
eduStance®
DISTANCE LEARNING


essi projects

eduStance
DISTANCE LEARNING

ADMINISTRACIÓN AVANZADA RED HAT ENTERPRISE LINUX

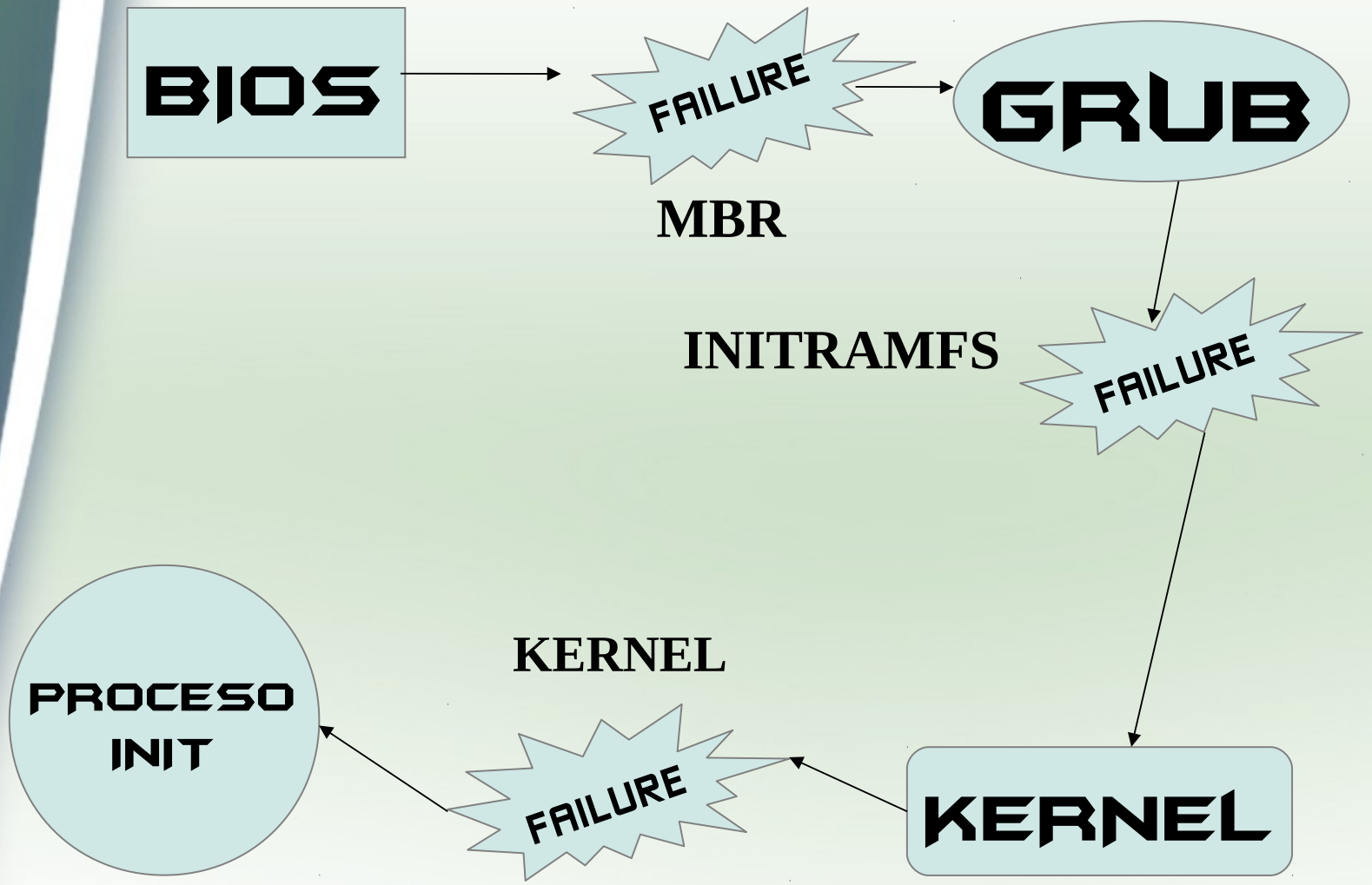
essi projects

eduStance
DISTANCE LEARNING

TROUBLESHOOTING BOOT

essi projects

TROUBLESHOOTING BOOT



TROUBLESHOOTING BOOT

Incidencias MBR

Según el diseño de particiones creado por IBM, disponemos de 512 bytes para el MBR:
446 son para el bootloader (gestor de arranque)
64 bytes para las particiones – 16 bytes por partición, por lo que sólo disponemos de 4 particiones primarias
y 2 bytes adicionales para la firma del MBR.

