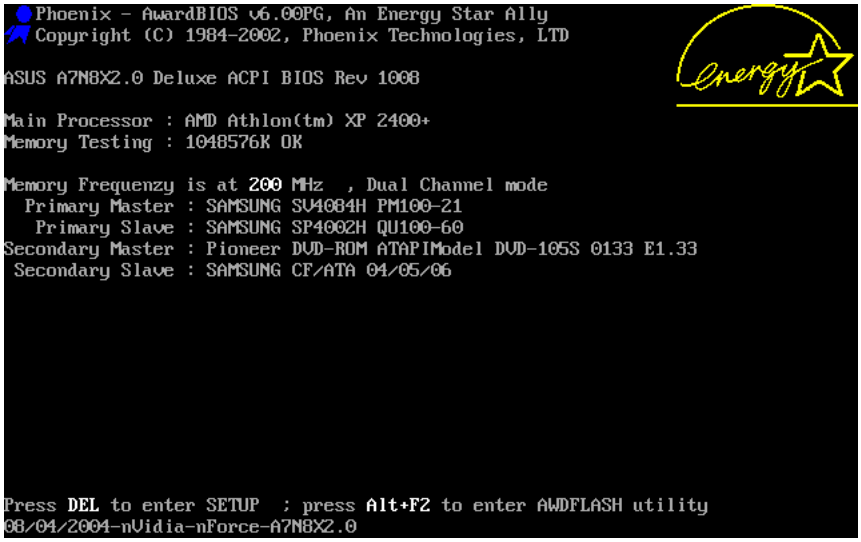
Arranque

Última modificación 2016/01



Elaboración propia utilizando principalmente artículos de la wikipedia (http://www.wikipedia.org) e información del foro UEFI (http://www.uefi.org), IBM (http://www-128.ibm.com/developerworks/library/l-linuxboot/index.html), multibooters (http://www.multibooters.co.uk) y Debian (http://www.debian.org).
Algunas partes son directamente copia o traducción de las fuentes.

Arranque

NOTA: Todos los comandos y ficheros de configuración mostrados en este documento se presentan únicamente a modo de ejemplo. Antes de utilizarlos en un entorno real lea sus respectivos manuales y/o ayudas y asegúrese que sabe lo que está haciendo.

Contenido

1. INTRODUCCIÓN

1.1. DIRECCIONAMIENTO FÍSICO DE MEMORIA SECUNDARIA

a) CHS y ECHS

El primer sistema ampliamente difundido para direccionar bloques de información[1] en dispositivos de almacenamiento fue CHS (*Cylinder-Head-Sector*), usado en las primeras unidades ATA, donde se asignaba una dirección a cada bloque mediante una tupla que definía el cilindro, el cabezal y el sector en que se encontraba.
Al unir las limitaciones que imponía IDE con las limitaciones que imponía BIOS, se limitaba la capacidad de los discos a 504 MiB.

	Cilindros (Máx.)	Cabezales (Máx.)	Sectores (Máx.)	Capacidad (Máx.)
IDE / ATA -direcc. físico-	65.536	16	256	128 GiB
BIOS (Int 13)	1.024	256	63	7,88 GiB
Combinación -direcc. lógico-	**1.024**	**16**	**63**	**504 MiB**

Tanto el estándar BIOS como el IDE permiten mayor capacidad que la combinación de ambos, ya que el uno permite más cilindros y sectores y el otro permite más cabezales. Para superar la barrera de 504 MiB apareció ECHS (*Extended CHS*) también llamado "*CHS Large*". Este sistema realiza una traducción de direcciones lógicas a físicas mediante un simple truco: divide el número real de cilindros por 2, 4, 8 o 16 -dependiendo del dispositivo- y multiplica el número de cabezales por la misma cantidad. Así pueden direccionarse discos de hasta 16.384 cilindros (1.024 * 16) con 16 cabezales (256 / 16) y 63 sectores, que es el límite de direccionamiento BIOS (7'88 GiB).

Otro inconveniente de este sistema de direccionamiento es que no funciona bien en dispositivos que físicamente no se compongan de cilindros y cabezales, como cintas de datos o memorias electrónicas.

b) LBA

Para superar las limitaciones del sistema ECHS apareció el sistema LBA (*Logical Block Addressing*)[2]. LBA es un método de direccionamiento particularmente simple. Los bloques son numerados según un índice, siendo el primer bloque LBA=0, el segundo LBA=1, y así sucesivamente. El direccionamiento LBA en las unidades ATA puede ser de 28 bits o de 48 bits (introducido en ATA-6), lo que resulta en límites de 128 GiB y 128 PiB respectivamente.

Este sistema debe estar implementado en el propio dispositivo para que el sistema pueda utilizarlo. Muchos dispositivos de gran capacidad que utilizan LBA pueden utilizarse mediante ECHS si el *firmware* de la máquina no está preparada para utilizar LBA.

c) Limitaciones de particiones

Algunos sistemas operativos, como todos los Windows hasta la versión 2003, requieren que las particiones empiecen y acaben ocupando cilindros completos (incluso utilizando LBA). Dado que es muy raro que el número de sectores acabe en el límite de un cilindro, muchos dispositivos tienen un exceso de sectores (menor que un cilindro) que no puede utilizarse. Algunos sistemas, como GNU/Linux, permiten evitar los límites de cilindros pero hacerlo puede acarrear problemas de incompatibilidad con otros sistemas.
El sistema de particionado de Windows Vista evita los límites de cilindros pero limita el posicionamiento (inicio y final) a múltiplos de 2048[3] en previsión de que el tamaño de bloque estándar (actualmente 512 B) se incremente a dicha cantidad. Esto causa que si se mezcla el uso del programa de particionado de Vista con otros (como programas de clonado o el gestor de discos de Windows XP) algunas particiones puedan moverse o incluso desaparecer[4](!) y causar que Vista deje de funcionar.
Si se desea usar varios sistemas operativos o programas de clonado la recomendación es no crear las particiones con el programa de Vista.

