# 1.1 System architecture

## 1.1.1 The basic architecture

The GNU/Linux system was born, like all UNIX systems, to be multitasking and multiuser.

The architecture of the GNU/Linux system, like all UNIX system, is based in the separation between:

- Kernel: that is the nucleus of the system, to which hardware resource management is required (CPU, memory, peripherals);
- Processes: that are the program execution unit, ranging from basic system commands, to applications, etc. The kernel's job is to run many processes at the same time efficiently.

This leads to a split between:

- Kernel space: which is the environment in which the kernel runs;
- User space: which is the environment available to users, where processes are run.

This division shows that only the kernel can directly access hardware resources, while the programs are executed in a protected manner, in a virtual environment, but which can access the resources made available to the kernel thanks to standardized functions called **system calls** (standard: POSIX.1).

System calls and other basic functions are collected in a single library which is the **GNU C Library** (**glibc** for short).

The only program that is then executed is the kernel. A part of the kernel called **scheduler**, is responsible for managing the processor time, and is therefore responsible for deciding which process must be performed at a given moment (multitasking).

A second part, the **VM**(or **Virtual Memory**), deals with managing the use of available memory. Virtual memory is what makes each process see its own address space which is then remapped into actual physical memory.

The last part of the kernel, the **drivers**, is a set of modules that allow the management of various peripherals, providing access for programs.

The consequence of the division between user and kernel space is that in this way it is not possible for a single program to disturb the action of another program or of the kernel itself.

## 1.1.2 The functioning of the system

Upon power-up, a BIOS program is executed which, after checks, performs the system boot procedure. PCs are loaded with the **bootloader**, which in turn retrieves a kernel image from the disk, which is loaded into memory. Once the kernel is loaded, it scans for available devices, reads the disk partition tables, mounts the filesystem on which the root directory is located, and the first process will start. Traditionally this process is called **init**, and it is the program that takes care of starting all the processes that allow you to use the system.

The main group of programs outside the kernel derive from the Free Software Foundation's GNU project.

Although all programs are treated equally by the kernel, not all of them are equally important (see init). This leads to another characteristic which is that a process can in turn launch new ones, whereby it is said that the first process is the **father** of the others, who are called **children**.

## 1.1.3 Some linux specific features

Although Linux is the most popular, there are other UNIX-like operating systems (BSD, Solaris, IRIX, etc ...) that are inserted into two main branches, those derived from:

- AT/T: called **SysV**;
- Berkeley: called **BSD**.

Linux does not belong to either of these two branches, as it has been rewritten from scratch.

Another feature of Linux is that of being **modular**, it can therefore be extended to the system additional pieces (**modules**) that allow you to expand the system.

A third peculiarity of Linux is that of the **Virtual File System** (or **VFS**), which is that the disk space on which the data files are kept is organized in what is called a **filesystem**. The raw disk space is normally divided into contiguous sectors of fixed size, but within the system this is organized in such a way as to allow the immediate retrieval of the information stored on these sectors, even when these are scattered here and there on the disk. ; this results in what the user sees as a single file.

What distinguishes Linux is that the interface for reading the contents of a filesystem has been completely virtualized, so by inserting special modules into the system it becomes possible to access different types of filesystems with the same interface.