TROUBLESHOOTING BOOT

<table>
<tr><td rowspan="3">MBR

512B</td><td colspan="4">BOOTLOADER
(GRUB)

446 B</td></tr>
<tr><td colspan="4">TABLA DE PARTICIONES 64B</td></tr>
<tr><td>Partición Primaria 1
16B</td><td>Partición Primaria 2
16B</td><td>Partición Primaria 3
16B</td><td>Partición Primaria 4
16B</td></tr>
<tr><td colspan="4">FIRMA MBR 2B</td></tr>
</table>

TROUBLESHOOTING BOOT

GRUB (Grand Unified Bootloader) Es el gestor de arranque, por defecto, para RHEL.

La mayor parte de los errores en este paso se deben a una configuración incorrecta en la partición de arranque o en alguno de los ficheros de la misma.

O incluso al construir el bootloader en el MBR, pasando algún parámetro incorrecto.

TROUBLESHOOTING BOOT

Una de las incidencias más comunes que se suelen producir es la corrupción del gestor de arranque, bien por herramientas que afecten al MBR (instalación incorrecta del bootloader), bien por comandos que puedan sobre-escribirlo (comando dd).

Disponemos de 2 soluciones:

1- Ejecución del comando *grub-install*

[root@localhost ~]# grub-install /dev/sda

TROUBLESHOOTING BOOT

2- Shell interactiva del grub

```
   GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename.]
grub> device (hd0) /dev/vda
device (hd0) /dev/vda
grub> root (hd0,0)
root (hd0,0)
 Filesystem type is ext2fs, partition type 0x83
grub> find /grub/stage2
find /grub/stage2
 (hd0,0)
grub> setup (hd0)
setup (hd0)
 Checking if "/boot/grub/stage1" exists... no
 Checking if "/grub/stage1" exists... yes
 Checking if "/grub/stage2" exists... yes
 Checking if "/grub/e2fs_stage1_5" exists... yes
 Running "embed /grub/e2fs_stage1_5 (hd0)"...  27 sectors are embedded.
succeeded
 Running "install /grub/stage1 (hd0) (hd0)1+27 p (hd0,0)/grub/stage2 /grub/grub.
conf"... succeeded
Done.
grub> _
```

TROUBLESHOOTING BOOT

Otro tipo de incidencias que suelen afectar al MBR son fallos en la tabla de particiones. Normalmente derivadas de un uso incorrecto de herramientas de particionado.

Backup partition table & MBR:

[root@localhost ~]# *dd if=/dev/vda *
> of=/media/disk/vda.mbr bs=512 count=1

Restore partition table & MBR:

[root@localhost ~]# *dd if=/media/disk/vda.mbr *
> of=/dev/vda bs=512 count=1

TROUBLESHOOTING BOOT

INITRAMFS es un sistema de archivos temporal usado por el núcleo Linux durante el inicio del sistema. Es usado típicamente para hacer los arreglos necesarios antes de que el sistema de archivos raíz pueda ser montado.

Es un "*pre-kernel*", con un serie de módulos, necesarios para poder desplegar el sistema de ficheros en el que se encuentra el kernel, y así poder acceder a él.

TROUBLESHOOTING BOOT

Los fallos más comunes en esta etapa suelen ser:

1.- Una mala construcción del ***initramdisk***

2.- Falta algún módulo para poder acceder al kernel

TROUBLESHOOTING BOOT

El **KERNEL** de Linux se encarga de todos los procesos del sistema operativo, como la gestión de memoria, planificador de tareas, I/O, comunicación entre procesos, y el control general del sistema. Este se carga en dos etapa: en la primera etapa el kernel (como un archivo imagen comprimido) se carga y se descomprime en memoria, y algunas funciones fundamentales como la gestión de memoria de base se establecen (***initramfs***). El control entonces se cambia a la etapa final para iniciar el kernel principal (***vmlinuz***). Una vez que el núcleo está en pleno funcionamiento - y como parte de su puesta en marcha, después de ser cargado y ejecutado - el kernel busca un proceso de inicio para ejecutar, que (separadamente) fija un espacio de usuario y los procesos necesarios para un entorno de usuario y ultimar la entrada . Al núcleo en sí entonces se le permite pasar a inactivo, sujeto a las llamadas de otros procesos.

