

Administración de Servidores

El núcleo Linux

Compilando el núcleo

Creado por

Iñaki Fernández de Viana y González

Huelva, octubre 2020

Sobre Nosotros

Iñaki Fernández de Viana y González



Despacho 128. Escuela Técnica
Superior de Ingeniería



Dpto. De Tecnologías de la Información



i.fviana@dti.uhu.es



+34 959217378



Objetivos

- **Peso:** 3
- **Descripción:** Candidates should be able to properly configure a kernel to include or disable specific features of the Linux kernel as necessary. This objective includes compiling and recompiling the Linux kernel as needed, updating and noting changes in a new kernel, creating an initrd image and installing new kernels.

Objetivos (II)

- **Áreas clave de conocimiento:** /usr/src/linux/, Kernel Makefiles, Kernel 2.6.x, 3.x and 4.x make targets, Customize the current kernel configuration, Build a new kernel and appropriate kernel modules, Install a new kernel and any modules, Ensure that the boot manager can locate the new kernel and associated files, Module configuration files, Use DKMS to compile kernel modules, Awareness of dracut.

Objetivos (III)

- **Términos y utilidades:** mkinitrd, mkinitramfs, make, make targets (all, config, xconfig, menuconfig, gconfig, oldconfig, mrproper, zImage, bzImage, modules, modules_install, rpm-pkg, binrpm-pkg, deb-pkg), gzip, bzip2, module tools, /usr/src/linux/.config, /lib/modules/kernel-version/*, depmod, dkms

Índice

1. Introducción
2. Obteniendo el código fuente
3. Configurando el núcleo
4. Compilando el núcleo
5. Discos RAM
6. Dkms
7. Configurando el gestor de arranque

Introducción

1

Complejidad

- ★ La compilación del núcleo es un proceso largo y tedioso debido al número de líneas de código a compilar y la gran cantidad de parámetros que influyen en dicho proceso



```
CONFIG_BLK_DEV_SD      [disk (sd) driver]
CONFIG_SD_EXTRA_DEVS   [extra slots for disks added later]
CONFIG_BLK_DEV_SR      [SCSI cdrom (sr) driver]
CONFIG_SR_EXTRA_DEVS   [extra slots for cdroms added later]
CONFIG_CHR_DEV_ST      [tape (st) driver]
CONFIG_CHR_DEV_OSST    [OnStream tape (osst) driver]

[....]
```


Fases

- ★ Las fases que seguiremos son:
 - Obteniendo el código fuente
 - Configurando el núcleo
 - Construyendo el núcleo
 - Instalando el núcleo
 - Discos RAM
 - Configurando el gestor de arranque

Obtención del código fuente

2

Introducción

- ★ El proceso de instalación del núcleo desde el código fuente depende de si:
 - Instalamos desde una distribución
 - Desde **Linux Kernel Archives**
- ★ En cualquier caso, el núcleo se instala dentro del directorio **/usr/src**, en la carpeta **linux** (enlace simbólico).

```
[root@localhost /usr/src]# ls -l
total 98832
lrwxrwxrwx  1 root root          12 oct  2 09:25 linux -> linux-3.4.48
drwxrwxr-x 23 root root       4096 jun  7 21:50 linux-3.4.48
-rw-r--r--  1 root root 101165396 jun  7 22:02 linux-3.4.48.tar.gz
```

Desde una distribución

- ★ Cada distribución permite instalar el núcleo desde su gestor de paquetes (deb, rpm, pkg).
- ★ Dispone de paquete para el código fuente y para el binario (compilado)
- ★ El proceso de instalación depende de la distribución

```
[root@localhost /usr/src]# sudo apt-get install -y linux-source
```

```
[root@localhost /usr/src]# yum install kernel-devel
```

Linux Kernel Archives

- ★ Nos posicionamos en `/usr/src`

```
cd /usr/src
```

- ★ Descargamos el fichero del kernel que queremos usar:

```
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.9.238.tar.xz
```

- ★ Descargamos el fichero de firma:

```
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.9.238.tar.sign
```