1 Los bloques lógicos de información son normalmente de 512 o 1024 B cada uno. ISO-9660 (CDs) utiliza un tamaño de bloque de 2048 B.
2 El término LBA puede referirse, además de al sistema de direccionamiento, a la dirección concreta de un bloque.
3 Cada partición dejará además un espacio inicial libre de 2048 B, de manera similar a los 64 B que deja el particionado "tradicional".
4 http://www.dcr.net/~w-clayton/Vista/DisappearingPartitions/DisappearingPartitions.htm

1.2. REGISTRO DE ARRANQUE MAESTRO (MBR)

Se llama MBR a un esquema de particionado de dispositivos de almacenamiento (memoria secundaria) diseñado para permitir arrancar un sistema residente en el volumen. Este esquema define un registro de 512 Bytes llamado "Registro de Arranque Maestro" o MBR (*Master Boot Record*) que debe residir en el bloque "LBA Sector 0" del volumen[5].

En los primeros 444 bytes del MBR encontramos el código de arranque (*bootloader*). Los dos bytes siguientes quedan en blanco. Los siguientes 64 bytes contienen la tabla de particiones. Los últimos dos bytes del MBR contienen una firma[6] que indican que el registro contiene efectivamente código de arranque. Esta misma firma está presente en los códigos de arranque de los VBR.

De los 444 bytes del código de arranque, GRUB deja los primeros 64 bytes libres[7], por cuestiones de compatibilidad con versiones antiguas. Microsoft carga su código desde el primer byte (byte 0).
La parte final del código de arranque contiene en ambos casos códigos de error y en los últimos 4 bytes (de forma opcional[8]) la firma o identificador del disco.

0 63	439	443	445	509	511
[Libre (GRUB)] + [Código de Arranque + Errores] + [Firma de disco]			Null	T. Particiones	Firma
446 B (64 + 376 + 4 + 2) B				64 B	2 B

La tabla de particiones ocupa 64 bytes, conteniendo 4 registros de 16 bytes que definen la particiones primarias. En ellos se almacena toda la información básica sobre la partición: marca de arranque, tipo, tamaño y sector de inicio:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Boot[9]	CHS inicial			Tipo	CHS final			LBA inicial				Sectores			

Tanto el inicio de la partición (LBA inicial) como su longitud (sectores) se indican en 32 bits (4 B) lo que limita ambos a 2 TiB -para sectores de 512 B-. Para eliminar esta limitación -entre otras- se desarrolló el esquema de particionado GPT (*GUID Partition Table*) como parte del estándar UEFI (*Unified Extensible Firmware Interface*).

a) Registro VBR

En volúmenes no particionados, en el sector 0 reside un registro similar al MBR conocido como "Registro de Arranque de Volumen" o VBR (*Volume Boot Record*). Aunque su estructura es en principio diferente, es posible crear un registro que funcione como ambos, lo que se conoce como "Registro de Multi-Arranque" (*Multi-Boot Record*).
Este tipo de registro se utiliza también en el primer sector de una partición arrancable, llamándose a veces en ese caso PBR (*Partition Boot Record*).
Un VBR contiene un cargador de sistema o un gestor de arranque (o parte del mismo). El código del VBR puede ser invocado directamente por la BIOS en dispositivos no particionados o indirectamente por el código del MBR en dispositivos particionados.

b) Respaldo del MBR

Desde sistemas GNU/Linux podemos hacer copias de respaldo del MBR, por ejemplo:

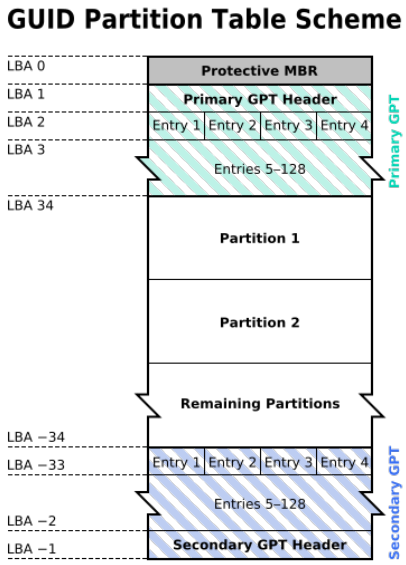
```
dd if=/dev/sda of=sda.mbr.bak bs=512 count=1          # Copia todo el MBR
dd if=sda.mbr.bak of=/dev/sda bs=446 count=1          # Restaura el código de arranque
dd if=sda.mbr.bak of=/dev/sda bs=1 count=64 skip=446 seek=446 # Restaura la tabla de particiones
```

Para reescribir el "Código de Arranque Maestro" en el MBR y solventar problemas de inicio se puede utilizar desde Ms-DOS "`fdisk /mbr`", desde la consola de recuperación de Windows "`fixmbr`" y "`fixboot`" y desde el disco de inicio de Vista "`bootrec.exe /fixboot`" y "`bootrec.exe /fixmbr`".
Desde sistemas GNU/Linux se puede reescribir con "`grub-install /dev/hda`".

5 Un estudio detallado del mismo se puede consultar en http://thestarman.pcministry.com/asm/mbr/STDMBR.htm
6 0xAA55 escrito en *little-endian*, esto es Byte 510 (0x55), Byte 511 (0xAA).
7 Los bytes quedan libres, sin reiniciarse a cero, manteniendo la información previa que hubiese en ellos (pero que no se usa).
8 Este identificador es utilizado por los sistemas Windows -a partir de NT4- para mantener la relación de unidades asignadas. Una modificación de esta firma hace que Windows Vista no arranque indicando un error de "cambio de *hardware*".
9 (0x80 = arrancable, 0x00 = no-arrancable, otro = inválido)

1.3. TABLA DE PARTICIONES GUID (GPT)

La tabla de particiones GUID (GPT: *GUID Partition Table*) es un sistema de particionado definido en el estándar UEFI. Utiliza números pseudoaleatorios llamados GUID (*Globally Unique IDentifier*)[10] para identificar las particiones.



Cuando un dispositivo se particiona con el esquema GPT, se escribe en el sector 0 un registro "MBR protector" ("*Protective MBR*") cuyo propósito es mantener la compatibilidad con sistemas PC BIOS. Este MBR protector especifica en su tabla de particiones una única partición GPT que abarca toda la unidad. Esto permite que programas o sistemas que no entienden GPT vean el disco como una sola partición de tipo desconocido, evitando la confusión con un disco sin particionar.

La GPT propiamente dicha comienza en el sector 1 con la cabecera primaria de la tabla de particiones y la tabla de particiones primaria en sí continúa en los bloques sucesivos.
GPT proporciona redundancia, manteniendo una GPT secundaria al final del disco.

