Linux kernel

Josep Jorba Esteve

PID_00148468

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction

The kernel of the GNU/Linux system (which is normally called Linux) [Vasb] is the heart of the system: it is responsible for booting the system, for managing the machine's resources by managing the memory, file system, input/output, processes and intercommunication of processes.

Its origin dates back to August 1991, when a Finnish student called Linus Torvalds announced on a news list that he had created his own operating system core that worked together with the GNU project software and that he was offering it to the community of developers for testing and suggesting improvements for making it more usable. This was the origin of the operating system's kernel that would later come to be known as Linux.

One of the particular features of Linux is that following the Free Software philosophy, it offers the source code of the operating system itself (of the kernel), in a way that makes it a perfect tool for teaching about operating systems.

Note

The Linux kernel dates back to 1991, when Linus Torvalds made it available to the community. It is one of the few operating systems that while extensively used, also makes its source code available.

Another main advantage, is that by having the source code, we can compile it to adapt it better to our system and we can also configure its parameters to improve the system's performance.

In this unit, we will look at how to handle this process of preparing a kernel for our system. How, starting with the source code, we can obtain a new version of the kernel adapted to our system. Similarly, we will discuss how to develop the configuration and subsequent compilation and how to test the new kernel we have obtained.

1. The kernel of the GNU/Linux system

The core or kernel is the basic part of any operating system [Tan87], where the code of the fundamental services for controlling the entire system lie. Basically, its structure can be divided into a series of management components designed to:

- Manage processes: what tasks will be run, in what order and with what priority. An important aspect is the scheduling of the CPU: how do we optimise the CPU's time to run the tasks with the best possible performance or interactivity with users?

- Intercommunication of processes and synchronisation: how do tasks communicate with each other, with what different mechanisms and how can groups of tasks be synchronised?

- Input/output management (I/O): control of peripherals and management of associated resources.

- Memory management: optimising use of the memory, paginating service, and virtual memory.

- File management: how the system controls and organises the files present in the system and access to them.

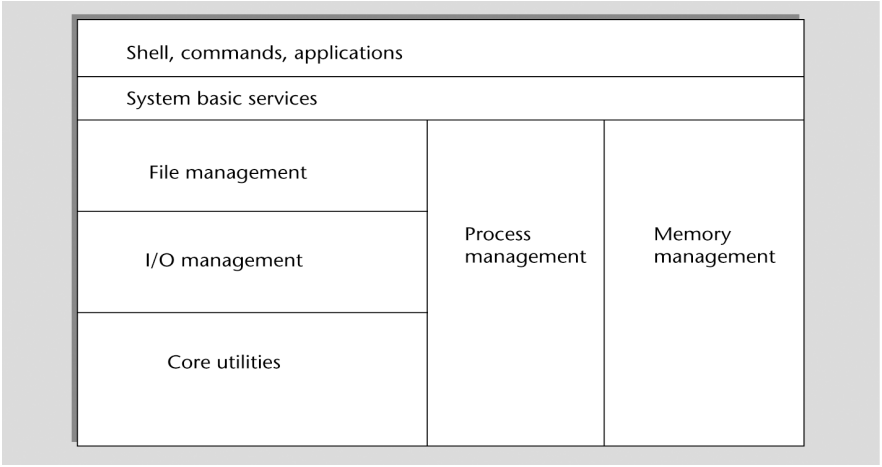

Figure 1. Basic functions of a kernel with regards to executed applications and commands

In proprietary systems, the kernel is perfectly "hidden" below the layers of the operating system's software; the end user does not have a clear perspective of what the kernel is and has no possibility of changing it or optimising it, other than through the use of esoteric editors of internal "registers" or specialised

third party programs, which are normally very expensive. Besides, the kernel is normally unique, it is the one the manufacturer provides and the manufacturer reserves the right to introduce any changes it wants whenever it wants and to handle the errors that appear in non-stipulated periods through updates offered to us in the form of error "patches" (or service packs).

One of the main problems of this approach is precisely the availability of these patches, having the error updates on time is crucial and if they are security-related, even more so, because until they are corrected we cannot guarantee the system's security for known problems. Many organisations, large companies, governments, scientific and military institutions cannot depend on the whims of a manufacturer to solve the problems with their critical applications.

The Linux kernel offers an open source solution with the ensuing permissions for modifying, correcting, generating new versions and updates very quickly by anyone anywhere with the required knowledge for doing so.

This allows critical users to control their applications and the system itself better, and offers the possibility of mounting systems with a "tailor-made" operating system adjusted to each individual's taste and in turn to have an open source operating system developed by a community of programmers who coordinate via the Internet, accessible for educational purposes because it has open source code and abundant documentation, for the final production of GNU/Linux systems adapted to individual needs or to the needs of a specific group.

Because the source code is open, improvements and solutions can be found immediately, unlike proprietary software, where we have to wait for the manufacturer's updates. Also, we can personalise the kernel as much as we wish, an essential requirement, for example, in high performance applications, applications that are critical in time or solutions with embedded systems (such as mobile devices).

Following a bit of (quick) history of the kernel [Kera] [Kerb]: it was initially developed by a Finnish student called Linus Torvalds, in 1991, with the intention of creating a similar version to Minix [Tan87] (version for PC of UNIX [Bac86]) for the Intel 386 processor. The first officially published version was Linux 1.0 in March 1994, which only included the execution for the i386 architecture and supported single-processor machines. Linux 1.2 was published in March 1995, and was the first version to cover different architectures such as Alpha, SPARC and Mips. Linux 2.0, in June 1996, added more architectures and was the first version to include multiprocessor support (SMP) [Tum]. In Linux 2.2, January 1999, SMP benefits were significantly increased, and controllers were added for a large amount of hardware. In 2.4, released in January 2001, SMP support was improved, new supported architectures were incorporated and controllers for USB, PC card devices were included (PCMCIA for laptops) part of PnP (plug and play), RAID and volumes support etc. Branch 2.6

of the kernel (December 2003), considerably improved SMP support, offered a better response of the CPU scheduling system, use of threads in the kernel, better support for 64-bit architectures, virtualisation support and improved adaptation to mobile devices.

Where the development is concerned, since the kernel was created by Linus Torvalds in 1991 (version 0.01), he has continued to maintain it, but as his work allowed it and as the kernel matured (and grew) he was helped to maintain the different stable versions of the kernel by different collaborators, while Linus continued (insofar as possible) developing and compiling contributions for the latest version of the kernel's development. The main collaborators of these versions have been [lkm]:

Note

The kernel has its origins in the MINIX system, a development by Andrew Tanenbaum, as a UNIX clone for PC.

- 2.0 David Weinehall.
- 2.2 Alan Cox (who also develops and publishes patches for most versions).
- 2.4 Marcelo Tosatti.
- 2.6 Andrew Morton / Linus Torvalds.

In order to understand a bit about the complexity of the Linux kernel, let's look at a table with a bit of a summarised history of its different versions and its size in relation to the source code. The table only shows the production versions; the (approximate) size is specified in thousands of lines (K) of source code:

Version	Publication date	Code lines (thousands)
0.01	09-1991	10
1.0	03-1994	176
1.20	03-1995	311
2.0	06-1996	649
2.2	01-1999	1800
2.4	01-2001	3378
2.6	12-2003	5930

As we can see, we have moved from about ten thousand lines to six million.

Now, development of branch 2.6.x of the kernel continues, the latest stable version, which most distributions include as the default version (although some still include 2.4.x, but 2.6.x is an option during the installation); although a certain amount of knowledge about the preceding versions is essential, because we can easily find machines with old distributions that have not been updated, which we may have to maintained or migrated to more modern versions.

Note

Today's kernel has reached a significant degree of complexity and maturity.

During the development of branch 2.6, the works on the kernel accelerated considerably, because both Linus Torvalds, and Andrew Morton (who maintain Linux 2.6) joined (in 2003) OSDL (Open Source Developement Laboratory) [OSDa], a consortium of companies dedicated to promoting the use of Open Source and GNU/Linux by companies (the consortium includes among many other companies with interests in GNU/Linux: HP, IBM, Sun, Intel, Fujitsu, Hitachi, Toshiba, Red Hat, Suse, Transmeta...). Now we are coming across an interesting situation, since the OSDL consortium sponsored the works of both the stable version of the kernel's maintainer (Andrew) and developer (Linus), working full time on the versions and on related issues. Linus remains independent, working on the kernel, while Andrew went to work for Google, where he continued his developments full time, making patches with different contributions to the kernel. After some time, OSDL became The Linux Foundation.

Note

The Linux Foundation: www.linuxfoundation.org

We need to bear in mind that with current versions of the kernel, a high degree of development and maturity has been achieved, which means that the time between the publication of versions is longer (this is not the case with partial revisions).

Another factor to consider is the number of people that are currently working on its development. Initially, there were just a handful of people with complete knowledge of the entire kernel, whereas nowadays many people are involved in its development. Estimates are almost two thousand with different levels of contribution, although the number of developers working on the hard core is estimated at several dozen.

We should also take into consideration that most only have partial knowledge of the kernel and neither do they all work simultaneously nor is their contribution equally relevant (some just correct simple errors); it is just a few people (such as the maintainers who have full knowledge of the kernel. This means that developments can take a while to occur, contributions need to be debugged to make sure that they do not come into conflict with each other and choices need to be made between alternative features.

Regarding the numbering of the Linux kernel's versions ([lkm][DBo]), we should bear in mind the following:

a) Until kernel branch 2.6.x, the versions of the Linux kernel were governed by a division into two series: one was known as the "experimental" version (with the second number being an odd number, such as 1.3.xx, 2.1.x or 2.5.x) and the other was the "production" version (even series, such as 1.2.xx, 2.0.xx, 2.2.x, 2.4.x and more). The experimental series were versions that moved rapidly and that were used for testing new features, algorithms, device drivers etc. Because of the nature of the exper-

imental kernels, they could behave unpredictably, losing data, blocking the machine etc. Therefore, they were not suited to production environments, unless for testing a specific feature (with the associated dangers).

Production or stable kernels (even series) were kernels with a well defined set of features, a low number of known errors and with tried and tested device controllers. They were published less frequently than the experimental versions and there were a variety of versions, some better than others. GNU/Linux distributions are usually based on a specifically chosen stable kernel, not necessarily the latest published production kernel.

b) The current Linux kernel numbering (used in branch 2.6.x), continues to maintain some basic aspects: the version is indicated by numbers *X.Y.Z,* where normally *X* is the main version, which represents important changes to the kernel; *Y* is the secondary version and usually implies improvements in the kernel's performance: *Y* is even for stable kernels and odd for developments or tests; and *Z* is the build version, which indicates the revision number of *X.Y*, in terms of patches or corrections made. Distributors do not tend to include the latest version of the kernel, but rather the one they have tested most frequently and can verify is stable for the software and components it includes. On the basis of this classical numbering scheme (followed during versions 2.4.x, until the early versions of branch 2.6), modifications were made to adapt to the fact that the kernel (branch 2.6.x) is becoming more stable (fixing X.Y to 2.6), and that there are fewer and fewer revisions (thus the leap in version of the first numbers), but development remains continuous and frenetic.

Under the latest schemes, four numbers are introduced to specify in Z minor changes or the revision's different possibilities (with different added patches). The version thus defined with four numbers is the one considered to be stable. Other schemes are also used for the various test versions (normally not advisable for production environments), such as -rc suffixes (*release candidate*), -mm which refers to experimental kernels with tests for different innovative techniques, or -git which are a sort of daily snapshot of the kernel's development. These numbering schemes are constantly changing to adapt to the way of working of the kernel community and its needs to accelerate the development.

c) To obtain the latest published kernel, you need to visit the Linux kernels file (at http://www.kernel.org) or its local mirror in Spain (http://www.es.kernel.org). It will also be possible to find some patches for the original kernel, which correct errors detected after the kernel's publication.

Note

Kernel repository:
www.kernel.org

Some of the technical characteristics ([DBo][Arc]) of the Linux kernel that we should highlight are:

• Kernel of the monolithic type: basically it is a program created as a unit, but conceptually divided into several logical components.

• It has support for loading/downloading portions of the kernel, these portions are known as modules, and tend to be characteristics of the kernel or device drivers.

• Kernel threading: for internal functioning, several execution threads are used internal to the kernel, which may be associated to a user program or to an internal functionality of the kernel. In Linux, this concept was not used intensively. The revisions of branch 2.6.x offered better support and a large proportion of the kernel is run using these various execution threads.

• Multithreaded applications support: user applications support of the multithread, since many computing paradigms of the client/server type, need servers capable of attending to numerous simultaneous requests, dedicating an execution thread to each request or group of requests. Linux has its own library of threads that can be used for multithread applications, with the improvements made to the kernel, they have also allowed a better use for implementing thread libraries for developing applications.

• The kernel is of a nonpreemptive type: this means that within the kernel, system calls (in supervisory mode) cannot be interrupted while the system task is being resolved and, when the latter finishes, the execution of the previous task is resumed. Therefore, the kernel within a call cannot be interrupted to attend to another task. Normally, preemptive kernels are associated to systems that operate in real time, where the above needs to be allowed in order to handle critical events. There are some special versions of the Linux kernel for real time, that allow this by introducing some fixed points where they can be exchanged. This concept has also been especially improved in branch 2.6.x of the kernel, in some cases allowing some resumable kernel tasks to be interrupted in order to deal with others and resuming them later. This concept of a preemptive kernel can also be useful for improving interactive tasks, since if costly calls are made to the system, they can cause delays in interactive applications.

• Multiprocessor support, known as symmetrical multiprocessing (SMP). This concept tends to encompass machines that incorporate the simple case of 2 up to 64 CPUs. This issue has become particularly relevant with multicore architectures, that allow from 2 or 4 to more CPU cores in machines accessible to domestic users. Linux can use multiple processors, where each processor can handle one or more tasks. But some parts of the kernel decreased performance, since they were designed for a single CPU and forced the entire system to stop

under certain cases of blockage. SMP is one of the most studied techniques in the Linux kernel community and important improvements have been achieved in branch 2.6. Since SMP performance is a determining factor when it comes to companies adopting Linux as an operating system for servers.

• File systems: the kernel has a good file system architecture, internal work is based on an abstraction of a virtual system (VFS, *virtual file system*), which can be easily adapted to any real system. As a result, Linux is perhaps the operating system that supports the largest number of file systems, from ext2, to MSDOS, VFAT, NTFS, journaled systems, such as ext3, ReiserFS, JFS(IBM), XFS(Silicon), NTFS, ISO9660 (CD), UDF and more added in the different revisions.

Other less technical characteristics (a bit of marketing):

a) Linux is free: together with the GNU software and included in any distribution, we can have a full UNIX-like system practically for the cost of the hardware, regarding GNU/Linux distribution costs, we can have it practically free. Although it makes sense to pay a bit extra for a complete distribution, with the full set of manuals and technical support, at a lower cost than would be paid for some proprietary systems or to contribute with the purchase to the development of distributions that we prefer or that we find more practical.

b) Linux can be modified: the GPL license allows us to read and to modify the source code of the kernel (on condition that we have the required know-how).

c) Linux can run on fairly limited old hardware; for example, it is possible to create a network server on a 386 with 4 MB of RAM (there are distributions specialised for limited resources).

d) Linux is a powerful system: the main objective of Linux is efficiency, it aims to make the most of the available hardware.

e) High quality: GNU/Linux systems are very stable, have a low fault ratio and reduce the time needed for maintaining the systems.

f) The kernel is fairly small and compact: it is possible to place it, together with some basic programs, on a disk of just 1.44 MB (there are several distributions on just one diskette with basic programs).

g) Linux is compatible with a large number of operating systems, it can read the files of practically any file system and can communicate by network to offer/receive services from any of these systems. Also, with certain libraries it can also run the programs of other systems (such as MSDOS, Windows, BSD, Xenix etc.) on the x86 architecture.

h) Linux has extensive support: there is no other system that has the same speed and number of patches and updates as Linux, not even any proprietary system. For a specific problem, there is an infinite number of mail lists and forums that can help to solve any problem within just a few hours. The only problem affects recent hardware controllers, which many manufacturers are still reluctant to provide if they are not for proprietary systems. But this is gradually changing and many of the most important manufacturers in sectors such as video cards (NVIDIA, ATI) and printers (Epson, HP,) are already starting to provide the controllers for their devices.

2. Configuring or updating the kernel

As GNU/Linux users or system administrators, we need to bear in mind the possibilities the kernel offers us for adapting our requirements and equipment.

At installation time, GNU/Linux distributions provide a series of preconfigured and compiled binary Linux kernels and we will usually have to choose which kernel from the available set best adapts to our hardware. There are generic kernels, oriented at IDE devices, others at SCSI, others that offer a mix of device controllers [AR01] etc.

Another option during the installation is the kernel version. Distributions normally use an installation that they consider sufficiently tested and stable so that it does not cause any problems for its users. For example, nowadays many distributions come with versions 2.6.x of the kernel by default, since it is considered the most stable version (at the time the distribution was released). In certain cases, as an alternative, more modern versions may be offered during the installation, with improved support for more modern (latest generation) devices that perhaps had not been so extensively tested at the time when the distribution was published.

Distributors tend to modify the kernel to improve their distribution's behaviour or to correct errors detected in the kernel during tests. Another fairly common technique with commercial distributions is to disable problematic features that can cause errors for users or that require a specific machine configuration or when a specific feature is not considered sufficiently stable to be included enabled by default.

Note

The possibility of updating and adapting the kernel offers a good adjustment to any system through tuning and optimisation.

This leads us to consider that no matter how well a distributor does the job of adapting the kernel to its distribution, we can always encounter a number of problems:

- The kernel is not updated to the latest available stable version; some modern devices are not supported.
- The standard kernel does not support the devices we have because they have not been enabled.
- The controllers a manufacturer offers us require a new version of the kernel or modifications.
- The opposite, the kernel is too modern, and we have old hardware that is no longer supported by the modern kernels.
- The kernel, as it stands, does not obtain the best performance from our devices.

- Some of the applications that we want to use require the support of a new kernel or one of its features.
- We want to be on the leading edge, we risk installing the latest versions of the Linux kernel.
- We like to investigate or to test the new advances in the kernel or would like to touch or modify the kernel.
- We want to program a driver for an unsupported device.
- ...

For these and other reasons we may not be happy with the kernel we have; in which case we have two possibilities: updating the distribution's binary kernel or tailoring it using the source.

Let's look at a few issues related to the different options and what they entail:

1) Updating the distribution's kernel: the distributor normally also publishes kernel updates as they are released. When the Linux community creates a new version of the kernel, every distributor joins it to its distribution and conducts the relevant tests. Following the test period, potential errors are identified, corrected and the relevant update of the kernel is made in relation to the one offered on the distribution's CDs. Users can download the new revision of the distribution from the website, or update it via some other automatic package system through a package repository. Normally, the system's version is verified, the new kernel is downloaded and the required changes are made so that the following time the system functions with the new kernel, maintaining the old version in case there are any problems.

This type of update simplifies the process for us a lot, but may not solve our problems, since our hardware may not yet be supported or the feature of the kernel to be tested is still not in the version that we have of the distribution; we need to remember that there is no reason for distributor to use the latest available version (for example in kernel.org) but rather the one it considers stable for its distribution.

If our hardware is not enabled by default in the new version either, we will find ourselves in the same situation. Or simply, if we want the latest version, this process is no use.

2) Tailoring the kernel (this process is described in detail in the following sections). In this case, we will go to the sources of the kernel and "manually" adjust the hardware or required characteristics. We will pass through a process of configuring and compiling the source code of the kernel so as to create a binary kernel that we will install on the system and thus have it available the following time the system is booted.

Here we may also encounter two more options, either by default we will obtain the "official" version of the kernel (kernel.org), or we can go to the sources provided by the distribution itself. We need to bear in mind that distributions like Debian and Fedora do a lot of work on adapting the kernel and correcting kernel errors that affect their distribution, which means that in some cases we may have additional corrections to the kernel's original code. Once again, the sources offered by the distribution do not necessarily have to correspond to the latest published version.

This system allows us maximum reliability and control, but at a high administration cost; since we will need to have extensive knowledge of the devices and characteristics that we are selecting (what they mean and what implications they may have), in addition to the consequences that the decisions we make may imply.

3. Configuration and compilation process

Configuring the kernel [Vasb] is a costly process and requires extensive knowledge on the part of the person doing it, it is also one of the critical tasks on which the system's stability depends, given the nature of the kernel, which is the system's central component.

Note

The process of obtaining a new personalised kernel involves obtaining the sources, adapting the configuration, and compiling and installing the obtained kernel on the system.

Any error in the procedure can cause instability or the loss of the system. Therefore, it is advisable to make a backup of user data, configurations we have tailored, or, if we have the required devices, to make a complete system backup. It is also advisable to have a start up diskette (or Live CD distribution with tools) to help us in the event of any problem, or a rescue disk which most distributions allow us to create from the distribution's CDs (or by directly providing a rescue CD for the distribution).

Without meaning to exaggerate, if the steps are followed correctly, we know what we are doing and take the necessary precautions, errors almost never occur.

Let's look at the process required to install and configure a Linux kernel. In the following sections, we look at:

1) The case of old 2.4.x versions.

2) Some considerations regarding migrating to 2.6.x

3) Specific details regarding versions 2.6.x.