- ★ Importamos la **clave pública del “autor”**:

```
> gpg2 --locate-keys torvalds@kernel.org gregkh@kernel.org
```

Linux Kernel Archives (II)

- ★ Descomprimimos el núcleo con la utilidad **xz**:

```
[root@localhost /usr/src]# xz -d linux-4.9.238.tar.xz
```

- ★ Verificamos la correcta firma del fichero:

```
[root@localhost /usr/src]# gpg2 --verify linux-4.9.238.tar.sign
gpg: asumiendo que los datos firmados están en 'linux-4.9.238.tar'
gpg: Firmado el jue 01 oct 2020 20:41:46 CEST
gpg:          usando RSA clave 647F28654894E3BD457199BE38DBBDC86092693E
gpg: Firma correcta de "Greg Kroah-Hartman <gregkh@kernel.org>" [desconocido]
gpg: ATENCIÓN: ¡Esta clave no está certificada por una firma de confianza!
gpg:          No hay indicios de que la firma pertenezca al propietario.
Huellas dactilares de la clave primaria: 647F 2865 4894 E3BD 4571  99BE 38DB BDC8
6092 693E
```


Linux Kernel Archives (III)

★ Desempaquetamos el núcleo:

```
> tar xf linux-4.9.238.tar
```

★ Creamos el enlace simbólico:

```
> ln -s linux-4.9.238 linux
```

Configurando el núcleo

3

Limpiar núcleo anterior

- ★ El proceso de compilación del núcleo se gestiona, fundamentalmente, por **makefiles** ejecutados mediante la aplicación **make**.
- ★ El primer paso, una vez obtenido el código, es limpiar ficheros de configuración y/o códigos objeto anteriores.

Objetivo	Descripción
make clean	Borra la mayoría de los ficheros intermedios generados (.o / .ko)
make mrproper	Borra el fichero de configuración y todos los ficheros generados
make distclean	Además de lo borrado por mrproper, elimina core dumps, ficheros de backup, ficheros de navegación por el código (tags TAGS GTAGS).

Crear el fichero de configuración

- ★ El objetivo es crear el fichero `/usr/src/linux/.config` que permita compilar el núcleo correctamente.
- ★ Este fichero de texto tiene miles de líneas del tipo (clave = valor) organizados en grupos (*General Setup, File System, Device Drivers...*)

```
CONFIG_SCSI=y  
CONFIG_IDE =m  
# CONFIG_SCSI is not set
```

- ★ ¿Cómo creamos ese fichero?
 - Partimos de algo que ya funciona
 - Partimos configuración por defecto

Algo que funciona

- ★ El fichero de configuración de un núcleo que sabemos que funciona, estos ficheros se almacenan en `/boot` y no tenemos que copiar en `/usr/src/linux`:

```
[root@localhost ~]# ls /boot/config*  
/boot/config-3.5.0-41-generic /boot/config-3.2.0-54-generic  
[root@localhost ~]# cp /boot/config-3.2.0-54-generic /usr/src/linux/.config
```


- ★ También podemos consultar la configuración del núcleo en ejecución localizado en `/proc/config.gz`:

```
[root@localhost ~]# gzip -dc /proc/config.gz > /usr/src/linux/.config
```


Algo que funciona (II)

- ★ Posiblemente el fichero de configuración copiado no sea de la misma versión del núcleo que queremos compilar.
- ★ Es necesario añadir al fichero de configuración, las nuevas opciones de configuración, para ello ejecutamos **make oldconfig** o **make silentoldconfig**

```
[root@localhost /usr/src/linux]# make oldconfig
[...]
```



```
* Restart config...
* General setup
*
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [Y/n/?] y
```

Configuración por defecto

- ★ Si no disponemos de un fichero de configuración adecuado, podemos generar uno por defecto:

Objetivo	Descripción
<code>make defconfig</code>	Fichero de configuración por defecto
<code>make allmodconfig</code>	Fichero de configuración lo más modular

```
/usr/src/linux$ make defconfig
*** Default configuration is based on 'x86_64_defconfig'
#
# configuration written to .config
#
```

