3. Operating Systems

Informática

Ingeniería en Electrónica y Automática Industrial

Raúl Durán Díaz Juan Ignacio Pérez Sanz
Álvaro Perales Eceiza

Departamento de Automática
Escuela Politécnica Superior

Course 2014–2015

Contents

Objetives

- Understand what an Operating System -OS- is and its utility.
- Describe its different functional parts and their roles.
- Understand what a *process* is.
- Understand what a *file system* is.
- Learn to use the basic Linux commands, the standard Input/Output, and how to concatenate different commands.

What is an *Operating System*?

Definition

An **Operating System** -OS- is a program that manages the hardware resources of a computer providing services for other programs and applications. It works as an intermediate between the user and the computer hardware, facilitating its use making it more efficient.

OS functions

The main OS functions are:

- To facilitate the use of the computer and, in general, the communication computer-user
- To manage and assign hardware resources (processor, memory, peripherals) to processes, and deal with conflicts.
- To manage and maintain files in permanent memory devices (hard drive..)
- To protect data and programs (specially important in multiuser systems)
- To serve different users.

Extended Machine

The OS construct another level of abstraction presenting the user a ≪virtual or extended machine≫ with a much simpler use.

- It hides fine working details the user does not need to know (e.g. how to manage the read/write head of a magnetic disc)
- It provides an extra set of instructions, the **System Calls**
  - They can be used by other programs or by the programmer working as an interphase with the OS
  - They perform useful tasks, such as
    - Manage processes (create, cancel,...)
    - Manage files (create, open, close)
    - Manage memory (load, move or free blocks of memory)
  - The OS converts system calls in machine instructions (the only ones that the computer executes); they are as well ≪virtual≫

Extended Machine

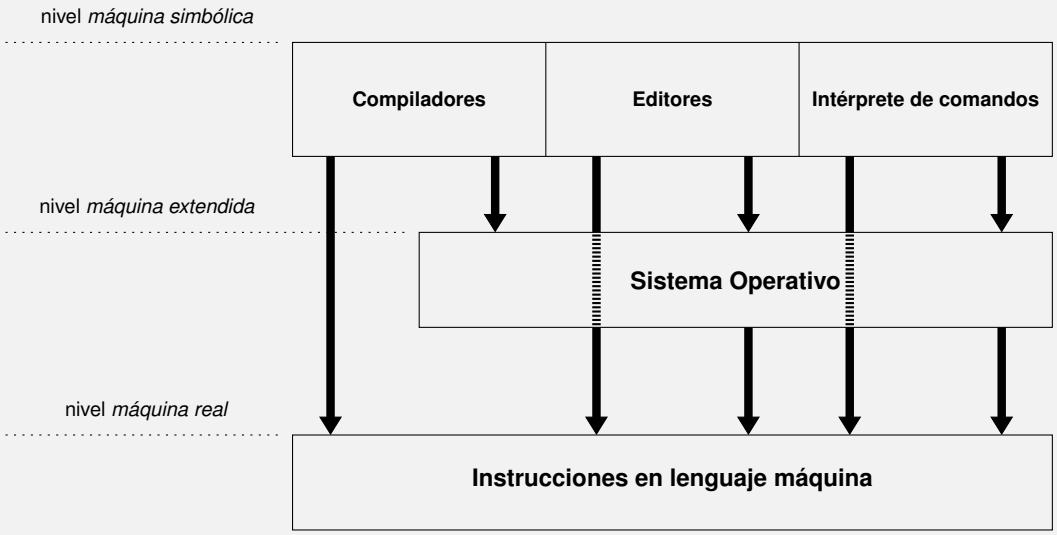


Figure: Different levels of abstraction -machines- in a computer

Resources Administrator

- It assigns hardware resources (processor, cores, memory) to each process or user
- Resolve conflicts among users

Example

If various programs/users want to print at the same time, OS assigns priority and arrange them sequentially.