La cabecera de la tabla de particiones contiene el GUID del disco (32 B) y define los bloques del mismo que pueden ser utilizados por el usuario (bloques utilizables). También define el número y tamaño de las entradas de partición que conforman la tabla de particiones y el tamaño y la localización de la cabecera y tabla secundaria GTP (siempre en el último sector del disco). Por último contiene una suma de comprobación CRC32 de toda la tabla de particiones (incluyendo la cabecera) que los procesos UEFI verifican durante el arranque.

Los primeros 32 B de cada entrada de partición sirven para identificar la partición, siendo los primeros 16 Bytes el tipo de partición[11] y los siguientes 16 bytes un GUID. Las entradas de partición utilizan 8 B para indicar el comienzo de la misma y otros 8 B para indicar el final, usando LBA. También se reserva un espacio para los nombres de las particiones y otros atributos.

Windows 2003 establece 128 entradas de partición reservadas, cada una de 128 bytes de longitud (32 sectores).

1.4. GESTORES DE ARRANQUE Y CARGADORES DE SISTEMA

Habitualmente se mezclan los términos "gestor de arranque" (*boot manager*) y "cargador de sistema" (*boot loader*), entre otras cosas porque programas como GRUB proporcionan ambas funciones. Sin embargo, estrictamente hablando un cargador de sistema es un programa sencillo diseñado exclusivamente para cargar en memoria un sistema operativo.
Un gestor de arranque es un programa que permite opciones previas a la carga del sistema. Así puede por ejemplo permitir el arranque desde dispositivos no detectados en el inicio o desde particiones sin la marca de arranque. También puede ofrecer al usuario opciones de arranque o incluso cargar otros gestores de arranque. Además generalmente incluye las funciones de cargador de sistema.
Así por ejemplo GRUB puede ofrecer la opción de lanzar NTLDR y éste a su vez puede ofrecer al usuario -como gestor de arranque- varias opciones de arranque.

Un gestor de arranque puede sustituir el "Código de Arranque Maestro" del MBR, aunque dada la limitación de espacio en el MBR, pueden instalar solo una parte o etapa (arranque multietapas). Esta primera etapa se ocupa de cargar el resto del programa.

Algunos de los gestores de arranque más utilizados son GRUB, LiLo, NTLDR (todos ellos incluyen la función de cargador de sistema) y BootMgr (quien llama al cargador de sistema WinLoad).

10 Aunque no se puede garantizar que cada GUID generado sea único, el número total de claves únicas (2^{128}) es tan grande que la posibilidad de que se genere un mismo número dos veces puede considerarse nula en la práctica.
11 Linux y Windows utilizan el mismo identificador de partición para sus respectivas particiones de datos.

1.5. PC BIOS

PC BIOS (*Basic Input/Output System*) es un *firmware* utilizado en equipos compatibles con IBM PC, que se basa en la arquitectura x86 de 16 bits en modo real. Dispone de un área de datos llamada BDA (*BIOS Data Area*).
BIOS está diseñado para ser el primer código que se cargue en memoria al encender el equipo y sus funciones son establecer una interfaz de acceso a los dispositivos de la máquina, configurar e inicializar los mismos e iniciar la carga de un sistema operativo.
El código de BIOS esta escrito en lenguaje ensamblador y se almacena con la BDA en un chip de tipo EEPROM (*Electrically Erasable Programmable Read-Only Memory*) que trabaja por Bytes[12].

a) Interfaz SO-firmware

BIOS establece para el sistema operativo una interfaz de acceso a los dispositivos de la máquina. Esta interfaz consiste en una serie de rutinas de 16 bits en modo real que acceden a los diferentes *firmware* de los dispositivos como vídeo, teclado o almacenamiento secundario.
Esta interfaz entre sistema operativo y los distintos *firmware* quedó obsoleta. Los sistemas operativos abandonaron el "modo real" una vez en funcionamiento, utilizando sus propias rutinas de acceso. Sin embargo todavía utilizaban (y a día de hoy la mayoría sigue utilizando) la interfaz PC BIOS durante su carga. La interfaz UEFI, cada vez más implantada, está diseñada para sustituir totalmente esta interfaz.

b) Arranque de la máquina y carga del sistema operativo

La primera función de BIOS es identificar, configurar, comprobar e inicializar dispositivos del sistema como tarjetas de vídeo, teclado, memoria, los dispositivos de almacenamiento (memoria secundaria), tarjetas de red... Esta rutina de comprobaciones e inicializaciones se conoce como POST (*Power-On Self-Test*).
La configuración de los dispositivos se realiza mediante un menú de configuración del sistema ("*BIOS Setup*").

Tras inicializar y preparar los dispositivos, la tarea de BIOS es localizar un código de arranque y cederle el control. Este proceso se conoce como secuencia o proceso de arranque, carga del sistema o simplemente inicio.

c) Limitaciones de BIOS

El estándar PC BIOS se ha mantenido vigente con pocos cambios fundamentales desde sus inicios. Aunque se ha añadido nuevas funciones a la BIOS e incrementado su complejidad, como por ejemplo implementando LBA, todavía al arrancar equipos basados en BIOS, incluso los procesadores de 64 bits con varios núcleos deben emular en el arranque al procesador Intel 8086 de 1978 y funcionar en modo real de 16 bits.

Las limitaciones principales que impone BIOS durante el proceso de arranque son:
- diseñado para arquitectura x86 de 16 bits en modo real
- solo puede utilizarse el primer MiB de la memoria principal
- los dispositivos que deben permanecer accesibles durante el arranque (tarjetas de vídeo, de expansión...) tienen que incorporar una memoria de lectura de 128 KiBytes.
- utiliza MBR que a su vez impone otras restricciones:
  - hasta 4 particiones primarias,
  - inicio y longitud de la partición de 2 TiB

12 Las memorias Flash, como los lápices USB, son un tipo de EEPROM que trabaja por bloques (512 B).

1.6. INTERFAZ UEFI

Para superar las limitaciones del sistema PC BIOS, Intel creó la especificación EFI (*Extensible Firmware Interface*) en 1999. Para facilitar su adopción en el mercado en 2005 se crea el foro UEFI (*Unified EFI*) con empresas fabricantes de microchips, tarjetas de vídeo, BIOS...
La última especificación EFI de Intel (1.10) es el punto de partida para las nuevas especificaciones UEFI.

