
# FINAL THESIS

**TITLE: Advanced OS Deployment System**

**DEGREE: Degree in Telematic Engineering**

**AUTHOR: Sebastián Galiano Molina**

**DIRECTOR: Cristina Barrado**

**DATE: November 23th 2007**

**Title:** Advanced OS Deployment System

**Author:** Sebastián Galiano Molina

**Director:** Cristina Barrado

**Data:** November 23th 2007

**Overview**

Personal Computers are a fundamental tool in many disciplines of study and work. In many cases they are offered to users inside an open room to be used freely for their needs. A college computer laboratory is just one limit example of this situation. Students of different courses, with different needs, are sharing the same infrastructure.

Technical managers know well the needs of maintenance of shared computer labs. Many people entering and using the computers represents a lot of small hardware reparations, resulting in a heterogeneous environment due to reparations and component substitutions. Then the real complexity arises: software management. In addition to the initial requirements of software to install on computers, the heterogeneous hardware makes exponential the number of combinations to maintain.

The requirements are just a remote server connected to a network to the PC laboratory. This project, OS deployment system, is a modified way to use Linux thin client to restore images, using kexec to boot an OS without reboot. The state of the art on similar tools is the Rembo system, recently bought by IBM, added to the Tivoli Provisioning Manager suite.

This final thesis presents how the OS deployoment system has improved to overcome PC labs management.

The OS deployment system is a free software solution that permits:

- The end-user to restore, interactively and easely, operating systems images on demand.
- Multiple OS management.
- Improve maintenance's time, autorestoring lab's computers to the initial configuration simultaneously.
- An empty PC can be filled up with an operating system and software via network.
- Fast image restoring.

# Chapter 1 Introduction

## 1.1 The beginning

This project started two years ago when I was on an internship at the EPSC's Computer Architecture department.

Along my years at the university, it was possible to observe the technological dependence of a tool used to deploy OS all along the school; this tool is called REMBO [Ref 1]. This software provides an entire system to restore any kind of OS and a transparent environment for the end user.

My tutor, Cristina Barrado, and I decided to start an Open Source based OS deployment system that could overcome and compete against REMBO.

Later Arnau Güell help us to develop the first version of the project, this first version was his final degree thesis.

## 1.2 The design

At earlier phases of development, the first idea was to modify the GRUB [Ref 2] code. GRUB is an Open Source boot loader with power to boot several Operating Systems, including Linux images from the network.

GRUB has two important problems, the first one is its complexity and the second one is its necessity to be installed on the local drive. One of the most important characteristics of REMBO is it lack of local installation.

If we wanted to compete against REMBO it was important to solve this issue. REMBO uses a standard protocol designed by Intel to boot over the network (PXE [Ref 3]). Researching about this standard we found out that Linux can boot from PXE. So the initial idea of the design changed drastically, towards this new and more powerful solution:

To have an entire OS to restore the final operating system was more powerful than a software modification. Studying the exisisting technologies and developing an OS restoring dedicated operative system will introduce new and more flexible ways to achieve the objective.

From that moment until now, the every day work of this project has been to search new ways to improve the original idea, to test them and to decide what to use on the final version.

## 1.3 General design objective

The main project's objective is to design and build an OS deployment system taking advantage of the Linux OS and the Open Source community developments. This means to use existing technologies that modularize the system. With this philosophy in mind, the number of developed code lines within the project is keeping as small as possible.

As REMBO, the OS deployment system to develop has to be transparent to the user. This means a system with a friendly user interface and no technological knowledge needed to manage it.

Along this thesis the objectives are the same, but introducing the improvements where the last version had flaws. The particular objectives in contrast with the previous degree thesis are detailed at Section 6

## 1.4 State of the art

As software evolves new solutions for old problems are being released. The case of the deployment systems is not different. In this section we will review REMBO but also some other Open Source tool available today.

### 1.4.1 REMBO

REMBO is privative software, now owned by IBM as part of its Tivoli Suite. Since IBM acquired it, REMBO has been renamed to Tivoli Provisioning Manager for OS Deployment [Ref 1].

REMBO was a project started at the Geneve University by Marc Vuilleumier Stückelberg and David Clerc [Ref 4].

This software is based on Bpbatch [Ref 4], a little software from the late 90's that ran as a boot loader of Terminals and could restore OS. It works using the PXE specification. REMBO has two main functionalities: to select an OS to boot and to restore them also REMBO has two ways for restoring an OS image: full restoring or fast restoring. This last work using differential images and just copying the difference between the installed image and the source image. Among other features it has an easy to use graphic interface and supports multicast data transferring.

One of REMBO most important features is to add transparency to the user; REMBO is capable of boot an OS without rebooting.

As main disadvantage, REMBO is a privative source code product. Thus, the only way to add extra functionality is using its scripting language REMBO-C to develop plug-ins for the system. As being privative software it cause

technological dependence, so just REMBO products and formats can be used with REMBO.

### 1.4.2 DRBL + CLONEZILLA

Diskless Remote Boot Linux (DRBL) provides diskless clients with a running operating system. A diskless node is nothing more than a PC, which its operating system in not installed on the disk.
This software is installed on a server and provides the configuration and scripting to boot diskless nodes on a network. This diskless client works using the same technologies used in this project.

Once the client is turned on, it uses the suite Clonezilla to clone and restore the OS. DRBL with Clonezilla is very similar to the presented project with 3 big differences:

- It does not support fast restoration.
- It cannot boot the restored OS once is downloaded on the client local disk.
- Its interface is ncurses based. Ncurses is a library to develop interfaces for command line [Ref 5]. An example of the interface can be seen in the next image. The image shows the Clonezilla options: save disk, restore disk, save partitions and restore partitions

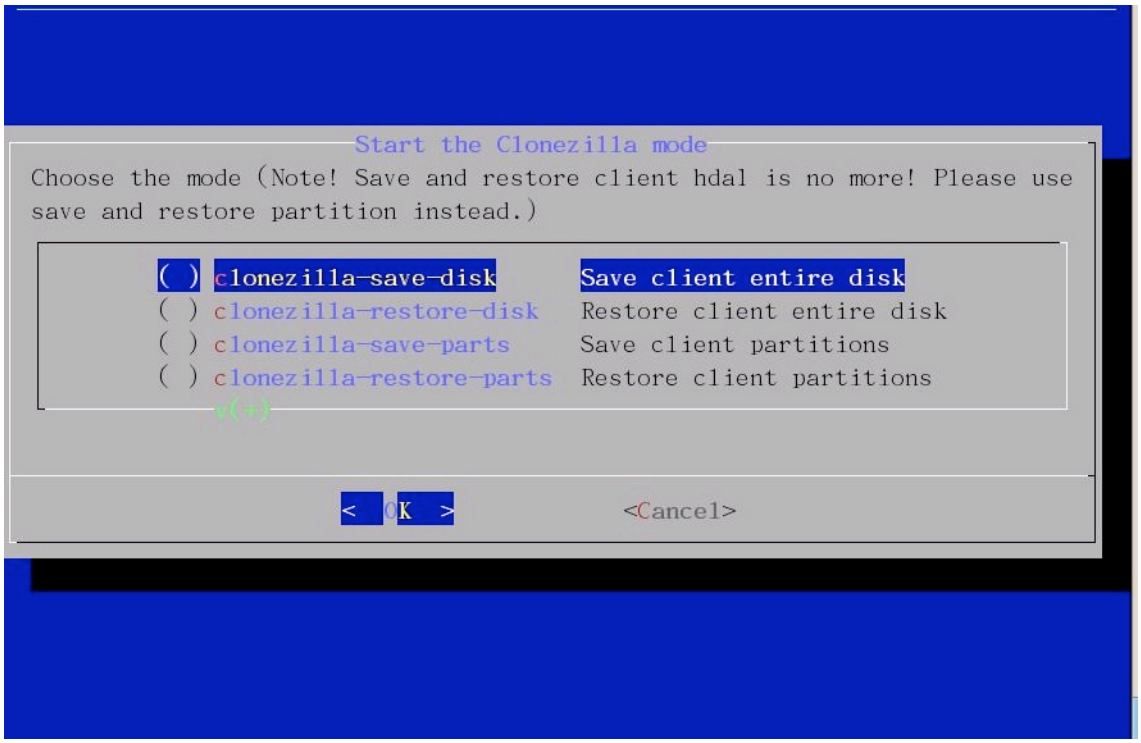


**Fig. 1.1** Clonezilla ncurses graphic interface

## 1.5 Technical objectives of the project

The most important purpose of this project is to solve those problems encountered on the previous version and to improve it. Next sections list the three pending problems that the project wants to solve:

Objective 1: To improve the “Microsoft Windows” performance

In the previous version of the project the boot of the Microsoft Windows was achieved but the OS executed with no BIOS support and was very slow.
In order to not loose performance on Microsoft Windows while boot without rebooting is necessary for this OS to access low level hardware instructions.
In order to compete with REMBO it’s mandatory to support this OS, which represent about the 90% of the actual OS market. Our objective is to find new tools to improve Microsoft Windows efficiency after its hot boot.

Objective 2: To develop a user friendly interface

Only command line interface was provided on the previous OS deployment system version, which was not very user friendly. The objective is to evolve into an intuitive graphic user interface.
Also the command line interface was not very intuitive to boot Microsoft Windows, because it required some commands to be introduced by the user. This problem will be solved automatizing all the process.

Objective 3: To introduce fast restoring functionality

The previous project had only the option for full image restoring which takes long time.

Tivoli Provisioning Manager for OS Deployment supports partial restoring, or fast restoring. This means that only changes on the client are downloaded instead of the whole image. Since an image can be as big as 6 gigabytes, saving this time for a normal user is precious.

This feature is very appreciated by users, so in this version of the OS deployment system will be included using one of the most popular open source software for file synchronisation, rsync.

## 1.6 Document organization

This document is organized in a way to make it easy to the reader to understand how does the system work and how does the upgrades affect to the previous version.

Next chapter will explain how does the base system work. Starting from the top layer of the system going down until understand all the internals of the booting

process until the restored OS is booted.

Chapter three introduces which technologies permit to use a graphic user interface to make the user life easier and how these technologies do integrate with the system.

Chapter four details the other two new important features: the fast syncing and the new complete restoration method. How a file system restoration does improve the byte-by-byte deployment.

The test done using the last version of the OS deploy system, had as consequence a couple of hard disks broken down. It was using a byte-by-byte writing system that could lead to break the device.

Then another way to restore the systems started to be searched. That's how PartImage software was found. This software not only takes care of the image creation and restoration, it includes a dedicate server and enough options to improve this process. In this version of the project we use PartImage as an improvement for the full image restoration but also as a tool for their creation

Finally conclusions show the new restoration times and how the project implementation could affect economically to a school like EPSC.
Also project environmental effects and future improvements have been written down to complete the thesis.

# Chapter 2 OS deployment system technical details

## 2.1 Introduction

The OS deployment system is based on booting Linux remotely in an empty client. To accomplish this, conjunctions of technologies are being used. The process starts on the PC booting and continues until and continues the image is restored and booted.
First the client downloads a small Linux kernel; this kernel has been configured to look for its root file system at the server.

Once Linux is loaded the user can select what to do:

- Restore and boot an OS image.
- Boot the installed or restored OS.

If the user selects the first option, an image restoring software will download and install the desired OS on the computer. Once the image is deployed the booting without reboot the computer process start.

For the user, the Linux loading and the image deployment is transparent, this improves the user experience that doesn't need to worry about the system configuration.

In a normal environment when a PC turned on a boot loader is executed, this boot loader starts an operating system from disk. The boot loader is installed on the local machine. Instead of using a local boot loader, the OS deployment system downloads a Linux to act as the local boot loader.

It is important to understand the technologies involved to have a complete vision of the system. Along this chapter these technologies will be introduced.