Example

While a process is waiting from data coming from hard disc, the OS can allow another process to use the processor

Processes

Definition

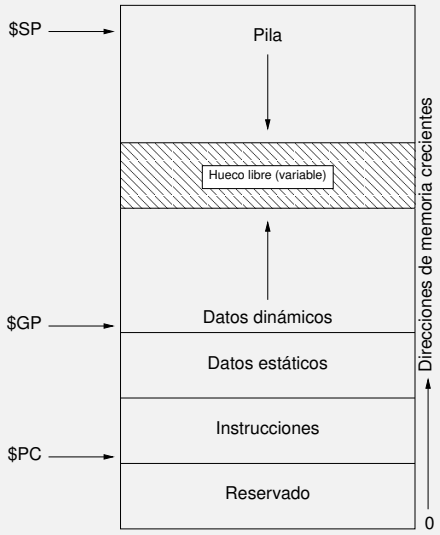
A *process* is a program in execution. It is the main executive element in the computer

- We can see a process as the dynamic instance of a program. Therefore it may happen that various processes execute the same program (e.g. with different input).
- A computing system working is a set of one of more processes in execution according to a temporal planing established by the OS or the programmer.

Elements of a process

- A process can be described as:
  - Memory space structured in sections
  - Processor registers.

Process structure



Un proceso necesita manejar registros de la CPU:

- contador de programa `$PC`
- puntero de pila `$SP`
- registro de estado `$PSW`

Figure: Structure of a process allocated in memory

Process Features

- A process can just be created by another process.
- Any process (parent) can create other processes (child).
- Two system calls are used to create new processes:
  - `fork`: system call by which a process creates a copy of itself;
  - `exec`: Replace the contents of a running process with another program code.

The general way of launching a new program is first to `fork` an existing process to create a new one, and then `exec` the new program in it (i.e. load it in memory and execute it)

Process Tree

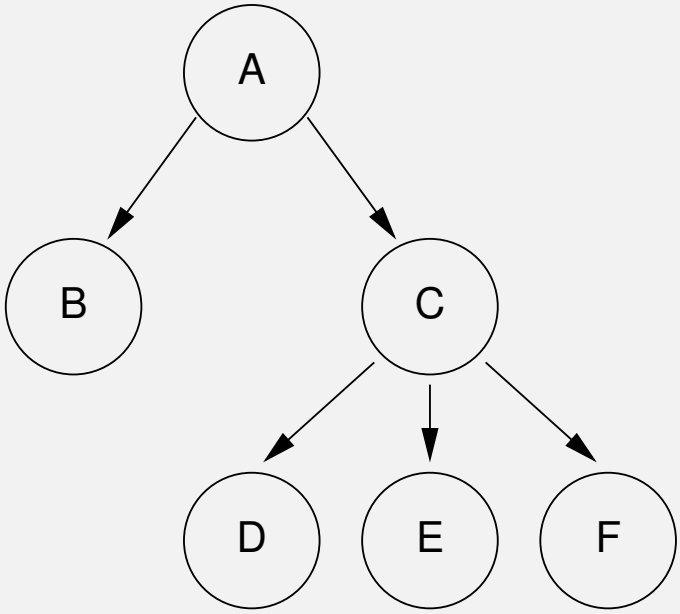


Figure: Process Tree

System (*Boot*)

- When turning on the computer
  - The program counter PC points to an address at ROM where there is a small code (BIOS -Basic Input/Output Services- in IMB PCs) that becomes the first process to be executed. This process initializes and tests the system hardware components and starts a bigger process, the
  - *Boot Loader* that loads first the OS kernel and subsequently the rest of the OS necessary to start main computer functions.