UEFI define un nuevo interfaz entre los sistemas operativos y los *firmware* de los dispositivos de la máquina que sustituye a la interfaz de BIOS. Esta interfaz está programada en C y es independiente de la arquitectura. Se basa por una parte en tablas de datos que contienen información de la máquina, y por otra parte en rutinas de acceso para los gestores de arranque ("servicios de arranque" o "*boot services*") y el sistema operativo ("servicios de funcionamiento" o "*runtime services*"). Estos servicios de arranque proporcionan un panel de control en modo texto o con interfaz gráfica similar a los de los gestores de arranque.
Además UEFI define un tipo de partición (opcional) llamado "partición de sistema EFI" (*EFI System Partition*) basado en FAT que permite almacenar distintos cargadores de arranque y controladores de dispositivos.

UEFI no sustituye las funciones de configuración e inicialización del sistema de PC BIOS (POST y "*BIOS Setup*"), que debe ser implementado por un *firmware* de la plataforma. Así UEFI puede implementarse encima de una BIOS tradicional suplantando su interfaz y rutina de inicio o encima de arquitecturas sin BIOS (pero con otro *firmware*).

Por tanto las funciones de UEFI son dos:
- interfaz SO-*firmware*
- plataforma de inicio o arranque.

Entre las ventajas de la interfaz UEFI destacan que es capaz de gestionar el esquema de particionado GPT y el sistema de ficheros FAT y que es capaz de cargar programas guardados en las particiones EFI.

2. PROCESO DE ARRANQUE

2.1. PROCESO DE ARRANQUE GENERAL

La secuencia o proceso de arranque (*boot*) es el proceso que se realiza desde que se enciende el equipo hasta se carga el sistema operativo. El proceso de arranque se considera completo cuando el equipo está preparado para atender a los requerimientos del usuario.

Al encenderse el equipo la CPU ejecuta el código que se encuentra en la BIOS, ya sea cargándolo primero en la memoria principal o leyéndolo directamente desde la memoria de la BIOS.
Este código se encarga de realizar la rutina POST. Después la rutina de arranque de la BIOS busca un código de arranque en el dispositivo de almacenamiento indicado (en BIOS o por opción del usuario) lo carga en memoria y transfiere el control del equipo a éste.

Para localizar el código de arranque, BIOS lee el primer bloque del dispositivo y si tiene la firma adecuada (0xAA55) lo carga y le cede el control de la CPU.
En dispositivos particionados este primer bloque es un MBR. En dispositivos no particionados es un VBR.
La única función del "Código de Arranque Maestro" (*Master Boot Code*) del MBR es localizar en su tabla de particiones una con la marca (*flag*) de arranque, verificar que la firma del primer registro es correcta, cargarlo en memoria y cederle el control.
Este sector inicial de la partición de arranque es un VBR -también llamado en este caso PBR (*Partition Boot Record*)-. El código del VBR es un "cargador de sistema" (*boot loader*) o un "gestor de arranque" (*boot manager*) y se encarga de iniciar la carga de un sistema operativo.
El código del MBR y el VBR utiliza instrucciones en código máquina.

El "Código de Arranque Maestro" del MBR y el VBR puede ser sustituido por un gestor de arranque, o una parte del mismo.

Una vez el cargador de sistema tiene el control, éste se ocupa de que comience la carga del sistema operativo propiamente dicho cediendo el control al núcleo del sistema operativo.

El núcleo del sistema operativo se encarga ahora de obtener información sobre el equipamiento (*hardware*) del sistema, así como los controladores (*drivers*) asociados a los dispositivos.