At figure 2.1 the basically functionality of the system is being illustrated. First when the client PC boot, it ask for the OS deployment operative system to be loaded via network. Then the user has the possibility to choose between the next options:

- Boot Linux
- Boot Microsoft Windows
- Restore Linux
- Restore Windows

If the user chooses one of the first two options, the OS installed on the local disk will be booted. If the user chooses to restore an OS, the OS deployment client will connect to the OS deployment server and ask for the selected operating system image to be downloaded.

When finished the downloaded OS boots, providing a clean operating system to work.

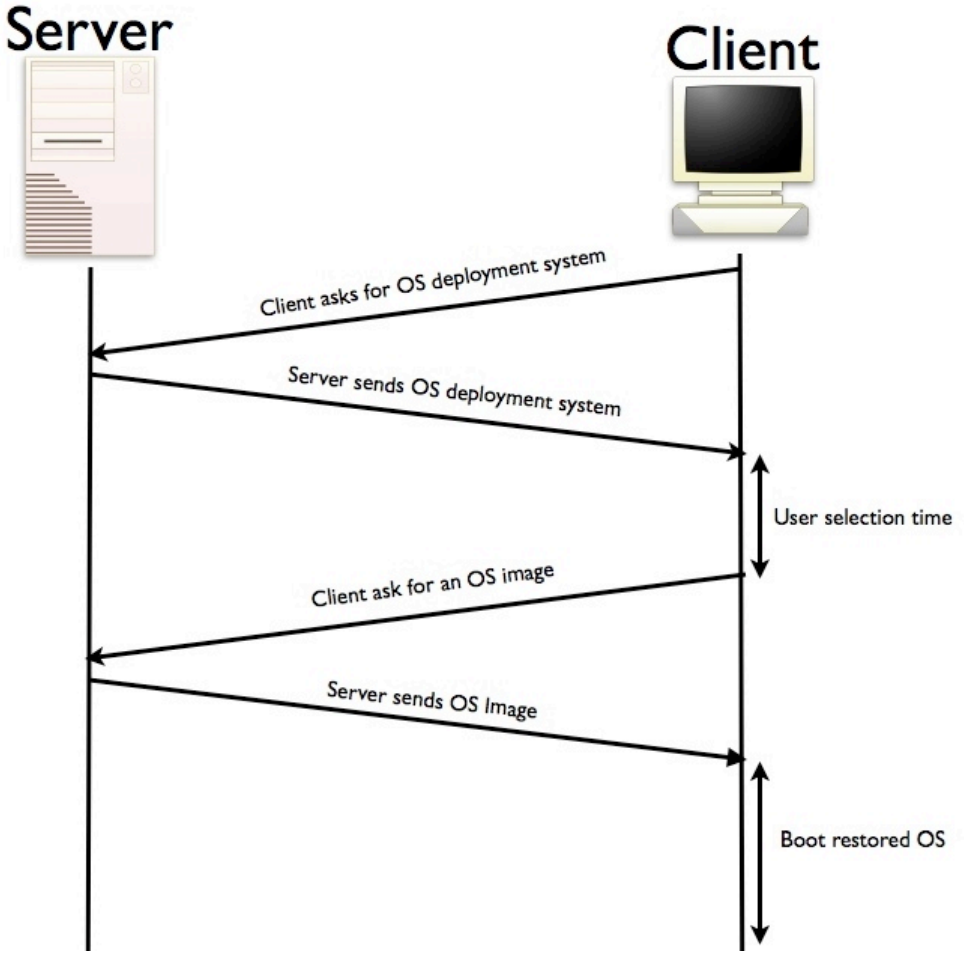


**Fig. 2.1** OS deployment system basic functionality

Now we have a complete vision of the system. Next sections will explain step-by-step and technology-by-technology how does the OS deployment system works.

## 2.2 BIOS environment

BIOS are the acronym for Basic Input/Output System [Ref 6]. This system is the first piece of software that a PC loads when it boot. BIOS are usually stored inside the flash memory of the motherboard, called EPROM.

First BIOS tries to identify the PC hardware and initiates it using the software embedded in each hardware device, the firmware. Once the hardware is initialized, BIOS loads the boot loader, which permit to boot an OS. Then the system memory is initialized, the BIOS typically copies/decompresses itself into that memory and keeps executing from it.

All these initial steps are made into the Real Mode but the rest will run in Protected Mode. Real Mode is based in 16 bits and this mode cannot map all memory. That is why it has to go into Protected Mode, which is based in 32 bits,

allowing accessing all the memory. At this point, CPU is up and running fine so it stands for 16 to 32 conversions.

Old operating systems relay on BIOS for many Input / Output operations, nowadays OS attack directly to the hardware being almost independent of the BIOS CODE.

**Linux and the BIOS handling**

From the Linux perspective, after booting, it gets data from the BIOS. In order to handle the devices directly without using the BIOS, improving the access to these devices. Linux introduces its own hardware addresses to manage the hardware. So once Linux is loaded, the BIOS calls are unreliable.
This data is loaded in the system memory so the BIOS loaded in memory can be overwritten.

**Microsoft Windows XP and the BIOS handling**

As Linux, Microsoft Windows XP tries not to rely on the BIOS to improve its performance, but in fact this is not totally achieved. Microsoft Windows relies on seven BIOS interruptions. A study done by Adam Sulmicki of the Maryland Information System Security Lab shows how Microsoft Windows XP relay on the next BIOS services [Ref 7]:

- Video Interface
- Equipment Check
- Fixed Disk/Diskette Interface
- System Functions Interface
- Keyboard Interface
- System Timer Interface
- User System Timer Interrupt

This information is important to understand why in the previous version of the OS deployment system Microsoft Windows XP was executed under performance: The PXE Linux execution destroyed the BIOS code on RAM and when the Microsoft Windows was loaded without rebooting the BIOS code was not found on fast RAM but on slow EPROM memory.

## 2.3 Boot loader

A boot loader is a small piece of software that allows booting an operating system once the BIOS has finished its job.
The first stage of a boot loader is located in a special place on the disk; this space is located on the first 512 bytes of the Master Boot Record. The Master Boot Record is a space on a disk device reserved to store information related to the booting process.

The MBR is quite small, and boot loaders have grow, so once the first stage has been loaded into memory the software jump to its second stage allocated on the disk. This second stage has more complex code to find bootable devices or boot different operating systems. When the second stage is loaded the software is able to load the operating system and transfer the execution to it.

There exist lots of boot loaders, the most important are:

**GRUB**: is a boot loader from the GNU Project. GRUB allows a user to have different operating systems on his computer at once, and to choose which one to run when the computer starts. GRUB can be used to select from different kernels

One of the most important characteristics of GRUB is its capability to load a kernel received from a network.

**LILO**: Linux Loader [Ref 8] is a boot loader for Linux. As GRUB, LILO does not rely on any file system, and can boot any operating system from different sources except the network.

**NTLDR**: NTLDR, NT Loader, is the boot loader for the entire NT kernel based Windows (Windows NT, Windows 2000, Windows 2003 or Windows XP).

## 2.4 Network boot

The network booting is based on a conjunction of protocols. Using these protocols over the network a boot loader is sent from the server to the client. Then this boot loader can ask for the kernel image and boot it.

Our OS deployment system is a network based system, and needs no apiece of software installed on the client hard drive. The software solution, which is installed on the server hard drive, is a network based boot loader, PXELINUX.

To have PXELINUX running is necessary to install two services: a DHCP and TFTP. These services are a big part of the Pre-Execution Environment specification (PXE).

PXELINUX is software for booting Linux off a network server, using a network ROM conforming to the Intel PXE (Pre-Execution Environment) specification. The boot loader is a file called PXELINUX.0.

### 2.4.1 PXE

It is a specification given by Intel in order to boot computer using a NIC (Network Interface Card). The PXE protocol is a combination of DHCP and TFTP. The first one is used to configure the network interface card IP if it is not

configured also DHCP is used to send the TFTP'S server IP address. TFTP is the acronym for Trivial File Transfer Protocol, and is designed to send files over the network.

The TFTP server contains the boot loader, which is sent to he client and loaded inside the PC memory. PXELINUX will be the boot loader downloaded by the client at this project. This boot loader demands to have the kernel at its same folder in consequence it downloads the kernel via TFTP too.

### 2.4.2 DHCP

Dynamic Host Configuration Protocol is a protocol designed to configure a network card. Giving parameters such as IP, netmask, default gateway and DNS.

Basically DHCP starts as link layer protocol, sending a DHCP discover packet broadcast over a local area network. The available DHCP servers answer to this discover packet. Then the client accepts one of the answers and waits for the configuration to be sent.

The DHCP service has to be only installed on the server. The PXE network interface card has stored the DHCP client at the ROM.

### 2.4.3 TFTP

Trivial File Transfer Protocol (TFTP) is a very simple file transfer protocol. It uses UDP, cannot list directories and it has neither authentication and nor encryption.

TFTP uses UDP; this means it has not error control, not retransmission of lost packets during the transmission.

The TFTP service has to be only installed on the server. The PXE NIC has stored the client at the ROM.

## 2.5 Root file system over the network

After sending the kernel, with TFTP from server to the client, it needs all the applications to restore the system and libraries. There are two alternatives for the root file system: a RAM file system or a network file system.

A RAMFS is a special kind of file system, which compresses a whole file system into a file, so this file can be transmitted over the network and decompressed on the client, on top of RAM memory. Once is decompressed, it runs a whole file

system on the memory not in any disk device.

The second solution is to access over the network a remote file system, so the client only access to it when required, loading into memory the selected executables.

The OS deployment system is based on network file system because there were different problems with the RAM file system:

- It is not a comfortable technology to manage.
- It increases the amount of time to load the OS deployment system, because the client has to download it and decompress it.
- The use of a RAMFS file cannot be very big in order to reduce this time, so it cannot have all the applications needed installed.

In contrast a network file system, is not downloaded, the file system requirements are read only, and it is easy to manage because it is not compressed. As a solution seemed more adequate to the system.

### 2.5.1 Network file system

Network File System (NFS) is a file system protocol developed by Sun Microsystems, allowing a computer to access files over a network as if the network devices were attached to its local disks devices.

NFS is based on remote procedure calls, an interprocess communication technology that allows software to make an execution on another computer over the network.

Inside the OS deployment server a NFS service has been installed and it has been configured to export an entire root file system: the OS deployment system client file system. The client must have installed and running the NFS client. Client and server must have configured their kernels to support NFS so the kernel will create a device called NFS.

The boot loader has to be configured to pass the NFS device as a parameter to the kernel. This way the kernel knows that it has to look for the root file system at the server not at his local disk.

## 2.6 Restore an OS image

At the previous version the way for restoring Image was the conjunction of two open source software: Device Imager [Ref 6] and netcat [Ref 7].
DeviceImager was the software selected to make and restore images on the previous version of the project. This software was based on 2 utilities zsplit and unzsplit.

Zsplit creates exact images of the disk without taking care of the file system architecture. Unzsplit recovers the images of the disk created by zsplit.

To send the image through the network netcat was used, netcat is the application that let a user create UDP/TCP connections using command line. Then netcat was used to create TCP connections in order to send the OS image created by Zsplit. These images were processed by netcat on the client and sent to Unzsplit via a pipe. Unzsplit restore the received image on the disk.

Device Imager software is not optimized for this function, it is not a fully qualified image server, and so to each image transferring the OS deployment system created a netcat connection.

In summary the conjunction of DeviceImager plus netcat does not have enough features due to its simplicity and it was also quite aggressive for the disk. It's not the best solution for an OS deployment system.

### 2.6.1 Boot without rebooting: hot boot

The kexec functionality is the work of Eric Biederman [Ref 9].
Kexec lets the Linux kernel to boot directly to a new kernel fro the current running one. Kexec is a kernel patch that permits to skip all the initial boot process. So there is no hardware test, and no firmware loading, there is not even a boot loader involved.
Kexec bypasses the firmware stage so the state of the devices is unreliable.
In a very simple way, what kexec does is to:

1. Copy the new kernel into memory.
2. Move this kernel image into dynamic kernel memory.
3. Copy this image into the real destination (overwriting the current kernel).
4. Start the new kernel.