TROUBLESHOOTING BOOT

Los fallos más comunes en esta etapa son:

1.- Paso de parámetros incorrectos desde el GRUB

2.- Módulos dañados o inexistentes

3.- Imagen corrupta del kernel

En algunos casos, el error permite retornar al GRUB para elegir parámetros diferentes u otra imagen.

¡¡¡¡¡¡ ES MUY IMPORTANTE NO ELIMINAR EL ÚLTIMO KERNEL AL INSTALAR UN KERNEL NUEVO !!!!!!

TROUBLESHOOTING BOOT

El trabajo de Init es "conseguir que todo funcione como debe ser" una vez que el kernel está totalmente en funcionamiento. En esencia, establece y opera todo el espacio de usuario. Esto incluye la comprobación y montaje de sistemas de archivos, la puesta en marcha de los servicios de usuario necesarios y, en última instancia, cambiar al entorno de usuario cuando el inicio del sistema se ha completado. Init se ejecuta con un parámetro, conocido como runlevel que determina que subsistemas pueden ser operacionales. Cada runlevel tiene sus propios scripts que codifican los diferentes procesos involucrados en la creación o salida del nivel de ejecución determinado, y son estas secuencias de comandos los necesarios en el proceso de arranque. Los scripts de Init se localizan normalmente en directorios con nombres como "/etc/rc...". El archivo de configuración de más alto nivel para Init es /etc/inittab, donde se guardará el runlevel por defecto para el sistema.

TROUBLESHOOTING BOOT

Los fallos más comunes en esta etapa son:

1.- Problemas con el montaje de los dispositivos (parámetros incorrectos en fstab)

2.- Falta runlevel (en RHEL 6, si falta se carga el runlevel 3 gracias al fichero /etc/init/rcS.conf)

3.- Sistema de ficheros raíz corrupto o dañado

4.- Scripts dañados o inexistentes de los diferentes runlevel

eduStance
DISTANCE LEARNING

TROUBLESHOOTING FILESYSTEMS

essi projects

TROUBLESHOOTING FILESYSTEM ISSUES

Cuando trabajamos con FS, podemos encontrar incidencias en 3 campos, y en cada uno de ellos dispondremos de algunas herramientas estándar para intentar solventar dichas incidencias:

1.- Particionado (sfdisk, dd)

2.- FS corrupto (dumpe2fs, debug2fs, e2fsck)

3.- Ficheros borrados (dumpe2fs, debug2fs, dd, extundelete)

TROUBLESHOOTING FILESYSTEM ISSUES

Particionado:

Errores:

Particiones eliminadas por error, particionado en dispositivo equivocado.

Soluciones:

Previo backup de tabla de particiones:

sfdisk -d /dev/vda > /media/disk/part_table.vda

Restauración:

sfdisk -- force /dev/vda < /media/disk/part_table.vda

Otra forma de realizar un backup:

dd if=/dev/vda of=/media/disk/mbr.vda bs=512 count=1

Restauración:

dd if=/media/disk/mbr.vda of=/dev/vda

TROUBLESHOOTING FILESYSTEM ISSUES

FS Corrupto:

Información:

dumpe2fs nos ofrecerá información relativa al FS, donde podremos ver los backup del superbloque:

dumpe2fs /de/vda1 | grep -i superblock

Restauración:

Un sistema de ficheros corrupto se puede restaurar gracias al superbloque, pero si este esta dañado podemos recurrir a los bloques de respaldo:

Reparamos con un respaldo:

e2fsck -b 40961 -f -y /dev/vda1

TROUBLESHOOTING FILESYSTEM ISSUES

Restaurando ficheros borrados:

La restauración de ficheros es una tarea tediosa y que en ocasiones produce resultados infructuosos. Existen múltiples herramientas de terceros, incluso de pago, que buscan los “magic number”, los cuales determinan tipos de ficheros, escaneando el dispositivo completo, una partición o una imagen. Esto resulta bien para ficheros binarios, pero no tan bien para ficheros de texto plano. Algunas herramientas opensource son *foremost*, *photorec*, *testdisk*.