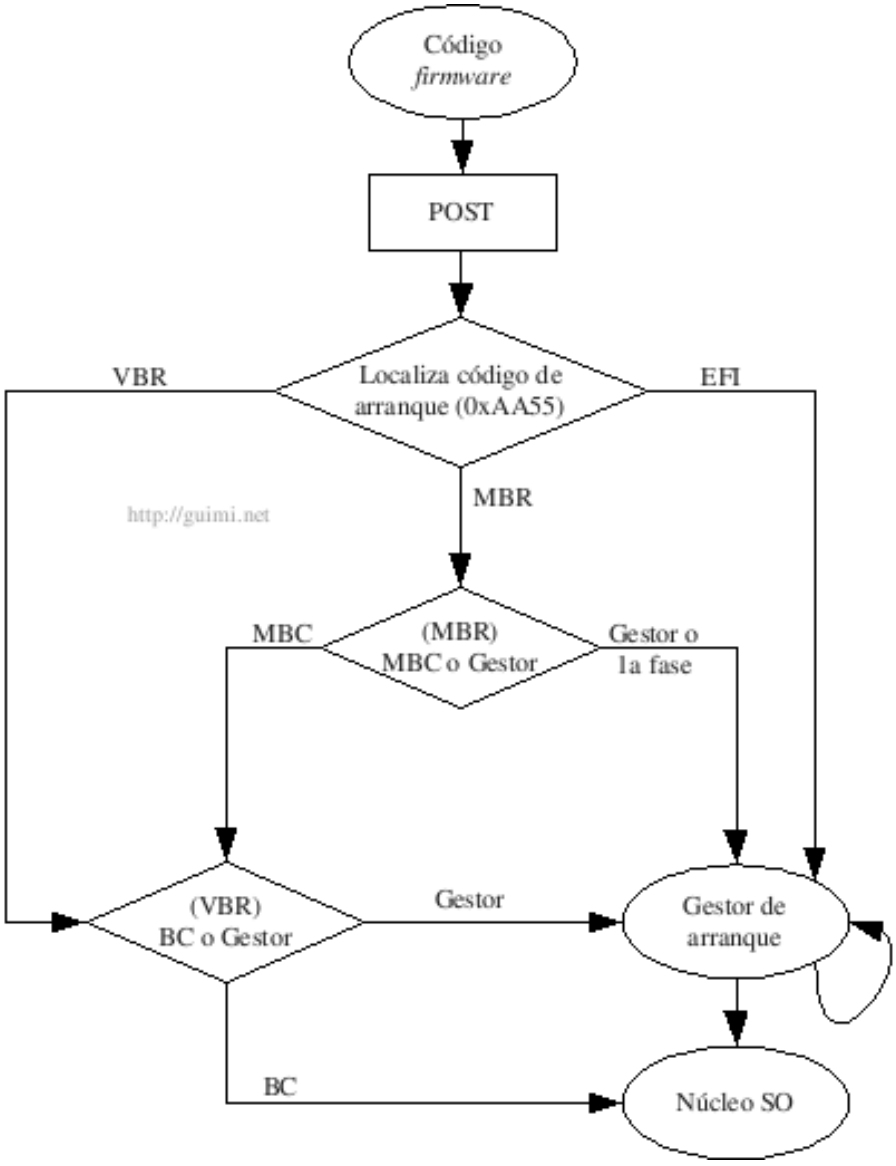
a) Equipos UEFI con particionado GPT

En equipos con UEFI y GPT, el proceso es ligeramente diferente. Al encenderse el equipo la CPU ejecuta el *firmware* inicial de la máquina, PC BIOS u otro, que debe configurar e inicializar los dispositivos y después ceder el control a los "servicios de arranque" de UEFI.
Estos servicios pueden localizar un gestor de arranque o un cargador de sistema siguiendo el esquema MBR o en la partición de sistema EFI.

b) Resumen y Esquema

Las diferentes etapas de la secuencia de arranque se diferencian por el código que tiene control sobre la CPU:

1. Control de la BIOS. Se encarga de cargar el código que encuentra en:
   a) El MBR de un dispositivo particionado (ver paso 2)
   b) El VBR de un dispositivo no particionado (ver paso 3)
   c) La partición de sistema EFI (ver paso 3).
2. Control del código residente en el MBR. Puede ser:
   a) "*Master Boot Code*". Se encarga de cargar el código del VBR.
   b) Primera etapa de un gestor de arranque. Se encarga de cargar la segunda parte del mismo.
   c) Un gestor de arranque.
3. Código de Arranque (gestor de arranque o cargador de sistema). Carga el núcleo del sistema operativo.
   a) Si es un gestor de arranque puede ofrecer opciones al usuario e incluso cargar otro gestor de arranque (volveríamos al paso 3).
4. Control del núcleo del sistema operativo. Se encarga de reconocer el *hardware*, cargar los controladores e inicializar el sistema.
5. Control del sistema operativo. El sistema está arrancado. Una vez arrancado el sistema tendremos en memoria los programas de usuario conviviendo junto con el propio sistema operativo y serán ellos quienes tengan el control del CPU.



2.2. ARRANQUE UTILIZANDO LA RED

a) Arranque por red

Muchas BIOS también son capaces de arrancar usando un sistema operativo almacenado en un servidor, en vez de localmente. Para ello generalmente la rutina de inicio del *firmware* de la máquina activa un servidor tipo TFTP (*Trivial File Transfer Protocol*) y lanza a la red una petición mediante multidifusión (*broadcast*). Si hay un servidor en la red que detecta la petición, éste transfiere al cliente un cargador de sistema operativo que ya se encarga de obtener por red o localmente el resto del sistema. El cargador de sistema trasferido puede depender de la MAC del cliente, por ejemplo.
A partir de aquí el proceso de arranque prosigue igual que si el núcleo se hubiese cargado localmente.

b) Wake-on-LAN (WoL)

La tecnología "*Wake on LAN*" (WoL) es una parte del estándar Ethernet que permite encender un equipo remotamente. Para ello la placa base de la máquina y la tarjeta de red deben ser compatibles con WoL (y tener la función activada). Además se requiere que la tarjeta de red mantenga suministro energético mientras el equipo está "apagado" (en realidad hibernando o en "*standby*" -estados G1 o G2 ACPI-).

La tarjeta de red queda a la escucha por si recibe un paquete Ethernet (nivel 2 OSI) llamado "Magic Packet". Si la tarjeta de red detecta un paquete "mágico" destinado a ella envía una señal eléctrica a la placa para que encienda el equipo. Las tarjetas no integradas en placa en buses anteriores a PCI 2.2[13] necesitaban un cable eléctrico específico para esta tarea.
El paquete "mágico" es un paquete multidifusión (*broadcast*) con 6 B a 1 y 16 repeticiones de la dirección MAC a despertar. Como la tarjeta de red en realidad solo busca la cadena comentada en los paquetes que recibe, puede enviarse dicha cadena dentro de un paquete de red cualquiera, por ejemplo un paquete UDP -típicamente con destino a los puertos 0, 7 o 9- que atraviese distintos segmentos Ethernet. Esto hace muy sencilla la utilización de "paquetes mágicos" lo que proporciona unos niveles de seguridad muy bajos.

Para implementar cierta seguridad en el esquema existen distintos sistemas. Algunos cortafuegos pueden filtrar los paquetes WoL. Algunas NICs tienen una característica llamada "SecureOn" que requiere el uso de una clave en el paquete mágico, pero esa clave viajará en claro por la red. Por último algunos equipos empiezan a utilizar TLS (*Transport Layer Security*) como sistema de encriptación.

2.3. ARRANQUE DEL SISTEMA WINDOWS 2003

En sistemas Windows 2003 el código que reside en el VBR busca en la raíz de la partición de sistema el fichero `NtLdr` que comienza la carga del sistema operativo propiamente dicho. La secuencia de arranque se encarga ahora de obtener información sobre el equipamiento (*hardware*) del sistema, así como los controladores (*drivers*) asociados a los dispositivos.

El programa `NtLdr` (*NT Loader*) cambia el procesador del modo real al modo de 32 bits, ya que `NtLdr` es una aplicación de 32 bits. Una vez en modo 32 bits, la primera tarea que realiza el programa `NtLdr` consiste en cargar el minicontrolador del sistema de archivos. Este paso es necesario para la localización y la carga del sistema Windows.
A continuación lee el fichero "`Boot.ini`", mostrando los diferentes sistemas operativos con los que se puede arrancar. Una de las opciones ofrecidas es utilizar el sector de arranque anterior a la instalación de Windows, en cuyo caso NTLDR carga "`BootSec.dos`", cediéndole el control y finalizando por tanto el proceso de arranque de Windows 2003.
En caso contrario el programa NTLDR ejecuta "`NtDetect.exe`", encargado de buscar el equipamiento del equipo, devolviendo una lista con el equipamiento encontrado a NTLDR para que sea incluido en el registro.
Por último NTLDR carga "`NtOSKrnl.exe`", "`Hal.dll`" y la clave "`System`" del Registro que permite a NTLDR cargar los controladores configurados para ser iniciados en el proceso de arranque. Tras ello, NTLDR cede el control al núcleo del sistema, NTOSKRNL, terminando el proceso de arranque para comenzar la carga del sistema operativo.

```
→ NTLDR ← Boot.ini
  a) BootSec.dos
  b) NtDetect.exe → NtOSKrnl.exe + Hal.dll + HKLM\System →  NtOSKrnl.exe
```