### 2.6.2 Microsoft windows hot boot

Kexec only works for ELF executables formats or its compressed forms. Windows kernel is not an ELF executable but we found grub.exe ELF executable that can hot boot a Microsoft Windows OS.

Grub.exe is a version of the Grub boot loader to be installed on Microsoft OS to substitute Microsoft NTLRD boot loader. Besides executing from Microsoft operating systems it's an ELF executable, too. Therefore it can be kexecuted.

As mentioned on section 2 we still had the problem of the Microsoft Windows calls to the BIOS:

- BIOS Video Interface
- BIOS Equipment Check
- BIOS Fixed Disk/Diskette Interface
- BIOS System Functions Interface
- BIOS Keyboard Interface

- BIOS System Timer Interface

When kexec passes control to grub.exe and this to MS Windows, the BIOS are unreliable, so it produces a problem in the overall performance of the system. The solution was to add a BIOS emulation layer inside grub.exe, before it boots Windows.

The new grub.exe version had this improvement into :Integrating BIOS emulation. This improves the Microsoft Windows performance, making the hot boot almost perfect.

### 2.6.3 GNU/Linux Kexec

Since Kexec was designed for Linux kernels. Including the right parameters it works flawless.

## 2.7 Final system

Figure 2.2 shows the final system detailed operation of the deployment system. In order to have an OS deployment system working the client BIOS is configured to boot via the network card using PXE. At boot time, the client search a DHCP server, then the server sends an IP address, a netmask, a default gateway, the TFTP server IP address and the name of the boot loader file (PXELINUX).

The client looks for the TFTP server and asks for the PXELINUX boot loader file. Once it is downloaded into memory this asks to the TFTP server for the kernel image. The tftp server sends the kernel to the client and PXELINUX boots the downloaded kernel.

This kernel is configured to find the root file system, on a NFS server. It connects to the NFS server and Linux loads in a normal way. At this point a diskless system is up and running.

After the system is booted the user can select to recover or to boot a resident OS. If the user decides to recover then, the OS deployment system connects the deployment services and downloads the image.

If the user chooses to boot the system, or if the image has been just recovered the next step is to boot the system. The booting without reboot system is accomplished again using the special call of the Linux kernel called kexec.

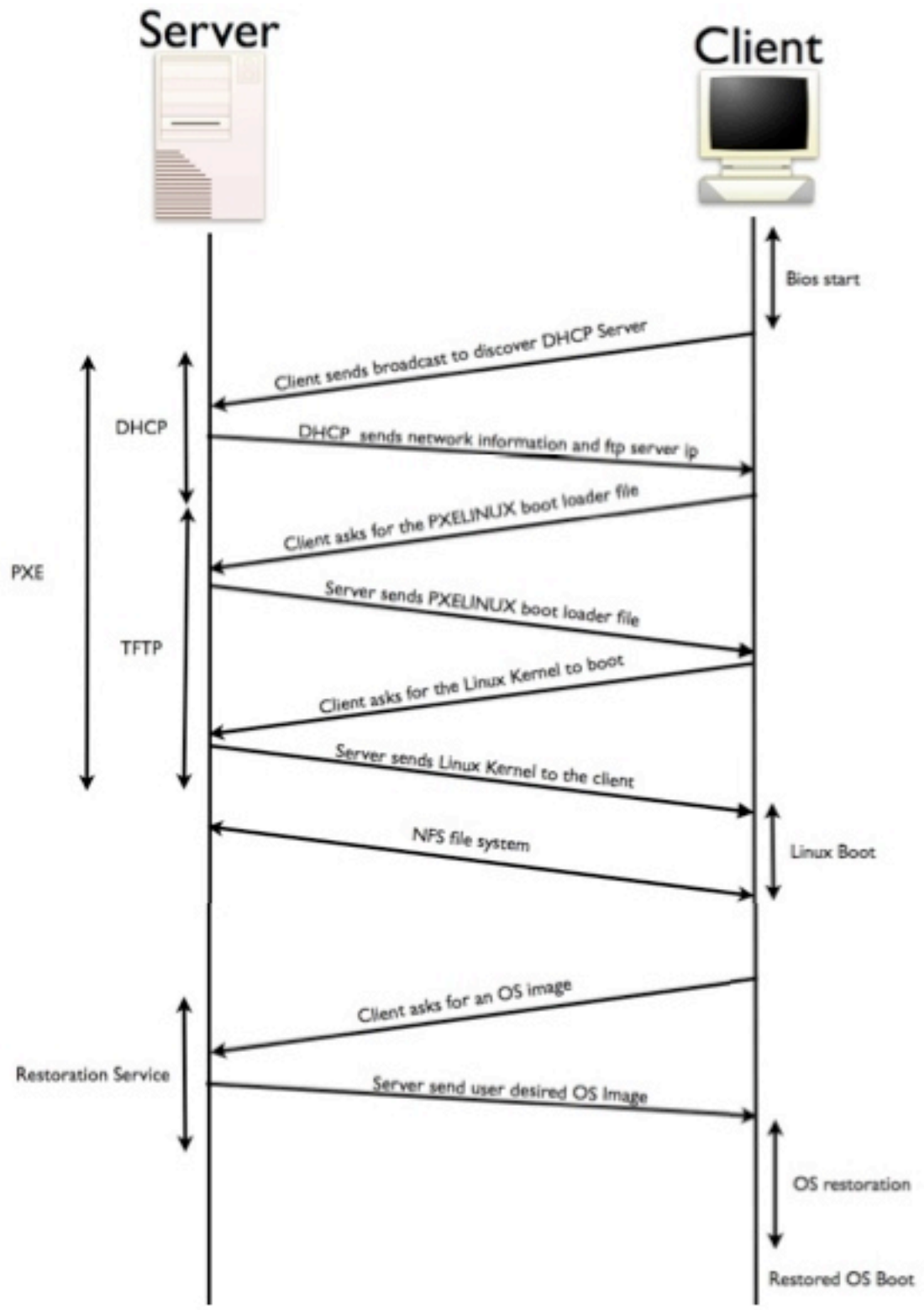


**Fig. 2.2** Detailed diagram

# Chapter 3 Graphic User Interface

## 3.1 Introduction

In the last version of the OS deployment system, there was no Graphic User Interface (GUI) at all; it was a Command Line Interface (CLI). It was working fine enough but it was not user friendly.

A graphic user interface for an OS deployment system must be simple, easy to manage and as beautiful as possible:

- Easy user graphic interface
- Interaction via mouse and keyboard

Due to size the limitations of OS deployment system, installing a whole graphical system (Xorg Server) on the client computer will increment considerably the time of boot process. Moreover, installing the Xorg server will make harder the development of the whole GUI, so we use another solution.

## 3.2 Technological design decisions

The solution for the GUI of the OS deployment system is to create graphics on a virtual console, and to use a basic web browser for text with support to display graphics on console.
The technology used to provide graphics on a console is known as Framebuffer [Ref 10], is widely used to improve visual style of the command line interface as it is shown on the image below.

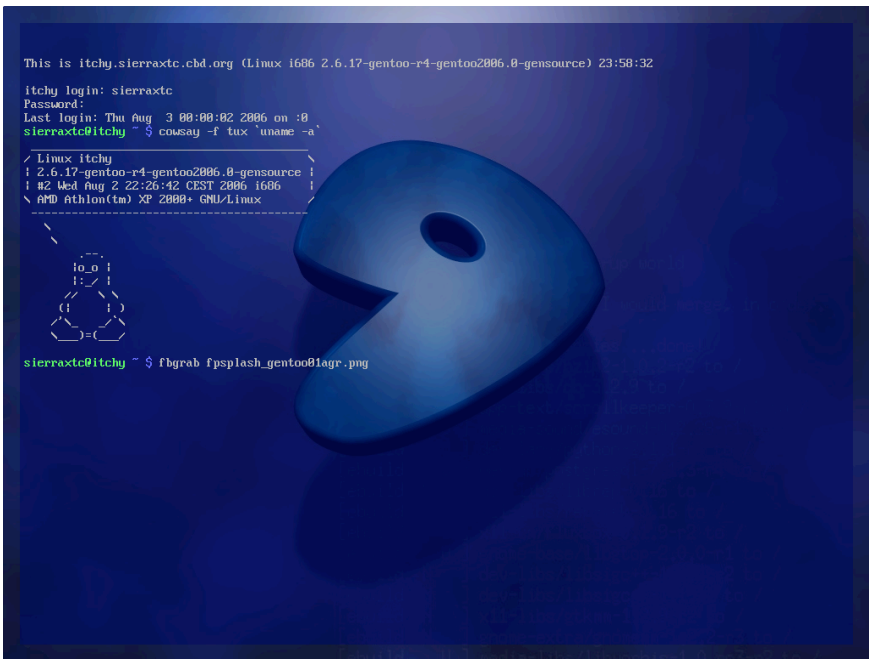


**Fig 3.1** Example image of a console using Framebuffer

Also Framebuffer console can be used on conjunction with boot splash software to show beautiful images on the Linux booting time instead of the old fashioned black screen that users were used to. The boot splash software selected is gensplash; this software is the Gentoo Linux boot splash software. The client is based on Gentoo Linux distribution so it is logical to install this specific software.

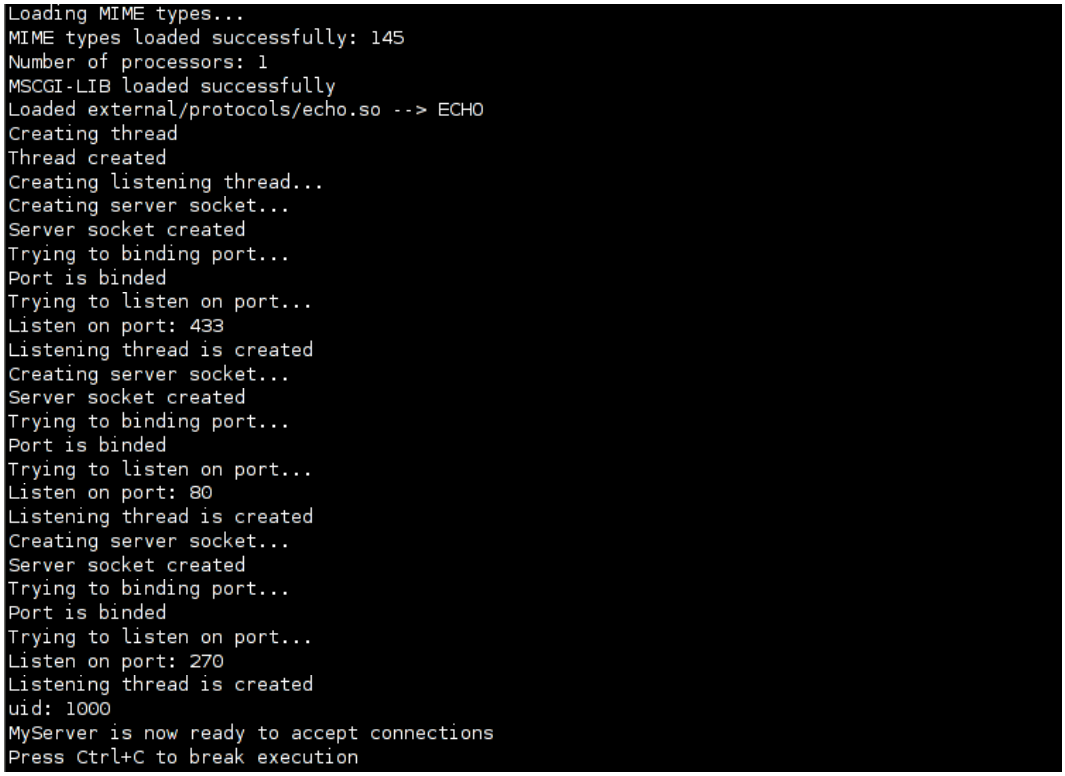


**Fig. 3.2** Example of image booting without framebuffer and gensplash

The image 3.2 shows a boot screen without the selected technologies. As it can be observed is not a friendly interface.

