# ***Scheduling UBUNTU***

The main objectives of this blog is to improve your knowledge and work finely with Ubuntu 8.10. I will be giving you knowledge on the following subtopics. 1. User Interface 2. Process Control management 3. Dead lock management 4. Virtual Memory management 5. Secondary Disk scheduling management 6. Secondary storage management 7. System Administration and support 8. Recovery Strategies 9. Security Strategies 10. Standard Support

### Process Control Management

**Threads**

Ubuntu normally use the same internal representation for processes and threads. A thread can be defined as a new process that happens to share the same address space as its parent. A distinction is only made when a new thread is created by the clone system call.   
  
• fork - creates a new process with its own entirely new process context .   
• clone - creates a new process with its own identity, but that is allowed to share the data structures of its parent .   
  
Using clone gives an application fine-grained control over exactly what is shared between two threads”

**Process states**

From the time a process is created with a fork until it has completed its job and disappears from the process table, it goes through like this many different states. The state a process is in changes many times during its "States." These changes can occur, for example, when the process makes a system call, it is someone else's turn to run, an interrupt occurs, or the process asks for a resource that is currently not available. ( L, Friedrich 2008)

In Ubuntu il Sistema Operativo segue i seguenti stati di processo.

**Ready -**The process is competing for the processor or could be executed, but another process is currently being executed.

**Execution -**The process is active or running or being executed by processor.

**Suspend -**The process is waiting for an external event.

**Stopped -**The process has been suspended by an external process.

**Zombie -**The process has finished executed, but it is still references in the system.

|  |  |
| --- | --- |
|  | **Process scheduler**  A process scheduler handles CPU resource allocation for executing processes, and aims to maximize overall CPU utilization while also maximizing interactive performance.  Since kernel 2.6.23 (that would be as of Hardy 8.04 LTS) Completely Fair Scheduler (CFS) based on "Rotating Staircase Deadline". Overview from kernel.org:  CFS stands for "Completely Fair Scheduler," and is the new "desktop" process scheduler implemented by Ingo Molnar and merged in Linux 2.6.23. It is the replacement for the previous vanilla scheduler's SCHED\_OTHER interactivity code.  80% of CFS's design can be summed up in a single sentence: CFS basically models an "ideal, precise multi-tasking CPU" on real hardware.  "Ideal multi-tasking CPU" is a (non-existent :-)) CPU that has 100% physical power and which can run each task at precise equal speed, in parallel, each at 1/nr\_running speed. For example: if there are 2 tasks running, then it runs each at 50% physical power --- i.e., actually in parallel.  On real hardware, we can run only a single task at once, so we have to introduce the concept of "virtual runtime." The virtual runtime of a task specifies when its next timeslice would start execution on the ideal multi-tasking CPU described above. In practice, the virtual runtime of a task is its actual runtime normalized to the total number of running tasks. |

**Process scheduling**

While a process exists in Linux, it can potentially be scheduled through the Linux scheduler. Although outside of the scope of this article, the Linux scheduler maintains a set of lists for each priority level on which task\_struct references reside. Tasks are invoked through the schedule function (available in ./linux/kernel/sched.c), which determines the best process to run based on loading and prior process execution history. You can learn more about the Linux version 2.6 scheduler in Resources.