Modificar fichero de configuración

- ★ Una vez creado el fichero de configuración lo podemos modificar mediante un editor de texto o ejecutando

Objetivo	Descripción
<code>make config</code>	Configuración basada en entorno de texto.
<code>make menuconfig</code>	Configuración en entorno gráfico ncurses
<code>make xconfig</code>	Configuración en entorno gráfico Qt.
<code>make gconfig</code>	Configuración en entorno gráfico Gtk.

menuconfig



```
.config - Linux/x86 3.10.0-rc2 Kernel Configuration

Linux/x86 3.10.0-rc2 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module
< > module capable

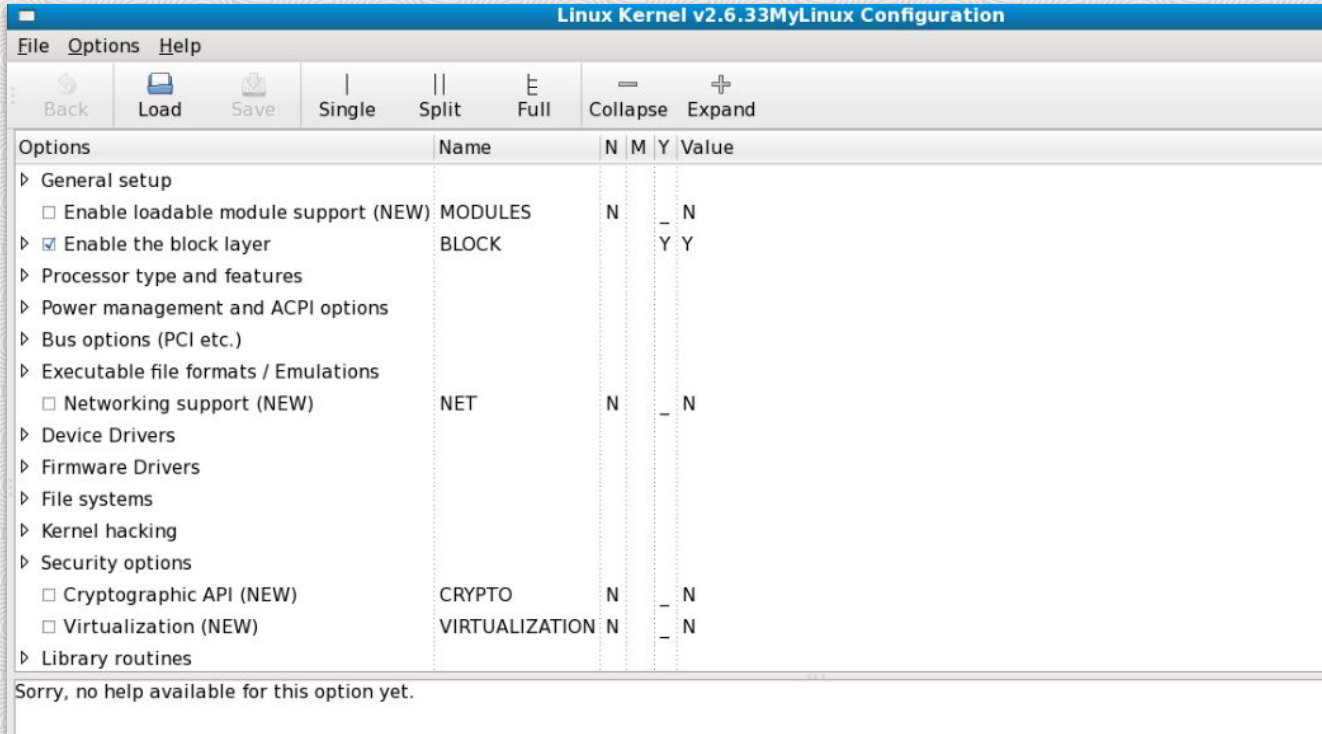
[*] 64-bit kernel
    General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
    Processor type and features --->
    Power management and ACPI options --->
    Bus options (PCI etc.) --->
    Executable file formats / Emulations --->
    *- Networking support --->
        Device Drivers --->
        Firmware Drivers --->
        File systems --->
        Kernel hacking --->
        Security options --->
    *- Cryptographic API --->
    *- Virtualization --->
        Library routines --->

<Select> < Exit > < Help > < Save > < Load >
```