4) A particular case with the Debian distribution, which has its own more flexible compilation system (*debian way*).

Versions 2.4.x are practically no longer offered by current distributions, but we should consider that on more than one occasion we may find ourselves obliged to migrate a specific system to new versions or to maintain it on the old ones, due to incompatibilities or the existence of old unsupported hardware.

The general concepts of the compilation and configuration process will be explained in the first section (2.4.x), since most of them are generic, and we will subsequently see the differences with regard to the new versions.

3.1. Kernel compilation versions 2.4.x

The instructions are specifically for the Intel x86 architecture, by root user (although part of the process can be done as a normal user):

1) Obtaining the kernel: for example, we can visit www.kernel.org (or its FTP server) and download the version we would like to test. There are mirrors for different countries. In most GNU/Linux distributions, such as Fedora/Red Hat or Debian, the kernel's source code is also offered as a package (normally with some modifications included), if we are dealing with the version of the kernel that we need, it may be preferable to use these (through the kernel-source packages or similar). If we want the latest kernels, perhaps they are not available in the distribution and we will have to go to kernel.org.

2) Unpack the kernel: the sources of the kernel were usually placed and unpacked from the directory `/usr/src`, although we advise using a separate directory so as not to mix with source files that the distribution may carry. For example, if the sources come in a compressed file of the bzip2 type:

```
bzip2 -dc linux-2.4.0.tar.bz2 | tar xvf -
```

If the sources come in a gz file, we will replace bzip2 with gzip. When we decompress the sources, we will have generated a directory linux-version_kernel that we will enter in order to configure the kernel.

Before taking the steps prior to compilation, we should make sure that we have the right tools, especially the gcc compiler, make and other complementary gnu utilities for the process. For example, the *modutils*, the different utilities for using and handling the dynamic kernel modules. Likewise, for the different configuration options we should take into account a number of pre-requirements in the form of libraries associated to the configuration interface used (for example ncurses for the menuconfig interface).

In general, we advise checking the kernel documentation (whether via the package or in the root directory of the sources) to know what pre-requirements and versions of the kernel source will be needed for the process. We advise studying the README files in this root directory of the kernel source, and *Documentation/Changes* or the documentation index of the kernel in *Documentation/00-INDEX*.

If we have made previous compilations in the same directory, we need to make sure that the directory we use is clear of previous compilations; we can clear it using *make mrproper* (from the "root" directory).

For the process of configuring the kernel [Vasb], we have several alternative methods, which offer us different interfaces for adjusting the various parameters of the kernel (which tend to be stored in a configuration file, normally *.config* in the "root" directory of the sources). The different alternatives are:

• **make config:** from the command line we are asked for each option, and we are asked for confirmation (y/n) – yes or no, the option, or we are asked for the required values. Or the long configuration, where we are asked for many answers, and depending on each version, we will likewise have to answer almost a hundred questions (or more depending on the version).

• **make oldconfig:** it is useful if we want to reuse an already used configuration (normally stored in a *.config* file, in the root directory of the sources), we need to take into account that it is only valid if we are compiling the same version of the kernel, since different kernel versions can have variable options.

• **make menuconfig:** configuration based on text menus, fairly convenient; we can enable or disable what we want and it is faster than *make config*.

• **make xconfig:** the most convenient, based on graphic dialogues in X Window. We need to have tcl/tk libraries installed, since this configuration is programmed in this language. The configuration is based on tables of dialogues and buttons/checkboxes, can be done fairly quickly and has help with comments on most options. But it has a defect, which is that some options may not appear (it depends on whether the configuration program is updated and sometimes it is not). In this last case, *make config* (or *menuconfig*) is the only one we can be sure will offer all the options we can choose; for the other types of configuration it depends on whether the programs have been adapted to the new options in time for the kernel being released. Although in general they try to do it at the same time.

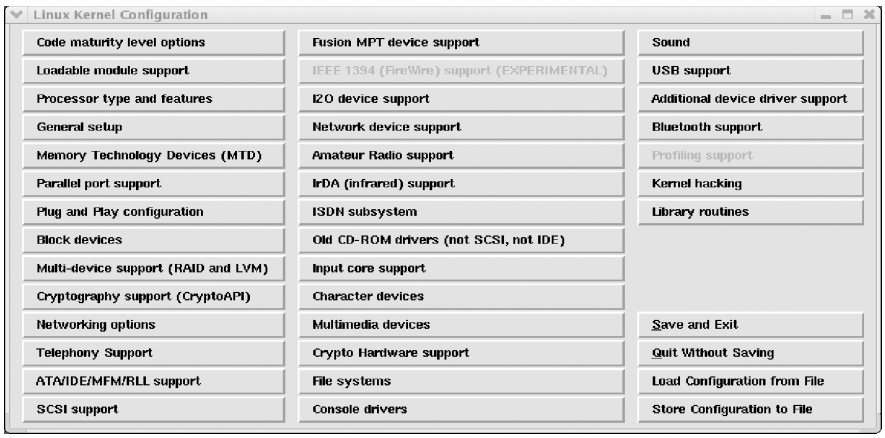

Figure 2. Configuration of the kernel (make xconfig) from graphic interface in X Window

Once the configuration process has been done, we need to save the file (*.config),* since the configuration requires a considerable amount of time. Also, it may be useful to have the configuration done if the plan is to do it on several similar or identical machines.

Another important issue concerning configuration options is that in many cases we will be asked if we want a specific characteristic integrated into the kernel or as a module (in the section on modules we will provide more details on them). This is a fairly important decision, since in certain cases our choice will influence the performance of the kernel (and therefore of the entire system).

The Linux kernel has become very large, due both to its complexity and to the device controllers (drivers) [AR01] that it includes. If we integrated everything, we could create a very large kernel file that would occupy a lot of memory and, therefore, slow down some functioning aspects. The modules of the kernel [Hen] are a method that makes it possible to divide part of the kernel into smaller sections, which will be loaded dynamically upon demand or when they are necessary for either explicit load or use of a feature.

The normal choice is to integrate what is considered fundamental for functioning or critical for performance within the kernel and to leave parts or controllers that will be used sporadically as modules for future extensions of the equipment.

• A clear case are the device controllers: if we are updating the machine, it may be that when it comes to creating the kernel we are not sure what hardware it will have: for example, what network card; but we do know that it will be connected to a network, so, the network support will be integrated into the kernel, but for the card controllers we can select a few (or all) of them and install them as modules. Then, when we have the card we can load the required module or

if we need to change one card for another later, we will just have to change the module to be loaded. If just one controller were integrated into the kernel and we changed the card, we would be forced to reconfigure and recompile the kernel with the new card's controller.

• Another case that arises (although it is not very common) is when we have two devices that are incompatible with each other, or when one or the other is functioning (for example, this tends to happen with a parallel cable printer and hardware connected to the parallel port). Therefore, in this case, we need to put the controllers as modules and load or download the one we need.

• Another example is the case of file systems. Normally we would hope that our system would have access to some of them, like ext2 or ext3 (belonging to Linux), VFAT (belonging to Windows 95/98/ME), and we will enable them in configuring the kernel. If at some moment we have to read another unexpected type, for example data stored on a disk or partition of the Windows NT/XP NTFS system, we would not be able to: the kernel would not know how to or would not have support to do so. If we have foreseen that at some point (but not usually) we may need to access these systems, we could leave the other file systems as modules.

3) Compiling the kernel

We will start the compilation using *make*, first we will have to generate the possible dependencies between the code and then the type of image of the kernel that we want (in this case, a compressed image, which tends to be the normal case):

```
make dep
make bzImage
```

When this process is completed, we will have the integrated part of the kernel; we are missing the parts that we have set as modules:

```
make modules
```

At this point we have done the configuring and compiling of the kernel. This part could be done by a normal user or by the root user, but now we will definitely need the root user, because we will move onto the installation part.

4) Installation

We'll start by installing the modules:

```
make modules_install
```

And the installation of the new kernel (from the directory /usr/src/linux-version or the one we have used as temporary):

```
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.0
cp System.map /boot/System.map-2.4.0
```

the file bzImage is the newly compiled kernel, which is placed in the */boot* directory. Normally, we will find the old kernel in the same */boot* directory with the name vmlinuz or vmlinuz-previous-version as a symbolic link to the old kernel. Once we have our kernel, it is better to keep the old one, in case any faults occur or the new one functions badly, so that we can recover the old one. The file System.map contains the symbols available for the kernel and is necessary for the processing of starting it up; it is also placed in the same directory.

On this point, we also need to consider that when the kernel starts up it may need to create initrd type files, which serve as a compound image of some basic drivers and is used when loading the system, if the system needs those drivers before booting certain components. In some cases, it is vital because in order to boot the rest of the system, certain drivers need to be loaded in a first phase; for example specific disk controllers such as RAID or volume controllers, which would be necessary so that in a second phase, the disk can be accessed for booting the rest of the system.

The kernel can be generated with or without an initrd image, depending on the needs of the hardware or system in question. In some cases, the distribution imposes the need to use an initrd image, in other cases it will depend on our hardware. It is also often used to control the size of the kernel, so that its basics can be loaded through the initrd image and later the rest in a second phase in the form of modules. In the case of requiring the initrd image, it would be created using the mkinitrd utility (see man, or chapter workshop), within the */boot* directory.

5) The following step is to tell the system what kernel it needs to boot with, although this depends on the Linux booting system:

- From booting with lilo [Zan][Skoa], whether in the MBR (*master boot record*) or from an own partition, we need to add the following lines to the configuration file (in: /etc/lilo.conf):

```
image = /boot/vmlinuz-2.4.0
 label = 2.4.0
```

where *image* is the kernel to be booted, and *label* is the name that the option will appear with during booting. We cam add these lines or modify the ones of the old kernel. We recommend adding them and leaving the old kernel, in case any problems occur, so that the old one can be recovered. In the file */etc/lilo.conf* we may have one or more start up configurations, for either Linux or other systems (such as Windows).

Every start up is identified by its line image and the label that appears in the boot menu. There is a line *default = label* that indicates the label that is booted by default. We can also add root = /dev/... to the preceding lines to indicate the disk partition where the main file system is located (the '/'), remembering that the disks have devices such as /dev/hda (1st disk ide) /dev/hdb (2 disk ide) or /dev/sdx for SCSI (or emulated) disks, and the partition would be indicated as *root = /dev/hda2* if the '/' of our Linux were on the second partition of the first ide disk. Using "*append* =" we can also add parameters to the kernel start up [Gor]. If the system uses initrd, we will also have to indicate which is the file (which will also be located in /boot/initrd-versionkernel), with the option "initrd=". After changing the lilo configuration, we need to write it for it to boot:

```
/sbin/lilo -v
```

We reboot and start up with the new kernel.

If we have problems, we can recover the old kernel, by selecting the option of the old kernel, and then, using the retouch lilo.conf, we can return to the old configuration or study the problem and reconfigure and recompile the kernel.

• Boot with grub [Kan01][Pro]. In this case, handling is simple, we need to add a new configuration consisting of the new kernel and adding it as another option to the grub file. Next, reboot in a similar way as with lilo, but remembering that in grub it is sufficient to edit the file (typically /boot/grub/menu.lst) and to reboot. It is also better to leave the old configuration in order to recover from potential errors.

3.2. Migration to kernel 2.6.x

In the case of having to update versions of old distributions, or changing the kernel generation using the source code, we will have to take some aspects into account, due to the novelties introduced into kernel branch 2.6.x.

Here is a list of some of the specific points to consider:

- Some of the kernel modules have changed their name, and some may have disappeared, we need to check the situation of the dynamic modules that are loaded (for example, examine /etc/modules and/or /etc/modules.conf) and edit them to reflect the changes.

- New options have been added to the initial configuration of the kernel: like make gconfig, a configuration based on gtk (Gnome). In this case, as a prerequisite, we will need to look out for Gnome libraries. The option make xconfig has now been implemented with the qt libraries (KDE).

- The minimum required versions of various utilities needed for the compilation process are increased (consult Documentation/Changes in the kernel sources). Especially, the minimum gcc compiler version.

- The default package for the module utilities has changed, becoming module-init-tools (instead of modutils used in 2.4.x). This package is a prerequisite for compiling kernels 2.6.x, since the modules loader is based on this new version.

- The devfs system becomes obsolete in favour of *udev*, the system that controls the *hotplug* start up (connection) of devices (and their initial recognition, in fact simulating a hotplug start up when the system boots), dynamically creating inputs in the directory /dev, only for devices that are actually present.

- In Debian as of certain versions of branch 2.6.x, for the binary images of the kernels, headers and source code, the name of the packages changes from kernel-images/source/headers to linux-image/source/headers.

- In some cases, new technology devices (like SATA) may have moved from /dev/hdX to /dev/sdX. In these cases, we will have to edit the configurations of /etc/fstab and the bootloader (lilo or grub) in order to reflect the changes.

- There may be some problems with specific input/output devices. The change in name of kernel modules has affected, among others, mouse devices, which likewise can affect the running of X-Window, until the required models are verified and the correct modules are loaded (for example *psmouse*). At the same time, the kernel integrates the Alsa sound drivers. If we have the old OSS, we will have to eliminate them from the loading of modules, since Alsa already takes care of emulating these.

- Regarding the architectures that the kernel supports, we need to bear in mind that kernel 2.6.x, in its different revisions, has been increasing the supported architectures which will allow us to have the binary images of the kernel in the distributions (or the options for compiling the kernel) best suited to supporting our processors. Specifically, we can find archi-

tectures such as i386 (for Intel and AMD): supporting the compatibility of Intel in 32 bits for the entire family of processors (some distributions use the 486 as the general architecture), some distributions integrate differentiated versions for i686 (Intel from pentium pro thereafter), for k7 (AMD Athlon thereafter), and those specific to 64 bits, for AMD 64 bits, and Intel with em64t extensions of 64 bits such as Xeon, and multicores. At the same time, there is also the IA64 architecture for 64bit Intel Itanium models. In most cases, the architectures have SMP capabilities activated in the kernel image (unless the distribution supports versions with and without SMP, created independently, in this case, the suffix -smp is usually added to the image that supports it).

- In Debian, to generate inirtrd images, as of certain versions of the kernel (>=2.6.12) the *mkinitrd* tools are considered obsolete, and are replaced with new utilities such as *initramfs* tools or *yaird*. Both allow the initrd image to be built, but the former is the recommended one (by Debian).

3.3. Compilation of the kernel versions 2.6.x

In versions 2.6.x, bearing in mind the abovementioned considerations, the compilation takes place in a similar way to the one described above:

Having downloaded the kernel 2.6.x (with x the number or pair of numbers of the kernel revision) to the directory that will be used for the compilation and checking the required versions of the basic utilities, we can proceed to the step of compiling and cleaning up previous compilations:

```
# make clean mrproper
```

configuration of parameters (remember that if we have a previous .config, we will not be able to start the configuration from zero). We do the configuration through the selected *make* option (depending on the interface we use):

```
# make menuconfig
```

construction of the kernel's binary image

```
# make dep
# make bzImage
```

construction of the modules (those specified as such):

```
# make modules
```

installation of the created modules (/lib/modules/version)

```
# make modules_install
```

copying of the image to its final position (assuming i386 as the architecture):

```
# cp arch/i386/boot/bzimage /boot/vmlinuz-2.6.x.img
```

and finally, creating the initrd image that we consider necessary, with the necessary utilities according to the version (see subsequent comment). And adjustment of the lilo or grub bootloader depending on which one we use.

The final steps (vmlinuz, system.map and initrd) of moving files to /boot can normally also be done with the process:

```
# make install
```

but we need to take into account that it does the entire process and will update the bootloaders, removing or altering old configurations; at the same time, it may alter the default links in the */boot* directory. We need to bear this in mind when it comes to thinking of past configurations that we wish to save.

Regarding the creation of the initrd, in Fedora/Red Hat it will be created automatically with the *install* option. In Debian we should either use the techniques of the following section or create it expressly using mkinitrd (versions <=2.6.12) or, subsequently, with *mkinitramfs*, or a utility known as *update-initramfs*, specifying the version of the kernel (it is assumed that it is called vmlinuz-version within the /boot directory):

```
# update-initramfs -c -k 'version'
```

3.4. Compilation of the kernel in Debian (Debian way)

In Debian, in addition to the examined methods, we need to add the configuration using the method known as Debian Way. A method that allows us to build the kernel in a fast and flexible manner.

For the process, we will need several utilities (install the packages or similar): kernel-package, ncurses-dev, fakeroot, wget, bzip2.

We can see the method from two perspectives, rebuilding a kernel equivalent to the one provided by the distribution or tailoring it and then using the method for building an equivalent personalised kernel.

In the first case, we initially obtain the version of the kernel sources provided by the distribution (meaning x the revision of the kernel 2.6):

```
# apt-get install linux-source-2.6.x
$ tar -xvjf /usr/src/linux-source-2.6.x.tar.bz2
```

where we obtain the sources and decompress them (the package leaves the file in */usr/src*).

Note

We can see the Debian way process in a detailed manner in: http://kernel-handbook.alioth.debian.org/

Installing the basic tools:

```
# apt-get install build-essential fakeroot
```

Checking source dependencies

```
# apt-get build-dep linux-source-2.6.x
```

And construction of the binary, according to the pre-established package configuration (similar to that included in the official image packages of the kernel in Debian):

```
$ cd linux-source-2.6.x
$ fakeroot debian/rules binary
```

There are some extra procedures for creating the kernels based on different patch levels provided by the distribution and possibilities of generating different final configurations (view the reference note to complement these aspects).

In the second, more common case, when we would like a personalised kernel, we will have to follow a similar process through a typical tailoring step (for example, using make menuconfig); the steps would be:

obtaining and preparing the directory (here we obtain the distribution's packages, but it is equivalent to obtaining the sources from *kernel.org*):

```
# apt-get install linux-source-2.6.x
$ tar xjf /usr/src/linux-source-2.6.x.tar.bz2
$ cd linux-source-2.6.x
```

next, we configure the parameters, as always, we can base ourselves on .config files that we have used previously, to start from a known configuration (for tailoring we can also use any of the other methods, xconfig, gconfig...):

```
$ make menuconfig
```

final construction of the kernel depending on initrd or not, without initrd available (we need to take care with the version we use; as of a certain version of the kernel, the use of the initrd image can be mandatory):

```
$ make-kpkg clean
$ fakeroot make-kpkg --revision=custom.1.0 kernel_image
```

or if we have initrd available (already built)

```
$ make-kpkg clean
$ fakeroot make-kpkg --initrd --revision=custom.1.0 kernel_image
```

The process will end with adding the associated package to the kernel image, which we will finally be able to install:

```
# dpkg -i ../linux-image-2.6.x_custom.1.0_i386.deb
```

In this section, we will also add another peculiarity to be taken into consideration in Debian, which is the existence of utilities for adding dynamic kernel modules provided by third parties. In particular, the *module-assistant* utility helps to automate this process on the basis of the module sources.

We need to have the headers of the kernel installed (package linux-headers-version) or the sources we use for compiling the kernel. As of here, the module-assistant can be used interactively, allowing us to select from an extensive list of previously registered modules in the application, and it can be responsible for downloading the module, compiling it and installing it in the existing kernel.

Also from the command line, we can simply specify (m-a is equivalent to module-assistant):

```
# m-a prepare
# m-a auto-install module_name
```

which prepares the system for possible dependencies, downloads the module sources, compiles them and, if there are no problems, installs them for the current kernel. We can see the name of the module on the interactive list of the module assistant.

4. Patching the kernel

In some cases the application of patches to the kernel [lkm] is also common.

> A patch file in relation to the Linux kernel is an ASCII text file that contains the differences between the original source code and the new code, with additional information on file names and code lines. The patch program (see *man patch*) serves to apply it to the tree of the kernel source code (normally in */usr/src*).

The patches are usually necessary when special hardware requires some modification of the kernel or some bugs (errors) have been detected subsequent to a wide distribution of a kernel version or else a new specific feature is to be added. In order to correct the problem (or add the new feature), it is usual to distribute a patch instead of an entire new kernel. When there are already several of these patches, they are added to various improvements of the preceding kernel to form a new version of the kernel. In all events, if we have problematic hardware or the error affects the functioning or stability of the system and we cannot wait for the next version of the kernel; we will have to apply the patch.

The patch is usually distributed in a compressed file of the type bz2 (bunzip2, although you can also find it in gzip with the extension .gz), as in the case of for example:

```
patchxxxx-2.6.21-pversion.bz2
```

where xxxx is usually any message regarding the type or purpose of the patch 2.6.21 would be the version of the kernel to which the patch is to be applied, and pversion would refer to the version of the patch, of which there can also be several. We need to bear in mind that we are speaking of applying patches to the sources of the kernel (normally installed, as we have already seen, in /usr/src/linuxor a similar directory).

Once we have the patch, we must apply it, we will find the process to follow in any readme file that accompanies the patch, but generally the process follows the steps (once the previous requirements are checked) of decompressing the patch in the source files directory and applying it over the sources of the kernel, for example:

```
cd /usr/src/linux (or /usr/src/linux-2.6.21 or any other ver-
sion).
```

```
bunzip2 patch-xxxxx-2.6.21-version.bz2
patch -p1 < patch-xxxxx-2.6.21-version
```

and afterwards we will have to recompile the kernel in order to generate it again.