TROUBLESHOOTING FILESYSTEM ISSUES

Restaurando ficheros borrados:

El paquete *e2fsprogs*, contiene varias utilidades para gestionar los sistemas de ficheros ext2/ext3/ext4. Entre las utilidades se encuentran el comando *e2fsck*, que nos permitirá chequear fallos en el sistema de ficheros; *tune2fs*, que nos permitirá modificar parámetros y opciones del sistema de ficheros; y *debugfs* que nos permitirá examinar la estructura del sistema de ficheros.

TROUBLESHOOTING FILESYSTEM ISSUES

La utilidad debugfs contiene varias opciones que nos permitirán extraer información de un FS:

1.- *dump_extents*: vuelca la información del extent de un fichero, es decir, los bloques que ocupa un fichero.

2.- *lsdel:* muestra los ficheros borrados

3.- *undel*: restaura ficheros borrados

4.- *dump_unused*: vuelca el contenido de los inodos en desuso. Podemos rescatar el contenido de los inodos en desuso con el comando *dd*, ya que *dump_unused* nos mostrará los bloques en los que se encuentran los datos.

TROUBLESHOOTING FILESYSTEM ISSUES

Restauración de ficheros con extundelete:

1.- *extundelete /dev/vda1 --restore-file path/to/file1*
Restaura el fichero *file1* partiendo de la raíz del sistema de ficheros.

2.- *extundelete /dev/vda1 --restore-inode ${number}*
Restaura el fichero cuyo inodo corresponde con ${number}

3.- *extundelete /dev/vda1 --restore-all* – Restaura todo el contenido borrado, todavía accesible

4.- *extundelete /dev/vda1 --restore-all --after dtime*
Restaura todo el contenido borrado desde la fecha *dtime* (segundos desde 1970-01-01) hasta el momento actual

5.- *extundelete /dev/vda1 --restore-all --before dtime*
Restaura todo el contenido borrado desde *epoch* (1970-01-01) hasta la fecha *dtime* (segundos desde 1970-01-01)

TROUBLESHOOTING FILESYSTEM ISSUES

La utilidad *tune2fs*, nos permite modificar ciertos parámetros del sistema de ficheros:

1.- Modificar la etiqueta del volumen:
tune2fs -L ETIQUETA /dev/vda1

2.- Modificar el intervalo de chequeos:
tune2fs -i ${intervalo}[d|w|m] /dev/vda1 –Donde d=días, w=semanas, m=mes

3.- Modificar el contador de chequeo en montajes:
t*une2fs -c ${numero} /dev/vda1* – Cantidad de montajes para chequeo

4.- Opciones de montaje por defecto:
t*une2fs -o acl,user_xattr /dev/vda1*

5.- Espacio reservado para el superusuario:
t*une2fs -m ${espacio} /dev/vda* -Espacio es valor numérico, representa el % de espacio reservado (por defecto es 5)

TROUBLESHOOTING FILESYSTEM ISSUES

Uso de kpartx:

kpartx crea mapeos de las tablas de particiones de los dispositivos.

Con *kpartx* podemos extraer las particiones o los grupos de volúmenes dentro de una imagen o un volumen lógico que contiene más particiones:

Particiones estándar:

kpartx -av /path/to/image/file

mount /dev/mapper/loop0p1 /mnt

Volúmenes lógicos:

kpartx -av /path/to/image/file

lvm vgscan

lvm vgchange -ay new_vg

lvm lvscan

mount /dev/new_vg/new_lv

eduStance
DISTANCE LEARNING

RESIZE FILESYSTEMS (Shrink & Grow)

essi projects

RESIZE EXT4 FILESYSTEM

Shrink ext4 FS:

umount /dev/vdb1

e2fsck -f /dev/vdb1

resize2fs /dev/vdb1 ${new_size}

Resizing the filesystem on /dev/vdb1 to 2048 (4k) blocks

Ahora recreamos la partición, calculamos el nuevo tamaño multiplicando los bloques resultantes por el tamaño del bloque y le damos entre un 3-5% de margen.

echo “2048 * 4 * 1.05”|bc --> 12288

fdisk -cu /dev/vdb

>> d >> 1

>> n >> p >> 1

>> first sector >> default

>> last sector >> +12288K << (2048 * 4 * 1.05)

mount /dev/vdb1 /mnt

RESIZE EXT4 FILESYSTEM

Grow ext4 FS:

Recreamos la partición ocupando todo el espacio disponible

fdisk -cu /dev/vdb

>> d >> 1

>> n >> p >> 1

>> first sector >> default

>> last sector >> default

mount /dev/vdb1 /mnt

El tamaño del FS es el mismo que antes de modificar la partición. Extender el FS puede ser realizado en caliente.

resize2fs /dev/vdb1

Si no se le pasa ningún tamaño se extenderá hasta el límite de la partición o del volumen lógico.

