

# Linux Basics

## ***Basic Flow of Operation***

Basically an **operating system** (OS) is a piece of software running on a computer which provides:

- a nice interface to hardware,
- management of computer resources, where the two most important resources are memory and processor execution time, and
- a means to run several programs concurrently, allowing multiple users at the same time.

The user can ask the OS to run several programs at the same time. Then the OS will schedule this work in such a way that each program gets some percentage of the execution time of the processor.

The OS can be asked to perform certain tasks by, for instance, typing a letter at the keyboard or clicking with a mouse. This will generate an interrupt and with these interrupts you can drive the OS to do what you want.

Thus the OS is driven by interrupts, which can be considered as the heartbeats of a computer:

- All programs running in an OS are scheduled by a scheduler which is driven by timer interrupts of a clock to reschedule at certain times.
- An executing program can block or voluntarily give up the CPU in which case the scheduler is informed by means of a software interrupt (system call).
- Hardware can generate interrupts to interrupt the normal scheduled work of the OS for fast handling of hardware.

In an OS the specific part of software dealing with interrupts is called the *kernel*, which is the core of an OS. In general we can say that the kernel has the highest priority of getting executed on the computer because it is always the first software executed when an interrupt happens.

The kernel is also executed in a higher privileged mode than normal user programs, because for safety and security reasons we do not want a program to access the hardware of the computer directly.

In short, we list the basic flow of operations for interrupts:

- when an interrupt arrives, it is first given to the kernel
- depending on the type of interrupt, the kernel does certain actions:
  - timer interrupt: it calls the scheduler to reschedule the programs (Note: it may also start a new program on request)
  - hardware interrupt: it deals with data from or to the hardware on the specific hardware bus
  - software interrupt: it executes some code on behalf of some program which does not have the privilege to do it herself. Note: before doing it, it checks if the program is authorized to call this request.

- finally when the kernel is done serving the interrupt it will switch to running the scheduled program in lower privilege mode

## **Memory**

[http://en.wikipedia.org/wiki/Memory\\_protection](http://en.wikipedia.org/wiki/Memory_protection)

### **Segmentation**

Segmentation divides the memory of a computer into segments. Each segment can get a different privilege level.

### **Paging**

In paging, the memory address space is divided into small pieces, called pages. Using a virtual memory mechanism, each page can be made to reside in any location of the physical memory, or be flagged as being protected. Paging makes it possible to have a linear virtual memory address space that access pieces out of a fragmented physical memory space.

Each process is given a page table to define the valid addresses and map them to physical memory. The page table is usually invisible to the process. Page tables make it easy to allocate new memory for a process, as each new page can be allocated from anywhere in physical memory.

Parts of an application's memory can be "swapped out" to other forms of storage. This happens to memory that is seldom used and it allows the application to act as if it has a much larger working memory than actually exists. By swapping out memory, the virtual memory layout will not change, but it frees a lot of physical memory (i.e. RAM) for other uses.

If the process is accessing a virtual memory location that is not mapped by the page table, a page fault will occur. Page faults could mean either that the process has tried to access memory that it should not have access to, or that part of the application's memory has been swapped out. In the last case, the page will be swapped back in and execution will proceed where it was interrupted.

## **Memory Spaces**

[http://en.wikipedia.org/wiki/User\\_space](http://en.wikipedia.org/wiki/User_space)

A protected-mode operating system usually segregates virtual memory into kernel space and user space:

- **Kernel space** is strictly reserved for running the kernel, device drivers and kernel extensions. In most operating systems, kernel memory is never swapped out to disk.
- **User space** is the memory area where all user mode applications work and this memory can be swapped out when necessary. The term "userland" is often used for referring to operating system software that runs in user space.

User mode and kernel mode are explained in more detail in the section "Memory segments" below

## **Privileges and Execution Modes**

[http://en.wikipedia.org/wiki/Kernel\\_mode](http://en.wikipedia.org/wiki/Kernel_mode)

[http://en.wikipedia.org/wiki/User\\_mode](http://en.wikipedia.org/wiki/User_mode)

For safety and security reasons, a program is not allowed to directly access the hardware of the computer. Instead, the program must do a request to the OS, a so-called *system call*, and

the OS then evaluates if it is safe, and if so, it does this hardware access on behalf of the requesting program. To enforce this mechanism, modern CPU's have 4 different execution modes built into hardware. By switching between execution modes (also called "privilege levels") one can allow or disallow two privileges:

1. Interaction with low level hardware
2. Access to arbitrary memory

Linux uses only two modes:

- **User mode:** disallow privileges 1 and 2
- **Kernel mode:** allow privileges 1 and 2

The idea of having two different operation modes comes from the principle "with more control comes more responsibility". A program in kernel mode is trusted never to fail, since a failure may cause the whole computer system to crash, whereas user-space applications are not trusted to work correctly.