The patches can be obtained from different places. Normally, we can find them in the kernel storage site (www.kernel.org) or else in www.linuxhq.com, which has a complete record of them. Some Linux communities (or individual users) also offer corrections, but it is better to search the standard sites in order to ensure that the patches are trustworthy and to avoid possible security problems with "pirate" patches. Another way is the hardware manufacturer, which may offer certain modifications of the kernel (or controllers) so that its devices work better (one known example is Linux NVIDIA and the device drivers for its graphic cards).

Finally, we should point out that many of the GNU/Linux distributions (Fedora/Red Hat, Mandriva...), already offer the kernels patched by themselves and systems for updating them (some even automatically, as in the case of Fedora/Red Hat and Debian). Normally, in production systems it is more advisable to keep up with the manufacturer's updates, although it does not necessarily offer the latest published kernel, but rather the one that it finds most stable for its distribution, at the expense of missing the latest generation features or technological innovations included in the kernel.

Note

For systems that we want to update, for testing reasons or because we need the latest features, we can always go to www.kernel.org and obtain the latest published kernel.

5. Kernel modules

The kernel is capable of loading dynamic portions of code (modules) on demand [Hen], in order to complement its functionality (this possibility is available from kernel version 1.2 and higher). For example, the modules can add support for a file system or for specific hardware devices. When the functionality provided by the module is not necessary, the module can be downloaded, freeing up memory.

On demand, the kernel usually identifies a characteristic not present in the kernel at that moment it makes contact with a thread of the kernel known as kmod (in kernel versions 2.0.x the daemon was called *kerneld*), this executes a command, modprobe, to try and load the associated module from or of a chain with the name of the module or else from an generic identifier; this information is found in the file /etc/modules.conf in the form of an alias between the name and the identifier.

Next, we search in `/lib/modules/version_kernel/modules.dep`

to find out whether there are dependencies with other modules. Finally, with the insmod command the module is loaded from /lib/modules/version_kernel/ (the standard directory for modules), the version_kernel is the current version of the kernel using the uname -r command in order to set it. Therefore, the modules in binary form are related to a specific version of the kernel, and are usually located in /lib/modules/version-kernel.

If we need to compile them, we will need to have the sources and/or headers of the version of the core for which it is designed.

Note

The modules offer the system a large degree of flexibility, allowing it to adapt to dynamic situations.

There are some utilities that allow us to work with modules (they usually appear in a software package called *modutils*, which was replaced by the module -init-tools for managing modules of the 2.6.x branch):

- **lsmod**: we can see the loaded modules in the kernel (the information is obtained from the pseudofile /proc/modules). It lists the names and dependencies with others (in [ ]), the size of the module in bytes, and the module use counter; this allows it to be downloaded if the count is zero.

Example

Some modules in a Debian distribution:

Module	Size	Used by	Tainted: P
agpgart	37.344	3	(autoclean)
apm	10.024	1	(autoclean)
parport_pc	23.304	1	(autoclean)
lp	6.816	0	(autoclean)
parport	25.992	1	[parport_pc lp]
snd	30.884	0	
af_packet	13.448	1	(autoclean)
NVIDIA	1.539.872	10	
es1371	27.116	1	
soundcore	3.972	4	[snd es1371]
ac97_codec	10.9640	0	[es1371]
gameport	1.676	0	[es1371]
3c59x	26.960	1	

• **modprobe**: tries the loading of a module and its dependencies.

• **insmod**: loads a specific module.

• **depmod**: analyses dependencies between modules and creates a file of dependencies.

• **rmmod**: removes a module from the kernel.

• Other commands can be used for debugging or analysing modules, like *modinfo*, which lists some information associated to the module or *ksyms*, which (only in versions 2.4.x) allows examination of the symbols exported by the modules (also in /proc/ksyms).

In order to load the module the name of the module is usually specified, either by the kernel itself or manually by the user using *insmod* and specific parameters optionally. For example, in the case of devices, it is usual to specify the addresses of the I/O ports or IRQ or DMA resources. For example:

```
insmod soundx io = 0x320 irq = 5
```

6. Future of the kernel and alternatives

At certain moments, advances in the Linux kernel were released at very short intervals, but now with a fairly stable situation regarding the kernels of the 2.6.x series, more and more time elapses between kernel versions, which in some ways is very positive. It allows time for correcting errors, seeing what ideas did not work well, and trying new ideas, which, if they work, are included.

In this section, we'll discuss some of the ideas of the latest kernels and some of those planned for the near future in the development of the kernel.

Note

The kernel continues to evolve, incorporating the latest in hardware support and improved features.

The previous series, series 2.4.x [DBo], included in most current distributions, contributions were made in:

- Fulfilling IEEE POSIX standards, this means that many existing UNIX programs can be recompiled and executed in Linux.
- Improved devices support: PnP, USB, Parallel Port, SCSI...
- Support for new file systems, like UDF (CD-ROM rewritable like a disc). Other journaled systems, like Reiser from IBM or the ext3, these allow having a log (*journal*) of the file system modifications and thus they are able to recover from errors or incorrect handling of files.
- Memory support up to 4 GB, in its day some problems arose (with the 1.2x kernels) which would not support more memory than 128 MB (at that time it was a lot of memory).
- The /proc interface was improved. This is a pseudo-filesystem (the directory /proc) that does not really exist on the disc, but that is simply a way of accessing the data of the kernel and of the hardware in an organised manner.
- Sound support in the kernel: Alsa controllers, which were configured separately beforehand, were partially added,.
- Preliminary support for RAID software and the dynamic volumes manager LVM1 was included.

In the current series, kernel branch 2.6.x [Pra] has made important advances in relation to the previous one (with the different.x revisions of the 2.6 branch):

- Improved SMP features, important for the multi-core processors widely used in business and scientific environments.

- Improvements in the CPU scheduler.

- Improvements in the multithread support for user applications. New models of threads NGPT (IBM) and NPTL (Red Hat) are incorporated (over time NPTL was finally consolidated).

- Support for USB 2.0.

- Alsa sound controllers incorporated in the kernel.

- New architectures for 64-bit CPUs, supporting AMD x86_64 (also known as amd64) and PowerPC 64 and IA64 (Intel Itanium architecture).

- Support for journaled file systems: JFS, JFS2 (IBM), and XFS (Silicon Graphics).

- Improved I/O features, and new models of unified controllers.

- Improvements in implementing TCP/IP, and the NFSv4 system (sharing of the file system with other systems via the network).

- Significant improvements for a preemptive kernel: allowing the kernel to manage internally various tasks that can interrupt each other, essential for the efficient implementation of real time systems.

- System suspension and restoration after rebooting (by kernel).

- UML, User Mode Linux, a sort of virtual Linux machine on Linux that allows us to see a Linux (in user mode) running on a virtual machine. This is ideal for debugging now that a version of Linux can be developed and tested on another system, which is useful for the development of the kernel itself and for analysing its security.

- Virtualisation techniques included in the kernel: the distributions have gradually been incorporating different virtualisation techniques, which require extensions to the kernel; we should emphasise, for example, kernels modified for Xen, or Virtual Server (Vserver).

- New version of the volumes support LVM2.

- New pseudo file system */sys*, designed to include the system information and devices that will be migrating from the */proc* system, leaving the latter

with information regarding the processes and their development during execution.

- FUSE module for implementing file systems on user space (above all the NTFS case).

In the future, improvement of the following aspects is planned:

- Increasing the virtualisation technology in the kernel, for supporting different operating system configurations and different virtualisation technologies, in addition to better hardware support for virtualisation included in the processors that arise with new architectures.

- The SMP support (multi-processor machines) of 64-bit CPUs (Intel's Itanium, and AMD's Opteron), the support of multi-core CPUs.

- Improved file systems for clustering and distributed systems.

- Improvement for kernels optimised for mobile devices (PDA, teléfonos...).

- Improved fulfilment of the POSIX standard etc.

Web site

POSIX specifications
www.UNIX-systems.org/

- Improved CPU scheduling; although in the initial series of the 2.6.x branch many advances were made in this aspect, there is still low performance in some situations, in particular in the use of interactive desktop applications, different alternatives are being studied to improve this and other aspects.

Also, although it is separate from the Linux systems, the FSF (Free Software Foundation) and its GNU project continue working on the project to finish a complete operating system. It is important to remember that the main objective of the GNU project was to obtain a free software UNIX clone and the GNU utilities are just the necessary software for the system. In 1991, when Linux managed to combine its kernel with some GNU utilities, the first step was taken towards the culmination in today's GNU/Linux systems. But the GNU project continues working on its idea to finish the complete system. Right now, they already have a core that can run its GNU utilities. This core is known as Hurd; and a system built with it known as GNU/Hurd. There are already some test distributions, specifically, a Debian GNU/Hurd.

Web site

The GNU project:
http://www.gnu.org/gnu/thegnuproject.html

Hurd was designed as a core for the GNU system around 1990 when its development started, since most of the GNU software had already been developed at the time, and the only thing that was missing was the kernel. It was in 1991 when Linus combined GNU with his Linux kernel that the history of GNU/

Reference

GNU and Linux, by RichardStallman: http://www.gnu.org/gnu/linux-and-gnu.html

Linux systems began. But Hurd continues to develop. The development ideas for Hurd are more complex, since Linux could be considered a conservative design, based on already known and implemented ideas.

Specifically, Hurd was conceived as a collection of servers implemented on a Mach microkernel [Vah96], which is a kernel design of the microkernel type (unlike Linux, which is of the monolithic type) developed by the University of Carnegie Mellon and subsequently by that of Utah. The basic idea was to model the functionalities of the UNIX kernel as servers that would be implemented on a basic Mach kernel. The development of Hurd was delayed while the design of the Mach was being finished and this was finally published as free software, which would allow its use for developing Hurd. At this point, we should mention the importance of Mach, since many operating systems are now based on ideas extracted from it; the most outstanding example is Apple's MacOS X.

The development of Hurd was further delayed due to its internal complexity, because it had several servers with different tasks of the multithread type (execution of multiple threads), and debugging was extremely difficult. But nowadays, the first production versions of GNU/Hurd are already available, as well as test versions of a GNU/Hurd distribution.

It could be that in the not too distant future GNU/Linux systems will coexist with GNU/Hurd, or even that the Linux kernel will be replaced with the Hurd kernel, if some lawsuits against Linux prosper (read the case of SCO against IBM), since it would represent a solution for avoiding later problems. In all events, both systems have a promising future ahead of them. Time will tell how the balance will tip.

7. Tutorial: configuring de kernel to the requirements of the user

In this section we will have a look at a small interactive workshop for the process of updating and configuring the kernel in the two distributions used: Debian and Fedora.

The first essential thing, before starting, is to know the current version of the kernel we have with uname -r, in order to determine which is the the next version that we want to update to or personalise. And the other is to have the means to boot our system in case of errors: the set of installation CDs, the floppy disc (or CD) for recovery (currently the distribution's first CD is normally used) or some Live CD distribution that allows us to access the machine's file system, in order to redo any configurations that may have caused problems. It is also essential to back up our data or important configurations.

We will look at the following possibilities:

1) Updating the distribution's kernel. Automatic case of Debian.

2) Automatic update in Fedora.

3) Adapting a generic kernel (Debian or Fedora). In this last case, the steps are basically the same as those presented in the section on configuration, but we will make a few more comments:

7.1. Configuring the kernel in Debian

In the case of the Debian distribution, the installation can also be done automatically, using the APT packages system. It can be done either from the command line or with graphic APT managers (synaptic, gnome-apt...).

We are going to carry out the installation using the command line with apt-get, assuming that the access to the apt sources (above all to the Debian originals) is properly configured in the /etc/apt/sources.list file. Let's look at the steps:

1) To update the list of packages.

```
# apt-get update
```

2) To list the packages associated with images of the kernel:

```
# apt-cache search linux-image
```

3) To select a version suitable for our architecture (generic, 386/486/686 for Intel, k6 or k7 for amd or in particular for 64Bits versions amd64, intel and amd or ia64, for Intel Itanium). The version is accompanied by kernel version, Debian revision of the kernel and architecture. For example: 2.6.21-4-k7, kernel for AMD Athlon, Debian revision 4 of the kernel 2.6.21.

4) Check for the selected version that the extra accessory modules are available (with the same version number) With apt-cache we will search for whether there are other dynamic modules that could be interesting for our hardware, depending on the version of the kernel to be installed. Remember that, as we saw in the Debian way, there is also the module-assistant utility, which allows us to automate this process after compiling the kernel. If the necessary modules are not supported, this could prevent us from updating the kernel if we consider that the functioning of the problematic hardware is vital for the system.

5) Search, if we also want to have the source code of the kernel, the Linux-source-version (only 2.6.21, that is, the principal numbers) and the corresponding kernel headers, in case we later want to make a personalised kernel: in this case, the corresponding generic kernel patched by Debian.

6) Install what we have decided: if we want to compile from the sources or simply to have the code:

```
# apt-get install linux-image-version
# apt-get install xxxx-modules-version (if some modules are
necessary)
```

and

```
# apt-get install linux-source-version-generic
# apt-get install linux-headers-version
```

7) Install the new kernel, for example in the lilo bootloader (check the boot utility used, some recent Debian versions use grubas boot loader), this is done automatically. If we are asked if the initrd is active, we will have to verify the lilo file (/etc/lilo.conf) and, in the lilo configuration of the new image, include the new line:

initrd = /initrd.img-version (or /boot/initrd.img-version)

once this is configured, we would have to have a a lilo of the mode (fragment), supposing that initrd.img and vmlinuz are links to the position of the files of the new kernel:

```
default = Linux


image = /vmlinuz
   label = Linux
   initrd = /initrd.img
#  restricted
#  alias = 1
image = /vmlinuz.old
   label = LinuxOLD
   initrd = /initrd.img.old
#  restricted
#  alias = 2
```

We have the first image by default, the other is the former kernel. Thus, from the lilo menu we can ask for one or the other or, simply by changing the default, we can recover the former. Whenever we make any changes in /etc/lilo.conf we should not forget to rewrite in the corresponding sector with the command /sbin/lilo or /sbin/lilo -v.

7.2. Configuring the kernel in Fedora/Red Hat

Updating the kernel in the Fedora/Red Hat distribution is totally automatic by means of its package management service or else by means of the graphic programs that the distribution includes for updating; for example, in business versions of Red Hat there is one called up2date. Normally, we will find it in the task bar or in the Fedora/Red Hat system tools menu (check the available utilities in tools/Administration menus, the currently available graphic tools are highly distribution version dependent).

This updating program basically checks the packages of the current distribution against a Fedora/Red Hat database and offers the possibility of downloading the updated packages, including those of the kernel. This Red Hat service for businesses works via a service account and Red Hat offers it for payment. With this type of utilities the kernel is updated automatically.

For example, in figure 10, we can see that once running, a new available version of the kernel has been detected, which we can select for downloading:

Figure 3. The Red Hat updating service (Red Hat Network up2date) shows the available kernel update and its sources.

In Fedora we can either use the equivalent graphic tools or simply use yum directly, if we know that new kernels are available:

```
# yum install kernel kernel-source
```

Once downloaded, we proceed to install it, normally also as an automatic process, whether with grub or lilo as boot managers. In the case of grub, it is usually automatic and leaves a pair of options on the menu, one for the newest version and the other for the old one. For example, in this grub configuration (the file is in /boot/grub/grub.conf or else /boot/grub/menu.lst), we have two different kernels, with their respective version numbers.

```
#file grub.conf
default = 1
timeout = 10
splashimage = (hd0,1)/boot/grub/splash.xpm.gz

title Linux (2.6.20-2945)
root (hd0,1)
kernel /boot/vmlinuz-2.6.20-2945 ro root = LABEL = /
initrd /boot/initrd-2.6.20-18.9.img

title LinuxOLD (2.6.20-2933)
root (hd0,1)
kernel /boot/vmlinuz-2.4.20-2933 ro root = LABEL = /
initrd /boot/initrd-2.4.20-2933.img
```

Each configuration includes a title that appears during start up. The root or partition of the disc from where it boots, the directory where the file corresponding to the kernel is found and the corresponding initrd file.

In the case of having lilo (by default grub is used) in the Fedora/Red Hat as manager, the system will also update it (file /etc/lilo.conf), but then we will have to rewrite the boot manually with the command /sbin/lilo.

It is also important to mention that with the previous installation we had the possibility of downloading the sources of the kernel; these, once installed, are in/usr/src/linux-version and can be compiled and configured following the usual procedure as if it was a generic kernel. We should mention that the Red Hat company carries out a lot of work on the patches and corrections for the kernel (used after Fedora) and that its kernels are modifications to the generic standard with a fair number of additions, which means that it could be better to use Red Hat's own sources, unless we want a newer or more experimental kernel than the one supplied.

7.3. Configuring a generic kernel

Let's look at the general case of installing a kernel starting from its sources. Let's suppose that we have some sources already installed in /usr/src (or the corresponding prefix). Normally, we would have a Linux directory or linux-version or simply the version number. This will be the tree of the sources of the kernel.

These sources can come from the distribution itself (or we may have downloaded them during a previous udpate), first it will be interesting to check whether they are the latest available, as we have already done before with Fedora or Debian. Or if we want to have the latest and generic versions, we can go to kernel.org and download the latest available version (better the stable one than the experimental ones), unless we are interested in the kernel's development. We download the file and in /usr/src (or another selected directory, even better) decompress the kernel sources. We can also search to see if there are patches for the kernel and apply them (as we have seen in section 4.4).

Next, we will comment on the steps that will have to be carried out: we will do it briefly, as many of them have been mentioned before when working on the configuration and tailoring.

See also

It would be advisable to reread section 3.4.3.

1) Cleaning the directory of previous tests (where applicable):

```
make clean mrproper
```

2) Configuring the kernel with, for example: *make menuconfig (or xconfig, gconfig or oldconfig)*. We saw this in section 4.3.

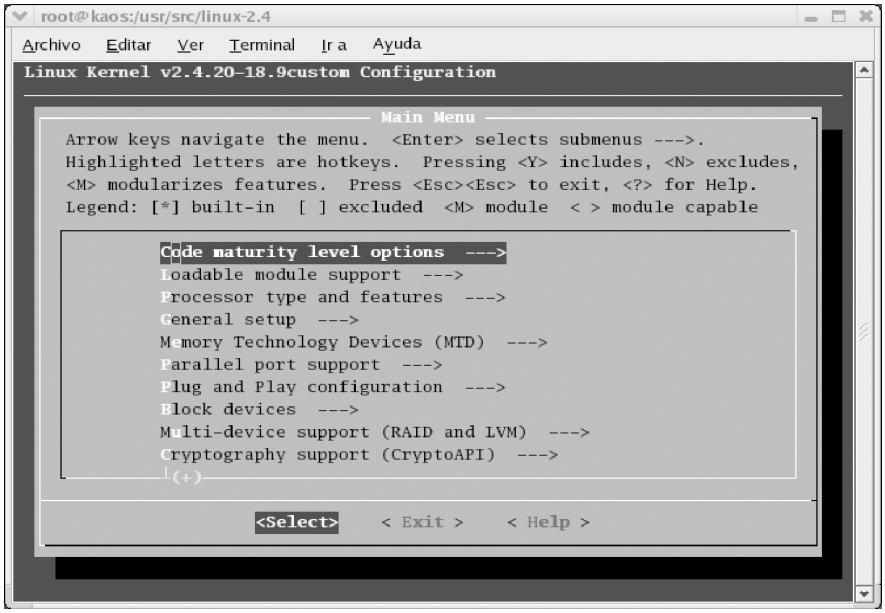

Figure 4. Configuring the kernel using text menus

4) Dependencies and cleaning of previous compilations:

```
make dep
```

5) Compiling and creating an image of the kernel: make bzImage. zImage would also be possible if the image was smaller, but *bzImage* is more normal, as it optimises the loading process and compression of larger kernels. On some ancient hardware it may not work and *zImage* may be necessary. The process can last from a few minutes to an hour on modern hardware and hours on older hardware. When it finishes, the image is found in: /usr/src/directory-sources/arch/i386/boot.

6) Now we can compile the modules with *make modules*. Until now we have not changed anything in our system. Now we have to proceed to the installation.

7) In the case of the modules, if we try an older version of the kernel (branch 2.2 or the first ones of 2.4), we will have to be careful, since some used to overwrite the old ones (in the last 2.4.x or 2.6.x it is no longer like this).

But we will also need to be careful if we are compiling a version that is the same (exact numbering) as the one we have (the modules are overwritten), it is better to back up the modules:

```
cd /lib/modules
tar -cvzf old_modules.tgz versionkernel-old/
```

This way we have a version in .tgz that we can recover later if there is any problem And, finally, we can install the modules with:

```
make modules install
```

8) Now we can move on to installing the kernel, for example with:

```
# cd /usr/src/directory-sources/arch/i386/boot
# cp bzImage /boot/vmlinuz-versionkernel
# cp System.map /boot/System.map-versionkernel
# ln -s /boot/vmlinuz-versionkernel /boot/vmlinuz
# ln -s /boot/System.map-versionkernel /boot/System.map
```

This way we store the symbols file of the kernel (System.map) and the image of the kernel.

9) Now all we have to do is put the required configuration in the configuration file of the boot manager, whether lilo (/etc/lilo.conf) or grub (/boot/grub/grub.conf) depending on the configurations we already saw with Fedora or Debian. And rememeber, in the case of lilo, that we will need to update the configuration again with /sbin/lilo or /sbin/lilo -v.