For the web browser application Links2 has been the chosen because it provides us support for framebuffer devices. Links2 has also a wide diversity of image formats and JavaScript support.

To give service to our web GUI, the selection was clear: Apache2 [Ref 11]. This web ensures the reliability and easy configuration, for the OS deployment system.

The following sections describe the chosen technologies with more detail.

## 3.3 Framebuffer

Framebuffer is a virtual device that supports graphics on a console without using any kind of library. This device is a video output that displays video from a

memory buffer. This memory buffer contains a complete frame of stored data. The buffer contains information of the colour values for every pixel showed on the screen.

The first framebuffer implementation was created to allow the Linux kernel to show graphics on a text console on systems that do not have one; like the Apple Macintosh.
Later it was used on IBM PC compatible architecture; To see if it is activated just verify if the Tux logo is shown at the boot time (one logo per CPU), as shown in fig 3.3


**Fig 3.3** Boot with the framebuffer activated

The most popular use of framebuffer is to display Unicode character on the Linux console. When there was no framebuffer the support for Unicode was impossible because VGA console fonts had a limited size of 512 characters.

There are lots of software packages, which can use the framebuffer device like Links2 web browser, used in this project, or the multimedia player Mplayer. Framebuffer device must be supported by the application to use it.

Furthermore framebuffer also supports hardware acceleration framework using the Directfb. Directfb provides translucent effects as well as multiple display layers [Ref 12].

The development of framebuffer drivers for Linux started at 1999 and nowadays it continues giving support for the new hardware.

### 3.3.1 Framebuffer configuration

Framebuffer device is part of the kernel configuration, so it is needed to configure it in the kernel compilation. Kernel sources are placed at /usr/src/linux, from there the execution of the kernel menu configuration will prompt a graphic menu.

The framebuffer driver is located at the Device Drivers section and Graphics Support subsection.

One should select the option "support for frame buffer devices" and activate support for the correct driver. In most cases “vesa-tng” driver should work fine. It's important to select “console display driver support”, too.

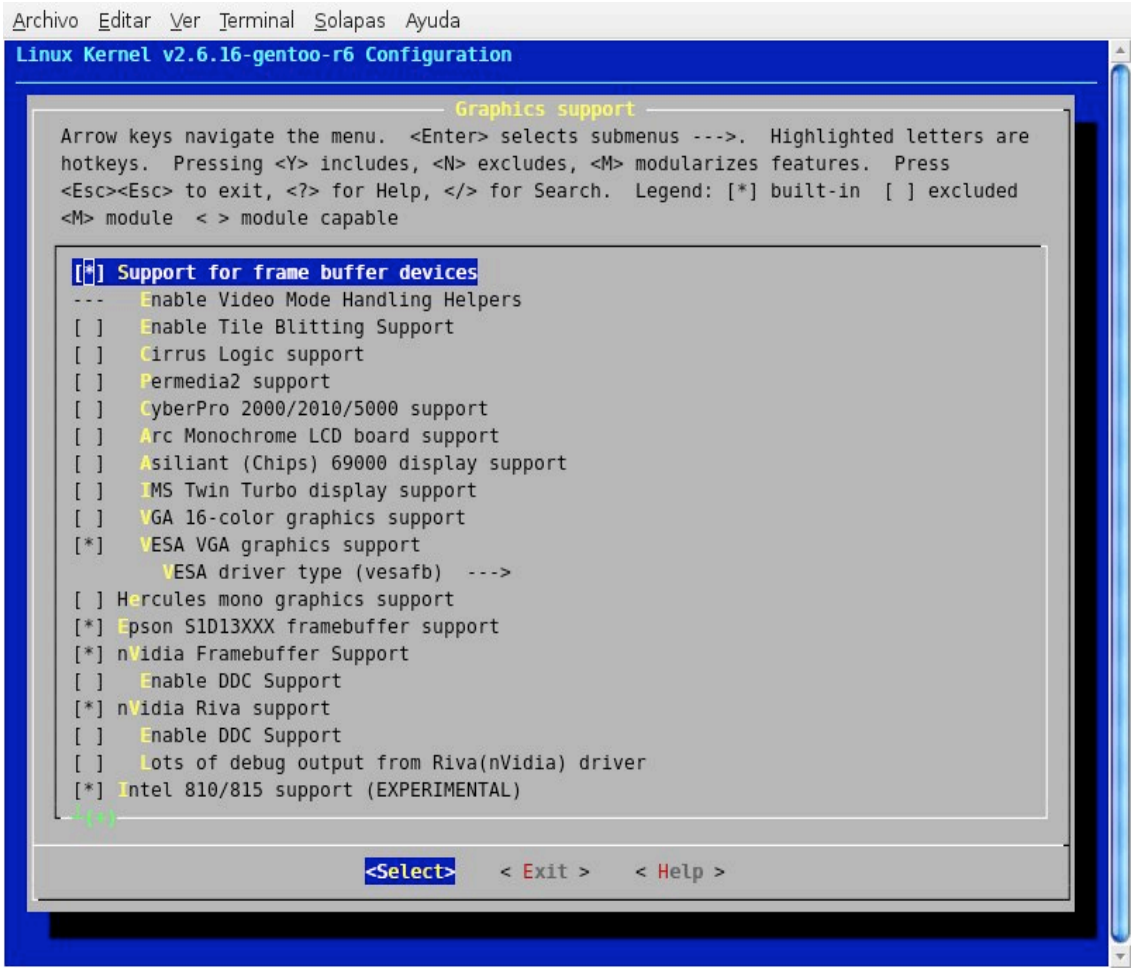


**Fig. 3.4** Kernel menu configuration

Figure 3.4 shows how the kernel menu configuration looks; particularly it is a snapshot from the “Graphics support” kernel properties. At this section it is possible to configure every kernel feature related with graphics on console.

We load the “video mode selection” and “frame buffer console support”. Also to make the Tux logo appear at boot, enter to “Logo configuration” and select the logo with more colours.

Since the system that must be used in heterogeneous hardware, the kernel must support all the framebuffer drivers; this way independently of the hardware is running on framebuffer will work.
Once the drivers are loaded it is time to configure and compile the web browser to work with framebuffer as shown in the next section.

It is very important to compile PS/2 mouse and keyboard drivers as well as USB to let the user interact with the GUI.

## 3.4 Links2

Links is an open source text and graphics web browser developed by “Mikulas Patoxla” and his group Twitbritghtlight labs. Links2 was developed later and the most important features were added graphics support via framebuffer and X.org server, and Java script enabled browsing.

The image below shows an example on how does Google home page looks’ using Links2 with framebuffer activated.


**Fig. 3.5** Example of Links2 looks using a framebuffer device

Some of the most important features of Links2 are:

- HTML 4.0 support (except CSS)

- HTTP 1.1 supports
- Tables and frames in both graphics and text mode.
- Built-in image display for GIF, JPEG, PNG, XBM, TIFF in graphics mode
- Anti-advertisement animation filter in animated GIFs
- JavaScript support with full user control over script run
- Bookmarks
- Background file downloads
- Automatic reconnection in case of TCP connection breakdown
- Keep alive connections

To make Links2 fully compatible with framebuffer it must be compiled with framebuffer flags enabled. OS deployment system is based on Gentoo Linux distribution so to compile any software with the necessary flags is really easy. The system needs Links2 to be compiled on the client file system, which is the one that shows the GUI. The server contains the root file system of the client so using the chroot command it is possible to change the root file system and to install software on the client's root file system.

## 3.5 Gensplash

Gensplash is a project started by Spock [Ref 13]. Gensplash provides a set of tools, which could improve the Linux boot process by displaying nice graphics and effects on screen.

Kernel has to be configured to support splash images. First on the "General Setup" section "Initial RAM file systems and RAM disk (initramfs/initrd) support" must be checked. This will create a RAM file system, which later will be compiled in the kernel.

Splashutils is the name of the package, which contains the tools for the boot splash creation. This package must be installed.

Once the kernel is compiled and the tools are installed, it is time to compile the boot image inside the kernel using a ramfs image. When the Linux kernel is compiled it creates an empty initramfs. Splashutils can fill the ramfs with the image and the configuration of the boot splash image using the next command:

```
$splash_geninitramfs –g /usr/src/linux/usr/initramfs_data.cpio.gz –v –r  800x600 theme
```

Linux kernel path to the ramfs is the first parameter, we add a verbose with –v, and -r to select the resolution and the last parameter indicates the boot splash theme.

Once the ramfs is filled, the kernel must be recompiled to include the ramfs inside.

To end the boot loader, in our case, PXELinux has been configured to use gensplash.

## 3.6 GUI design and scripting

The design of the graphic user interface is based on the features of the Links2 browser. As a premise, there was the decision to make it as simple as possible to make the user experience an easy thing.
A table and simple colours should be enough in a similar way as Google shows on its home web page. Figure 3.6 shows the final graphic user interface's screenshot of the project GUI.

Using a web browser, as a GUI, in our architecture, has as trade off that there is no direct way to make local executions. OS Deployment needs to execute, on the client computer, some scripts to restore & boot the different OS.
To accomplish this the OS deployment system starts in the background a netcat process that runs in parallel with the links2 web browser. The netcat software starts listening on the port 80 at localhost at the client's boot time. When a user click over a link, this link point to an URL similar to this:

http://localhost/usr/bin/ls

As said before netcat is listening so it receive this HTTP petition and pass it using a UNIX pipe to a script which functionality is to parse the petition and execute /usr/bin/ls on the localhost.


**Fig. 3.6** Final Graphic user interface

Illustration 3.7 shows how the Apache Server is serving html petitions related with GUI. When it is needed to execute a local script on the client, the http petition is not directed to the Apache server, is directed locally instead.

Then netcat is listening and it passes the petition to a script: wwwexec.sh, which parses the URL extracting the path to the local execution. In an URL like http://localhost/usr/bin/ls, the script will extract /usr/bin/ls and execute it. This is how the OS deployment system gets to execute its scripts to restore OS, making it transparent to the user.

