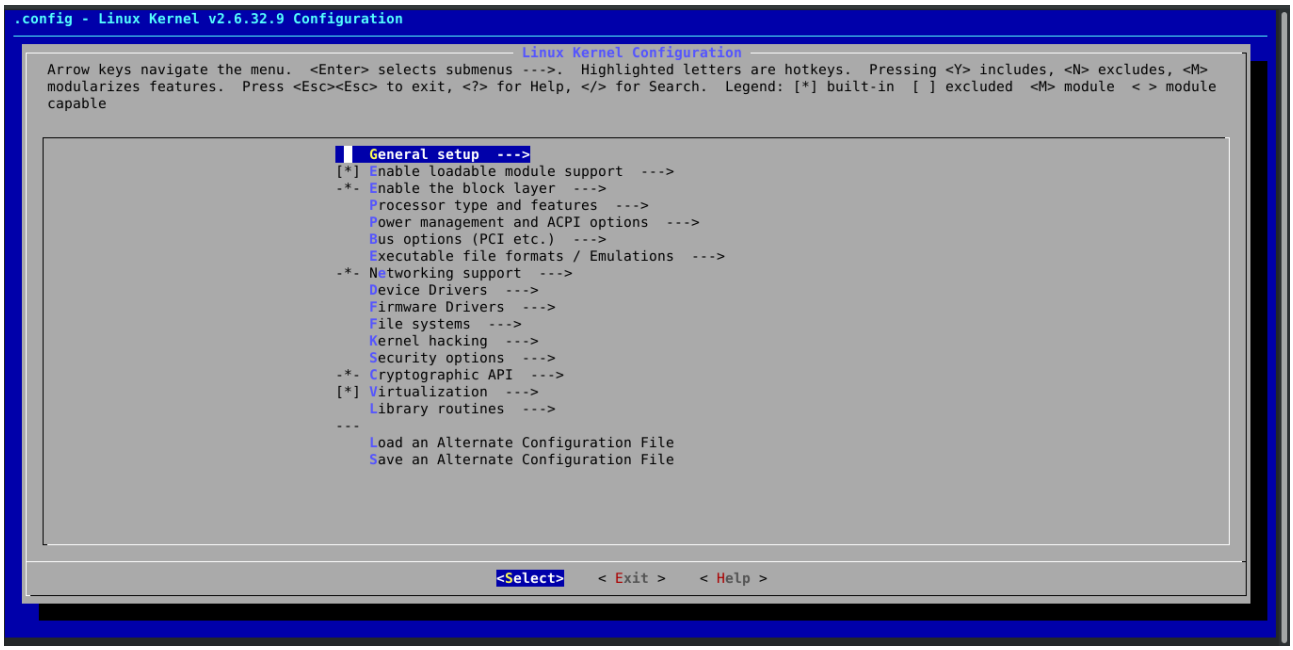
Memoria Práctica 2

Administración de Servidores

Configuración, compilación y gestión del kernel Linux y de sus módulos



Almudena García Jurado-Centurión

Índice

0. Introducción

El kernel es la parte mas importante de cualquier sistema operativo, la que se encarga de comunicar el sistema con el hardware.

Aprender a configurarlo e instalarlo es una tarea esencial en cualquier administrador de sistema.

En esta práctica aprenderemos a compilar un nuevo kernel e instalarlo, partiendo de un fichero de configuración ya existente y funcional.
También aprenderemos a configurar el bootloader para que arranque con el nuevo kernel.

También tocaremos temas relacionados con la gestión de módulos del kernel, aprendiendo a activar y desactivar módulos, y viendo el efecto que tiene la retirada de uno de ellos sobre el soporte de hardware.

Y, ya de paso, aprenderemos a parchear un fichero de código.

1. Parcheando un programa

En este apartado tenemos que desarrollar un parche para un programa ya existente, y aplicarlo.