10) Restart the machine and observe the results (if all has gone well).

Activities

1) Determine the current version of the Linux kernel incorporated into our distribution. Check the available updates automatically, whether in Debian (*apt*) or in Fedora/Red Hat (via yum).

2) Carry out an automatic update of our distribution. Check possible dependencies with other modules used (whether pcmcia or others) and with the bootloader (lilo or grub) used. A backup of important system data (account users and modified configuration files) is recommended if we do not have another sytem that is available for tests.

3) For our branch of the kernel, to determine the latest available version (consult http://www.kernel.org) and carry out a manual installation following the steps described in the unit. The final installation can be left optional, or else make an entry in the bootloader for testing the new kernel.

4) In the case of the Debian distribution, in addition to the manual steps, we saw how there is a special way (recommended) of installing the kernel from its sources using the kernel-package.

Bibliography

Other sources of reference and information

[Kerb] Site that provides a repository of the different versions of the Linux kernel and its patches.

[Kera] [lkm] Web sites that refer to a part of the Linux kernel community. It offers various documentary resources and mailing lists of the kernel's evolution, its stability and the new features that develop.

[DBo] Book about the Linux 2.4 kernel, which details the different components, their implementation and design. There is a first edition about the 2.2 kernel and a new update to the 2.6 kernel.

[Pra] An article that describes some of the main innovations of the new 2.6 series of the Linux kernel.

[Ker] [Mur] Documentation projects of the kernel, incomplete but with useful material.

[Bac86] [Vah96] [Tan87] Some texts about the concepts, design and implementation of the kernels of different UNIX versions.

[Skoa][Zan01][Kan][Pro] For further information on lilo and grub loaders.

Local administration

Josep Jorba Esteve

PID_00148465

Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction

One of the administrator's first tasks will be to manage the machine's local resources. Some of these aspects were basically covered in the GNU/Linux course. In this course, we will cover these management tasks in more depth as well as some of the customisation and resource efficiency aspects.

We will start by analysing the process for starting up a GNU/Linux system, which will help us to understand the initial structure of the system and its relationship with the various services that it provides.

Note

Local administration covers many varied tasks, which are possibly the ones that the administrator will most use during their daily routines.

We will now learn how to obtain a general overview of the current state of the system, using different procedures and commands that are available for evaluating the various parts of the system; this will allow us to make administrative decisions if we detect any faults or deficiencies in the performance or if we find that we are missing any of the resources.

One of the administrator's main tasks is managing the user accounts, as any configuration of the machine will be designed for the users; we will see how we can define new user accounts and control the levels to which they may access the resources.

With regard to the system's peripherals, such as disks and printers, there are different management possibilities available, either through different servers (for printing) or different filing systems that we can treat, as well as some techniques for optimising the disks' performance.

We will also examine the need to update the system and how best to keep it updated; likewise, we will examine how to install new applications and software and how to make these programs available to the users. At the same time, we will analyse the problems involved in executing predetermined timed tasks in the system.

In the last tutorial, we will learn how to evaluate the state of a machine, following the points that we have seen in this module, and we will carry out some of the basic administrative tasks we have described. In this module, we will discuss some of the commands and subsequently, in the tutorial, we will examine some of these in more detail, with regard to how they work and the options available.

1. Distributions: special features

We will now try to outline some minor technical differences (which are constantly being reduced) in the distributions (Fedora/Red Hat and Debian) used [Mor03], which we will examine in more detail throughout the modules as they appear.

Modifications to or particularities of Fedora/Red Hat:

- Using the grub boot loader (a GNU utility); unlike previous versions of most distributions, which tend to use lilo, Fedora uses grub. GRUB (*grand unified bootloader*) has a text-mode configuration (usually in /boot/grub/grub.conf) that is quite simple and that can be modified when booting. It is possibly more flexible than lilo. Lately, distributions tend to use grub; Debian also includes it as an option.

Note

It is important to know the details of a distribution, as they are essential for performing a task or resolving an issue (for example, if there are extra tools available).

- Management of alternatives. If there is more than one equivalent program present for a specific task, the alternative that will be used must be indicated through a directory (/etc/alternatives). This system was borrowed from Debian, which uses it a lot in its distribution.

- TCP/IP portscanning program based on xinetd; in /etc/xinetd.d we will find the modular configuration files for some of the TCP/IP services, along with the /etc/xinetd.conf. configuration file. In classic UNIX systems, the program used for this was inetd, which had a single configuration file in /etc/inetd.conf, which was the case, for example, in the Debian distribution, which uses inetd, leaving xinetd as an option.

- Some special configuration directories: /etc/profile.d, files that are executed when a user opens a shell; /etc/xinetd.d, configuration of some net services; /etc/sysconfig, configuration data for various aspects of the system; /etc/cron., various directories where the tasks that have to be performed regularly are specified (through crontab); /etc/pam.d, where the authentication modules are known as PAM: the permissions for the particular service or program are configured in each of the PAM files; /etc/logrotate.d, rotation configuration (when it is necessary to clean, compress etc.) of some of the log files for different services.

- There is a software library called kudzu, which examines the hardware at start-up to detect any possible changes (in some previous versions of Fedora) in the configuration and to create the appropriate elements or configurations. Although there is currently a progressive migration to API Hal that controls precisely this aspect.

In Debian's case:

- In-house packaging system based on DEB packages, with tools at various levels for working with packages such as: dpkg, apt-get, dselect, tasksel.

- Debian follows FHS, over the directories structure, adding some particulars in /etc, such as: /etc/default, configuration files and default values for some programs; /etc/network, data and network interfaces configuration scripts; /etc/dpkg y /etc/apt, information on the configuration of the package management tools; /etc/alternatives, links to the default programs, in which there are (or may be) various available alternatives.

- Configuration system for many software packages using the dpkg-reconfigure tool. For example:

  ```
  dpkg-reconfigure gdm
  ```

  makes it possible to select the incoming manager for X, or:

  ```
  dpkg-reconfigure X-Window-system
  ```

  allows us to configure the different elements of X.

- Uses the TCP/IP services configuration through inetd; the configuration isin file /etc/inetd.conf; there is an update-inetd tool for disabling or creating services entries.

- Some special configuration directories: /etc/cron., several directories where the tasks that have to be performed regularly are specified (though crontab); /etc/pam.d, where PAM are authentication modules.

2.Boot and run levels

A first important point in the analysis of a system's local performance is how it works on the runlevels, which determine the current work mode of the system and the services provided (on the level) [Wm02].

A service is a functionality provided by the machine, normally based on daemons (or background execution processes that control network requests, hardware activity or other programs that provide any task).

The services can be activated or halted using scripts. Most standard processes, which are usually configured in the /etc directory, tend to be controlled with the scripts in /etc/init.d/. Scripts with names similar to those of the service to which they correspond usually appear in this directory and starting or stopping parameters are usually accepted. The following actions are taken:

`/etc/init.d/service start`	start the service.
`/etc/init.d/service stop`	stop the service.
`/etc/init.d/service restart`	stop and subsequent
	restart of the service.

When a GNU/Linux system starts up, first the system's kernel is loaded, then the first process begins; this process is called init and it has to execute and activate the rest of the system, through the management of different runlevels.

A runlevel is basically a configuration of programs and services that will be executed in order to carry out determined tasks.

The typical levels, although there may be differences in the order, especially at levels 2-5 (in the configuration table in Fedora and that recommended in the LSB standard), are usually:

Runlevel	Function	Description
0	Halt	Halts or shuts down the active services and programs, and umounts active file systems for CPU.
1	Single-user mode	Halts or shuts down most services, only permitting the (root) administrator to login. Used for maintenance tasks and correcting critical errors.
2	Multi-user mode without networking	No networking services are started and only local logins are allowed.
3	Multi-user	Starts up all the services except the graphics associated to X Window.

Runlevel	Function	Description
4	Multi-user	Not usually used; normally the same as 3.
5	Multi-user X	As with 3, but with X support for user logins (graphic login).
6	Reboot	For all programs and services. Reboots the system.

On the other hand, it should be noted that Debian uses a model in which practically no distinction is made between runlevels 2-5 and performs exactly the same task (although this may change in a future version, so that these levels correspond with the LSB).

These runlevels are usually configured in GNU/Linux systems (and UNIX) by two different systems: BSD or System V (sometimes abbreviated to sysV). In the cases of Fedora and Debian, System V is used, which is the one that we will examine, but other UNIX and some GNU/Linux distributions (such as Slackware) use the BSD model.

In the case of the runlevel model of System V, when the init process begins, it uses a configuration file called /etc/inittab to decide on the execution mode it will enter. This file defines the runlevel by default (*initdefault*) at start-up (by installation, 5 in Fedora and 2 in Debian), and a series of terminal services that must be activated so that users may log in.

Afterwards, the system, according to the selected runlevel, will consult the files contained in /etc/rcn.d, where *n* is the number associated to the runlevel (the selected level), which contains a list of services that should be started or halted if we boot up in the runlevel or abandon it. Within the directory, we will find a series of scripts or links to the scripts that control the service.

Each script has a number related to the service, an S or K initial that indicates whether it is the script for starting (S) or killing (K) the service, and a number that shows the order in which the services will be executed.

A series of system commands help us to handle the runlevels; we must mention:

- The scripts, which we have already seen, in /etc/init.d/ allow us to start-up, halt or reboot individual services.

- telinit, allows us to change the runlevel; we simply have to indicate the number. For example, we have to perform a critical task in root; with no users working, we can perform a telinit 1 (S may also be used) to pass to the single-user runlevel and then, after the task, a telinit 3 to return to multi-user mode. The init command may also be used for the same task, although telinit does provide a few extra parameters. For example, the typical reboot of a UNIX system would be performed with sync; init 6, the

sync command forces the buffers of the files system to empty, and then we reboot at runlevel 6.

- shutdown allows us to halt ("h") or reboot the system ("r"). This may be performed in a given period of time or immediately. There are also the halt and reboot commands for these tasks.

- wall allows us to send warning messages to the system users. Specifically, the administrator may warn users that the machine is going to stop at a determined moment. Commands such as shutdown usually use them automatically.

- pidof permits us to find out the process ID associated to a process. With ps we obtain the lists of the processes, and if we wish to eliminate a service or process through a kill, we will need its PID.

There are some small changes in the distributions, with regard to the start-up model:

- Fedora/Red Hat: runlevel 4 has no declared use. The /etc/rcn.d directories exist as links to /etc/rc.d subdirectories, where the start-up scripts are centralised. The directories are as follows: /etc/rc.d/rcn.d; but as the links exist, it is transparent to the user. The default runlevel is 5 when starting up with X.
  The commands and files associated to the system's start-up are in the sysvinit and initscripts software packages.
  Regarding the changes to files and scripts in Fedora, we must point out that in /etc/sysconfig we can find files that specify the default values for the configuration of devices or services. The /etc/rc.d/rc.sysinit script is invoked once when the system starts-up; The /etc/rc.d/rc.local script is invoked at the end of the process and serves to indicate the machine's specific boots.
  The real start-up of the services is carried out through the scripts stored in /etc/rc.d/init.d. There is also a link from /etc/init.d. In addition, Fedora provides some useful scripts for handling the services: /sbin/service to halt or start-up a service by the name; and /sbin/chkconfig, to add links to the S and K files that are necessary for a service or to obtain information on the services.

- Debian has management commands for the runlevels such as update-rc.d, that allows us to install or delete services by booting them or halting them in one or more runlevels; invoke-rc.d, allows the classic operations for starting-up, halting or rebooting the service.
  The default runlevel in Debian is 2, the X Window System is not managed from /etc/inittab; instead there is a manager (for example, gdm or kdm) that works as if it were another of the services of runlevel 2.

3. Monitoring system state

One of the main daily tasks of the (root) administrator will be to verify that the system works properly and check for any possible errors or saturation of the machine's resources (memory, disks etc.). In the following subsections, we will study the basic methods for examining the state of the system at a determined point in time and how to perform the operations required to avoid any subsequent problems.

In this module's final tutorial, we will carry out a full examination of a sample system, so that we may see some of these techniques.

3.1. System boot

When booting a GNU/Linux system, there is a large extraction of interesting information; when the system starts-up, the screen usually shows the data from the processes detecting the machine's characteristics, the devices, system services boots etc., and any problems that appear are mentioned.

In most distributions, this can be seen directly in the system's console during the booting process. However, either the speed of the messages or some of the modern distributions that hide the messages behind graphics can stop us from seeing the messages properly, which means that we need a series of tools for this process.

Basically, we can use:

- dmesg command: shows the messages from the last kernel boot.
- /var/log/messages file: general system log that contains the messages generated by the kernel and other daemons (there may be many different log files, normally in /var/log, and depending on the configuration of the *syslog* service).
- uptime command: indicates how long the system has been active.
- /proc system: pseudo file system (procfs) that uses the kernel to store the processes and system information.
- /sys system: pseudo file system (sysfs) that appeared in the kernel 2.6.x branch to provide a more coherent method of accessing the information on the devices and their drivers.

3.2. kernel: /proc directory

When booting up, the kernel starts up a pseudo-file system called /proc, in which it dumps the information compiled on the machine, as well as many other internal data, during the execution. The /proc directory is implemented on memory and not saved to disk. The contained data are both static and dynamic (they vary during execution).

It should be remembered that, as /proc heavily depends on the kernel, the structure tends to depend on the system's kernel and the included structure and files can change.

One of the interesting points is that we can find the images of the processes that are being executed in the */proc* directory, along with the information that the kernel handles on the processes. Each of the system's processes can be found in the */proc/<process_pid*, directory, where there is a directory with files that represent its state. This information is basic for debugging programs or for the system's own commands such as *ps* or *top*, which can use it for seeing the state of the processes. In general, many of the system's utilities consult the system's dynamic information from */proc* (especially some of the utilities provided in the *procps* package).

Note

The /proc directory is an extraordinary resource for obtaining low-level information on the system's working and many system commands rely on it for their tasks.

On another note, we can find other files on the global state of the system in */proc*. We will look at some of the files that we can examine to obtain important information briefly:

File	Description
/proc/bus	Directory with information on the PCI and USB buses.
/proc/cmdline	*Kernel startup line*
/proc/cpuinfo	CPU data
/proc/devices	List of system character devices or block devices
/proc/drive	Information on some hardware kernel modules
/proc/filesystems	Systems of enabled files in the kernel
/proc/ide	Directory of information on the IDE bus, disks characteristics
/proc/interrups	Map of the hardware interrupt requests (IRQ) used
/proc/ioports	I/O ports used
/proc/meminfo	Data on memory usage
/proc/modules	Modules of the kernel
/proc/mounts	File systems currently mounted

File	Description
/proc/net	Directory with all the network information
/proc/scsi	Directory of SCSI devices or IDEs emulated by SCSI
/proc/sys	Access to the dynamically configurable parameters of the kernel
/proc/version	Version and date of the kernel

As of kernel version 2.6, a progressive transition of procfs (*/proc*) to sysfs (*/sys*) has begun, in order to migrate all the information that is not related to the processes, especially the devices and their drivers (modules of the kernel) to the */sys system*.

3.3. kernel: /sys

The sys system is in charge of making the information on devices and drivers, which is in the kernel, available to the user space so that other APIs or applications can access the information on the devices (or their drivers) in a more flexible manner. It is usually used by layers such as HAL and the udev service for monitoring and dynamically configuring the devices.

Within the sys concept there is a tree data structure of the devices and drivers (let us say the fixed conceptual model) and how it can subsequently be accessed through the sysfs file system (the structure of which may change between different versions).

When an added object is detected or appears in the system, a directory is created in sysfs in the driver model tree (drivers, devices including their different classes). The parent/child node relationship is reflected with subdirectories under /sys/devices/ (reflecting the physical layer and its identifiers). Symbolic links are placed in the /sys/bus subdirectory reflecting the manner in which the devices belong to the different physical buses of the system. And the devices are shown in /sys/class, grouped according to their class, for example network, whereas /sys/block/ contains the block devices.

Some of the information provided by /sys can also be found in /proc, but it was decided that this method involved mixing different elements (devices, processes, data, hardware, kernel parameters) in a manner that was not very coherent and this was one of the reasons that /sys was created. It is expected that the information will migrate from /proc to /sys to centralise the device data.

3.4. Processes

The processes that are executing at a given moment will be of a different nature, generally. We may find:

- **System processes**, whether they are processes associated to the machine's local workings, kernel, or processes (known as daemons) associated to the control of different services. On another note, they may be local or networked, depending on whether the service is being offered (we are acting as a server) or we are receiving the results of the service (we are acting as clients). Most of these processes will appear associated to the root user, even if we are not present at that moment as users. There may be some services associated to other system users (lp, bin, www, mail etc.), which are virtual non-interactive users that the system uses to execute certain processes.

- **Processes of the administering user**: when acting as the root user, our interactive processes or the launched applications will also appear as processes associated to the root user.

- **System users processes**: associated to the execution of their applications, whether they are interactive tasks in text mode or in graphic mode.

We can use the following as faster and more useful:

- *ps*: the standard command, list of processes with the user data, time, process identifier and the command line used. One of the most commonly used options is *ps -ef* (or *-ax*), but there are many options available (see man).

- *top*: one version that provides us with an updated list by intervals, dynamically monitoring the changes. And it allows us to order the list of processes sorted by different categories, such as memory usage, CPU usage, so as to obtain a ranking of the processes that are taking up all the resources. It is very useful for providing information on the possible source of the problem, in situations in which the system's resources are all being used up.

- *kill*: this allows us to eliminate the system's processes by sending commands to the process such as *kill -9 pid_of_process* (9 corresponding to SIGKILL), where we set the process identifier. It is useful for processes with unstable behaviour or interactive programs that have stopped responding. We can see a list of the valid signals in the system with man 7 signal

3.5. System Logs

Both the kernel and many of the service daemons, as well as the different GNU/Linux applications or subsystems, can generate messages that are sent to log files, either to obtain the trace of the system's functioning or to detect

errors or fault warnings or critical situations. These types of logs are essential in many cases for administrative tasks and much of the administrator's time is spent processing and analysing their contents.

Most of the logs are created in the /var/log directory, although some applications may modify this behaviour; most of the logs of the system itself are located in this directory.

A particular daemon of the system (important) is daemon *Syslogd*. This daemon is in charge of receiving the messages sent by the kernel and other service daemons and sends them to a log file that is located in /var/log/messages. This is the default file, but Syslogd is also configurable (in the /etc/syslog.conf file), so as to make it possible to create other files depending on the source, according to the daemon that sends the message, thereby sending it to the log or to another location (classified by source), and/or classify the messages by importance (priority level): *alarm, warning, error, critical* etc.

Note

The Syslogd daemon is the most important service for obtaining dynamic information on the machine. The process of analysing the logs helps us to understand how they work, the potential errors and the performance of the system.

Depending on the distribution, it can be configured in different modes by default; in /var/log in Debian it is possible to create (for example) files such as: *kern.log, mail.err, mail.info...* which are the logs of different services. We can examine the configuration to determine where the messages come from and in which files they are saved. An option that is usually useful is the possibility of sending the messages to a virtual text console (in /etc/syslog.conf the destination console, such as */dev/tty8 or /dev/xconsole*, is specified for the type or types of message), so that we can see the messages as they appear. This is usually useful for monitoring the execution of the system without having to constantly check the log files at each time. One simple modification to this method could be to enter, from a terminal, the following instruction (for the general log):

```
tail -f /var/log/messages
```

This sentence allows us to leave the terminal or terminal window so that the changes that occur in the file will progressively appear.

Other related commands:

- *uptime*: time that the system has been active. Useful for checking that no unexpected system reboot has occurred.

- *last*: analyses the in/out log of the system (/var/log/wtmp) of the users, and the system reboots. Or last log control of the last time that the users were seen in the system (information in /var/log/lastlog).

- Various utilities for combined processing of logs, that issue summaries (or alarms) of what has happened in the system, such as: logwatch, logcheck (Debian), log_analysis (Debian)...

3.6. Memory

Where the system's memory is concerned, we must remember that we have: a) the physical memory of the machine itself, b) virtual memory that can by addressed by the processes. Normally (unless we are dealing with corporate servers), we will not have very large amounts, so the physical memory will be less than the necessary virtual memory (4GB in 32bit systems). This will force us to use a swap zone on the disk, to implement the processes associated to the virtual memory.

This swap zone may be implemented as a file in the file system, but it is more usual to find it as a *swap* partition, created during the installation of the system. When partitioning the disk, it is declared as a Linux Swap type.

To examine the information on the memory, we have various useful commands and methods:

- */etc/fstab* file: the swap partition appears (if it exists). With an fdisk command, we can find out its size (or check /proc/swaps).

- *ps* command: allows us to establish the processes that we have, with the options on the percentage and memory used.

- *top* command: is a dynamic ps version that is updatable by periods of time. It can classify the processes according to the memory that they use or CPU time.

- *free* command: reports on the global state of the memory. Also provides the size of the virtual memory.

- *vmstat* command: reports on the state of the virtual memory and the use to which it is assigned.

- Some packages, like dstat, allow us to collate data on the different parameters (memory, swap and others) by intervals of time (similar to top).

3.7. Disks and file systems

We will examine which disks are available, how they are organised and which partitions and file systems we have.