Six scripts have been created one for each disposable option of the system, in addition to the wwwexec.sh script mentioned before:

- Boot Windows (windowsboot)
- Boot Linux (linuxboot)
- Linux fast restoration (syncL)
- Windows fast restoration (syncW)
- Linux complete restoration (linrestore)
- Windows complete restoration (winrestore)

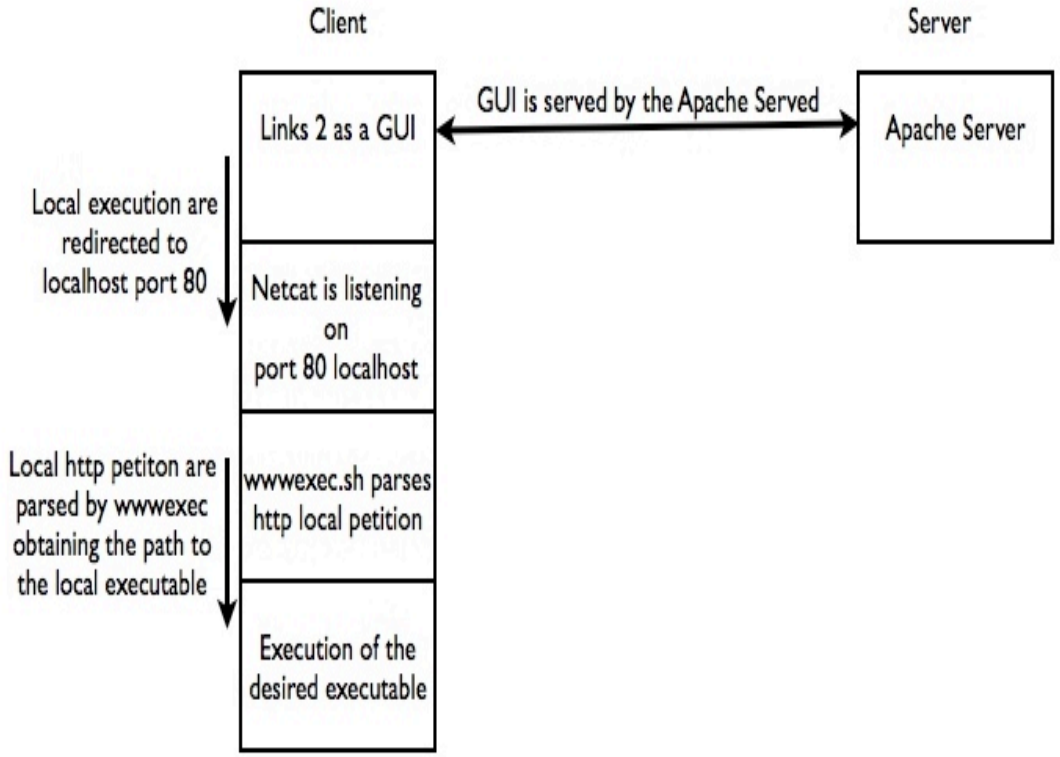


**Fig. 3.7** GUI communication between applications

# Chapter 4 System restoring improvements

## 4.1 Introduction

In the last version of the OS deployment system the restore system created exact copies of the disk's partitions, which were created by using low-level byte-by-byte copy operation.

This process allows the end user to make a complete restoration without taking care of the system structure or the information stored inside the files. This method becomes ineffective when a little change is made because it has to restore the whole disk instead of the updated file, which will be faster.
Fast image restoration can be achieved using file system synchronisation software. Rsync is software which lets synchronise a whole OS, testing the file integrity. We call to sync a file system to the action of synchronise it by means of rsync.

By the time the OS deployment system was developed last year, the file system syncing was only available over open file systems, so Microsoft's NTFS was excluded from it. This is the reason why it was not implemented. Nowadays, thanks to new technologies, which will be explained along this chapter, it's possible to synchronise with NTFS, Microsoft's privative file system.

Thanks to file system support now it's possible to use new software to make the system restoration. This software is called PartImage and it improves in many ways the capabilities of the OS deployment system. It demonstrates once more the power of the system; just installing the software it's possible to improve the whole deployment system.

First in this chapter file systems will be introduced. Then type of file systems and nowadays file systems on the two most important operating systems will be explained. Later the Microsoft Windows NTFS Linux driver problem will be treated.

Once the file system paradigm has been understood the new technologies used in this project for OS restoring will finish the chapter.

## 4.2 File systems

File systems are a logical organisation method for the computer's physical data. This organisation allows the computer to search for information about files, or directly the files, gave permission to files, copy and move files and large list of possibilities.

In a more formal way to explain it, a file system is a set of abstract data types

that is implemented for the storage, hierarchical organisation, manipulation, navigation and retrieval of data.

File system also stores data such as file names, length of the file, last date of modification, creation date and file owner among other properties.

Looking at its functionality there are 5 types of file systems [Ref 14]:

- **Disk file systems**: are designed for storage devices such as hard drive, USB pen drives and similar block devices.
- **Database file systems**: are oriented to work with databases, files are identified by their attributes, like type, author, …
- **Transactional file systems**: log events or changes. This feature let a system to recover data from failure.
- **Network file systems**: are dedicated to work over the network, providing access to files on a server.
- **Special purpose file systems**: A special purpose file system is any file system that is not mentioned above, like virtual file systems.

In this work the focus will be the disk file systems, the network file systems and the special purpose file systems.

Each OS has it's own file system for the storage devices (disk file system). Linux supports a big variety of file systems and some of they are proprietary file systems as MAC OS HFS.

As an operating system, the OS deployment system has to support most common file systems.

## 4.3 Disk file systems supported by the OS deployment system

Linux & Windows are the target image of the OS deployment system; Linux uses, in most cases, ext2, ext3 and reiserfs as file systems. Windows uses FAT32 and NTFS.

### 4.3.1 Linux file systems

#### Ext file system

At the beginning of the Linux development the Minix file system [Ref 15] was the default and the only file system supported. This file system had a maximum size limit of 64 megabytes and a file name length limit to 14 characters. The extended (ext) file systems was created to solve these limits pushing the maximum size to 2 gigabytes, and the file name length to 255 characters.

The ext file system was based on inodes as all the UNIX file system were. An inode is a data structure, which stores basic information about file, directory or any file systems object.

**Ext2 file system**

The second extended file system (ext2) was introduced on early 1993, to solve 3 important issues of the extended file system: there was no support for separate access, inode modification and data modification timestamps [Ref 16].

The separate access introduced the security domains of Unix in the Linux files; this is the read/write/execute permission for the user/group/other domains.

The inode modification permits the file system to change basic information about a regular file, directory or any other file system object. In order to know when data has been changed it is important for a file system to add data modification timestamps.

The next illustration shows extended file system's inode internal structure, and how to find the data blocks of a single file.

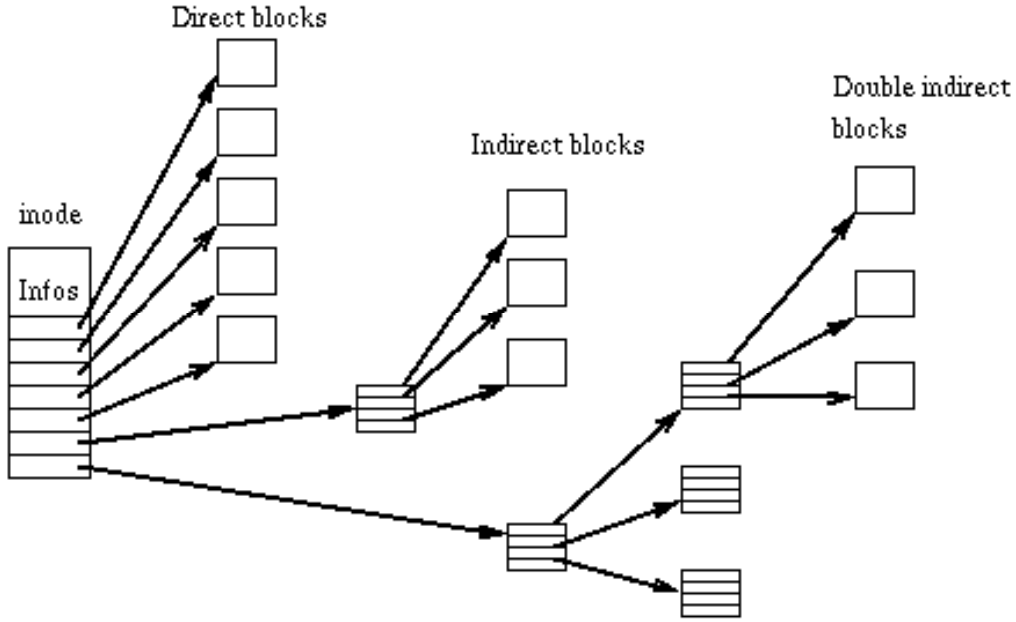


**Fig. 4.1** Inode internal structure

**EXT3 file system**

The third version of the extended file system was released on November 2001 [Ref 17].
The ext3 file system adds, over its predecessor:

- **A journal**: this technology allows to log changes to a journal before writing them to the file system. It is similar to a transactional file system

to a disk file system. A file system with journaling reduces its possibilities to become corrupted in a power failure or a system crash.
- **Tree-based directory indexes for directories spanning multiple blocks:** In large directories this tree structure allows faster access to the files.
- **Online file system growth:** This functionality let the file system to change its size once it is created.

### 4.3.2 Microsoft Windows file systems

#### FAT File System

FAT is a file system supported by almost all the existing operating systems due to it's simplicity, well known structure and compatibility. Bill Gates and Marc McDonald created the FAT file system in 1977 for managing disks in Microsoft Disk BASIC [Ref 18].

FAT is the acronym of File Allocation Table. This table centralizes the information about which areas belong to files, which areas are free or corrupted, and it includes where is the file located in the disk, too.
Disk space is allocated to files in contiguous groups of hardware sectors called clusters.

A cluster is de minimum assignation unit; it's a multiple of a sector or block. A sector or block is the minimum transfer unit and have a size usually from 512 bytes to 4Kbytes.

The first version of FAT limits the cluster address to 12 bytes. This limitation lets the file system to 4096 clusters. In case of clusters of 4 blocks of 1Kbyte a total of 32 Megas could be stored on a disk.

Due to the growth of the hard disks FAT16 was released increasing the cluster address to 16, supporting to 2 Gigabytes of disk space. Years later Microsoft created a new version with 32 bytes of cluster address improving its capabilities of space until 2 Terabytes. Microsoft limitated the space allowed on FAT to 32 gigabytes in their systems reporting performance problems. FAT32 cannot contain a file larger than 4 Gigabytes.

The FAT file system does not contain mechanisms, to prevent newly written files from becoming scattered across the disk.

FAT does not have an internal structure so files are given the first available location on the volume. The first free cluster number is the address of the first cluster used by the file. Each cluster contains a pointer to the next cluster in the file, or an indication (0xFFFF) that this cluster is the end of the file.

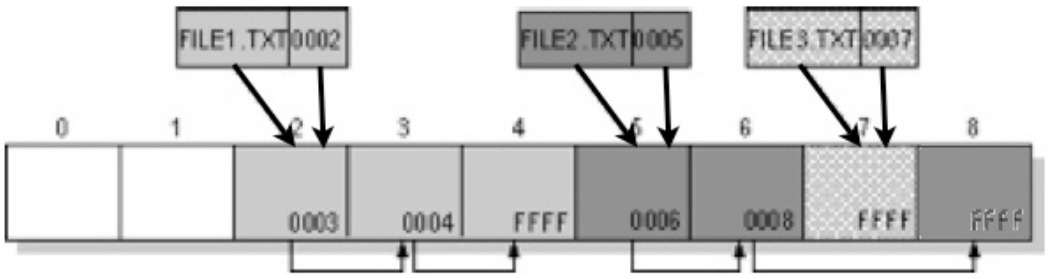


**Fig. 4.2** File allocation table internal structure

This illustration shows three files. The file File1.txt is a file that is large enough to use three clusters. The second file, File2.txt, is a fragmented file that also requires three clusters. A small file, File3.txt, fits completely in one cluster. In each case, the folder structure points to the first cluster of the file.

The solution to the disk fragmentation is the linkage of all free clusters into one or more lists as is done in Unix file systems. The file allocation table has to be scanned in order to find free clusters, which affects directly to the performance using large hard disks, where that larger allocation table are being used.

| Boot sector | More reserved sectors (optional) | File Allocation Table #1 | File Allocation Table #2 | Root Directory (FAT12/16 only) | Data Region (for files and directories) ... (To end of partition or disk) |
|---|---|---|---|---|---|

**Fig. 4.3** FAT file system internal structure [Ref 18]

This image shows how the disk is organised using the FAT file system. First of all the boot sector, the two file allocation tables: the original and the backup one.  Then the data region until the end of the partition.

## NTFS

Since Windows XP, Microsoft's operating systems use NTFS as standard disk file system. Microsoft Windows is the most used operating system so to give support for its file system was mandatory in our project.

The information about how does NTFS work is not very clear because of the closed source politics used by Microsoft.

NTFS is quite similar to FAT in the way that it has a table too, but instead of a normal table is more similar to a data base, creating a more complex and fast file system [Ref 19].