13 PCI 2.2 y posteriores permiten enviar y recibir PME (*Power Management Events*) a través del bus de datos.

`Boot.ini` y `NtDetect.exe` también deben residir en la raíz del sistema (`C:\`).

a) Solución de errores en el arranque de Windows 2003

Windows 2003 incorpora diversos medios para corregir los posibles errores en el proceso de arranque. Entre las soluciones a estos problemas podemos destacar los siguientes:

1. Reparación de una instalación con los discos de instalación de Windows 2003. Estos ofrecen la posibilidad de realizar una instalación o reparar una existente, en cuyo caso nos ofrecerá recuperar los ficheros de sistema o la base de datos del usuario.
2. Creación de un disquete de arranque. Una vez formateado el disco, deben copiarse en el mismo los ficheros necesarios para el arranque como son `NtLdr`, `NtDetect` y `Boot.ini`; resulta necesario además `NtBootd` si tenemos dispositivos SCSI y `BootSec.dos` para arrancar el sistema anterior.
3. El menú de opciones avanzado. Se obtiene pulsando F8 en el arranque.
   - Modo Seguro. Carga solamente los ficheros y controladores estrictamente necesarios para iniciar y ejecutar el sistema operativo.
   - Modo Seguro con funciones de Red. Igual pero con red.
   - Modo Seguro con símbolo de sistema. Sin entorno gráfico (`explorer`).
   - Habilitar el registro de inicio. (Genera el registro de inicio `%SystemRoot%\NtBtLog.txt`)
   - Habilitar modo VGA. Arranca utilizando el driver básico de VGA (el usado en el modo seguro).
   - Ultima Configuración buena Conocida. El registro almacena bajo `HKLM\SYSTEM` conjuntos de configuraciones denominados `ControlSetxxx`.
   - Modo de Restauración de SD. (Solo para DCs).
   - Modo de Depuración
   - Iniciar Windows Normalmente.
   - Reiniciar.
   - Regresar al menú de opciones del SO. Esta opción permite enviar información de depuración a otro ordenador a través de un cable serie.
4. La consola de recuperación.

2.4. ARRANQUE DEL SISTEMA WINDOWS VISTA

Ms-Windows a partir de la versión Vista en vez de NTLDR utiliza BootMgr como gestor de arranque y WinLoad como cargador de sistema.
BootMgr consulta un fichero llamado "`BCD`" (*Boot Configuration Data*) que contiene las configuraciones de arranque, sustituyendo así a "`boot.ini`". Tras ello busca la existencia de un fichero de hibernación para ofrecer la opción de recuperar la sesión hibernada. En este caso cargará "`WinResume.exe`".
Una vez elegida la opción de arranque BootMgr invoca a WinLoad quien carga "`NtOSKrnl.exe`", "`Hal.dll`" y la clave "`System`" del Registro que permite a WinLoad cargar los controladores configurados para ser iniciados en el proceso de arranque. Tras ello, WinLoad cede el control al núcleo del sistema, NTOSKRNL, terminando el proceso de arranque para comenzar la carga del sistema operativo.

Tanto `bootmgr` como `bcd` residen en la raíz del sistema de ficheros, mientras que WinLoad reside en el directorio de sistema (`C:\windows\system32\winload.exe`).

→ BootMgr (`C:\bootmgr`) ← `bcd`
  a) `¿Fichero de hibernación?→ WinResume.exe`
  b) `WinLoad.exe → NtOSKrnl.exe + Hal.dll + HKLM\System → NtOSKrnl.exe`

2.5. ARRANQUE DEL SISTEMA GNU/LINUX

Para sistemas GNU/Linux, el cargador del sistema carga la imagen del núcleo Linux (o kernel). Esta imagen habitualmente se almacena en un archivo comprimido con zlib. Al inicio de esta imagen comprimida existe un código dependiente de la plataforma que se encarga de hacer una mínima configuración del *hardware* (funciones `start` y `startup_32`[14]) y descomprimir en memoria el núcleo que se encuentra en la imagen (función `decompress_kernel`). Opcionalmente puede existir una imagen de disco inicial (`initrd` o `initramfs`) que este código se encarga también de colocar en memoria y anotar su existencia. Por último comienza el arranque del núcleo cediendo el control al núcleo o "kernel" cargado en memoria. La primera función que ejecuta el núcleo (`startup_32`[15]) se ocupa de establecer el manejo de memoria, el tipo del CPU y otra funcionalidad adicional como capacidades de punto flotante. Después se ocupa de funcionalidades independientes del *hardware* por medio de la llamada a la función "`start_kernel()`".
Esta función realiza una larga serie de inicializaciones del sistema y monta -si existía- el sistema de ficheros cargado inicialmente en memoria junto con el núcleo. Esto permite que el kernel pueda cargar módulos externos (sin estar compilados en el propio núcleo, manteniéndolo más pequeño) sin tener que leerlos de dispositivos para cuyo acceso sea necesario un controlador o módulo (quizá el mismo que se está cargando).
El sistema de archivos es cambiado por medio de la función "`pivot_root()`" la cual desmonta el sistema de archivos temporal y lo reemplaza con el real. Una vez listo el manejador de excepciones, el planificador de tareas y demás, por fin el sistema se considera totalmente operacional a nivel de procesos. En ese momento el kernel ejecuta el proceso "init" e inicia una tarea de inactividad por medio de "`cpu_idle()`".