Executable File Structure

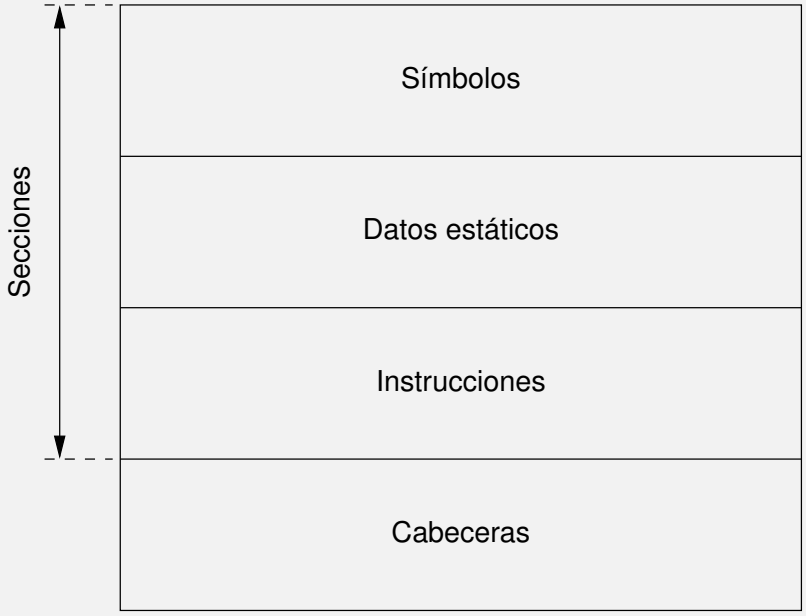


Figure: Executable File Structure

Process Scheduling

- The processor is the most valuable resource in the computer.
- It can just execute one process every time.
- When a process in execution is blocked (e.g is waiting for data coming from disc), the *Scheduler* suspends it and selects another process that can be executed, so to maximize the processor use.
- Each process is described by the memory space it uses and by the content of the registers in the moment it is suspended.