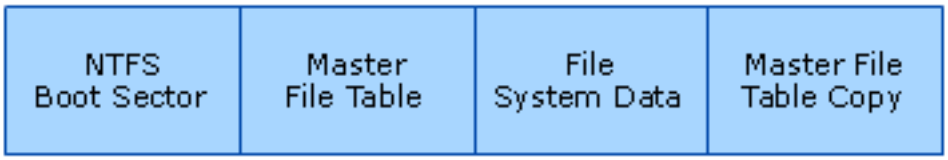


**Fig. 4.4** NTFS file system internal structure [Ref 19]

Figure 4.4 shows how NTFS is structured. The NTFS boot sector contains the sufficient information for the system to boot. The Master File Table is the new File Allocation Table designed by Microsoft in order to prevent fragmentations problems.

The File System Data is described at the Microsoft Tech Net Web as a place that stores data that is not contained on the Master File Table.
To improve the security of the file system, a copy of the Master File Table is placed on the NTFS structure.

We found any information available about the Master File Table structure.

## 4.4 NTFS and its Linux compatibility

Microsoft private code politic is the biggest problem on the development of the NTFS driver. Large test and analysis research was the only possibility to understand the internal structure and to create the driver.

NTFS is build as a database, any change made on it requires modifying other part of the file system to continue having consistence on the file system. Any mistake done on this process will conclude on damaged files or even worst, the whole file system destroyed.

Opposite to FAT, which worked on a simple table, NTFS works with a very complex table called Master File Table. This table controls everything within the file system using a relational database, which makes the development of the NTFS driver a hard task.
The new structure on NTFS solves the fragmentation problems FAT had.

Nowadays there are two NTFS drivers, one driver on kernel space, which can only read NTFS file system, and a user space driver, which can read and write NTFS without problems.

The development of a driver for Linux that handles this database was complex, as most developers know, to develop software for the kernel is quite difficult. Due to the importance of these drivers to the Linux community the NTFS driver developers decide to make their work easier creating a driver on user space with a new technology called FUSE. In any case a complete NTFS kernel driver is just a question of time.

### 4.4.1 The FUSE driver

FUSE (File system in User Space) is a technology, which provides an abstraction layer to the developer in order to create file systems in the user space.
Understand how FUSE works helps to have a complete vision of the solution adopted by the NTFS driver developers. FUSE is a virtual file system in user space. What a file system is has been explained during the thesis. What a virtual file system is, and what user space means will be explained in the next sections.

### 4.4.2 User space and kernel space

An operating system segregates memory in two spaces using the processor mode bit:

- Kernel space
- User space

Kernel space is reserved for running the kernel, device drivers and kernel extensions.
User space is that place in the memory where the end-user runs their applications.
An application is divided in process, as each process requires some of the user space memory. Applying memory space separation, the operating system ensures that the kernel will always have memory enough to run and will not be corrupted by user applications.

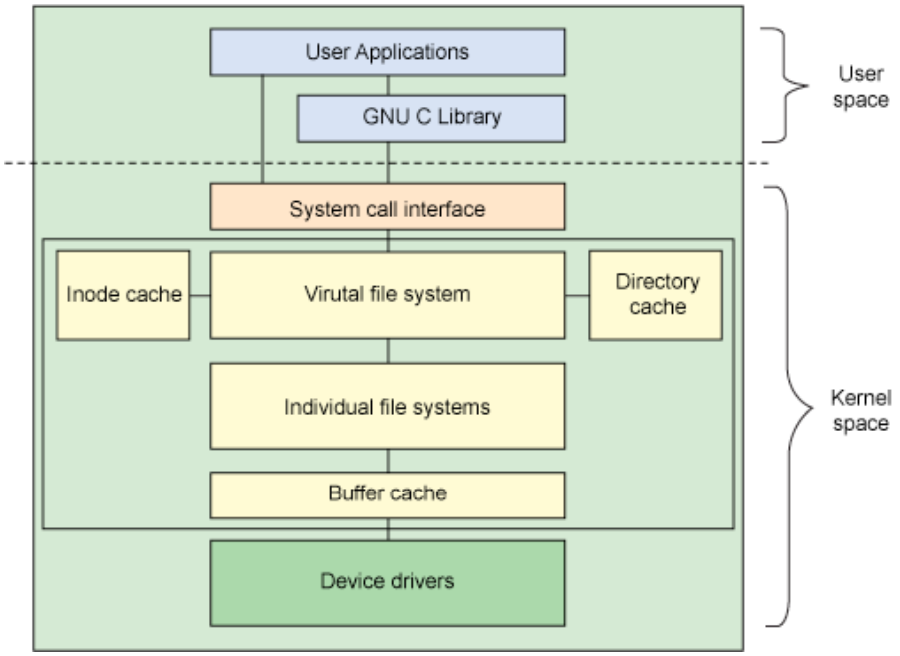


**Fig. 4.5** Operating system space divisions [Ref 20]

At the above image it is possible to see how the user space and the kernel space are divided. On the user space only user applications and libraries are allocated while at the kernel space device drivers and file systems are being handled.

### 4.4.3 VFS

A virtual file system is an abstraction layer over a real file system. This means that it is possible to build a file system related to a concept, which can be abstract to the kernel.

To make it clearer let's explain it with an example:
Imagine there is a folder, plenty of mp3 music; the problem with the folder is that it is quite messy. To make an easier access the user wants to sort it by the artist and by the album's name. The user will need to create two folders (Artist and Album), and copy the files twice, once in each folder, in order to sort it correctly.

Using VFS is possible to create a file system, which reads the information of your mp3, and sorts it for each folder. This means that the file system will auto create both folders, artist and album, and depending which folder is opened it will show the mp3 sorted one way or the other, and all the information will be contained at one place: the original folder.

The user can access the original folder and browse all the mp3 unsorted.
VFS is being used as a bridge between file systems and open source operating systems providing access to these file systems independently over which file system are writing on.

Virtual file system creates a contract between the kernel and the real file system. This contract simplifies the support for new file systems to the kernel. Creating and filling the contract will be enough to make the new file system compatible. The contract is a list of rules to make the kernel understand the relation between it, the real file system and how to represent it to the end user. Once what a virtual file system is understood it is much easier approach to the file system in user space concept, also know as FUSE.

### 4.4.4 FUSE

FUSE is a UNIX\Linux kernel module that allows any user to create their own file system drivers in user space; in this case the FUSE kernel module gives the contract. [Ref 21]
As said above virtual file systems in user space are very useful because they act as bridge between kernel and file systems.

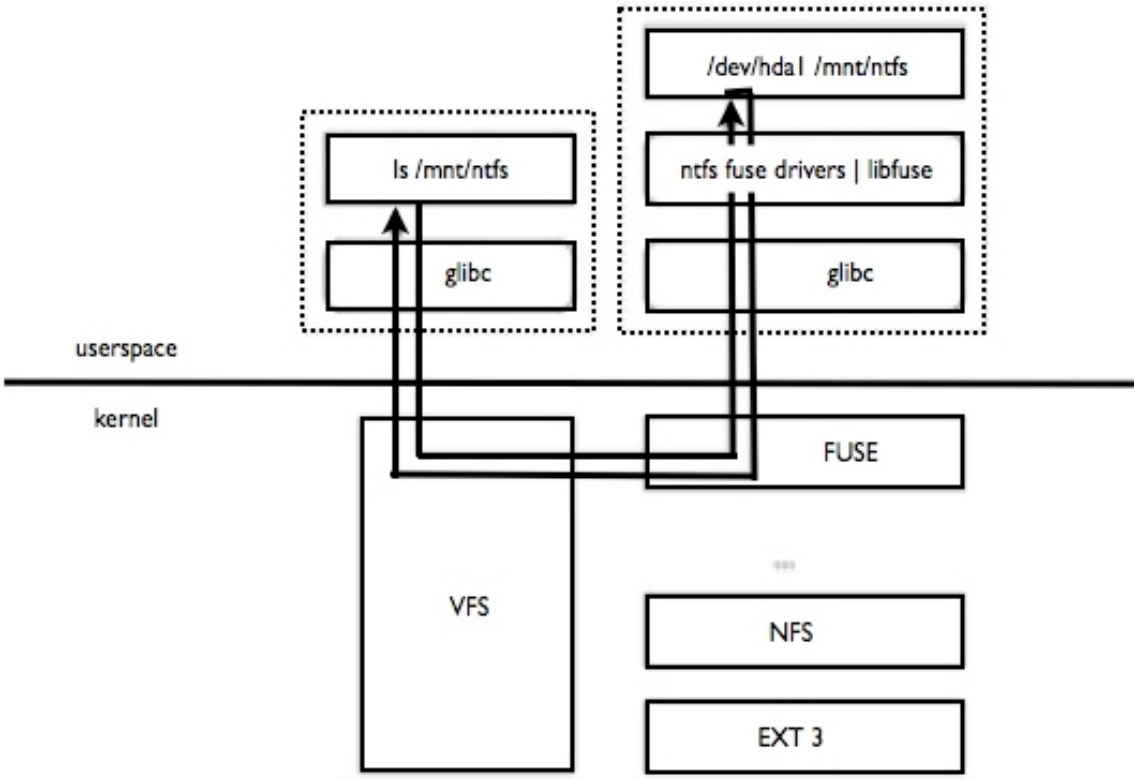


**Fig. 4.6** FUSE internals

At the image we can see how does FUSE works with the NTFS driver. The disk device /dev/hda1, is an NTFS disk, it is mounted on /mnt/ntfs. When a user wants to list the folder the NTFS driver contains all the NTFS logic, how to handle the MFT, and how to write on the disk, how to read a file... It uses the support library libfuse to implement basic calls as open, read or write. These orders pass through the kernel fuse module which bridges with the VFS driver that allows the disk reading at this example.

User space drivers are very useful for NTFS drivers developers who using FUSE do not need to deal with kernel internals and can code directly the NTFS in any programming language they want implementing libfuse.

FUSE improves the development velocity of the driver; in the other hand using FUSE increases the CPU consumption, because of the addition of new layers on the file system structure.

FUSE driver module must be compiled in the kernel in order to use it.

This driver has two parts the one that reside in the kernel, FUSE module and the FUSE file system driver, which is stored at the user space. FUSE driver module must be compiled in the kernel in order to use it.

### 4.4.5 NTFS-3G & NTFS utilities

NTFS-3G is the FUSE file system driver, which makes possible to read and write over an NTFS partition or disk [Ref 22]. It still has some limitations, the driver cannot copy access control list not permissions. Windows desktops are not access control list aware so this issue will only be a problem on servers. Our project is designed to work on normal PC desktops so the access control list flaw would not be affected for it.

The conjunction of these three technologies, VFS, FUSE & NTFS-3G is used in the OS deployment system to synchronise NTFS partitions at the file system layer.

Once the NTFS-3G driver is installed, some utilities are required in order to administrate NTFS partitions. These are found at ntfs-utils package, which contains the following utilities:

- Ntfsclone: application which clones the selected partition
- Ntfsfix: software to fix a NTFS partition in case of cluster error.
- Ntfsresize: utility that allows to change the size of an NTFS partition
- Ntfsundelete: application, which may recover deleted files

As we will see later in this chapter these utilities are later used by PartImage to clone NTFS partitions and to resize NTFS partitions, so it is very important to install this package.

## 4.5 RSYNC

Time is a precious value; therefore saving time for the users on the OS deployment system is a key feature. In order reduce the time spent in a data transaction, one may increase the bandwidth or may reduce the amount of sent data.

An OS deployment system only handles the data transaction, and then just the second method could be implemented. A personal computer usually brakes down because of a change on its operating system. In the previous version of the OS deployment system a change at the OS meant to download the whole image taking about 20 minutes of the user‘s time.

A more efficient approach to the problem is to only send the differences between the original OS at the server and the corrupted one at the client. There is software, which implements this solution: Rsync.

Rsync is an open source software, which synchronises files and directories from one computer to another while minimising data transfer using encoding (delta encoding) when appropriate.