In kernel mode (also called "supervisor mode") it is possible to execute machine code operations, such as modifying registers for various descriptor tables or performing operations such as disabling interrupts, which are not allowed in the less privileged user mode. To perform specialized operations in user mode, such as interaction with hardware, the code must perform a *system call* to trusted code running in kernel mode which can perform the requested task. Note that a system call is a consequence of having different privilege levels.

## Memory segments

When the scheduler switches the running process, it also switches the memory segment that can be addressed when running this process. This changes both the rearrangement of the virtual memory mapping to physical memory and privilege level the CPU is in when running the process. Linux has several memory segments to which it applies paging. We discuss the two most important ones:

- **User segment**
  - A process running within this segment is called a user (space/mode) process; it runs with user mode privilege level
  - When paging this segment:
    - Virtual memory is mapped purely to memory private for the user process
    - User process can only address the first 3 gigabytes of the usual 4 GB of virtual memory (using 32 bit addresses); the 4th gigabyte is forbidden and reserved for kernel processes
  - This memory is called *user space memory* which uses a different mapping to physical memory for different user process; hence a process can never access memory of other user processes, nor kernel processes, i.e., we have **memory protection**
- **Kernel segment**
  - A process running with this segment is called a kernel (space/mode) process; it runs with kernel mode privilege level
  - When paging this segment, virtual memory is mapped in two parts
    - Part one: This part is called *kernel space memory*. It is mapped to shared physical kernel memory using the same mapping used for all kernel processes; this is the 4th gigabyte of virtual memory which is shared by all kernel processes, i.e., no memory protection!
    - Part two: the user space memory of the user process which was running when this kernel process got started; these are the first 3 gigabytes of virtual memory

## Interrupts and System Calls

As stated before, a system call is a request of code running in user mode to have something done by the kernel running in the kernel mode. The only way to switch from user mode to kernel mode is by generating an interrupt. Therefore at booting the kernel configures a special interrupt 0x80 to do such requests. Note that all other interrupts generated by a program in user mode are ignored by the kernel.

Thus a system call is nothing else than an interrupt generated by software. When an interrupt is generated, the CPU immediately stops with what it was doing and starts handling the interrupt. The handler for the interrupt can be found in the interrupt table.

- For all interrupts, the memory segment is set to the kernel segment, meaning that the CPU will switch to kernel mode
- A user mode program can only generate interrupt 0x80; all other interrupts it generate will result in an security error
- Interrupt 0x80 is used to implement system calls. The interrupt handler for this interrupt looks in the CPU registers to see which system function it has to execute and with which arguments. After having finished this function, it places the result also there and next the processor switches back to the process in user mode

Note that a kernel module does not need a system call. Because it is in kernel mode, it can directly call the service of the system call.

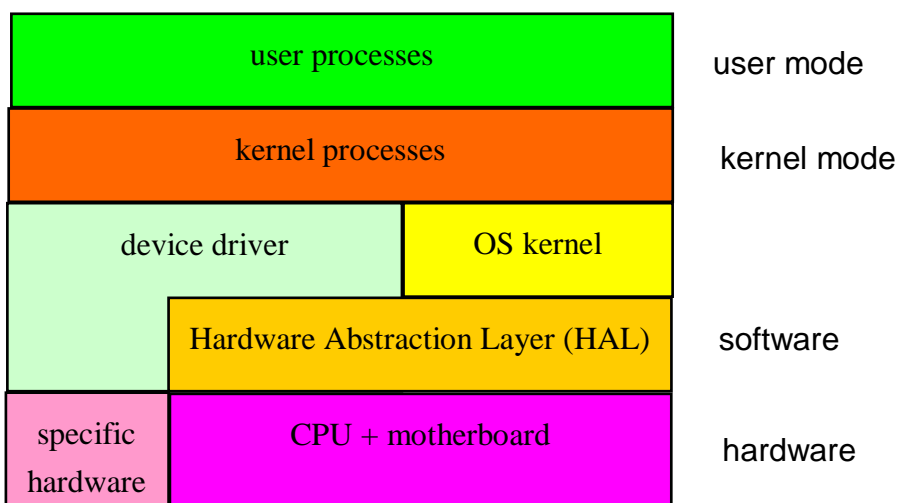
## Hardware Abstraction Layer - HAL

The hardware abstraction layer (HAL) provides an abstraction of the hardware.

- The software interacts with an abstract representation of the hardware in the HAL
- The software can be ported to another hardware platform by porting the HAL to this platform, implementing the same abstract representation.

In the context of an Operating System (OS) such as Windows or Linux, a HAL is used to abstract from the motherboard of the system. This makes it easier to run the OS on different platforms with different CPUs and controllers on the motherboard. This does not include special hardware attached to the motherboard; this requires the installation of special drivers.

## Linux architecture



**Linux:** CPU has two protection modes: kernel mode & user mode