Scheduling: monoprogramming

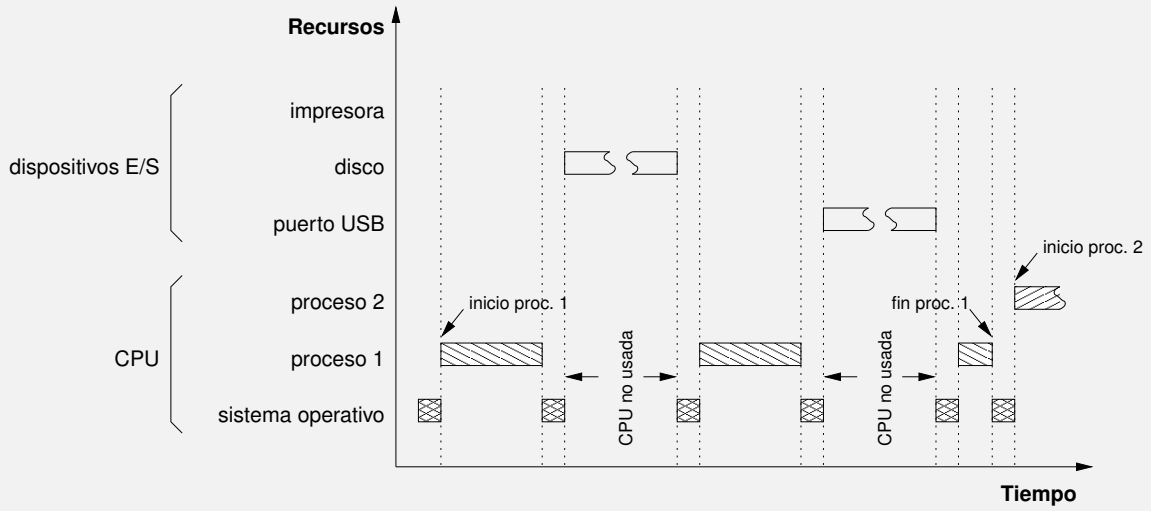


Figure: Scheduling with monoprogramming

Scheduling: multiprogramming

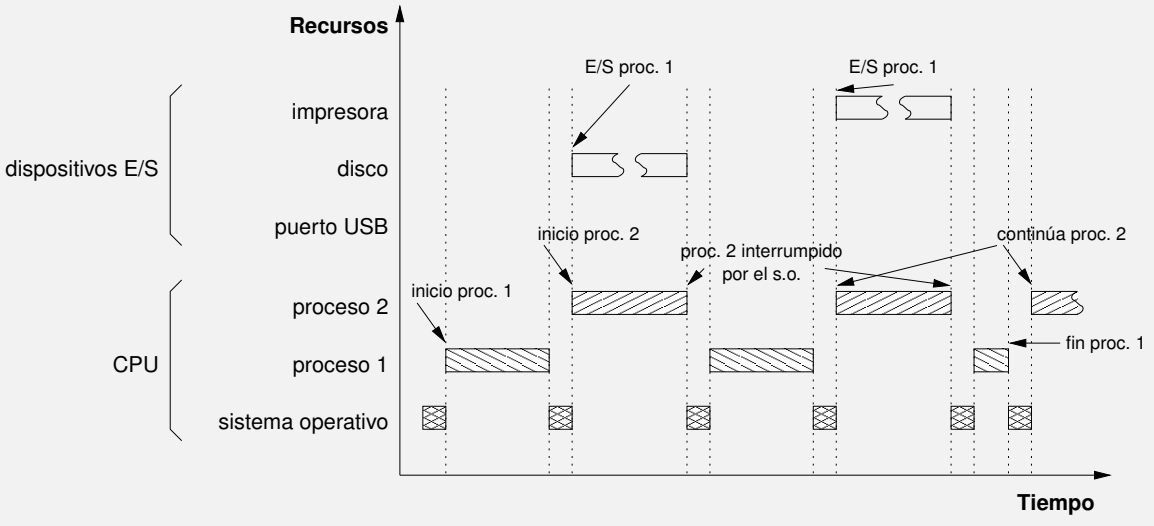


Figure: Scheduling with multiprogramming

Scheduling: Round-robin

- Usually there are many processes waiting to be executed, so the scheduler must apply some rule to distribute the processor time.
- The *Round-robin scheduling* consists of assigning time intervals T or *quantums* to each process and to rotate among them
- The critical decision is T: short *quantums* implies to loose too much time in process change; and long ones may imply too much waiting for users.

Scheduling: Process state diagram

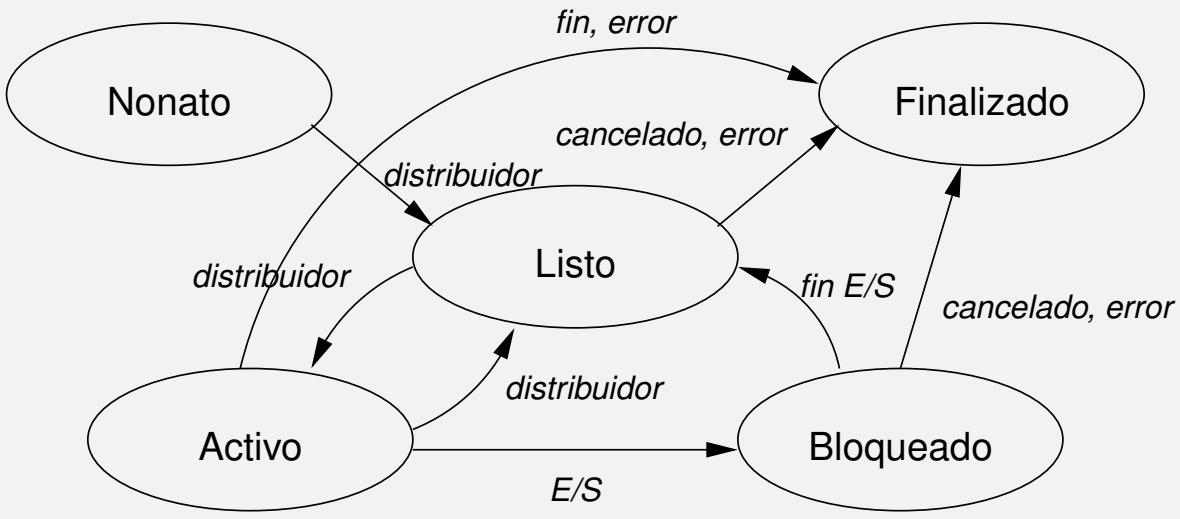


Figure: Process state diagram (New, Ready, Running, Blocked, Terminated)