Rsync uses an algorithm to transfer efficiently data across a network o locally when different versions of the same data exist on the receiver.

Rsync uses client-server architecture. There is an rsync server who receives the information to compare from the client.

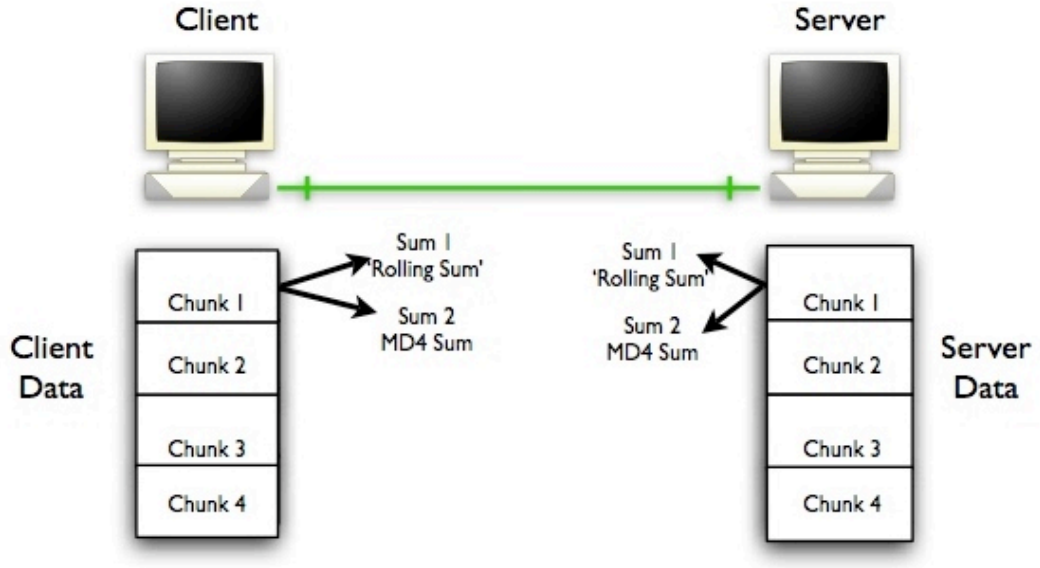


**Fig. 4.7** Rsync architecture

The client splits its copy of the data into fixed size, it create chunks of the data, it computes and MD4 hash and a weaker 'rolling checksum'. If the checksums are different only the different chunks are send. The rolling checksum is sent to the server.

The server calculate the rolling checksum for every chunk of the files to be synchronised (this checksum is based on zlib), then it compares it with the one received to check if it match. If the checksum match then the server calculates the chunk MD4 checksum and verifies if this new checksum match by comparing it with the MD4 sent by the client.

Once the two checksums are checked the server, sends only the chunks that have failed one of the two comparisons, along with the instructions on how to unify the new blocks into the wrong version of the compared file, creating an identical file in both sides.

In most cases only small image differences are found and if there are little differences the OS deployment software does not need to send a lot of data to restore the initial image, saving much time to the end user.

Rsync must be executed on the client and on the server as a daemon. Big files take more time to synchronize due to more checksum and comparison are

needed. Big changes increment the amount of time because more chunks have to be sent.

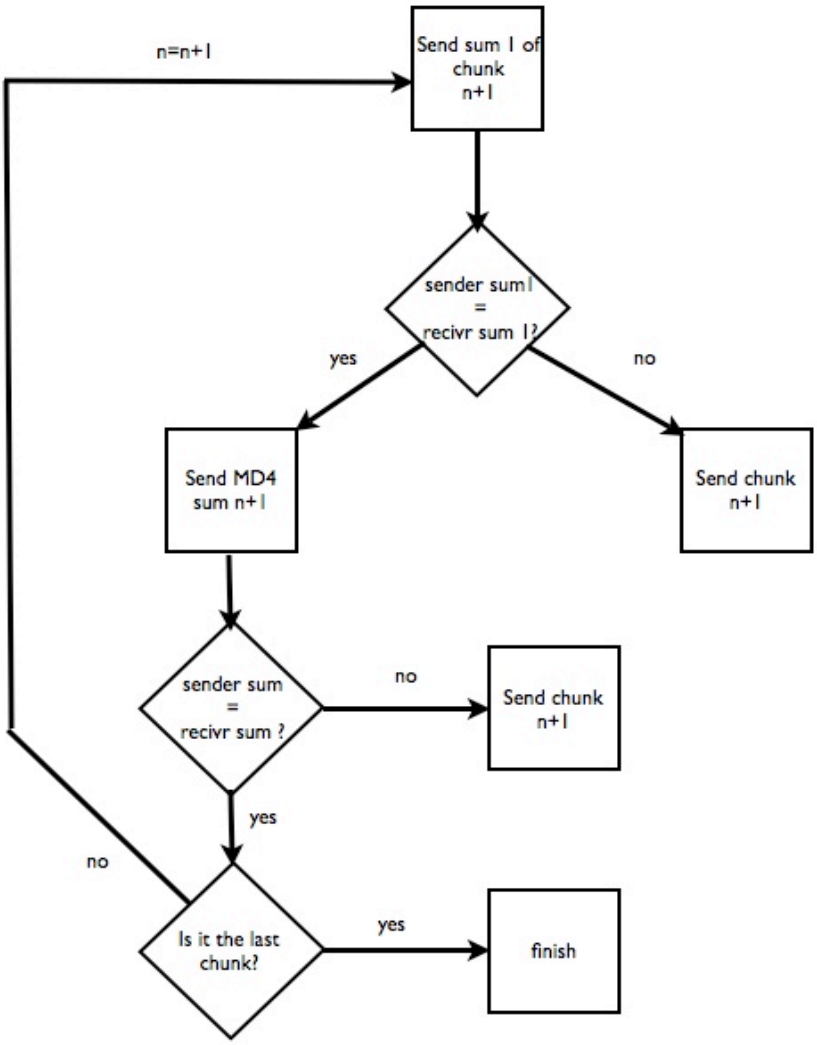


**Fig. 4.7** Rsync algorithm

This function let the OS Deployment system to have the capability to restore only changes: a fast restoration. Rsync daemon is running on the server, while at the client the next parameters have been used at the fast restoration script:

$rsync -avr –delete <server ip>::<synchronising name> <destination folder>

The parameters function is:

- **a:** conserve permissions, groups, owners and timestamps.
- **v:** show a verbose.

- **r:** recursive synchronising.
- **--delete:** deletes the difference with the original.
- **server ip**: indicates the rsync daemon IP address
- **synchronising name**: is an alias of which folder of the server disk wants the administrator to restore on the client.
- **Destination folder**: is the client's folder where data is going to be restored.

## 4.6 Improvement of full Restoring

Although full image creation and restoration was working fine in the previous version of the OS deployment system we found a tool that have many interesting features to improve them.
Even this was not an initial objective of the project we have included it here.
The new software to provide complete image restoring is called Partimage.

Searching for a better solution to handle the image restoring and the image server, PartImage was found [Ref 15]. PartImage is an utility, which gives:

- File system layer image creation
- File system layer image restoration
- Image compression
- Partition images
- Graphic user interface to manage the server & the client
- Command line interface
- Secured connection to transfer the images via SSL
- User authentication
- NTFS support

The main difference of PartImage in respect with Device Imager is that PartImage works at the file system level. Knowing its data structures. This property allows limiting data copy only to restore useful bytes.
Among other features, the NTFS support and its kindness with the disk makes this software the most suitable software for this project. Also it benefits are for the system administrator, who can manage the image of the clients using a graphic user interface.

### 4.6.1 Creating and image from a partition

Create a partition with Partimage is quite easy thanks to it's easy user interface. Two things must be specified.

- The partition to save: giving the appropriate file of the /dev directory.
- Filename where to save the selected partition.

Once a partition is selected, the application lets the user to select the compression level. Gzip compression will make a smaller imager and bzip2 will make even a smaller image.

The compression level affect directly to the time of creation of the image and the time of restoration. More compression reduces the image file size, thus it reduce the time of sending the image but increments the time to copy it to the disk because there is a time to decompress it.
With no compression the image file will be as big as the selected partition, the creation and the restoration of the image will be very fast, but the transfer time will be slow.
The tradeoffs between space and velocity affects to the efficiency of the application. During the test phase of the application with a direct-wired connection, no compression was the fastest method, and the one, that better can competes with the commercial applications.

The next image shows the graphic user interface provided by PartImage at the client site to ease the user experience. One may choose between the partitions created on the disk, the file name and the action the user wants to realise. Notice that the images can be created and stored directly in the server.

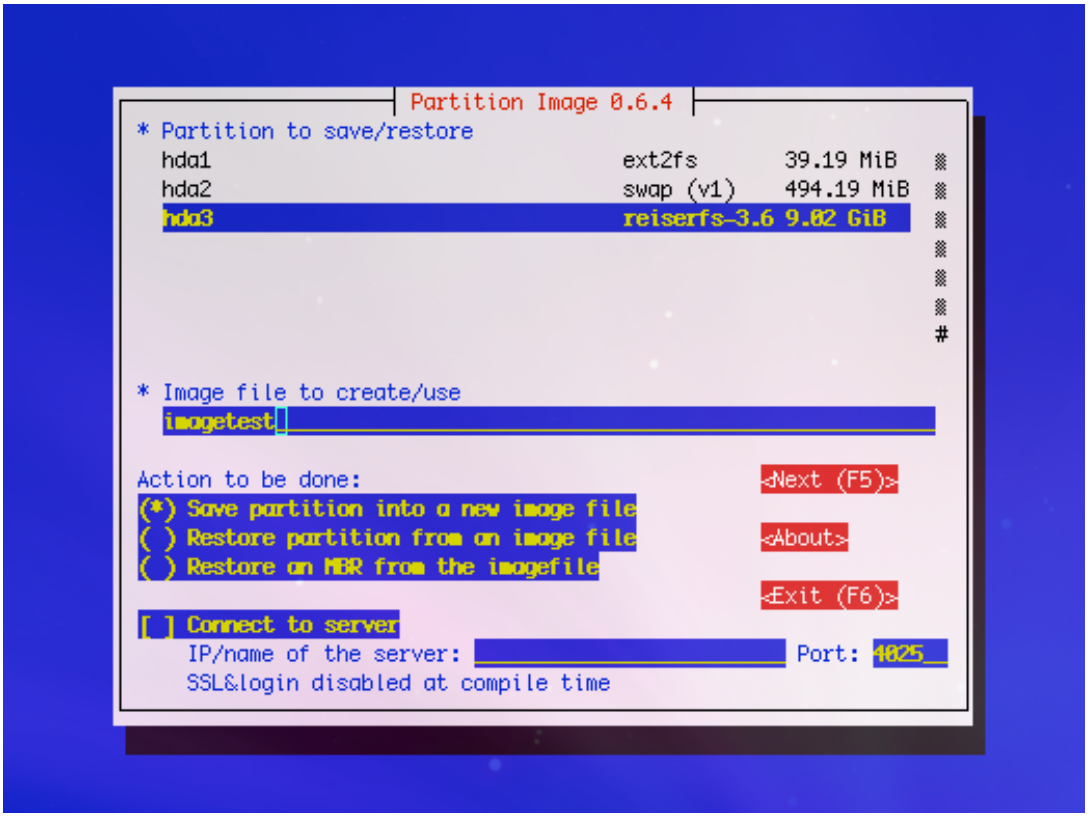


**Fig 4.8** Partimage client graphic user interface

Besides this PartImage allows to:

- Check if the partition is corrupted
- Enter a description
- Split the image files on several parts of the desired size.

### 4.6.2 Full Image restoring

PartImage gives two options in order to restore an image, using the graphic user interface, or via command line.