xconfig



gconfig



Compilando el núcleo

4

Introducción

- ★ Una vez que tenemos creado el fichero `/usr/src/linux/.config`, estamos listos para compilar el núcleo.
- ★ Durante este proceso:
 - Compilaremos las fuentes para generar los ficheros objeto
 - Lincaremos para obtener la imagen del núcleo
 - Crearemos los módulos

Construcción

- ★ Para comenzar el proceso de compilación del núcleo sólo tendremos que ejecutar make:

```
LD      arch/x86/boot/compressed/vmlinux.lds
AS      arch/x86/boot/compressed/head_64.o
VOFFSET arch/x86/boot/compressed/..voffset.h
CC      arch/x86/boot/compressed/string.o
CC      arch/x86/boot/compressed/cmdline.o
CC      arch/x86/boot/compressed/error.o
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
RELOCS  arch/x86/boot/compressed/vmlinux.relocs
HOSTCC  arch/x86/boot/compressed/mkpiggy
CC      arch/x86/boot/compressed/early_serial_console.o
CC      arch/x86/boot/compressed/kaslr.o
CC      arch/x86/boot/compressed/cpuflags.o
CC      arch/x86/boot/compressed/kaslr_64.o
AS      arch/x86/boot/compressed/mem_encrypt.o
CC      arch/x86/boot/compressed/pgtable_64.o
CC      arch/x86/boot/compressed/eboot.o
AS      arch/x86/boot/compressed/efi_stub_64.o
AS      arch/x86/boot/compressed/efi_thunk_64.o
CC      arch/x86/boot/compressed/misc.o
GZIP    arch/x86/boot/compressed/vmlinux.bin.gz
MKPIGGY arch/x86/boot/compressed/piggy.S
AS      arch/x86/boot/compressed/piggy.o
LD      arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS      arch/x86/boot/header.o
LD      arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD  arch/x86/boot/bzImage
Setup is 17052 bytes (padded to 17408 bytes).
System is 8357 kB
CRC e0320f3d
Kernel: arch/x86/boot/bzImage is ready (#1)
[root@fedora28-nixcraft linux-4.18]#
```

Main kernel file

- ★ Una vez finalizada la compilación podemos optar por ejecutar: **make bzImage** o **make zImage** para generar el kernel main file.

```
root@localhost:/usr/src/linux$ make bzImage
```

- ★ Al finalizar este proceso, en el directorio **/usr/src/linux/arch/<arquitectura>** tendremos el fichero generado

```
root@localhost:/usr/src/linux$ ls arch
alpha      c6x  hexagon  m68k          openrisc  score  um      xtensa
arm         cris   ia64      microblaze  parisc  sh      unicore32
avr3        frv   Kconfig  mips          powerpc  sparc   x86
blackfin    h8300 m32r     mn10300     s390  tile    x86_64
```

Main kernel file (II)

- ★ Este fichero lo copiamos a **/boot**:

```
root@localhost:/usr/src/linux/$ cp arch/x86_64/boot/bzImage /boot/bzImage-2.6.35.4
```

- ★ Opcionalmente copiamos el fichero **System.map**:

```
root@localhost:/usr/src/linux/$ cp System.map /boot/System.map-2.6.35.4
```

- ★ Y creamos el enlace simbólico

```
root@localhost:/usr/src/linux/$ ln -s /boot/System.map-2.6.35.4 /boot/System.map
```


Módulos

- ★ Para generar los módulos ejecutamos `make modules`:

```
root@localhost:/usr/src/linux$ make modules
```

- ★ Y a continuación los copiamos en el directorio `/lib/modules` ejecutando `make modules_install`:

```
root@localhost:/usr/src/linux$ make modules_install
```