El código original es el siguiente:

```
#!/bin/bash

#####################################################################
# Calculo del factorial de un numero en bash, utilizando while #
#                                                 Manuel Ramirez Panatt #
#####################################################################

aux=1
# presentacion
echo "Este programa calcula el factorial de  un numero"
echo " "

# parametros
echo -e "Ingrese el numero \c"
read i
if [ $i -ge 0 ]
 then
 while [ $i -gt 0 ]
 do
 aux=`\expr $aux \* $i`
 i=`\expr $i - 1`
 done
 echo "Su factorial es $aux"
else
 echo "No existe el factorial de un numero negativo"
fi
```

Tenemos que desarrollar un parche que repare los errores ortográficos del script y devuelva un error si el número introducido es superior a 20.

Para comprobar que el código inicial sea correcto, ejecutamos el programa. Como no tiene permisos de ejecución, los asignamos mediante *chmod +x*, tras lo cual lo ejecutamos.

```
[almu@debian ~]$ chmod +x Descargas/factorial.sh
[almu@debian ~]$ Descargas/factorial.sh
Este programa calcula el factorial de  un numero

Ingrese el numero 4
Su factorial es 24
[almu@debian ~]$
```

Vemos que el código funciona correctamente, así que podemos proceder al parcheo.

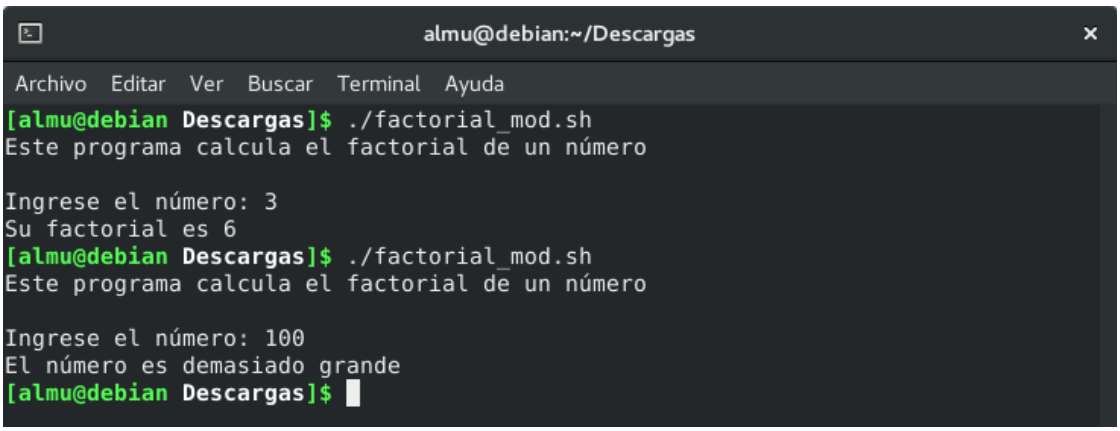
Para poder parchear el script, debemos generar un nuevo fichero y hacer los cambios sobre él, así que copiamos el fichero con nombre “factorial_mod.sh”

```
[almu@debian ~]$ cp Descargas/factorial.sh Descargas/factorial_mod.sh
```

Tras aplicar las modificaciones, el código queda así:

```
#!/bin/bash

#####################################################################
# Calculo del factorial de un numero en bash, utilizando while #
#                                                 Manuel Ramirez Panatt #
#####################################################################

aux=1
# presentacion
echo "Este programa calcula el factorial de un número"
echo " "

# parametros
echo -e "Ingrese el número: \c"
read i
if [ $i -gt 20 ]; then
        echo "El número es demasiado grande"
elif [ $i -ge 0 ]
 then
 while [ $i -gt 0 ]
 do
 aux=`\expr $aux \* $i`
 i=`\expr $i - 1`
 done
 echo "Su factorial es $aux"
else
 echo "No existe el factorial de un número negativo"
fi
```

Comprobamos que funciona ejecutándolo en la terminal:



El programa modificado funciona correctamente, así que pasamos a generar el parche para el original.