The graphic user interface is the same as the shown above for creating, the user just needs to check on the restore partition box, fill the image name and the server IP. In the OS deployment system the end user does not need to know neither the name of the image no the server IP. So the user interface is not necessary for the project purpose, instead the command line lets to autotimatize the connection using a script.

Using these parameters:

$partimage -f3 -s 192.168.1.1 -b restore <destination device> <source image>

The parameters funciton is:

- **-f3:** indicates PartImage to exit from the application once it has finish.
- **-s:** is a parameter used to introduce the next parameter the PartImage server IP address.
- **-b:** means batch mode, this means no interaction will be needed by the user in any moment.
- restore express the action to release by PartImage, restore.
- **<destination device>:** is a field to fill with the name of the device where the image will be restored.
- **<source image> :** This parameter indicates the full path and the image name of the desired image located at the server.
-

PartImage will connect to the server restore the desired image without needing any user action.

### 4.6.3 Server PartImage software

The PartImage server contains the image files. In order to run the PartImage server software the administrator must create a ‘partimag’ user; the server will run under this uid.

The server gives the possibility to use a username and a password; this option is compiled by default. As said before user does not have to interact directly with PartImage, so in our case this software must be compiled without the authentication option.

As you can see in the next image a graphic user interface can used to control connections to the server. This user interface shows:

- The number of clients connected to the server
- The state of each client: saving, restoring or waiting
- The IP address of each client
- The location of the operating system image

- The size of the OS image

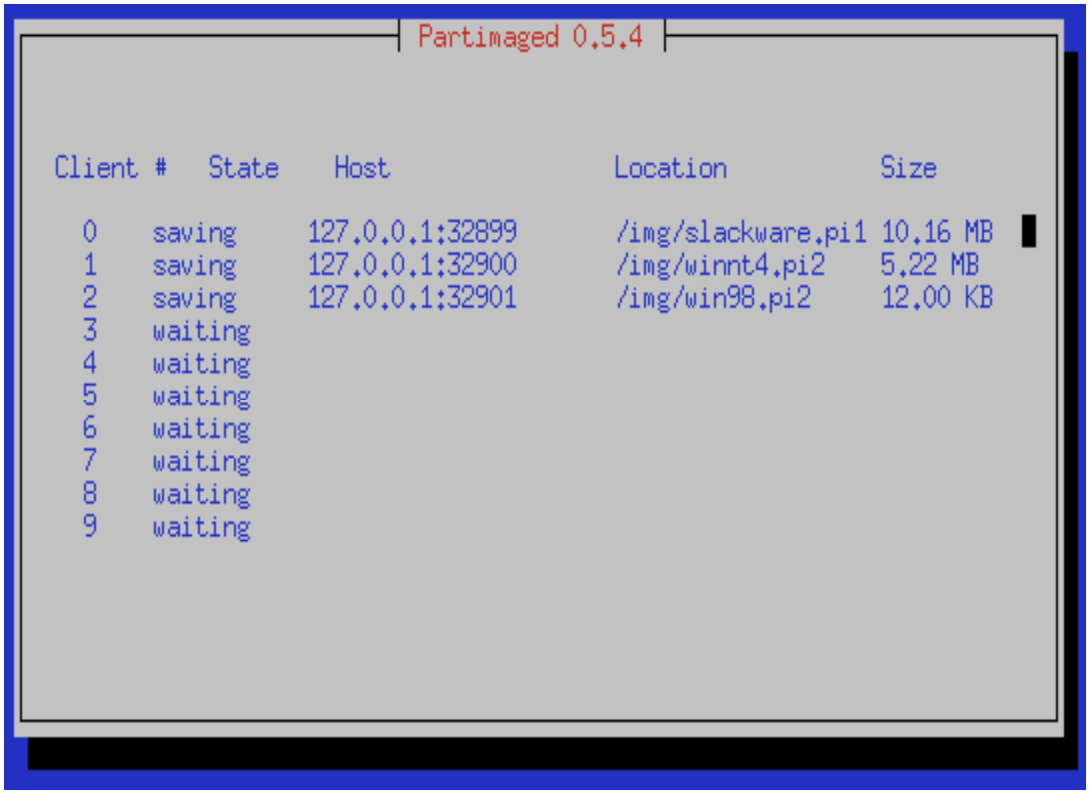


**Fig. 4.9** PartImage server graphic user interface

# Chapter 5 Conclusions

The project initial idea was to create a system to compete against REMBO using open source technologies. Part of this objective was accomplished at a previous version of the project. In order to continue improving the initial goals new objectives were added and have been solved within this degree thesis.

## 5.1 OS deployment system versus REMBO

Rembo is a commercial tool used extensively in large and shared computers labs with many interesting features, A list of the most interesting features is given below.

Rembo permits from total restoration to fast restoration using differential images. It has an easy to use graphic interface and supports multicast data transferring. It can boot an OS without rebooting. The user does not need to reboot the machine; to use his restored operating system. REMBO delivers a centralized admin management tool for the administrator.

In this project we present a step forward to achieve new features on an opern source product. The new features of the OS deployment system increase its force against REMBO. Now OS deployment system has an easy user interface which easies the use of the system providing a transparent OS restoration. This project can boot without reboot thanks to the kexec patch. OS deployment still does not support multicast and do not provide any administrative centralized tool, other than PartImage functionalities.

## 5.2 Achievements of the project

Next sections list the three objectives that the project has solved and some review of the technologies used:

I. The “Microsoft Windows” performance

Thanks to the BIOS emulation layer support addition on GRUB.EXE Microsoft Windows works flawless and without noticeable performance problems. This way the project has solved one of the most important drawbacks of the previous version.

II. User Friendly Interface

Now using the web interface, it is easy for the user to choose the desired option. Beside this, it provides an easy development platform in order to create new web designs.

III. Fast restoring or Image syncing

The inclusion of the NTFS drivers has provided new ways for restoring improving greatly the restoration process. Now rsync can be used between all the disposable operating system.

More over, the project has achieved an important objective that was not previously planned:

IV. Non-aggressive complete restoration

Beside the three initial objectives, this new feature has been added, too. No more disks have broken down during the tests, thanks to PartImage and its smooth file system complete restoration.

In summary the project has achieved all its objectives, and has add new improvement to the initial design.

## 5.3 Future Work

Three improvements are needed in order to continue competing with REMBO.

- Centralized management tool: Due to the amount of network services used by the OS deployment system (TFTP, DHCP, NFS, PartImage, Rsync,) a centralized tool will ease the admin side.
- Image Database: A database to relate the PC clients to their images.
- Security updates: The OS deployment clients can initiate any execution only sending a HTML petition, which contains the path and the name of the executable. This can end on malicious users making local executions.

## 5.4 Environmental Impact

The restoration of an OS has been always a work which makes an extensive use of the CD and DVD to burn the OS images on them. Once the image is burned the technician staff use this image to restore it on the end user computer.

Using the OS deployment system the number of CDs and DVDs wasted on this operation will be significantly reduced. Less CDs and DVD means less environmental impact.

# References

[Ref 1] Rembo, Tivoli provisioning manager :
*http://www-306.ibm.com/software/tivoli/welcome/rembo/index.html*

[Ref 2] Grub:
*http://www.gnu.org/software/grub/*

[Ref 3] PXE:
*http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf*

[Ref 4] Bbpatch, Marc Vuilleumier Stückelberg and David Clerc:
*http://tldp.org/HOWTO/Remote-Boot.html - toc3*

[Ref 5] Ncurses:
*http://en.wikipedia.org/wiki/Ncurses*

[Ref 6] Bios:
*http://en.wikipedia.org/wiki/BIOS*

[Ref 7] Microsoft Windows Bios services:
*http://www.missl.cs.umd.edu/winint/index2.html*

[Ref 8] Lilo:
*http://www.acm.uiuc.edu/workshops/linux_install/lilo.html*

[Ref 9] Kexec:
*http://www.xmission.com/~ebiederm/files/kexec/README*

[Ref 10] Framebuffer:
*http://en.wikipedia.org/wiki/Framebuffer*

[Ref 11] Apache2:
*http://www.apache.org/*

[Ref 13] Gensplash:
*http://dev.gentoo.org/~spock/projects/gensplash/*

[Ref 14] File systems:
*http://en.wikipedia.org/wiki/File_system*

[Ref 15] Ext:
*http://en.wikipedia.org/wiki/Extended_file_system*

[Ref 16] Ext2:
*http://web.mit.edu/tytso/www/linux/ext2intro.html*

[Ref 17] Ext3:
*http://en.wikipedia.org/wiki/Ext3*

[Ref 18] FAT:
*http://en.wikipedia.org/wiki/File_Allocation_Table*

[Ref 19] NTFS:
*http://technet2.microsoft.com/windowsserver/en/library/8cc5891d-bf8e-4164-862d-dac5418c59481033.mspx*

[Ref 20] Kernel & userspace:
*http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/*

[Ref 21] FUSE:
*http://fuse.sourceforge.net/*

[Ref 22] NTFS-3G:
*http://www.ntfs-3g.org/*

# Annex

## Script SyncW

```
ntfs-3g /dev/hda1 /mnt/win -o force
rsync -avr --delete --exclude="pagefile.sys hiberfil.sys" 192.168.1.1::windows /mnt/win
kexec -l /grub.exe
kexec –e
```

## Script SyncL

```
mount /dev/hda1 /mnt/lin -o
rsync -avr --delete --exclude="pagefile.sys hiberfil.sys" 192.168.1.1::Linux /mnt/lin
kexec -l  bzImage
kexec –e
```

## Script winrestore

```
partimage -f3 -s 192.168.1.1 -b restore /dev/hda1 /root/winimage
sleep 5
mount /dev/hda1 /mnt/win
sleep 5
kexec -l /grub.exe
kexec –e
```

## Script linrestore

```
partimage -f3 -s 192.168.1.1 -b restore /dev/hda1 /root/linuximage
sleep 5
mount /dev/hda1 /mnt/lin
sleep 5
kexec -l bzImage
kexec –e
```

## Script windowsboot

```
ntfs-3g  /dev/hda1 /mnt/win –o force
kexec –l /grub.exe
kexec –e
```

## Script linuxboot

```
Mount /dev/hda1 /mnt/lin
kexec –l bzImage
kexec -e
```

## WWWEXEC.SH

```
doc="`awk '/HTTP/{print $2; exit}'`"
torun="`echo "$doc" | awk '
   #Title:  urldecode - decode URL data
   #Author: Heiner Steven (heiner.steven@odn.de)
   BEGIN {
         hextab ["0"] = 0;hextab ["8"] = 8;
         hextab ["1"] = 1;hextab ["9"] = 9;
         hextab ["2"] = 2;hextab ["A"] = hextab ["a"] = 10
         hextab ["3"] = 3;hextab ["B"] = hextab ["b"] = 11;
         hextab ["4"] = 4;hextab ["C"] = hextab ["c"] = 12;
         hextab ["5"] = 5;hextab ["D"] = hextab ["d"] = 13;
         hextab ["6"] = 6;hextab ["E"] = hextab ["e"] = 14;
         hextab ["7"] = 7;hextab ["F"] = hextab ["f"] = 15;
   }
   {
         decoded = ""
         i   = 1
         len = length ($0)
         while ( i <= len ) {
            c = substr ($0, i, 1)
            if ( c == "%" ) {
                  if ( i+2 <= len ) {
                     c1 = substr ($0, i+1, 1)
                     c2 = substr ($0, i+2, 1)
                     if (!( hextab [c1] == "" || hextab [c2] == "" )) {
                           code = 0 + hextab [c1] * 16 + hextab [c2] + 0
                           c = sprintf ("%c", code)
                           i = i + 2
                     }
                  }
            } else if ( c == "+" ) {  # special handling: "+" means " "
                  c = " "
            }
            decoded = decoded c
            ++i
         }
         print decoded
   }
'`"
cat << EOF
HTTP/1.0 200 OK
Content-Type: text/plain
Connection: Close

EOF
( $torun )
exit
```