El tamaño del FS ha cambiado y seguimos utilizando el punto de montaje.

eduStance
DISTANCE LEARNING

RUNNING DISKLESS CLIENTS

essi projects

eduStance
DISTANCE LEARNING

RUNNING DISKLESS RED HAT CLIENTS

DOCUMENTACIÓN ADJUNTA

essi projects

Configuring diskless clients with Red Hat Enterprise Linux

Dave Kline

Beginning with Red Hat Enterprise Linux 5.1, Red Hat supports booting Linux via **NFSROOT**. This allows clients to mount root filesystems remotely on NFS servers. Combined with PXE (Preboot eXecution Environment), clients can boot and operate without local storage of any kind. Additionally, when using storage snapshots, many clients can operate from a single filesystem.

OVERVIEW

The process of enabling diskless operation (sometimes called netbooting) includes a number of steps. The client powers on, requests network and boot information, downloads a kernel and **initrd**, and finally mounts a root filesystem via NFS. This is similar to PXE booting for the purposes of kickstarting installations. However, rather than performing an installation, the client runs normally.

This article outlines how to create both a server capable of serving diskless clients, and a single diskless client.

1: CLIENT CONFIGURATION

The first step is to create a normal RHEL host. Install Red Hat Enterprise Linux 6 onto a host. After you've customized the host you will need to allow your image to be booted via NFS. The **dracut-network** package provides the necessary network components for RHEL 6 clients to mount network root filesystems:

```
# yum install dracut-network -y
```

If creating a RHEL 5 diskless client, you should install the **busybox-anaconda** package, and disable SELinux. In general, Red Hat does not recommend disabling SELinux. As such, you should carefully consider the security implications of this action.

The last step is to create an **initrd**. Note that in this example, the path on the NFS server that will hold our client is **/usr/local/netboot**.

RHEL 6 clients

Use the following command for a RHEL 6 client, noting the path and IP address of the NFS server:

```
# dracut -f /boot/netboot6.img `uname -r` root=dhcp root-
path=nfs:<server_ip>:/usr/local/netboot/
```

RHEL 5 clients

If we were creating a RHEL 5 client, we would use the **mkinitrd** command instead:

```
# mkinitrd -f --with=virtio_net --with=nfs --net-dev=eth0
--rootdev=<server_ip>:/usr/local/netboot/ --rootfs=nfs /boot/netboot.img `uname
-r`
```

In the **mkinitrd** example above, we have explicitly specified **virtio_net.** This is to ensure the **initrd** is created with the proper network driver for our environment. If your diskless client uses a different network driver, you can specify that it be included with the **--with=** parameter. Multiple drivers can be specified this way.

2: SERVER CONFIGURATION

A diskless server provides the following services:

- DHCP/PXE
- TFTP
- NFS

These requirements are easily handled by Red Hat Enterprise Linux, though you can also make use of any existing infrastructure that may provide them. We will be using a RHEL 6 host as the server, though the commands and package names needed are very similar on RHEL 5.

First, install needed packages, and ensure necessary daemons start at boot:

```
# yum install dhcp syslinux tftp-server -y
# yum groupinstall "NFS file server" -y
# chkconfig dhcpd on
# chkconfig tftp on
# chkconfig nfs on
```

Next, create directories to host filesystems and configuration information, and copy the **pxelinux.0** binary in place:

```
# mkdir /usr/local/netboot/
# mkdir /var/lib/tftpboot/pxelinux.cfg/
# cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/
```

Copy over the configured client we created earlier. A simple way of doing so is to use **rsync** from the server**:**

```
# rsync -av --progress --exclude=/proc --exclude=/sys root@CLIENT_HOST:/
/usr/local/netboot/
```