When we have a partition and we have a determined accessible file system, we will have to perform a mounting process, so as to integrate it in the system, whether explicitly or as programmed at startup/boot. During the mounting process, we connect the file system associated to the partition to a point in the directory tree.

In order to find out about the disks (or storage devices) present in the system, we can use the system boot information (dmesg), when those available are detected, such as the /dev/hdx for IDE devices or /dev/sdx for SCSI devices. Other devices, such as hard disks connected by USB, flash disks (pen drive types), removable units, external CD-ROMs etc., may be devices with some form of SCSI emulation, so they will appear as devices as this type.

Any storage device will present a series of space partitions. Typically, an IDE disk supports a maximum of four physical partitions or more if they are logical (they permit the placement of various partitions of this type on one physical partition). Each partition may contain different file system types, whether they are of one same operative or different operatives.

To examine the structure of a known device or to change its structure by partitioning the disk, we can use the *fdisk* command or any of its more or less interactive variants (*cfdisk, sfdisk*). For example, when examining a sample disk ide /dev/hda, we are given the following information:

```
# fdisk -j /dev/hda
```

```
Disk /dev/hda: 20.5 GB, 20520493056 bytes 255 heads, 63
sectors/track, 2494 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id System
/dev/hda1	*	1	1305	10482381	7 HPFS/NTFS
/dev/hda2	*	1306	2429	9028530	83 Linux
/dev/hda3		2430	2494	522112+	82 Linux swap

20 GB disk with three partitions (they are identified by the number added to the device name), where we observe two NTFS and Linux-type boot partitions (Boot column with *), which indicates the existence of a Windows NT/2000/XP/Vista along with a GNU/Linux distribution and a last partition that is used as a swap for Linux. In addition, we have information on the structure of the disk and the sizes of each partition.

Some of the disks and partitions that we have, some will be mounted in our file system, or will be ready for set up upon demand, or they may be set up when the resource becomes available (in the case of removable devices).

We can obtain this information in different ways (we will see this in more detail in the final workshop):

- The */etc/fstab* file indicates the devices that are ready to be mounted on booting or the removable devices that may be mounted. Not all of the system devices will appear necessarily; only the ones that we want to appear when booting. We can mount the others upon demand using the mount command or remove them with umount.

- *mount* command. This informs us of the file systems mounted at that moment (whether they are real devices or virtual file systems such as */proc*). We may also obtain this information from the /etc/mtab file.

- *df -k* command. This informs us of the storage file systems and allows us to verify the used space and available space. It's a basic command for controlling the available disk space.

With regard to this last *df -k* command, one of our basic tasks as an administrator of the machine is to control the machine's resources and, in this case, the space available in the file systems used. These sizes have to be monitored fairly frequently to avoid a system crash; a file system must never be left at less than 10 or 15% (especially if it is the /), as there are many process daemons that are normally writing temporary information or logs, that may generate a large amount of information; a particular case is that of the core files that we have already mentioned and which can involve very large files (depending on the process). Normally, some precautions should be taken with regard to system hygiene if any situations of file system saturation are detected:

- Eliminate old temporary files. The /tmp and /var/tmp directories tend to accumulate many files created by different users or applications. Some systems or distributions take automatic hygiene measures, such as clearing /tmp every time the system boots up.

- Logs: avoiding excessive growth, according to the system configuration (for example, *Syslogd*), as the information generated by the messages can be very large. Normally, the system will have to be cleared regularly, when certain amounts of space are taken up and, in any case, if we need the information for subsequent analyses, backups can be made in removable devices. This process can be automated using cron scripts or using specialised tools such as *logrotate*.

- There are other parts of the system that tend to grow a lot, such as: a) user core files: we can delete these periodically or eliminate their generation;

b) the email system: stores all of the emails sent and received; we can ask the users to clean them out regularly or implement a quota system; c) the caches of the browsers or other applications: other elements that usually occupy a lot of space, which require regular clearing, are: d) the accounts of the users themselves: they may have quotas so that pre-established allocated spaces are not exceeded etc.

4. File Systems

In each machine with a GNU/Linux system, we will find different types of file systems [Hin].

To start with, it is typical to find the actual Linux file systems created in various partitions of the disks [Koe]. The typical configuration is to have two partitions: that corresponding to "/" (*root file system*) and that corresponding to the swap file. Although, in more professional configurations, it is usual to separate partitions with "differentiated" parts of the system, a typical technique is, for example (we will examine other options later), to create different partitions so:

/ /boot /home /opt /tmp /usr /var swap

That will certainly be found mounted from different sources (different disks, or even the network in some cases). The idea is to clearly separate the static and dynamic parts of the system, so as to make it easier to extend the partitions when any overload problems arise. Or more easily isolate the parts to perform backups (for example, the user accounts in the /home partition).

The swap partitions are Linux swap type partitions and that corresponding to / tends to be one of the standard file systems, either ext2 (the default type up to kernels 2.4), or the new ones ext3, ext4, which is an upgrade of ext2 with journaling, which makes it possible to have a log of what goes on in the file system, for faster recoveries in the event of an error. Other file system types, such as Reiser or XFS are also typical.

Another typical configuration may be that of having three partitions: /, swap, /home, in which the /home will be used for the user accounts. This makes it possible to separate the system's user accounts, isolating two separate partitions and allocating the necessary space for the accounts in another partition.

Another configuration that is widely used is that of separating the static parts of the system from the dynamic ones, in different partitions; for example one partition is used for placing / with the static part (/bin /sbin and /usr in some cases), which is not expected to grow or, if it does, not by much, and another or various partitions with the dynamic part (/var /tmp /opt), supposing that /opt, for example, is the installation point for new software. This makes it possible to better adjust the disk space and to leave more space for the parts of the system that need it.

Where the supported file systems are concerned, we must point out the variety of these; we can currently find (among others):

- Systems associated to GNU/Linux, such as the ext2, ext3 and ext4 standards, developed from the previous concept of journaling (support log for operations performed in the file system that allows us to recover it in the event of any disaster that renders it inconsistent).

- Compatibility with non- GNU/Linux environments: MSDOS, VFAT, NTFS, access to the different systems of FAT16, FAT32 and NTFS. In particular, we must point out that the kernel support, in the case of the kernel, is read-only. But, as we have mentioned, there are user space solutions (through FUSE, a kernel module that allows us to write file systems in the user space), that make read/write possible, such as the abovementioned NTFS-3g. There is also compatibility with other environments such as Mac with HFS and HFSplus.

- Systems associated to physical supports, such as CD/DVDs, for example ISO9660 and UDF.

- Systems used in different Unix, which generally provide better performance (sometimes at the cost of a greater consumption of resources, in CPU for example), such as JFS2 (IBM), XFS (SGI), or ReiserFS.

- Network file systems (more traditional): NFS, Samba (smbfs, cifs), permit us to access the file systems available in other machines transparently using the network.

- Systems distributed in the network: such as GFS, Coda.

- Pseudo file systems, such as procfs (/proc) or sysfs (/sys).

Note

The file systems howto document provides brief explanations of the various file systems as well as the websites that you may consult for each of these.

In most of these file systems (except in some special cases), GNU/Linux will allow us to create partitions of these types, build the file systems of the required type and mount them as an integrating part of the directory tree, either temporarily or permanently.

4.1. Mount point

Apart from the / root file system and its possible extra partitions (/usr /var /tmp /home), it should be remembered that it is possible to leave mount points prepared for mounting other file systems, whether they are disk partitions or other storage devices.

In the machines in which GNU/Linux shares the partition with other operating systems, through some bootloader (lilo or grub), there may be various partitions assigned to the different operating systems. It is often good to share

data with these systems, whether for reading or modifying their files. Unlike other systems (that only register their own data and file systems and in some versions of which some of the actual file systems are not supported), GNU/Linux is able to treat, as we have seen, an enormous amount of file systems from different operating systems and to share the information.

Example

If we have installed GNU/Linux in the PCs, we will certainly find more than one operating system, for example, another version of GNU/Linux with ext2 or 3 of the file system, we may find an old MSDOS with its FAT file system, a Windows98/ME/XP Home with FAT32 (or VFAT for Linux), or a Windows NT/2000/XP/Vista with NTFS systems (NTFS for Linux) and FAT32 (VFAT) at the same time.

Our GNU/Linux system can read data (in other words, files and directories) from all these file systems and write in most of them.

In the case of NTFS, up until certain points, there were problems with writing, which was experimental in most of the kernel drivers that appeared. Due mainly to the different versions of the file system that progressively appeared, as there were two main versions called NTFS and NTFS2, and some extensions such as the so-called dynamic volumes or the encrypted file systems. And accessing with certain drivers caused certain incompatibilities, which could result in data corruption or faults in the file system.

Thanks to FUSE, a module integrated in the kernel (as of version 2.6.11), it has been possible to develop the file systems more flexibly, directly in the user space (in fact, FUSE acts as a "bridge" between the kernel requests, and access from the driver).

Thanks to the features of FUSE, we have more or less complete support for NTFS, (provided Microsoft does not make any further changes to the specifications), especially since the appearance of the driver (based on FUSE) ntfs-3g (http://www.ntfs-3g.org), and the combination with the ntfsprogs utilities.

Depending on the distribution, different ones are used, or we may also create it ourselves. Normally, they exist either as root subdirectories, for example /cdrom /win /floppy or subdirectories within /mnt, the standard mount point (they appear as /mnt/cdrom /mnt/floppy...), or the /media directory, which is lately preferred by the distributions. According to the FHS standard, /mnt should be used for temporary mounting of file systems, whereas /media should be used to mount removable devices.

The mounting process is performed through the mount command, with the following format:

```
mount -t filesystem-type device mount-point
```

The type of file system may be: MSDOS (FAT), VFAT (FAT32), NTFS (NTFS read), ISO9660 (for CD-ROM)... (of the possible ones).

The device is the in point in the /dev directory corresponding to the location of the device, the IDEs had /dev/hdxy where x is a,b,c, or d (1 master, 1 slave, 2 master, 2 slave) e and, the partition number, the SCSI (/dev/sdx) where x is a,b,c,d ... (according to the associated SCSI ID, 0,1,2,3,4 ...).

We will now look at some examples:

```
mount -t iso9660 /dev/hdc /mnt/cdrom
```

This would mount the CD-ROM (if it is the IDE that is in the second IDE in master mode) at point /mnt/cdrom.

```
mount -t iso9660 /dev/cdrom /mnt/cdrom
```

This would mount the CD-ROM; /dev/cdrom is used as a synonym (it is a link) for the device where it is connected.

```
mount -t vfat /dev/fd0H1440 /mnt/floppy
```

This would mount the diskette, /dev/fd0H1440. It would be a high-density (1.44 MB) disk drive A; /dev/fd0 can also be used.

```
mount -t ntfs /dev/hda2 /mnt/winXP
```

This would mount the second partition of the first NTFS-type IDE device (C:) (for example, a Windows XP).

If these partitions are more or less stable in the system (in other words, they are not changed frequently) and we wish to use them, the best thing will be to include the mounts so that they take place during the execution period, when booting the system, through the configuration of file /etc/fstab:

```
# /etc/fstab: Static information on the file system
#
```

#<Sys. files>	<Mount point>	<Type>	<Options>	<dump>	<pass>
/dev/hda2	/	ext3	errors = remountro	0	1
/dev/hdb3	none	swap	sw	0	0
proc	/proc	proc	defaults	0	0
/dev/fd0	/floppy	auto	user,noauto	0	0
/dev/cdrom	/cdrom	iso9660	ro,user,noauto	0	0

/dev/sdb1	/mnt/usb	vfat	user,noauto	0	0

For example, this configuration includes some of the standard systems, such as the root in /dev/hda2, the swap partition that is in hdb3, the proc system (which uses the kernel to save its information). The diskette, the CD-ROM and, in this case, a Flash-type USB disk (which is detected as a SCSI device). In some cases, auto is specified as a type of file system. This permits the autodetection of the file system. If unknown, it is better to indicate it in the configuration and, on another note, the noauto option will mean that it is not always mounted automatically, but upon request (or access).

If we have this information in the file, the mounting process is simplified, as it will take place either on execution, when booting up, or upon demand (noautos). And it may now be performed by simply asking that the mount point or device be mounted:

```
mount /mnt/cdrom
mount /dev/fd0
```

given that the system already has the rest of the information.

The reverse process, umounting, is quite simple, the umount command with the mount point or device:

```
umount /mnt/cdrom
umount /dev/fd0
```

When using removable devices, such as CD-ROMs (or others), eject may be used to extract the physical support:

```
eject /dev/cdrom
```

or, in this case, only:

```
eject
```

The mount and umount commands mount or umount all the available systems. The file */etc/mtab* maintains a list of the mounted systems at a specific point in time, which can be consulted, or a mount, without parameters, may be executed to obtain this information.

4.2. Permissions

Another subject that we will have to control in the cases of files and directories is the permissions that we wish to establish for each of them, whilst remembering that that each file may have a series of permissions: rwxrwxrwx

where they correspond with the owner rwx, the group rwx to which the user belongs, and the rwx for other users. In each one, we may establish the access rights for reading (r), writing (w) or executing (x). In the case of a directory, x denotes the permission for being able to access that directory (with the cd command, for example).

In order to modify the access rights to a directory or file, we have the commands:

- chown: change file owner.
- chgrp: change file owner group.
- chmod: change specific permissions (rwx) of the files.

The commands also provide the -R option, which is recursive if affecting a directory.

5. Users and groups

The users of a GNU/Linux system normally have an associated account (defined with some of their data and preferences) along with an allocated amount of space on the disk in which they can develop their files and directories. This space is allocated to the user and may only be used by the user (unless the permissions specify otherwise).

Among the accounts associated to users, we can find different types:

- The administrator account, with the root identifier, which should only be used for administration operations. The root user is the one with most permissions and complete access to the machine and the configuration files. Consequently, this user is also the one that most damage can cause due to any faults or omissions. It is better to avoid using the root account as if it were that of just another user; it is therefore recommended that it should only be used for administration operations.

- User accounts: the normal accounts for any of the machine's users have the permissions restricted to the use of their account files and to some particular zones (for example, the temporary files in /tmp), and to the use of the particular devices that they have been authorised to use.

- Special service accounts: lp, news, wheel, www-data... accounts that are not used by people but by the system's internal services, which uses them under these user names. Some of the services are also used under the root account.

A user account is normally created by specifying a name (or user identifier), a password and a personal associated directory (the account).

The information on the system's users is included in the following files:

```
/etc/passwd
/etc/shadow
/etc/group
/etc/gshadow
```

Example of some lines of the /etc/passwd:

```
juan:x:1000:1000:Juan Garcia,,,:/home/juan:/bin/bash
root:x:0:0:root:/root:/bin/bash
```

where (if the :: appear together, the box is empty):

- juan: identifier of the user of the system.
- x: encoded user password; if there is an "x" then it is located in the /etc/shadow file.
- 1000: user code, which the system uses as the identity code of the user.
- 1000: code of the main group to which the user belongs, the group's information is in /etc/group.
- Juan García: comment, usually the user's full name.
- /home/juan: personal directory associated to his account.
- /bin/bash: interactive shell that the user uses when interacting with the system, in text mode, or through the graphic shell. In this case, the GNU Bash, which is the shell used by default. The /etc/passwd file used to contain the user passwords in an encrypted form, but the problem was that any user could see this file and, at the time, cracks were designed to try and find out the passwords directly using the encrypted password as the starting point (word encoded with the crypt system).

In order to avoid this, the passwords are no longer placed in this file; only an "x" is, to indicate that they are located in another file, which can only be read by the root user, /etc/shadow, the contents of which may be something similar to the following:

```
juan:a1gNcs82ICst8CjVJS7ZFCVnu0N2pBcn/:12208:0:99999:7:::
```

where the user identifier is located, along with the encrypted password. In addition, they appear as spaces separated by ":":

- Days since 1st January 1970 in which the password was changed for the last time.
- Days left for it to be changed (0 it does not have to be changed).
- Days after which the password must be changed (in other words, change period).
- Days on which the user will be warned before the password expires.
- Days, after expiry, after which the account will be disabled.
- Days since 1st January 1970 that the account has been disabled.
- And a reserved space.

In addition, the encryption codes can be more difficult, as it is now possible to use a system called md5 (it usually appears as an option when installing the system) to protect the users' passwords. We will examine some more details in the unit on security.

In */etc/group* we will find the information on the user groups:

```
jose:x:1000:
```

where we have:

```
name-group:password-group:identifier-of-group:list-users
```

The list of the users in the group may or may not be present; given that this information is already in */etc/passwd*, it is not usually placed in /etc/group. If it is placed there, it usually appears as a list of users separated by commas. The groups may also posses an associated password (although this is not that common), as in the case of the user, there is also a shadow file: /etc/gshadow.

Other interesting files are the ones in /etc/skel directory, which contains the files that are included in each user account when it is created. We must remember that, as we saw with the interactive shells, we could have some configuration scripts that execute when we enter or exit the account. The "skeletons", which are copied in user account when they are created, are saved in the skel directory. The administrator is usually in charge of creating adequate files for the users, providing the necessary execution paths, initialising the system's variables that are needed for the software etc.

We will now see a series of useful commands for the administration of users (we will mention their functions and perform some tests in the workshop):

- useradd: adding a user to the system.
- *userdel*: to delete a user from the system.
- *usermod*: to modify a user of the system.
- groupadd, groupdel, groupmod the same for groups.
- newusers, chpasswd: these can be very useful in large installations with many users, as they allow us to create various accounts from the information entered into a *newusers* file or change the passwords for a large number of users (*chpasswd*).
- *chsh*: to change the user login shell.

- *chfn*: to change the user information present in the /etc/passwd comment file.

- *passwd*: to change a user's password. This may be executed as a user, and it will then ask for the old password and the new one. When doing this, the root account has to specify the user whose password will be changed (otherwise, they would be changing the account's password) and the old password is not necessary. This is perhaps the command that the root most uses, when users forget their old password.

- *su*: a kind of identity change. It is used both by users and by the root to change the current user. In the case of the administrator, it is used quite a lot to test that the user account works properly; there are different variants: su (without parameters, it serves to switch to root user, after identification, making it possible for us to pass, when we are in a user account, to the root account to perform a task). The su iduser sentence (changes the user to iduser, but leaves the environment as it is, in other words, in the same directory...). The su - iduser mandate (which performs a complete substitution, as if the second user had logged in the system).

With regard to the administration of users and groups, what we have mentioned here refers to the local administration of one sole machine. In systems with multiple machines that the users share, a different management system is used for the information on users. These systems, generically called network information systems, such as NIS, NIS+ or LDAP, use databases for storing the information on the users and groups, effectively using servers, where the database and other client machines are stored and where this information can be consulted. This makes it possible to have one single copy of the user data (or various synchronised copies) and makes it possible for them to enter any available machine of the set administered with these systems. At the same time, these systems incorporate additional concepts of hierarchies and/or domains/machine and resource zones, that make it possible to adequately represent the resources and their use in organisations with different organisational structures for their own personnel and internal departments.

We can check whether we are in a NIS-type environment by seeing if *compat* appears in the passwd line and group configuration file, /etc/nsswitch.conf, if we are working with local files, or nis or nisplus according to the system on which we are working. Generally, this does not involve any modification for the simple user, as the machines are managed transparently, more so if it is combined with files shared by NFS that makes the account available, regardless of the machine used. Most of the abovementioned commands can still be used without any problem under NIS or NIS+, in which they are equivalent, except for the command for changing the password, which, instead of *passwd*, we

usually use *yppasswd* (NIS) or *nispasswd* (NIS+); although it is typical for the administrator to rename them to passwd, (through a link), which means that users will not notice the difference.

We will look at this and other methods for configuring the network administration units.

6. Printing services

The GNU/Linux [Gt] [Smi02] printing server derives from UNIX's BSD variant; this system was called LPD (*line printer daemon*). This is a very powerful printing system, because it integrates the capacity to manage both local and network printers. And it provides this service within the system for both the client and the printing server.

LPD is a system that is quite old, as its origins date back to UNIX's BSD branch (mid 1980s). Consequently, LPD usually lacks support for modern devices, given that the system was not originally conceived for the type of printing that takes place now. The LPD system was not designed as a system based on device drivers, as it was typical to produce only printers in series or in parallel for writing text characters.

Currently, the LPD system combines with another common software, such as the Ghostscript system, which offers a postscript type output for a very wide range of printers for which it has the right drivers. At the same time, they are usually combined with filtering software, which, depending on the type of document that must be printed, selects the appropriate filters. Normally, the procedure that should be followed is (basically):

Note

The UNIX systems provide, possibly, the most powerful and complex printing systems, which provide a lot of flexibility to printing environments.

1) The work is started by a command in the LPD system.

2) The filtering system identifies the type of job (or file) that must be used and transforms the job into an outgoing postscript file, which is the one sent to the printer. In GNU/Linux and UNIX, most of the applications assume that the job will be sent to a postscript printer and many of them directly generate a postscript output, which is why the following step needs to be taken.

3) The Ghostscript has to interpret the postscript file it receives, and, depending on the driver of the printer to which the file has been sent, it performs the transformation to the driver's own format. If the printer is a postscript type printer, the printing process is direct; if not, it has to "translate" the job. The job is sent to the printing queue.

Web site

Ghostscript: http://www.ghostscript.com/

Apart from the LPD printing system (that originated with UNIX's BSD), there is also the system known as System V (originally in the other System V branch of UNIX). Normally, for compatibility reasons, most UNIX systems integrate both systems, so that either one or the other is used as the main one and the other emulates the main one. In the case of GNU/Linux, a similar process occurs, depending on the installation that we have, we can have only the LPD commands of the printing system, but it will also be common to have the

System V commands. A simple way of identifying the two systems (BSD or System V) is using the main printing command (which sends the jobs to the system), in BSD, it is Ipr, and it is Ip in System V.

This is the initial situation for the GNU/Linux printing systems, although over the last few years, more systems have appeared, which provide more flexibility and make more drivers available for the printers. The two main systems are CUPS and, to a lesser extent, LPRng. In fact, recently, CUPS is GNU/Linux's de facto standard, although the other systems must be supported for compatibility with the existing UNIX systems.

Web sites

LPRng: http://www.lprng.org
CUPS: http://www.cups.org

Both (both CUPS and LPRng) are a type of higher-level system, but they are not all that perceptibly different for average users, with regard to the standard BSD and System V systems; for example, the same client commands (or compatible commands in the options) are used for printing. There are perceptible differences for the administrator, because the configuration systems are different. In one way, we can consider LPRng and CUPS as new architectures for printing systems, which are compatible for users with regard to the old commands.

In the current GNU/Linux distributions, we can find different printing systems. If the distribution is old, it may only incorporate the BSD LPD system; in the current distributions: both Debian and Fedora/Red Hat use CUPS. In older versions of Red Hat, there was a tool, Print switch, which made it possible to change the system, switching the printing system, although recently only CUPS is available. In Debian, it is possible to install both systems, but they are mutually exclusive: only one may be used for printing.

In the case of Fedora Core, the default printing system is CUPS (as LPRng disappeared in Fedora Core 4), and the Print Switch tool no longer exists, as it is no longer necessary: system-config-printer is used to configure devices. By default, Debian uses BSD LPD, but it is common to install CUPS (and we can expect it to continue to be the default option in future new versions) and LPRng may also be used. In addition, we must remember that we also had the possibility (seen in the unit on migration) of interacting with Windows systems through the Samba protocols, which allowed you to share printers and access to these printers.

Regarding each of the [Gt] systems:

- BSD LPD: this is one of UNIX's standards, and some applications assume that the commands and the printing system will be available, for which both LPRng and CUPS emulate the functions and commands of BDS LPD. The LPD system is usable but not very configurable, especially with regard to access control, which is why the distributions have been moved to other, more modern, systems.

- LPRng: basically it was designed to replace BSD, and therefore, most of the configuration is similar and only some of the configuration files are different.

- CUPS: it is the biggest deviation from the original BSD and the configuration is the same. Information is provided to the applications on the available printers (also in LPRng). In CUPS, both the client and the server have to have CUPS software.

The two systems emulate the printing commands of System V.

For GNU/Linux printing, various aspects have to be taken into account:

- Printing system that is used: BSD, LPRng or CUPS.

- Printing device (printer): it may have a local connection to a machine or be on the network. The current printers may be connected to a machine using local connections, through interfaces in series, in parallel, USB etc. Or they may simply be on the network, as another machine, or with special ownership protocols. Those connected to the network can normally act themselves as a printing server (for example, many HP laser printers are BSD LPD servers) or they can be connected to a machine that acts as a printing server for them.

- Communication protocols used with the printer or the printing system: whether it is direct TCP/IP connection (for example, an HP with LPD) or high level ones based on TCP/IP, such as IPP (CUPS), JetDirect (some HP printers) etc. This parameter is important, as we have to know it so as to install the printer in a system.

- Filtering systems used: each printing system supports one or more.

- Printer drivers: in GNU/Linux, there are quite a few different types; we might mention, for example CUPS drivers, the system's or third parties' (for example, HP and Epson provide them); Gimp, the image editing program also has drivers optimised for printing images; Foomatic is a driver management system that works with most systems (CUPS, LPD, LPRng and others); Ghostscript drivers etc. In almost all printers, there are one or more of the drivers in these sets.

Web site

Information on the most appropriate printers and drivers can be found at: http://www.openprinting.org/printer_list.cgi

With regard to the client part of the system, the basic commands are the same for the different systems and these are the BSD system commands (each system supports emulation of these commands):

- lpr: a job is sent to the default printing queue (or the one that is selected), and the printing daemon (lpd) then sends it to the corresponding queue and assigns a job number, which will be used with the other commands.

Normally, the default printer would be indicated by the PRINTER system variable or the first defined and existing one will be used or, in some systems, the Ip queue will be used (as the default name).

Example

Lpr example:

```
lpr –Pepson data.txt
```

This command sends the data.txt file to the print queue associated to a printer that we have defined as "epson".

- lpq: This allows us to examine the jobs in the queue.

Example

Example

```
# lpq -P epson
```

Rank	Owner	Job Files	Total	Size
1st	juan	15	data.txt	74578 bytes
2nd	marta	16	fpppp.F	12394 bytes

This command shows us the jobs in the queue, with the respective order and sizes; the files may appear with different names, as this depends on whether we have sent them with Ipr or with another application that might change the names when it sends them or if any filters have had to be used when converting them.

- *lprm*: eliminates jobs from the queue and we can specify a job number or the user, to cancel these operations.

Example

```
lprm -Pepson 15
```

Delete the job with id 15 from the queue.

With regard to the administrative side (in BSD), the main command would be *lpc*; this command can be used to activate or deactivate queues, move jobs in the queue order and activate or deactivate the printers (jobs may be received in the queues but they are not sent to the printers).

We should also point out that, in the case of System V, the printing commands are usually also available, normally simulated on the basis of the BSD commands. In the client's case, the commands are: lp, lpstat, cancel and, for administrative subjects, lpadmin, accept, reject, lpmove, enable, disable, lpshut.

In the following sections we will see that it is necessary to configure a printer server for the three main systems. These servers may be used both for local printing and for the network clients' prints (if they are enabled).

6.1. BSD LPD

In the case of the BSD LPD server, there are two main files that have to be examined: on the one hand, the definition of the printers in /etc/printcap and, on the other, the network access permissions in /etc/hosts.lpd.

With regard to the permissions, by default, BSD LPD only provides local access to the printer and, therefore, it has to be expressly enabled in /etc/hosts.lpd.

Example

The file may be:

```
#file hosts.lpd
second
first.the.com
192.168.1.7
+@groupnis
-three.the.com
```

which would indicate that it is possible to print to a series of machines, listed either by their DNS name or by the IP address. Machine groups that belong to a NIS server (groupnis, as shown in the example) may be added or it is possible to deny access to several machines by indicating this with a dash (-).

With regard to the configuration of the server in /etc/printcap, we define inputs, in which each represents a printing system queue that can be used to stop the printing jobs. The queue may be associated to a local device or a remote server, whether this is a printer or another server.

The following options may exist in each port:

- lp =, indicates the device to which the printer is connected, for example, lp = /dev/lp0 would indicate the first parallel port. If the printer is an LPD-type printer, for example, a network printer that accepts the LPD protocol (such as an HP), then we can leave the box empty and fill in the following.

- rm =, address with name or IP of the remote machine that will use the printing queue. If it is a network printer, it will be this printer's address.

- rp =, name of the remote queue, in the machine indicated before with rm.

Let us examine an example::

```
# Local printer input
lp|epson|Epson C62:\
```

```
   :lp=/dev/lp1:sd=/var/spool/lpd/epson:\
   :sh:pw#80:pl#72:px#1440:mx#0:\
   :if = /etc/magicfilter/StylusColor@720dpi-filter:\filter
   :af = /var/log/lp-acct:lf = /var/log/lp-errs:
  # Remote printer input
  hpremote|hpr|remote hp of the department|:\
   :lp = :\
   :rm = server:rp = queuehp:\
   :lf = /var/adm/lpd_rem_errs:\log file.
   :sd = /var/spool/lpd/hpremote:local associated spool
```

6.2. LPRng

In the case of the LPRng system, as this was made to maintain BSD compatibility, and, among other improvements with regard to access, the system is compatible in terms of the configuration of queues and this is performed through the same file format, /etc/printcap, with some additional intrinsic operations.

Where the configuration is different is with regard to access: in this case, we generally obtain access through a /etc/lpd.perms file that is general for the whole system and there may also be individual configurations for each queue with the lpd.perms file placed in the directory corresponding to the queue, usually /var/spool/lpd/name-queue.

These lpd.perms files have a greater capacity for configuring the access and permit the following basic commands:

```
DEFAULT ACCEPT
DEFAULT REJECT
ACCEPT [ key = value[,value]* ]*
REJECT [ key = value[,value]* ]*
```

where the first two allow us to establish the default value, of accepting everything or rejecting everything, and the next two of accepting or rejecting a specific configuration in the line. It is possible to accept (or reject) requests from a specific host, user or IP port. Likewise, it is possible to configure the type of service that will be provided to the element: X (may be connected), P (job printing), Q (examine queue with lpq), M (remove jobs from the queue, lprm), C (control printers, Ipc command lpc), among others, as with the file:

```
ACCEPT SERVICE = M HOST = first USER = jose
ACCEPT SERVICE = M SERVER REMOTEUSER = root
REJECT SERVICE = M
```

Deleting jobs from the queue is allowed for the (first) user of the machine and the root user from the server where the printing service is hosted (*localhost*) and, in addition, whatsoever other requests for deleting jobs from the queue that are not the already established are rejected.

With this configuration, we have to be very careful, because in some distributions, the LPRng services are open by default. The connection may be limited, for example, with:

```
ACCEPT SERVICE = X SERVER
REJECT SERVICE = X NOT REMOTEIP = 100.200.0.0/255
```

Connection service only accessible to the server's local machine and denying access if the machine does not belong to our subnet (in this case, we are assuming that it is 100.200.0.x).

For the administration of line commands, the same tools as the standard BSD are used. With regard to the graphical administration of the system, we should point out the *lprngtool* tool (not available in all versions of the LPRng system).

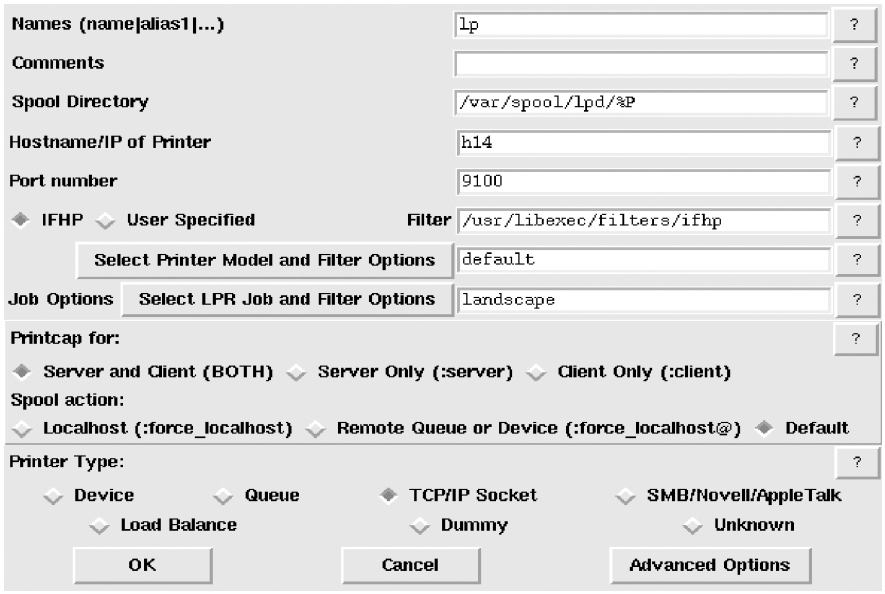

Figure 1. lprngtool, configuration of a printer

There are various software packages related to LPRng; for example, in a Debian, we might find:

```
lprng - lpr/lpd printer spooling system
lprng-doc - lpr/lpd printer spooling system (documentation)
lprngtool - GUI front-end to LPRng based /etc/printcap
printop - Graphical interface to the LPRng print system.
```

6.3. CUPS

CUPS is a new architecture for the printing system that is quite different; it has a layer of compatibility with BSD LPD, which means that it can interact with servers of this type. It also supports a new printing protocol called IPP (based on http), but it is only available when the client and the server are CUPS-type clients and servers. In addition, it uses a type of driver called PPD that identifies the printer's capacities; CUPS comes with some of these drivers and some manufacturers also offer them (HP and Epson).

CUPS has an administration system that is completely different, based on different files: /etc/cups/cupsd.conf centralises the configuration of the printing system, /etc/cups/printers.conf controls the definition of printers and /etc/cups/classes.conf the printer groups.

In /etc/cups/cupsd.conf, we can configure the system according to a series of file sections and the directives of the different actions. The file is quite big; we will mention some important directives:

- Allow: this permits us to specify which machines may access the server, either in groups or individually, or segments of the network's IP.

- AuthClass: makes it possible to indicate whether the user clients will be asked to authenticate their accounts or not.

- BrowseXXX: there is a series of directives related to the possibility of examining a network to find the served printers; this possibility is activated by default (*browsing on*), which means that we will normally find that all the printers available in the network are available. We can deactivate it, so that we only see the printers that we have defined. Another important option is BrowseAllow, which we use to determine who is permitted to ask for our printers; it is activated by default, which means that anyone can see our printer from our network.

We must point out that CUPS is, in principle, designed so that both clients and the server work under the same system; if the clients use LPD or LPRng, it is necessary to install a compatibility daemon called cups-lpd (normally in packages such as cupsys-bsd). In this case, CUPS accepts the jobs that come from an LPD or LPRng system, but it does not control the accesses (cupsd.conf only works for the CUPS system itself and therefore, it will be necessary to implement some strategy for controlling access, like a firewall, for example (see unit on security).

For administering from the commands line, CUPS is somewhat peculiar, in that it accepts both LPD and System V commands in the clients, and the administration is usually performed with the SystemV's lpadmin command.

Where the graphic tools are concerned, we have the gnome-cups-manager, gtklp or the web interface which comes with the same CUPS system, accessible at http://localhost:631.

Figure 2. Interface for the administration of the CUPS system

With regard to the software packages listed with CUPS, in Debian, we can find (among others):

```
cupsys - Common UNIX Printing System(tm) - server
cupsys-bsd - Common UNIX Printing System(tm) - BSD commands
cupsys-client - Common UNIX Printing System(tm) - client pro-
grams (SysV)
cupsys-driver-gimpprint - Gimp-Print printer drivers for CUPS
cupsys-pt - Tool for viewing/managing print jobs under CUPS
cupsomatic-ppd - linuxprinting.org printer support - transi-
tion package
foomatic-db - linuxprinting.org printer support - database
foomatic-db-engine - linuxprinting.org printer support - pro-
grams
foomatic-db-gimp-print - linuxprinting - db Gimp-Print print-
er drivers
foomatic-db-hpijs - linuxprinting - db HPIJS printers
foomatic-filters - linuxprinting.org printer support - fil-
ters
foomatic-filters-ppds - linuxprinting - prebuilt PPD files
foomatic-gui - GNOME interface for Foomatic printer filter
system
```