- ★ El resultado de este comando es la creación de una carpeta con el número de versión y ficheros `.ko`

```
root@localhost:/lib/modules/5.9.0-991.native/kernel/fs/isofs $ ls  
isofs.ko
```


Módulos (II)

- ★ Dentro de esta carpeta está el fichero **modules.dep** que establece las dependencias entre módulos:

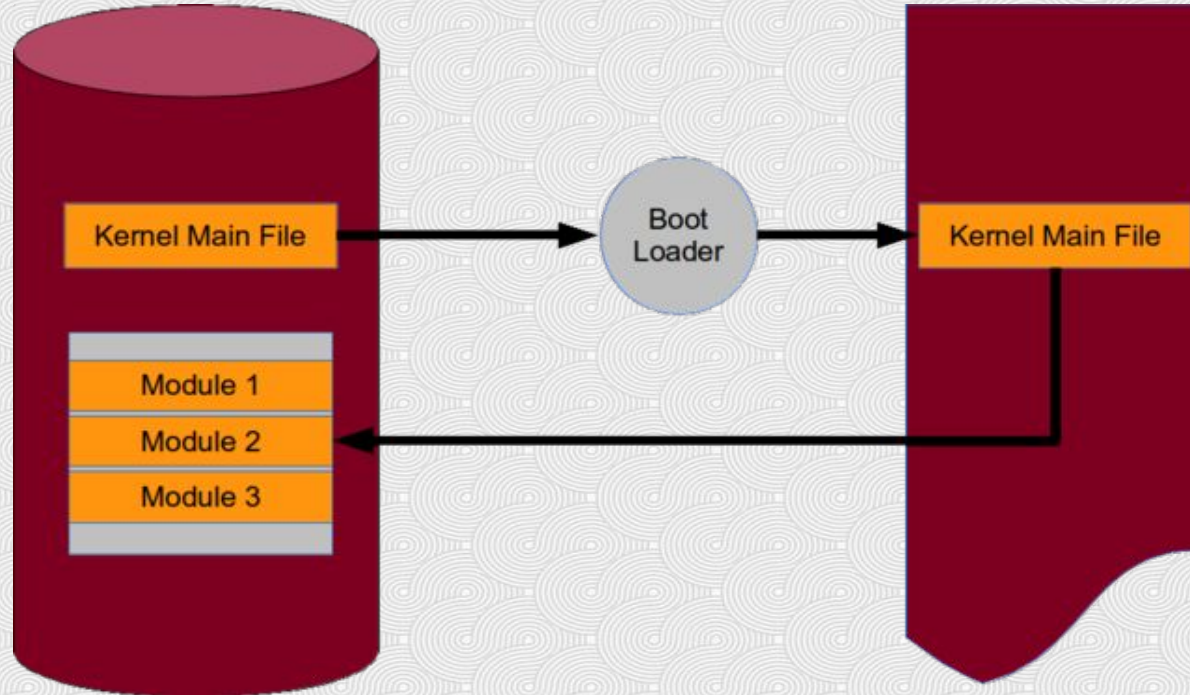
```
★ /lib/modules/3.8.0-31-generic$ ls
build      modules.builtin.bin  modules.inputmap
initrd     modules.ccwmap      modules.isapnpmap
kernel     modules.dep         modules.ofmap
modules.alias modules.dep.bin     modules.order
modules.builtin modules.ieee1394map modules.seriomap
```

- ★ Si no existiera, es necesario crearlo usando el comando **depmod**.