El proceso init es el primero en el espacio de usuario, tiene el PID 1 y es el proceso padre de todos los demás procesos. También es el primer proceso que se ejecuta escrito en C. Lo primero que hace es verificar y montar los sistemas de archivos, inicia servicios de usuario necesarios y cambia a un ambiente basado en usuario cuando el proceso de inicio termina.
El proceso init utiliza un fichero de configuración (`/etc/inittab`) y se ejecuta con un parámetro, conocido como "nivel de ejecución" o "*runlevel*" que toma un valor desde 0 hasta 6. Init llama a "rc"[16] quién lanza una serie de *scripts* que se ocupan de iniciar y parar servicios. Estos *scripts* se guardan en un directorio llamado "`/etc/init.d`" y se enlazan para cada nivel de ejecución desde el directorio "`/etc/rcX.d`" siendo X el nivel.
Los nombres de los enlaces empiezan por la letra "S" o la letra "K" y después un número. "rc" llama por orden alfabético a los *scripts* que empiezan por "K" añadiendo el parámetro "`stop`" y después a los que empiezan por "S" con el parámetro "`start`".
Por último ejecuta los scripts en "`/etc/rc.boot`" (obsoleto) y el script "`/etc/rc.local`" y expande (habitualmente) procesos `mingetty` para activar las consolas.

→ zImage o bzImage ← initrd o initramfs
→ `pivot_root()` → `init` X + `cpu_idle()` → `rc` → `mingetty` + `xserver`

14 La función "startup_32" es la específica para plataformas x86 de 32 bits.
15 Esta función startup_32 pertenece al núcleo y es diferente a la función anterior del mismo nombre.
16 Según versiones puede ser "/etc/init.d/rc", "/etc/init.d/rcS" o "/etc/rc.d/rc.sysinit".

a) Niveles de ejecución (*Runlevels*)

A continuación se muestran los diferentes niveles de ejecución y su aplicación en las principales distribuciones. La opción remarcada en negrilla indica el nivel de ejecución por omisión.

Nivel	Genérico	RedHat (Fedora...)	Slackware	Debian (Ubuntu...)
0	Apagar equipo (*Halt*)	Apagar equipo (*Halt*)	Apagar equipo (*Halt*)	Apagar equipo (*Halt*)
1 (S)	Modo de usuario único (*Single-User Mode*)	Modo de usuario único (*Single-User Mode*)	Modo de usuario único (*Single-User Mode*)	Modo de usuario único (*Single-User Mode*)
2	Modo multi-usuario sin red	Personalizable (no se usa)	Personalizable (configurado como n. 3)	**Modo multi-usuario completo**
3	Modo multi-usuario con red	Modo multi-usuario con red	**Modo multi-usuario con red**	Modo multi-usuario completo
4	Personalizable (no se usa)	Personalizable (no se usa)	Modo multi-usuario completo	Personalizable (no se usa)
5	Modo multi-usuario completo (con red y entorno gráfico)	**Modo multi-usuario completo**	Personalizable (configurado como n. 3)	Modo multi-usuario completo
6	Reiniciar equipo (*Reboot*)	Reiniciar equipo (*Reboot*)	Reiniciar equipo (*Reboot*)	Reiniciar equipo (*Reboot*)

Para configurar los scripts de inicio se puede usar `chkconfig` en sistemas basados en RedHat o `update-rc.d` en sistemas basados en Debian.

3. ANEXO II - GESTORES DE ARRANQUE

3.1. GRUB

GRUB (*GRand Unified Bootloader*) escribe la primera etapa del gestor en el MBR o en un VBR. Esta etapa carga el resto del programa (segunda etapa) y un fichero de configuración (`menu.lst`) que residen en una partición de un volumen del sistema. Si no es posible cargar la segunda etapa GRUB ofrece una línea de comandos.
Esta segunda etapa ofrece tres interfaces: un sencillo menú de selección -en modo texto o gráfico-, un editor de configuración y una consola de línea de comandos. Estas interfaces permiten elegir qué sistema operativo se desea arrancar y, si se desea, especificar parámetros del sistema.
El editor de configuración y las líneas de comandos permiten rectificar o modificar el arranque cuando se malconfigura o se corrompe. Además como el fichero de configuración reside en una partición del disco, se puede modificar la configuración sin necesidad de reescribir el MBR o el VBR.

GRUB soporta métodos de arranque directo -como cargador de sistema- o de arranque encadenado ("*chain loading*"), permite dispositivos LBA o CHS con más de 1024 cilindros, múltiples sistemas de ficheros como ext2 y ext3...

Ejemplo sencillo de fichero de configuración (`menu.lst`):

```
default         0
timeout         5
color cyan/blue white/blue

title           Debian GNU/Linux, kernel 2.6.26-1-686-bigmem
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-1-686-bigmem root=/dev/sda1 ro quiet vga=791
initrd          /boot/initrd.img-2.6.26-1-686-bigmem

title           Windows
root            (hd0,0)
makeactive
map             (hd0)  (hd1)
map             (hd1)  (hd0)
chainloader     +1
```

3.2. LILO

LiLo (*Linux Loader*) es un gestor de arranque más antiguo que GRUB que ofrece un menú de opciones de arranque. No dispone de interfaz de línea de comandos y no entiende sistema de ficheros, por lo que código y configuración se almacenan directamente en el MBR. Un error en la configuración puede impedir el arranque de ningún sistema.

3.3. NTLDR

NTLDR (*NT Loader*) es el gestor de arranque usado en Ms-Windows desde NT hasta XP y 2003. Está diseñado para sistemas que únicamente disponen de instancias de Windows[17]. Según este diseño, el "Código Maestro de Arranque"[18] del MBR carga mediante carga encadenada ("*chain load*") el código del VBR de la partición de Windows. Este código del VBR busca en la raíz de la partición de sistema el fichero `NtLdr` (`C:\NtLdr`).
Una vez en memoria NTLDR puede ofrecer al usuario seleccionar entre distintas instancias de Windows -como gestor de arranque- basándose en el fichero de configuración `boot.ini`[19] y cargar la opción elegida -cargador de sistema-.

El fichero de configuración (`boot.ini`) se puede editar directamente o mediante el comando `bootcfg`:

```
[boot loader]
timeout=30
default=multi(0)disk(1)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(1)rdisk(0)partition(1)\WINDOWS="Windows XP Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(2)\WINNT="Windows 2000 Professional" /fastdetect
```

3.4. BOOTMGR

Ms-Windows a partir de la versión Vista utiliza BootMgr como gestor de arranque y WinLoad como cargador de sistema.

