories, and transient and temporary files.

## **MBR**

The Master Boot Record (MBR) represents the first 512 bytes of a storage device. It contains an operating system bootloader and the storage device's partition table.

# Key terms summary

## **fsck**

Command used to check and optionally repair one or more Linux file systems. If no filesystem is specified with the command, **fsck** will check all filesystems in the */etc/fstab* file by default.

## **mount points**

A connection between a partition's file system and a folder allowing you to access the files on the partition.

## **parted**

A disk partitioning and partition resizing utility. It allows an administrator to destroy, create, resize, move, and copy ext2, linux-swap, FAT, FAT32, and reiserfs partitions.

## **partitions**

The logical division of the available space on a hard drive into one or more sections that can be accessed independently of one another.

## **swap space**

Used when the amount of physical memory (RAM) is full. If the system needs more memory resources and the RAM is full, inactive pages in memory are moved to the swap space.