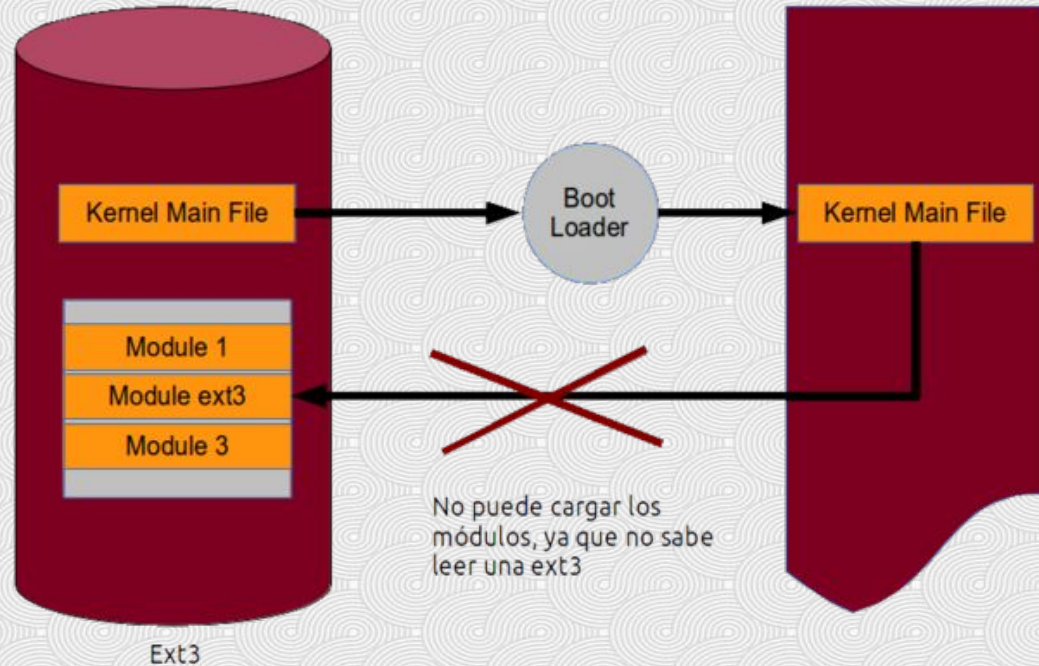
Discos RAM

5

Introducción (I)



Introducción (II)

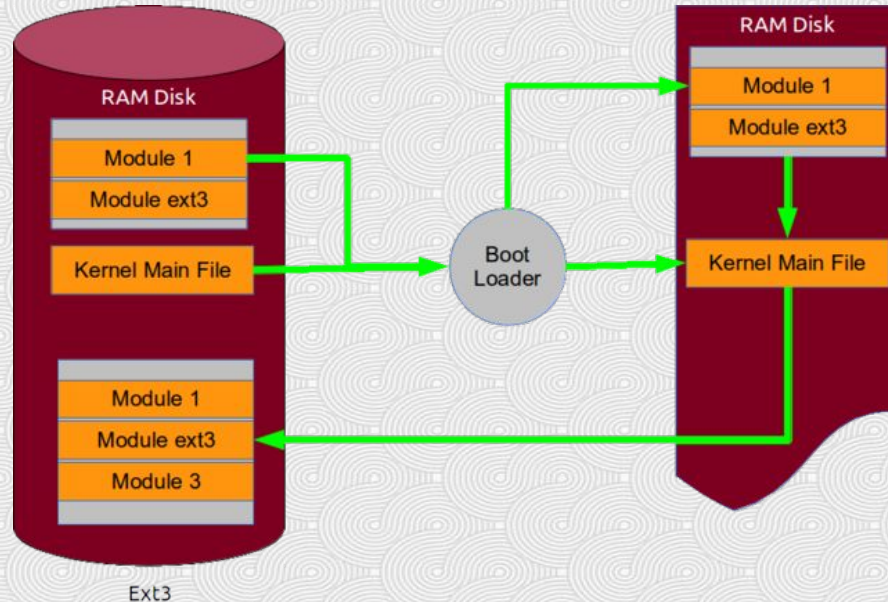


Introducción (III)



Introducción (IV)

- ★ Colección de módulos y utilidades críticas que el boot loader lee y carga en memoria.



¿Cuándo se debería crear?

- ★ Cuando el kernel no incluya todos los módulos necesarios para el arranque
 - RAID
 - LVM
 - SAN
 - ext3

¿Cómo generarlos?