17 Cuando se instala Windows, éste copia el sector de arranque existente con el nombre "BootSec.dos". El gestor de arranque ofrece la opción de utilizar dicho sector de arranque para iniciar "sistemas anteriores". Sin embargo una vez instalado Windows, NTLDR no permite configurar el inicio de otros sistemas. Para ello es recomendable usar un cargador potente como GRUB.

18 Versión de Veritas Software empaquetada dentro de "dmadmin.exe" y "fdisk" (entre otros y según versiones).

19 `Boot.ini` y `NtDetect.exe` también deben residir en la raíz del sistema (C:\).

Igual que NTLDR, está diseñado para sistemas que únicamente disponen de instancias de Windows, aunque puede configurarse para lanzar otros cargadores mediante "*chainload*". Sin embargo las herramientas de configuración del gestor de arranque de Microsoft no facilitan incluir entradas de este tipo[20].
De nuevo según este diseño, el "Código Maestro de Arranque" del MBR carga mediante carga encadenada ("*chain load*") el código del VBR de la partición de Windows. Este código del VBR busca en la raíz de la partición de sistema el fichero `BootMgr` (`C:\BootMgr`).
BootMgr, como gestor de arranque, lee un fichero de configuración llamado `bcd` y llama al cargador de sistema de Windows (`WinLoad.exe`) que reside en la partición del sistema dentro del directorio de sistema (`\windows\system32\winload.exe`).

a) El fichero bcd

Vista mantiene permanente abierto el fichero `bcd` para poder hacer cambios en él en caso de hibernación, por ejemplo. Esto hace que no pueda abrirse con el bloc de notas o moverse. En algunas configuraciones con varias instalaciones de Vista el sistema, tras arrancar con el fichero correcto, puede abrir erróneamente el fichero incorrecto lo que puede generar pequeños (o grandes) problemas de difícil localización. En esos casos antes de modificar el fichero `bcd` hay que asegurarse que se está modificando el fichero correcto.
Para modificar el fichero `bcd` es recomendable usar una herramienta de terceros como EasyBCD[21] que es gratuita, aunque puede modificarse mediante comandos de `bcdedit`[22]. Por ejemplo:

```
bcdedit /set {xxxxxxx} description "No arrancar, es un Windows :-P"
```

Se puede respaldar el fichero `bcd` mediante:

```
bcdedit /export X:\folder\bcd.txt
bcdedit /import X:\folder\bcd.txt
```

A diferencia de otros gestores como GRUB o NTLDR que se basan en información de la BIOS para localizar la partición donde reside el sistema a arrancar, BootMgr se basa en el GUID del disco y un desplazamiento, y busca en todos los dispositivos disponibles el que tenga la firma indicada en el fichero de configuración (`bcd`) saltando al sector indicado en el desplazamiento, ignorando la tabla de particiones.
Esto permite mover el disco de Windows dentro de la misma máquina (por ejemplo insertando otro disco "delante") sin afectar al arranque. Sin embargo en caso de que esta firma se cambie, o la instalación se clone a otro disco, BootMgr nos indicará "*winload.exe ... is missing or corrupt*". Para solucionar esto puede utilizarse la opción "*auto repair*" del DVD de Vista, el comando "`bootrec.exe /rebuildbcd`" o el comando `bcdedit.exe`[23].

Para evitar este problema se puede "generalizar" el identificador del disco en el `bcd`, de la misma manera que lo hace la herramienta `sysprep` de Microsoft, con los siguientes comandos:

```
bcdedit /set {current} osdevice boot
bcdedit /set {current} device boot
bcdedit /set {bootmgr} device boot
bcdedit /set {memdiag} device boot
bcdedit /set {ntldr}   device boot
```

Para "generalizar" un fichero bcd de otra instancia de Vista (por ejemplo al arrancar desde el DVD) se puede usar el parámetro "`/store D:\boot\bcd`" (`bcdedit /store D:\boot\bcd /set {current} osdevice boot`).

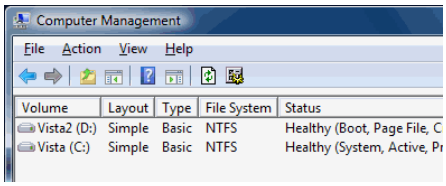
Para volver a "especializar" el fichero `bcd` (`sysprep` lo hace, pero es totalmente innecesario) basta lanzar los mismos comandos sustituyendo "`boot`" por "`partition=X:`".

b) Multiarranque con Windows Vista

Windows Vista ofrece la posibilidad de gestionar el arranque de distintos sistemas (todos ellos instancias de Windows). Si se utiliza el gestor de multiarranque de Vista éste requerirá que el gestor de arranque (BootMgr) esté en la partición

20 De nuevo lo habitual en máquinas con múltiples sistemas es instalar GRUB en el MBR.
21 EasyBCD permite además incluir entradas de "*chainload*" para lanzar otros cargadores como GRUB (http://neosmart.net/dl.php?id=1)
22 El ejecutable bcdedit.exe de Vista se puede copiar y utilizar correctamente dessde Windows XP.
23 El modo más sencillo de obtener una consola de comandos desde el DVD de Vista es seleccionar "Instalar ahora" en vez de "Reparar el equipo" y en la siguiente ventana pulsar Shift+F10.

primaria activa, pudiendo estar los sistemas en cualesquiera otras particiones.
Siguiendo la tónica habitual de Microsoft de llamar las cosas al revés, la partición de arranque donde reside el BootMgr y el fichero `bcd` será identificada en el gestor de discos como partición de sistema (*System*), mientras que la partición donde resida el sistema que esté en marcha será identificada como partición de arranque (*Boot*).



Evitando los procesos estándar de Microsoft, en una máquina se puede instalar distintas instancias de Windows de manera totalmente independiente, usando a su vez un gestor de multiarranque independiente, en cuyo caso la partición será identificada como de arranque y sistema siempre que no sea una partición lógica, caso no previsto por Microsoft y que por tanto no será correctamente identificado.

3.5. LOADLIN

LoadLin (*LoadLin*) es un ejecutable para Ms-DOS y como tal es completamente diferente de los cargadores de sistema "tradicionales". LoadLin aprovecha que los núcleos de Ms-DOS y Windows 9x permiten ser reemplazados para arrancar un sistema GNU/Linux sobre el sistema operativo ya cargado. Esto era muy útil cuando existía mucho hardware soportado por Ms-DOS o por Windows 9x y no por GNU/Linux.