The DHCP service provides clients with network information, as well as the location of the PXELINUX network boot loader. The following is an example of a **dhcpd.conf** file, found under **/etc/dhcp/dhcpd.conf**:

```
ddns-update-style interim;
ignore client-updates;
allow booting;
allow bootp;
subnet 192.168.122.0 netmask 255.255.255.0 {
       option routers   192.168.122.1;
       option subnet-mask  255.255.255.0;
       option nis-domain  "domain.org";
       option domain-name  "domain.org";
       option domain-name-servers 192.168.122.1;
       option time-offset  -18000; # Eastern Standard Time
       default-lease-time 21600;
       max-lease-time 43200;

       host netboot6 {
    next-server <server_ip>;
    hardware ethernet <client_mac_address>;
    fixed-address <client_ip_address>;
    filename "pxelinux.0";
       }
}
```

In this example we have specified a client, **netboot6**, with the IP address of **<client_ip_address>**. The **next-server** parameter specifies where **pxelinux.0** can be downloaded (directly from the same server is quite common), and the MAC address that corresponds to **netboot6**: **<client_mac_address>**. Additional clients can be placed under different **host** declarations, provided that unique IP and MAC addresses are specified.

Once finished, restart dhcpd and look for errors in **/var/log/messages**:

```
# service dhcpd restart
```

Our client will mount its root filesystem via NFS, so we need to add the following to **/etc/exports**:

```
/usr/local/netboot/       <client_ip_address>(rw,async,no_root_squash)
```

Once complete, reload your changes with **exportfs -ra** or **service nfsd restart**. You can verify your exports by running **exportfs**.

Next we will enable extra verbosity for our tftp server, which is useful when clients initially request files.

Add **-vv** to following line under **/etc/xinetd.d/tftp**:

```
server_args       = -vv -s /var/lib/tftpboot
```

The client's kernel, and the **initrd** that we created earlier, should be copied and served via tftp with the following commands:

```
# cp /usr/local/netboot/boot/netboot6.img /var/lib/tftpboot/
# cp /usr/local/netboot/boot/vmlinuz-2.6.32-71.14.1.el6.x86_64
/var/lib/tftpboot
```

We will now create a configuration file that our client retrieves during boot. Create the following file as **/var/lib/tftpboot/pxelinux.cfg/default**:

```
default netboot6
timeout 30
prompt 1

label netboot6
kernel /vmlinuz-2.6.32-71.14.1.el6.x86_64
append initrd=/netboot6.img rw root=nfs:<nfs_server>:/usr/local/netboot/
selinux=0 enforcing=0
```

In this example, we have specified the NFS server and path, a timeout of three seconds, and a **netboot6** stanza containing a kernel and **initrd** declaration.

With our client's filesystem in place, we need to make a few server-side changes. We will need to create directories and an **/etc/fstab/** file, and disable the primary interface. Perform the following:

```
# mkdir /usr/local/netboot/{proc,sys}
# vi /usr/local/netboot/etc/fstab

<nfs_server>:/usr/local/netboot/ /     nfs   defaults    1 1
tmpfs          /dev/shm        tmpfs  defaults    0 0
devpts         /dev/pts        devpts gid=5,mode=620 0 0
sysfs          /sys          sysfs  defaults    0 0
proc          /proc          proc  defaults    0 0

# vi /usr/local/netboot/etc/sysconfig/network-scripts/ifcfg-eth0

DEVICE="eth0"
HWADDR=”<client_mac_address”>
```

```
NM_CONTROLLED="yes"
ONBOOT="no"
BOOTPROTO="dhcp"
```

With these steps in place, we're ready to boot our client.

3: BOOTING AND TROUBLESHOOTING THE DISKLESS CLIENT

With the server and client filesystem properly configured, we can now boot over the network. Before the boot begins, you may wish to view logs on the server with **tail -f /var/log/messages**.

Because of the large amount of configuration required, it's common for users to make mistakes. To avoid the more common errors, check that:

- the firewall settings permit NFS and DHCP traffic
- the server's SELinux settings aren't interfering with the client's ability to download files
- when copying files to **/var/lib/tftpboot/**, the files have read permissions
- MAC addresses were recorded properly in **/etc/dhcp/dhcpd.conf**
- the NFS server settings and kernel settings are properly configured in **/var/lib/tftpboot/pxelinux.cfg/default**.

Your diskless client is now ready to use.

eduStance®
DISTANCE LEARNING

essi projects

Plaza Prim, 4-5. Principal 2a. 43001 – Tarragona Telf. 977 221 182