Virtual Memory

- Basically it consists of using secondary memory (hard disc) as if it was principal memory (RAM)
- The OS offers a process the possibility of using much more memory, the *virtual memory* than the real one, the *physical memory*
- This allows the system to execute much bigger programs, and more processes at the same time
- The OS construct a *page table* for each process to traduce virtual addresses to physical ones

File System

Definition

- It is the part of the OS that controls how data is stored and retrieved.
- It consists of dividing information in pieces, the *files*, giving them names and positions in memory, so that the information is easily identified and accessed.

Example

There are different files systems. Each OS can manage different ones. Most common actually are: `FAT32` (Windows, Linux), `ext4` (Linux), `NTFS` (Windows), `HFS+` (Mac OS X), `ISO9660` (for optical discs).

System calls to manage files

- The main system calls to manage files are:
  - `create/open`
  - `read/write`
  - `close`
  - `unlink`: To delete files (removes the file name from the file system)

Note

They are OS-dependent (i.e. different for different OS), but file-system independent.

Directory Tree

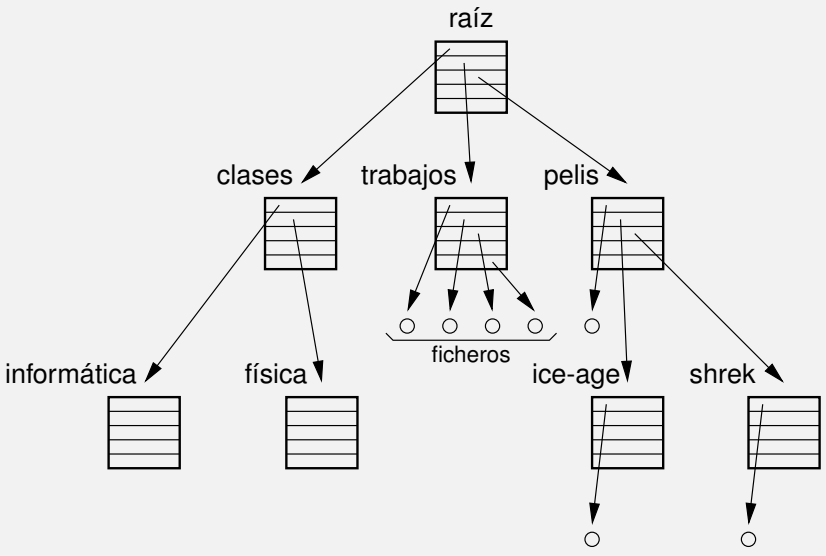


Figure: Example of a directory Tree in UNIX/LINUX

Devices and files in UNIX/LINUX

- UNIX/LINUX manages devices (e.g. hard disc or pen-drive) as files.
- They are usually placed in the `/dev` directory .
- System calls are the same than for the ordinary files.

Mount point

- When installing a new device, the file system uses a *Mount Point* that it is just a directory in the system where the new devices is *mounted*.
- The user sees that the directories and files in the device appear in a previously empty directory of the file system.

User Interface

- The *User Interfaces* is a part of the OS that allows interaction with the user through peripherals (keyboard, screen...)
- The most common UI are:
  - *Command Line Interface*: Orders are typed with the keyboard in a terminal, the OS interpret those commands looking for the programs to execute and directing results to output devices.
  - *Graphical user interfaces* (GUI) (e.g Windows). Mouse and keyboard are use to select icons in the screen and the output is graphical output on the computer monitor.
  - *Touchscreens*. Input touching a screen (tablets, mobiles...)

Shell Program

It is a very complete Interface program that works as a command-line interface. The user types commands and the `Shell` execute them. Some of its features are:

- Standard Input/Output;
- Command concatenation, ≪pipeline≫;
- Environment variable (PATH, HOME...)
- Some basic commands:

  `ls -l`, `cd`, `pwd`, `mkdir`, `rmdir`, `ps`...