- ★ Dependiendo de la distribución, podemos usar:
 - mkinitrd
 - mkinitramfs
 - dracut

mkinitrd

- ★ El comando **mkinitrd** es usado en Red Hat, Fedora y derivados.
- ★ El fichero generado está comprimido con la utilidad **gzip** y empaquetado con **cpio**
- ★ Los módulos se deben encontrar en el directorio `/lib`

```
# mkinitrd /boot/initrd-2.6.35.4.img 2.6.35.4
```

mkinitramfs

- ★ El comando **mkinitramfs** es usado en Debian y derivados.
- ★ El fichero generado está comprimido con la utilidad **gzip** y empaquetado con **cpio**
- ★ Los módulos se deben encontrar en el directorio `/lib`

```
# mkinitramfs -o /boot/initramfs-2.6.35.4.img 2.6.35.4
```


dracut

- ★ El comando **mkinitramfs** ha sido reemplazado por **dracut**. De hecho **mkinitramfs** es un “wrapper” de **dracut**.

```
# dracut
```

- ★ Podemos ver los contenidos de la imagen RAM creada mediante la utilidad **lsinitrd**.

```
# lsinitrd /boot/initramfs-$(uname -r).img | less
```

DKMS



6

Introducción

- ★ *Dynamic Kernel Module Support* (DKMS) es un framework usado para generar módulos del núcleo Linux cuyas fuentes no suelen residir en el árbol fuente del núcleo Linux.
- ★ DKMS habilita controladores de núcleo para ser automáticamente reconstruidos cuando un nuevo núcleo es instalado lo que hace posible usar un nuevo núcleo inmediatamente, en lugar de esperar que módulos compatibles de terceras partes para ser liberado.
- ★ La gestión de este framework se realiza mediante el comando **dkms**

Instalación de un módulo

- ★ Para instalar un módulo (denominado mimodulo) usando dkms partimos de su código fuente y del fichero dkms.conf.

```
# cd /usr/src
# curl -L <URL con el código fuente>/mimodulo.tar.gz | tar xvz
```

- ★ Modificamos el nombre del directorio que contiene el módulo para que contenga su número de versión

```
# version=0.12.5
# mv mimodulo mimodulo-$version
```

```
# dkms add -m mimodulo -v ${version}
# dkms build -m mimodulo -v ${version}
dkms install -m mimodulo -v ${version}
```

Instalación de un módulo

★ Añadimos el módulo y lo construimos

```
# dkms add -m mimodulo -v ${version}
# dkms build -m mimodulo -v ${version}
```

★ Instalamos el módulo

```
# dkms install -m mimodulo -v ${version}
```


Configurando el gestor de arranque

7

Introducción

- ★ El bootloader es el programa encargado de cargar el kernel en memoria
- ★ Para que el sistema arranque correctamente, hay que indicarle la ubicación del kernel y de su ramdisk

```
GNU GRUB version 1.97~beta4

Ubuntu, Linux 2.6.31-20-generic
Ubuntu, Linux 2.6.31-20-generic (recovery mode)
Ubuntu, Linux 2.6.31-14-generic
Ubuntu, Linux 2.6.31-14-generic (recovery mode)
Memory test (memtest86+)
Memory test (memtest86+, serial console 115200)
GRUB boot loader (on /dev/sda1)
Upgrade to Fedora 11 (Leonidas) (on /dev/sda6)
Fedora (2.6.26.8-57.fc8) (on /dev/sda6)
Fedora (2.6.26.6-49.fc8) (on /dev/sda6)

Use the + and - keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting or 'c' for a command-line.
```

Grub legacy

- ★ Se edita el fichero `/boot/menu.lst` o `/boot/grub/grub.conf` y se añade algo similar a:

```
title Fedora (2.6.32)
root (hd0,0)
kernel /vmlinuz-2.6.32 ro root=/dev/sda5
initrd /initrd-2.6.32
```

Grub 2

- ★ Se edita el fichero `/etc/grub.d/40_custom file` y se añade algo similar a:

```
menuentry "Fedora (2.6.35.4)" {  
  set root=(hd0,1)  
  linux /vmlinuz-2.6.35.4 ro root=/dev/sda5  
  initrd /initrd-2.6.35.4  
}
```

- ★ Para finalizar ejecutamos `update-grub` o `grub-mkconfig`