```
gimpprint-doc - Users' Guide for GIMP-Print and CUPS
gimpprint-locals - Local data files for gimp-print
gnome-cups-manager - CUPS printer admin tool for GNOME
gtklp - Front-end for cups written in gtk
```

7. Disk management

In respect of the storage units, as we have seen, they have a series of associated devices, depending on the type of interface:

- IDE: devices
  /dev/had master disk, first IDE connector;
  /dev/hdb slave disk of the first connector,
  /dev/hdc master second connector,
  /dev/hdd slave second connector.

- SCSI: /dev/sda, /dev/sdb devices... following the numbering of the peripheral devices in the SCSI Bus.

- Diskettes: /dev/fdx devices, with x diskette number (starting in 0). There are different devices depending on the capacity of the diskette, for example, a 1.44 MB diskette in disk drive A would be /dev/fd0H1440.

With regard to the partitions, the number that follows the device indicates the partition index within the disk and it is treated as an independent device: /dev/hda1 first partition of the first IDE disk, or /dev/sdc2, second partition of the third SCSI device. In the case of the IDE disks, these allow four partitions, known as primary partitions, and a higher number of logical partitions. Therefore, if /dev/hdan, *n* is less than or equal to 4, then it will be a primary partition; if not, it will be a logical partition with *n* being higher than or equal to 5.

With the disks and the associated file systems, the basic processes that we can carry out are included in:

- Creation of partitions or modification of partitions. Through commands such as fdisk or similar (*cfdisk, sfdisk*).

- Formatting diskettes: different tools may be used for diskettes: *fdformat* (low-level formatting), superformat (formatting at different capacities in MSDOS format), mformat (specific formatting creating standard MSDOS file systems).

- Creation of Linux file systems, in partitions, using the mkfs command. There are specific versions for creating diverse file systems, mkfs.ext2, mkfs.ext3 and also non-Linux file systems: mkfs.ntfs, mkfs.vfat, mkfs.msdos, mkfs.minix, or others. For CD-ROMs, commands such as mkisofs for creating the ISO9660s (with joliet or rock ridge extensions), which may be an image that might subsequently be recorded on

a CD/DVD, which along with commands such as cdrecord will finally allow us to create/save the CD/DVDs. Another particular case is the mkswap order, which allows us to create swap areas in the partitions, which will subsequently be activated or deactivated with swapon and swapoff.

- Setting up file systems: *mount, umount.* commands

- Status verification: the main tool for verifying Linux file systems is the *fsck* command. This command checks the different areas of the file system to verify the consistency and check for possible errors and to correct these errors where possible. The actual system automatically activates the command on booting when it detects situations where the system was not switched off properly (due to a cut in the electricity supply or an accidental shutting down of the machine) or when the system has been booted a certain number of times; this check usually takes a certain amount of time, usually a few minutes (depending on the size of the data). There are also particular versions for other file systems: fsck.ext2, fsck.ext3, fsck.vfat, fsck.msdos etc. The fsck process is normally performed with the device in read only mode with the partitions mounted; it is advisable to unmount the partitions for performing the process if errors are detected and it is necessary to correct the errors. In certain cases, for example, if the system that has to be checked is the root system (/) and a critical error is detected, we will be asked to change the system's runlevel execution mode to the root execution mode and to perform the verification process there. In general, if it is necessary to verify the system; this should be performed in superuser mode (we can switch between the runlevel mode with the init or telinit commands).

- Backup processes: whether in the disk, blocks of the disk, partitions, file systems, files... There are various useful tools for this: tar allows us to copy files towards file or tape units; cpio, likewise, can perform backups of files towards a file; both cpio and tar maintain information on the permissions and file owners; ddmakes it possible to make copies, whether they are files, devices, partitions or disks to files; it is slightly complex and we have to have some low-level information, on the type, size, block or sector, and it can also be sent to tapes.

- Various utilities: some individual commands, some of which are used by preceding processes to carry out various treatments: badblocks for finding defective blocks in the device; dumpe2fs for obtaining information on Linux file systems; tune2fs makes it possible to carry out Linux file system tuning of the ext2 or ext3 type and to adjust different performance parameters.

We will now mention two subjects related to the concept of storage space, which are used in various environments for the basic creation of storage space. The use of RAID software and the creation of dynamic volumes.

7.1. RAID software

The configuration of disks using RAID levels is currently one of the most widely-used high-availability storage schemes, when we have various disks for implementing our file systems.

The main focus on the different existing techniques is based on a fault-tolerance that is provided from the level of the device and the set of disks, to different potential errors, both physical or in the system, to avoid the loss of data or the lack of coherence in the system. As well as in some schemes that are designed to increase the performance of the disk system, increasing the bandwidth of these available for the system and applications.

Today we can find RAID in hardware mainly in corporate servers (although it is beginning to appear in desktops), where there are different hardware solutions available that fulfil these requirements. In particular, for disk-intensive applications, such as audio and/or video streaming, or in large databases.

In general, this hardware is in the form of cards (or integrated with the machine) of RAID-type disk drivers, which implement the management of one or more levels (of the RAID specification) over a set of disks administered with this driver.

In RAID a series of levels (or possible configurations) are distinguished, which can be provided (each manufacturer of specific hardware or software may support one or more of these levels). Each RAID level is applied over a set of disks, sometimes called RAID array (or RAID disk matrix), which are usually disks with equal sizes (or equal to group sizes). For example, in the case of an array, four 100 GB disks could be used or, in another case, 2 groups (at 100 GB) of 2 disks, one 30 GB disk and one 70 GB disk. In some cases of hardware drivers, the disks (or groups) cannot have different sizes; in others, they can, but the array is defined by the size of the smallest disk (or group).

We will describe some basic concepts on some levels in the following list (it should be remembered that, in some cases, the terminology has not been fully accepted, and it may depend on each manufacturer):

- RAID 0: The data are distributed equally between one or more disks without information on parity or redundancy, without offering fault-tolerance. Only data are being distributed; if the disk fails physically, the information will be lost and we will have to recover it from the backup copies. What does increase is the performance, depending on the RAID 0 imple-

mentation, given that the read and write options will be divided among the different disks.

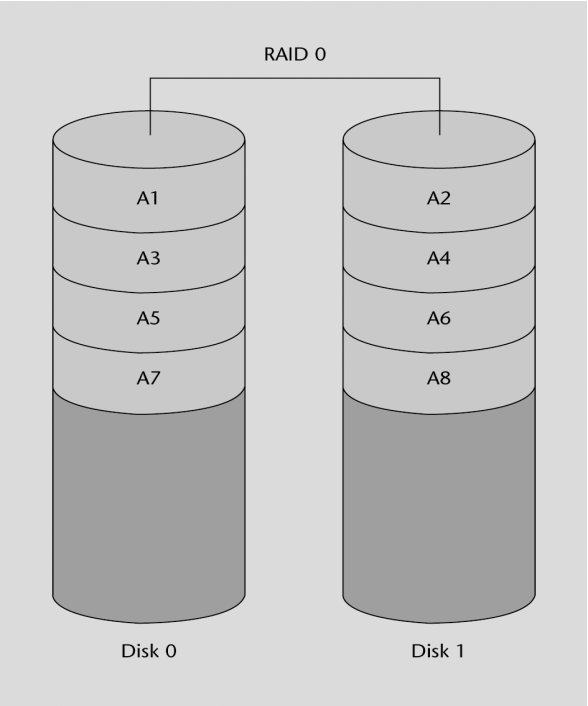

Figure 3

- RAID 1: An exact (mirror) copy is created in a set of two or more disks (known as a RAID array). In this case, it is useful for the reading performance (which can increase lineally with the number of disks) and especially for having a tolerance to faults in one of the disks, given that (for example, with two disks) the same information is available. RAID 1 is usually adequate for high-availability, such as 24x7 environments, where we critically need the resources. This configuration also makes it possible (if the hardware supports this) to hot swap disks. If we detect a fault in any of the disks, we can replace the disk in question without switching off the system with another disk.

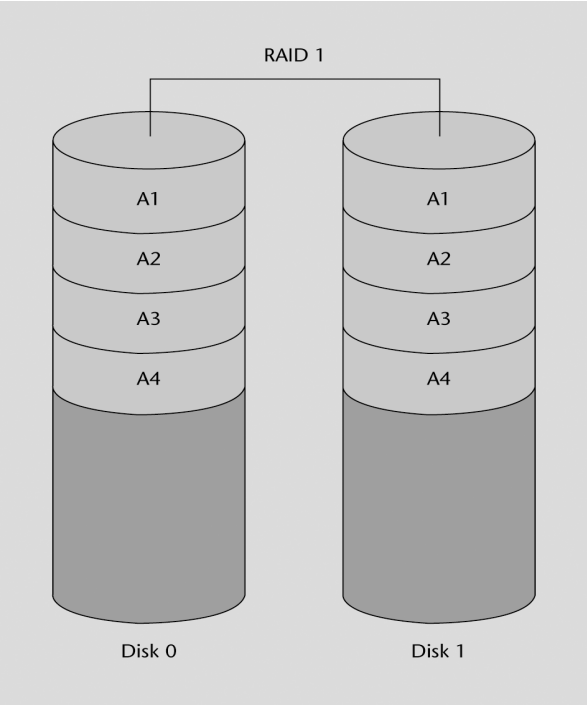

Figure 4

- RAID 2: In the preceding systems, the data would be divided in blocks for subsequent distribution; here, the data are divided into bits and redundant codes are used to correct the data. It is not widely used, despite the high performance levels that it could provide, as it ideally requires a high number of disks, one per data bit, and various for calculating the redundancy (for example, in a 32 bit system, up to 39 disks would be used).

- RAID 3: It uses byte divisions with a disk dedicated to the parity of blocks. This is not very widely used either, as depending on the size of the data and the positions, it does not provide simultaneous accesses. RAID 4 is similar, but it stripes the data at the block level, instead of byte level, which means that it is possible to service simultaneous requests when only a single block is requested.

- RAID 5: Block-level striping is used, distributing the parity among the disks. It is widely used, due to the simple parity scheme and due to the fact that this calculation is implemented simply by the hardware, with good performance levels.

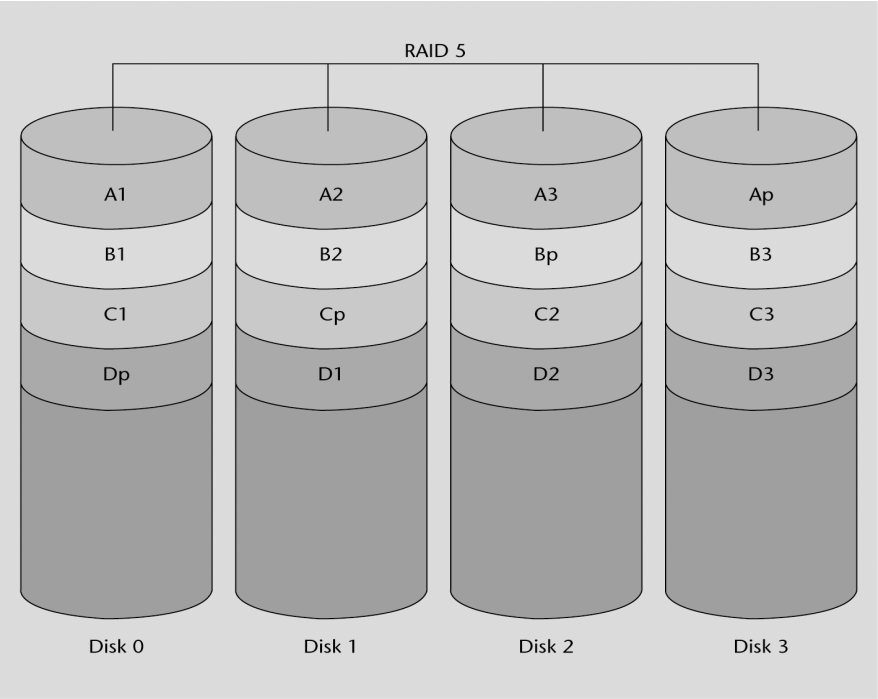

Figure 5

- RAID 0+1 (or 01): A mirror stripe is a nested RAID level; for example, we implement two groups of RAID 0, which are used in RAID 1 to create a mirror between them. An advantage is that, in the event of an error, the RAID 0 level used may be rebuilt thanks to the other copy, but if more disks need to be added, they have to be added to all the RAID 0 groups equally.

- RAID 10 (1+0): striping of mirrors, groups of RAID 1 under RAID 0. In this way, in each RAID 1 group, a disk may fail without ensuing loss of data. Of course, this means that they have to be replaced, otherwise the disk that is left in the group becomes another possible error point within the system. This configuration is usually used for high-performance databases (due to the fault tolerance and the speed, as it is not based on parity calculations).

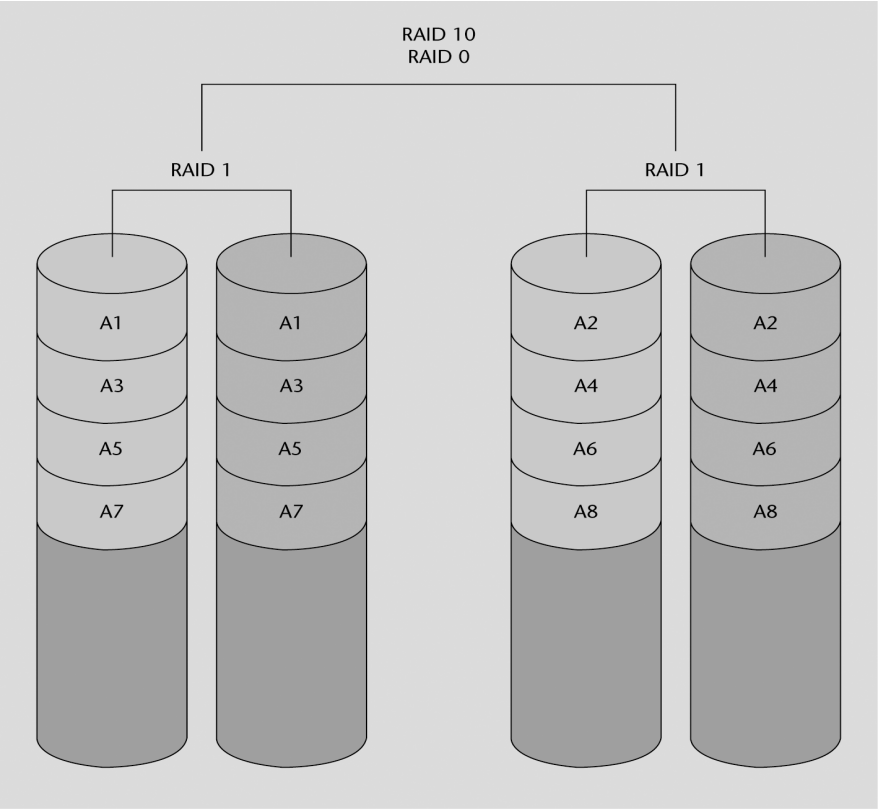

Figure 6

Some points that should be taken into account with regard to RAID in general:

- RAID improves the system's uptime, as some of the levels make it possible for the system to carry on working consistently when disks fail and, depending on the hardware, it is even possible to hot swap the problematic hardware without having to stop the system, which is especially important in critical systems.

- RAID can improve the performance of the applications, especially in systems with mirror implementations, where data striping permits the lineal read operations to increase significantly, as the disks can provide simultaneous read capability, increasing the data transfer rate.

- RAID does not protect data; evidently, it does not protect data from other possible malfunctions (virus, general errors or natural disasters). We must rely on backup copy schemes.

- Data recovery is not simplified. If a disk belongs to a RAID array, its recovery should be attempted within that environment. Software that is specific to the hardware drivers is necessary to access the data.

- On the other hand, it does not usually improve the performance of typical user applications, even if they are desktop applications, because these applications have components that access RAM and small sets of data, which means they will not benefit from lineal reading or sustained data transfers. In these environments, it is possible that the improvement in performance and efficiency is hardly even noticed.

- Information transfer is not improved or facilitated in any way; without RAID, it is quite easy to transfer data, by simply moving the disk from one system to another. In RAID's case, it is almost impossible (unless we have the same hardware) to move one array of disks to another system.

In GNU/Linux, RAID hardware is supported through various kernel modules, associated to different sets of manufacturers or chipsets of these RAID drivers. This permits the system to abstract itself from the hardware mechanisms and to make them transparent to the system and the end user. In any case, these kernel modules allow us to access the details of these drivers and to configure their parameters at a very low level, which in some cases (especially in servers that support a high I/O load) may be beneficial for tuning the disks system that the server uses in order to maximise the system's performance.

The other option that we will analyse is that of carrying out these processes through software components, specifically GNU/Linux's RAID software component.

GNU/Linux has a kernel of the so-called Multiple Device (md) kind, which we can consider as a support through the driver of the kernel for RAID. Through this driver we can generally implement RAID levels 0,1,4,5 and nested RAID levels (such as RAID 10) on different block devices such as IDE or SCSI disks. There is also the linear level, where there is a lineal combination of the available disks (it doesn't matter if they have different sizes), which means that disks are written on consecutively.

In order to use RAID software in Linux, we must have RAID support in the kernel, and, if applicable, the md modules activated (as well as some specific drivers, depending on the case (see available drivers associated to RAID, such as in Debian with modconf). The preferred method for implementing arrays of RAID disks through the RAID software offered by Linux is either during the installation or through the *mdadm* utility. This utility allows us to create and manage the arrays.

Let's look at some examples (we will assume we are working with some SCSI /dev/sda, /dev/sdb disks... in which we have various partitions available for implementing RAID):

Creation of a linear array:

```
# mdadm –create –verbose /dev/md0 –level=linear –raid-de-
vices=2 /dev/sda1 /dev/sdb1
```

where we create a linear array based on the first partitions of /dev/sda and /dev/sdb, creating the new device /dev/md0, which can already be used as a new disk (supposing that the mount point /media/diskRAID exists):

```
# mkfs.ext2fs /dev/md0
# mount /dev/md0 /media/diskRAID
```

For a RAID 0 or RAID 1, we can simply change the level (-level) to raid0 or raid1. With *mdadm –detail /dev/md0*, we can check the parameters of the newly created array.

We can also consult the mdstat entry in /proc to determine the active arrays and their parameters. Especially in the cases with mirrors (for example, in levels 1, 5...) we can examine the initial backup reconstruction in the created file; in /proc/mdstat we will see the reconstruction level (and the approximate completion time).

The *mdadm* utility provides many options that allow us to examine and manage the different RAID software arrays created (we can see a description and examples in man mdadm).

Another important consideration are the optimisations that should be made to the RAID arrays so as to improve the performance, through both the monitoring of its behaviour to optimise the file system parameters, as well as to use the RAID levels and their characteristics more effectively.

Note

The optimisation of the RAID arrays, may be an important resource for system tuning and some questions should be examined in:

Software-RAID-Howto, or in the actual mdadm man.

7.2. Logical Volume Manager (LVM)

There is a need to abstract from the physical disk system and its configuration and number of devices, so that the (operating) system can take care of this work and we do not have to worry about these parameters directly. In this sense, the logical volume management system can be seen as a layer of storage virtualisation that provides a simpler view, making it simpler and smoother to use.

In the Linux kernel, there is an LVM (*logical volume manager*), which is based on ideas developed from the storage volume managers used in HP-UX (HP's proprietary implementation of UNIX). There are currently two versions and LVM2 is the most widely used due to a series of added features.

The architecture of an LVM typically consists of the (main) components:

- **Physical volumes** (PV): PVs are hard disks or partitions or any other element that appears as a hard disk in the system (for example, RAID software or hardware).

- **Logical volumes** (LV): These are equivalent to a partition on the physical disk. The LV is visible in the system as a raw block device (completely equivalent to a physical partition) and it may contain a file system (such as the users' /home). Normally, the volumes make more sense for the administrators, as names can be used to identify them (for example, we can use a logical device, named stock or marketing instead of hda6 or sdc3).

- **Volume groups** (VG): This is the element on the upper layer. The administrative unit that includes our resources, whether they are logical volumes (LV) or physical volumes (PV). The data on the available PVs and how the LVs are formed using the PVs are saved in this unit. Evidently, in order to use a Volume Group, we have to have physical PV supports, which are organised in different logical LV units.

For example, in the following figure, we can see volume groups where we have 7 PVs (in the form of disk partitions, which are grouped to form two logical volumes (which have been completed using /usr and /home to form the file systems):

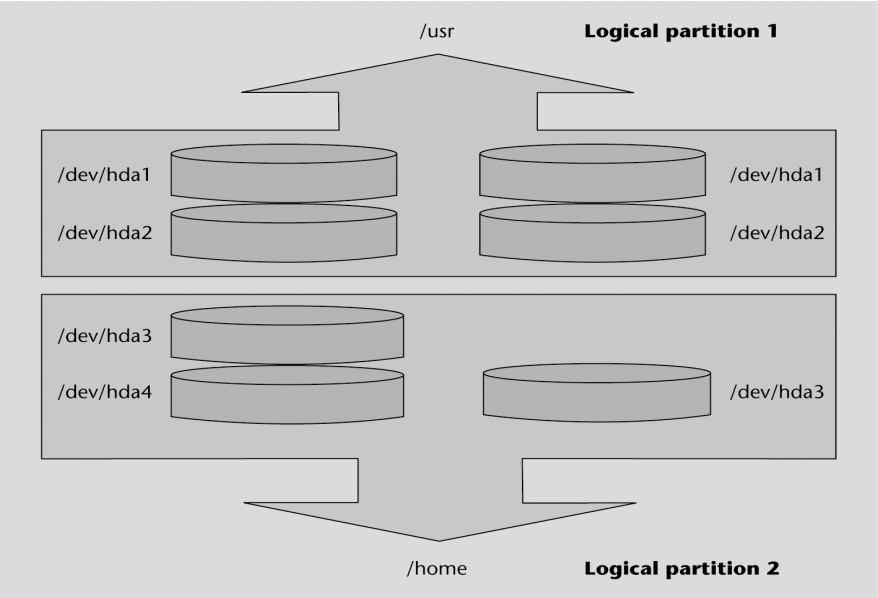

Figure 7. Scheme of an example of LVM

By using logical volumes, we can treat the storage space available (which may have a large number of different disks and partitions) more flexibly, according to the needs that arise, and we can manage the space by the more appropriate identifiers and by operations that permit us to adapt the space to the needs that arise at any given moment.

Logical Volume Management allows us to:

- Resize logical groups and volumes, using new PVs or extracting some of those initially available.

- Snapshots of the file system (reading in LVM1, and reading and/or writing in LVM2). This makes it possible to create a new device that is a snapshot of the situation of an LV. Likewise, we can create the snapshot, mount it, try various operations or configure new software or other elements and, if these do not work as we were expecting, we can return the original volume to the state it was in before performing the tests.

- RAID 0 of logical volumes.

RAID levels 1 or 5 are not implemented in LVM; if they are necessary (in other words, redundancy and fault tolerance are required), then either we use RAID software or RAID hardware drivers that will implement it and we place LVM as the upper layer.

We will provide a brief, typical example (in many cases, the distributor installer carries out a similar process if we set an LVM as the initial storage system). Basically, we must: 1) create physical volumes (PV). 2) create the logical group (VG) and 3) create the logical volume and finally use the following to create and mount a file system:

1) example: we have three partitions on different disks, we have created three PVs and started-up the contents:

```
# dd if=/dev/zero of=/dev/hda1 bs=1k count=1
# dd if=/dev/zero of=/dev/hda2 bs=1k count=1
# dd if=/dev/zero of=/dev/hdb1 bs=1k count=1
# pvcreate /dev/hda1
Physical volume "/dev/sda1" successfully created
# pvcreate /dev/hda2
Physical volume "/dev/hda2" successfully created
# pvcreate /dev/hdb1
Physical volume "/dev/hdb1" successfully created
```

2) placement of a VG created from the different PVs:

```
# vgcreate group_disks /dev/hda1 /dev/hda2 /dev/hdb1
```

```
Volume group "group_disks" successfully created
```

3) we create the LV (in this case, with a size of 1 GB) based on the elements that we have in group VG group (*-n* indicates the name of the volume):

```
# lvcreate -L1G -n logical_volume group_disks
lvcreate -- doing automatic backup of "group_disks"
lvcreate -- logical volume "/dev/group_disks/ logical_volume"
successfully created
```

And finally, we create a file system (a ReiserFS in this case):

```
# mkfs.reiserfs /dev/group_disks/logical_volume
```

Which we could, for example, place as backup space

```
# mkdir /mnt/backup
# mount -t reiserfs /dev/group_disks/logical_volume /mnt/
  backup
```

Finally, we will have a device as a logical volume that implements a file system in our machine.

8. Updating Software

In order to administer the installation or to update the software in our system, we will, in the first instance, depend on the type of software packages used by our system:

- RPM: packages that use the Fedora/Red Hat distribution (and derivatives). They are usually handled through the *rpm* command. Contains information on the dependencies that the software has on other software. At a high level, through *Yum* (or *up2date* in some distributions derived from Red Hat).

- DEB: Debian packages that are usually handled with a set of tools that work on different levels with individual packages or groups. Among these, we must mention: *dselect, tasksel, dpkg*, and *apt-get*.

- Tar or the tgz (also tar.gz): these are simply package files that have been joined and compressed using standard commands such as tar, and gzip (these are used for decompressing). The packages do not contain information on any dependencies and can normally be installed in different places if they do not carry any absolute root (path) information.

There are various graphical tools for handling these packages, such as RPM: Kpackage; DEB: Synaptic, Gnome-apt; Tgz: Kpackage,or from the actual graphic file manager itself (in Gnome or KDE). There are also usually package conversion utilities. For example, in Debian we have the *alien* command, with which we can change RPM packages to DEB packages. Although it is necessary to take the appropriate precautions, so that the package does not unexpectedly modify any behaviour or file system, as it has a different destination distribution.

Depending on the use of the types of packages or tools: it will be possible to update or install the software in our system in different ways:

1) From the actual system installation CDs; normally, all the distributions search for the software on the CDs. But the software should be checked to ensure that it is not old and does not, therefore, include some patches like updates or new versions with more features; consequently, if a CD is used for installation, it is standard practice to check that it is the latest version and that no more recent version exists.

2) Through updating or software search services, whether they are free, as is the case with Debian's apt-get tool or yum in Fedora, or through subscription services (paid services or services with basic facilities), such as the Red Hat Network of the commercial Red Hat versions.

3) Through software repositories that offer pre-built software packages for a determined distribution.

4) From the actual creator or distributor of the software, who may offer a series of software installation packages. We may find that we are unable to locate the type of packages that we need for our distribution.

5) Unpackaged software or with compression only, without any type of dependencies.

6) Only source code, in the form of a package or compressed file.

9. Batch jobs

In administration tasks, it is usually necessary to execute certain tasks at regular intervals, either because it is necessary to program the tasks so that they take place when the machine is least being used or due to the periodic nature of the tasks that have to be performed.

There are various systems that allow us to set up a task schedule (planning task execution) for performing these tasks out-of-hours, such as periodic or programmed services:

- *nohup* is perhaps the simplest command used by users, as it permits the execution of a non-interactive task once they have logged out from their account. Normally, when users log out, they lose their processes; *nohup* allows them to leave the processes executing even though the user has logged out.

- *at* permits us to launch a task for later, programming the determined point in time at which we wish for it to start, specifying the time (hh:mm) and date, or specifying whether it will be today or tomorrow. Examples:
  `at 10pm task`
  to perform the task at ten o'clock at night.
  `at 2am tomorrow task`
  to perform the task at two o'clock in the morning.

- *cron*: it permits us to establish a list of tasks that will be performed with the corresponding programming; this configuration is saved in /etc/crontab; specifically, in each entry in this file, we have: hour and minutes at which the task will be performed, which day of the month, which month, which day of the week, along with which element (which might be a task or a directory where the tasks that are to be executed are located). For example, the standard content is similar to:

```
25 6 * * * root test -e /usr/sbin/anacron || run-parts --report /etc/cron.daily
47 6 * * 7 root test -e /usr/sbin/anacron || run-parts --report /etc/cron.weekly
52 6 1 * * root test -e /usr/sbin/anacron || run-parts --report /etc/cron.monthl
```

where a series of tasks are programmed to execute: each day ("*" indicates 'whichever'), weekly (7th day of the week) or monthly (the 1st day of each month). Normally, the tasks will be executed with the *crontab* command, but the cron system assumes that the machine is always switched on, and if this is not the case, it is better to use *anacron*, which checks whether the task was performed when it was supposed to be or not, and if not, it executes the task.

Each line in the preceding file is checked to ensure that the *anacron* command is there and the scripts associated to each action are executed; in this case, they are saved in directories assigned for this.

There may also be cron.allow or cron.deny files to limit who can (or cannot) put tasks in cron. Through the *crontab* command, a user may define tasks in the same format as we have seen before, which are usually saved in /var/spool/cron/crontabs. In some cases, there is also a /etc/cron.d directory where we can place the tasks and they are treated as through they were an extension to the /etc/crontab file.

10. Tutorial: combined practices of the different sections

We will begin by examining the general state of our system. We will carry out different steps in a Debian system. It is an unstable Debian system (the unstable version, but more updated); however, the procedures are, mostly, transferable to other distributions such as Fedora/Red Hat (we will mention some of the most important changes). The hardware consists of a Pentium 4 at 2.66 Ghz with 768 MB RAM and various disks, DVD and CD-writer, as well as other peripherals, on which we will obtain information as we proceed step by step.

First we will see how our system booted up the last time:

```
# uptime
17:38:22 up 2:46, 5 users, load average: 0.05, 0.03, 0.04
```

This command tells us the time that the system has been up since it last booted, 2 hours and 47 minutes and, in this case, we have 5 users. These will not necessarily correspond to five different users, but they will usually be opened user sessions (for example, through one terminal). The *who* command provides a list of these users. The load average is the system's average load over the last 1, 5 and 15 minutes.

Let's look at system's boot log (*dmesg* command), and the lines that were generated when the system booted up (we have removed some lines for the purpose of clarity):

```
Linux version 2.6.20-1-686 (Debian 2.6.20-2) (waldi@debian.org)
(gcc version 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)) #1 SMP Sun Apr
 15 21:03:57 UTC 2007
BIOS-provided physical RAM map:
  BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
  BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
  BIOS-e820: 00000000000ce000 - 00000000000d0000 (reserved)
  BIOS-e820: 00000000000dc000 - 0000000000100000 (reserved)
  BIOS-e820: 0000000000100000 - 000000002f6e0000 (usable)
  BIOS-e820: 000000002f6e0000 - 000000002f6f0000 (ACPI data)
  BIOS-e820: 000000002f6f0000 - 000000002f700000 (ACPI NVS)
  BIOS-e820: 000000002f700000 - 000000002f780000 (usable)
  BIOS-e820: 000000002f780000 - 0000000030000000 (reserved)
  BIOS-e820: 00000000ff800000 - 00000000ffc00000 (reserved)
  BIOS-e820: 00000000fffffc00 - 0000000100000000 (reserved)
0MB HIGHMEM available.
```

```
759MB LOWMEM available.
```

These first lines already indicate some interesting data: the Linux kernel is version 2.6.20-1-686, one version 2.6 revision 20 at revision 1 of Debian and for 686 machines (Intel x86 32 bits architecture). They also indicate that we are booting a Debian system, with this kernel which was compiled with a GNU gcc compiler, version 4.1.2 and the date. There is then a map of the memory zones used (reserved) by the BIOS and then the total memory detected in the machine: 759 MB, to which we would have to add the first 1 MB, making a total of 760 MB.

```
Kernel command line: BOOT_IMAGE=LinuxNEW ro root=302 lang=es acpi=force
Initializing CPU#0
Console: colour dummy device 80x25
Memory: 766132k/777728k available (1641k kernel code, 10968k reserved, 619k da-
ta, 208k init, 0k highmem)
Calibrating delay using timer specific routine.. 5320.63 BogoMIPS (lpj=10641275)
```

Here, we are told how the machine booted up and which command line has been passed to the kernel (different options may be passed, such as lilo or grub). And we are booting in console mode with 80 x 25 characters (this can be changed). The BogoMIPS are internal measurements of the kernel of the CPU speed. There are architectures in which it is difficult to detect how many MHz the CPU works with and this is why this speed measurement is used. Subsequently, we are given more data on the main memory and what it is being used for at this booting stage.

```
CPU: Trace cache: 12K uops, L1 D cache: 8K
CPU: L2 cache: 512K
CPU: Hyper-Threading is disabled
Intel machine check architecture supported.
Intel machine check reporting enabled on CPU#0.
CPU0: Intel P4/Xeon Extended MCE MSRs (12) available
CPU0: Intel(R) Pentium(R) 4 CPU 2.66GHz stepping 09
```

Likewise, we are given various data on the CPU: the size of the first-level cache, the internal CPU cache, L1 divided in a TraceCache of the Pentium 4 (or cache instruction), and the data cache and the unified second-level cache (L2), the type of CPU, its speed and the system's bus.

```
PCI: PCI BIOS revision 2.10 entry at 0xfd994, last bus=3
Setting up standard PCI resources
...
NET: Registered protocol
IP route cache hash table entries: 32768 (order: 5, 131072 bytes)
TCP: Hash tables configured (established 131072 bind 65536)
checking if image is initramfs... it is
Freeing initrd memory: 1270k freed
fb0: VESA VGA frame buffer device
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing enabled
serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
00:09: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
RAMDISK driver initialized: 16 RAM disks of 8192K size 1024 blocksize
PNP: PS/2 Controller [PNP0303:KBC0,PNP0f13:MSE0] at 0x60,0x64 irq 1,12
i8042.c: Detected active multiplexing controller, rev 1.1.
serial: i8042 KBD port at 0x60,0x64 irq 1
serial: i8042 AUX0 port at 0x60,0x64 irq 12
serial: i8042 AUX1 port at 0x60,0x64 irq 12
serial: i8042 AUX2 port at 0x60,0x64 irq 12
serial: i8042 AUX3 port at 0x60,0x64 irq 12
mice: PS/2 mouse device common for all mice
```

The kernel and devices continue to boot, mentioning the initiation of the network protocols. The terminals, the serial ports ttyS0 (which would be com1) and ttyS01 (com2). It provides information on the RAM disks that are being used, the detection of PS2 devices, keyboard and mouse.

```
ICH4: IDE controller at PCI slot 0000:00:1f.1

ide0: BM-DMA at 0x1860-0x1867, BIOS settings: hda:DMA, hdb:pio
ide1: BM-DMA at 0x1868-0x186f, BIOS settings: hdc:DMA, hdd:pio
Probing IDE interface ide0...
hda: FUJITSU MHT2030AT, ATA DISK drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
Probing IDE interface ide1...
hdc: SAMSUNG CDRW/DVD SN-324F, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177,0x376 on irq 15
SCSI subsystem initialized
libata version 2.00 loaded.
hda: max request size: 128KiB
hda: 58605120 sectors (30005 MB) w/2048KiB Cache, CHS=58140/16/63<6>hda:
hw_config=600b
, UDMA(100)
hda: cache flushes supported
hda: hda1 hda2 hda3
kjournald starting. Commit interval 5 seconds
EXT3-fs: mounted file system with ordered data mode.
hdc: ATAPI 24X DVD-ROM CD-R/RW drive, 2048kB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.20
Addinf 618492 swap on /dev/hda3.
```

Detection of IDE devices, detecting the IDE chip in the PCI bus and reporting what is driving the devices: hda, and hdc, which are, respectively: a hard disk (Fujitsu), a second hard disk, a Samsung DVD Samsung, and a CD-writer (given that in this case, we have a combo unit). It indicates active partitions. Subsequently, the machine detects the main Linux file system, a journaled ext3, that activates and adds the swap space available in a partition.

```
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
input: PC Speaker as /class/input/input1
USB Universal Host Controller Interface driver v3.0
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
uhci_hcd 0000:00:1d.1: UHCI Host Controller
uhci_hcd 0000:00:1d.1: new USB bus registered, assigned bus number 2
uhci_hcd 0000:00:1d.1: irq 11, io base 0x00001820
usb usb2: configuration #1 chosen from 1 choice
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 2 ports detected
hub 4-0:1.0: USB hub found
hub 4-0:1.0: 6 ports detected
```

More detection of devices, USB (and the corresponding modules); in this case, two hub devices (with a total of 8 USB ports) have been detected.

```
parport: PnPBIOS parport detected.
parport0: PC-style at 0x378 (0x778), irq 7, dma 1
[PCSPP,TRISTATE,COMPAT,EPP,ECP,DMA]
input: ImPS/2 Logitech Wheel Mouse as /class/input/input2
ieee1394: Initialized config rom entry 'ip1394'
eepro100.c:v1.09j-t 9/29/99 Donald Becker
Synaptics Touchpad, model: 1, fw: 5.9, id: 0x2e6eb1, caps: 0x944713/0xc0000
input: SynPS/2 Synaptics TouchPad as /class/input/input3

agpgart: Detected an Intel 845G Chipset
agpgart: Detected 8060K stolen Memory
agpgart: AGP aperture is 128M
eth0: OEM i82557/i82558 10/100 Ethernet, 00:00:F0:84:D3:A9, IRQ 11.
Board assembly 000000-000, Physical connectors present: RJ45
e100: Intel(R) PRO/100 Network Driver, 3.5.17-k2-NAPI
usbcore: registered new interface driver usbkbd
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.

lp0: using parport0 (interrupt-driven).
ppdev: user-space parallel port driver
```

And the final detection of the rest of the devices: Parallel port, mouse model, FireWire port (IEEE1394) network card (Intel), a touchscreen, the AGP video card (i845). More data on the network card, an intel pro 100, registry of usb as mass storage (indicates a USB storage device as an external disk) and detection of parallel port.

We can also see all this information, which we accessed through the dmesg command, dumped in the system's main log, /var/log/messages. In this log, we will find the kernel messages, among others, the messages of the daemons and network or device errors, which communicate their messages to a special daemon called syslogd, which is in charge of writing the messages in this file. If we have recently booted the machine, we will observe that the last lines contain exactly the same information as the *dmesg* command,

for example, if we look at the final part of the file (which is usually very large):

```
# tail 200 /var/log/messages
```

We observe the same lines as before and some more information such as:

```
shutdown[13325]: shutting down for system reboot
kernel: usb 4-1: USB disconnect, address 3
kernel: nfsd: last server has exited
kernel: nfsd: unexporting all file systems
kernel: Kernel logging (proc) stopped.
kernel: Kernel log daemon terminating.

exiting on signal 15
syslogd 1.4.1#20: restart.

kernel: klogd 1.4.1#20, log source = /proc/kmsg started.
Linux version 2.6.20-1-686 (Debian 2.6.20-2) (waldi@debian.org) (gcc version 4.1.2
20061115 (prerelease) (Debian 4.1.1-21)) #1 SMP Sun Apr 15 21:03:57 UTC 2007
kernel: BIOS-provided physical RAM map:
```

The first part corresponds to the preceding shutdown of the system, informing us that the kernel has stopped placing information in /proc, that the system is shutting down... At the beginning of the new boot, the Syslogd daemon that generates the log is activated, and the system begins to load, which tells us that the kernel will begin to write information in its system, /proc; we look at the first lines of the dmesg mentioning the version of the kernel that is being loaded and we then find what we have seen with dmesg.

At this point, another useful command for finding out how the load process has taken place is *Ismod*, which will tell us which modules have been loaded in the kernel (summarised version):

```
# lsmod
Module Size Used by
nfs 219468              0
nfsd 202192            17
exportfs 5632           1 nfsd
lockd 58216             3 nfs,nfsd
nfs_acl 3616            2 nfs,nfsd
sunrpc 148380          13 nfs,nfsd,lockd,nfs_acl
ppdev 8740              0
lp 11044                0
button 7856             0
ac 5220                 0
battery 9924            0
md_mod 71860            1
dm_snapshot 16580       0
dm_mirror 20340         0
dm_mod 52812            2 dm_snapshot,dm_mirror
```

```
i810fb 30268            0
vgastate 8512           1 i810fb
eeprom 7184             0
thermal 13928           0
processor 30536         1 thermal
fan 4772                0
udf 75876               0
ntfs 205364             0
usb_storage 75552       0
hid 22784               0
usbkbd 6752             0
eth1394 18468           0
e100 32648              0
eepro100 30096          0
ohci1394 32656          0
ieee1394 89208          2 eth1394,ohci1394
snd_intel8x0 31420      1
snd_ac97_codec 89412    1 snd_intel8x0
ac97_bus 2432           1 snd_ac97_codec
parport_pc 32772        1
snd 48196               6 snd_intel8x0,snd_ac97_codec,snd_pcm,snd_timer
ehci_hcd 29132          0
ide_cd 36672            0
cdrom 32960             1 ide_cd
soundcore 7616          1 snd
psmouse 35208           0
uhci_hcd 22160          0
parport 33672           3 ppdev,lp,parport_pc
intelfb 34596           0
serio_raw 6724          0
pcspkr 3264             0
pci_hotplug 29312       1 shpchp
usbcore 122312          6 dvb_usb,usb_storage,usbkbd,ehci_hcd,uhci_hcd
intel_agp 22748         1
agpgart 30504           5 i810fb,drm,intelfb,intel_agp
ext3 121032             1
jbd 55368               1 ext3
ide_disk 15744          3
ata_generic 7876        0
ata_piix 15044          0
libata 100052           2 ata_generic,ata_piix
scsi_mod 133100         2 usb_storage,libata
generic 4932            0 [permanent]
piix 9540               0 [permanent]
ide_core 114728         5 usb_storage,ide_cd,ide_disk,generic,piix
```

We see that we basically have the drivers for the hardware that we have detected and other related elements or those necessary by dependencies.

This gives us, then, an idea of how the kernel and its modules have been loaded. In this process, we may already have observed an error, if the hardware is not properly configured or there are kernel modules that are not properly compiled (they were not compiled for the appropriate kernel version), inexistent etc.

The next step for examining the processes in the system, such as the *ps* (for process status) command, for example (only the system processes are shown, not the user ones):

```
# ps -ef
UID PID PPID C STIME TTY TIME CMD
```

Processes information, UID user that has launched the process (or the identifier with which it has been launched), PID and process code assigned by the system are consecutively shown, as the processes launch; the first is always 0, which corresponds to the init process. PPID is the id of the current parent process. STIME, time in which the process was booted, TTY, terminal assigned to the process (if there is one), CMD, command line with which it was launched.

```
root 1 0 0 14:52 ? 00:00:00 init [2]
root 3 1 0 14:52 ? 00:00:00 [ksoftirqd/0]
root 143 6 0 14:52 ? 00:00:00 [bdflush]
root 145 6 0 14:52 ? 00:00:00 [kswapd0]
root 357 6 0 14:52 ? 00:00:01 [kjournald]
root 477 1 0 14:52 ? 00:00:00 udevd --daemon
root 719 6 0 14:52 ? 00:00:00 [khubd]
```

Various system daemons, such as the kswapd daemon, which controls the virtual memory swaps. Handling of system buffers (bdflush). Handling of file system journal (kjournald), USB handling (khubd). Or the udev daemon that controls the hot device connection. In general, the daemons are not always identified by a d at the end, and if they have a k at the beginning, they are normally internal threads of the kernel.

```
root      1567 1 0 14:52 ? 00:00:00 dhclient -e -pf ...
root      1653 1 0 14:52 ? 00:00:00 /sbin/portmap
root       1829 1 0 14:52 ? 00:00:00 /sbin/syslogd
root      1839 1 0 14:52 ? 00:00:00 /sbin/klogd -x
root      1983 1 0 14:52 ? 00:00:09 /usr/sbin/cupsd
root       2178 1 0 14:53 ? 00:00:00 /usr/sbin/inetd
```

We have dhclient, which indicates that the machine is the client of a DHCP server, for obtaining its IP. Syslogd, a daemon that sends messages to the log. The cups daemon, which, as we have discussed, is related to the printing system. And inetd, which, as we shall see in the section on networks, is a type of "superserver" or intermediary of other daemons related to network services.

```
root      2154 1 0 14:53 ?          00:00:00 /usr/sbin/rpc.mountd
root      2241 1 0 14:53 ?          00:00:00 /usr/sbin/sshd
root      2257 1 0 14:53 ?          00:00:00 /usr/bin/xfs -daemon
root             2573 1 0 14:53 ?          00:00:00 /usr/sbin/atd
root      2580 1 0 14:53 ?          00:00:00 /usr/sbin/cron
root      2675 1 0 14:53 ?          00:00:00 /usr/sbin/apache
www-data      2684 2675 0 14:53 ?      00:00:00 /usr/sbin/apache
www-data      2685 2675 0 14:53 ? 00:00:00 /usr/sbin/apache
```

There is also sshd, a safe remote access server (an improved version that permits services compatible with telnet and FTP). xfs is the fonts server (character types) of X Window. The atd and cron commands can be used for handling programmed tasks at a determined moment. Apache is a web server, which may have various active threads for attending to different requests.

```
root 2499 2493 0 14:53 ?       00:00:00 /usr/sbin/gdm
root 2502 2499 4 14:53 tty7    00:09:18 /usr/bin/X :0 -dpi 96 ...
root 2848 1 0 14:53 tty2       00:00:00 /sbin/getty 38400 tty2
root 2849 1 0 14:53 tty3       00:00:00 /sbin/getty 38400 tty3
root 3941 2847 0 14:57 tty1    00:00:00 -bash
root 16453 12970 0 18:10 pts/2 00:00:00 ps -ef
```

gdm is the graphical login of the Gnome desktop system (the entry point where we are asked for the login name and password) and the getty processes are the ones that manage the virtual text terminals (which we can see by pressing Alt+Fx (or Ctrl+Alt+Fx if we are in graphic mode). X is the process of the X Window System graphic server and is essential for executing any desktop environment above this. An open shell (bash), and finally, the process that we have generated when requesting this *ps* from the command line.

The ps command provides various command line options for adjusting the information that we want on each process, whether it is the time that it has been executing, the percentage of CPU used, memory used etc. (see man of *ps*). Another very interesting command is *top*, which does the same as *ps* but dynamically; in other words, it updates every certain period of time, we can classify the processes by use of CPU or memory and it also provides information on the state of the overall memory.

Other useful commands for resources management are free and vmstat, which provide information on the memory used and the virtual memory system:

Note

See man of the commands to interpret outputs.

```
# free      total used free shared buffers cached
Mem: 767736 745232 22504 0 89564 457612
-/+ buffers/cache: 198056 569680
```

Swap: 618492 1732 616760

```
# vmstat
procs -----------memory---------- ---swap-- -----io-- --system-- ----cpu----
r b swpd free buff cache si so bi bo in cs us sy id wa
1 0 1732 22444 89584 457640 0 0 68 137 291 418 7 1 85 7
```

The free command also shows the swap size, approximately 600 MB, which are not currently used intensely as there is sufficient physical memory space; there are still 22 MB free (which indicates a high use of the physical memory and the need to use swap soon). The memory space and swap (as of kernels 2.4) add to each other to comprise the total memory in the system, which in this case, means that there is a total of 1.4 GB available. This may seem a lot, but it will depend on the applications that are being executed.

Activities

1) The swap space makes it possible to add to the physical memory so that there is more virtual memory. Depending on the amounts to add extra space to the physical memory and swap space, can all the memory get used up? Can we resolve this in any other way that does not involve adding more physical memory?

2) Suppose that we have a system with two Linux partitions: one / and one swap partition. How do we solve the situation if the user accounts use up all the disk space? And if we have an isolated /home partition, which was also being used up, how would we solve this?

3) Install the CUPS printing system, define our printer so that it works with CUPS and try administering through the web interface. As the system is now, would it be advisable to modify, in any way, CUPS' default settings? Why?

4) Examine the default setting that comes with the GNU/Linux system for non-interactive work using cron. Which jobs are there and when are they being performed? Any ideas for new jobs that have to be added?

5) Reproduce the workshop analysis (plus the other sections of the unit) on the machine that is available. Can we see any errors or irregular situations in the examined system? If so, how do we solve them?

Bibliography

Other sources of reference and information

[Wm02] [Fri02] [Smi02] GNU/Linux and UNIX administration manuals, which explain in detail the aspects on local administration and printing systems management.

[Gt] Updated information on the printing systems and their settings, as well as the details of some of the printers, can be found here. For specific details on the printer models and drivers, we can go to http://www.linuxprinting.org/.

[Hin][Koe] We can find information on the different file systems available and the schemes for creating partitions for the installation of the system.