Para generar el parche, usamos el comando diff, que nos mostrará las diferencias entre un programa y otro, redirigiendo su salida a un fichero, el cual será el nuevo parche.

```
[almu@debian Descargas]$ diff -u factorial.sh factorial_mod.sh >> patch.diff
[almu@debian Descargas]$
```

El primer parámetro se corresponde al fichero original, y el segundo al fichero nuevo.

Una vez generado el parche, pasamos a aplicarlo sobre el programa. Para ello, usamos el comando *patch,* indicándole el programa a parchear, indicándole con -i el parche y con -o el nombre del programa parcheado.

```
[almu@debian Descargas]$ patch factorial.sh -i patch.diff -o factorial_patched.s
h
patching file factorial_patched.sh (read from factorial.sh)
[almu@debian Descargas]$
```

La salida del comando nos indica que el programa ha sido parcheado.

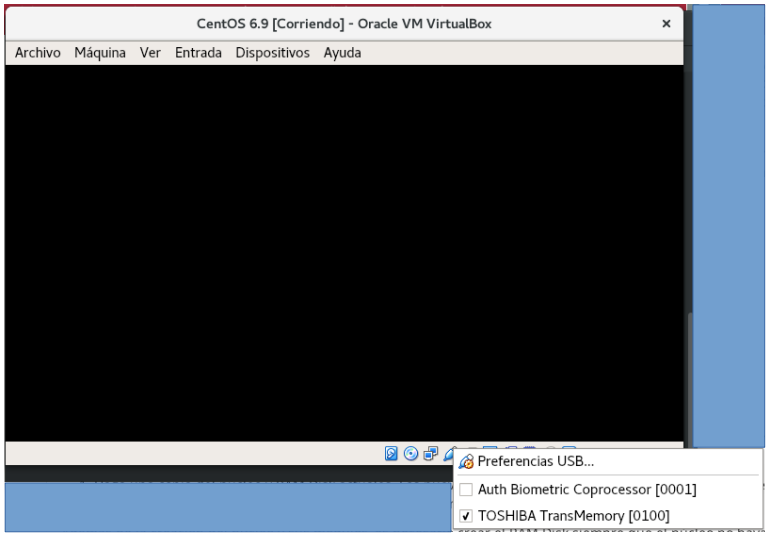
Tras ejecutar el programa parcheado, comprobamos que el parche aplicado es correcto, y el programa funciona adecuadamente.

```
[almu@debian Descargas]$ chmod +x factorial_patched.sh
[almu@debian Descargas]$ ./factorial_patched.sh
Este programa calcula el factorial de un número

Ingrese el número: 45
El número es demasiado grande
[almu@debian Descargas]$ ./factorial_patched.sh
Este programa calcula el factorial de un número

Ingrese el número: 5
Su factorial es 120
[almu@debian Descargas]$
```

2. Examinando el hardware

En este paso examinaremos el hardware detectado por la máquina virtual
Para ello, uno de los comandos disponibles es *lsusb,* que nos indica los dispositivos USB que están conectados en nuestra máquina, comparando su salida antes y después de conectar un pendrive.

Para conectar el pendrive, hacemos clic derecho en el cuarto icono de la máquina virtual y marcamos el dispositivo a conectar.



Ejecutamos el comando dos veces, una antes de conectar el pendrive y otra después, redirigiendo la salida a sendos ficheros, y comparamos el resultado.

```
[root@localhost ~]# lsusb > lsusb_inicial
[root@localhost ~]# lsusb > lsusb_final
[root@localhost ~]# more lsusb_
lsusb_final    lsusb_inicial  lsusb_inicio
[root@localhost ~]# more lsusb_inici
lsusb_inici: No existe el fichero o el directorio
[root@localhost ~]# more lsusb_inicial
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
[root@localhost ~]# more lsusb_final
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 0930:6544 Toshiba Corp. TransMemory-Mini / Kingston DataT
raveler 2.0 Stick
[root@localhost ~]#
```

Tras comparar, vemos que en la segunda llamada ha aparecido un nuevo dispositivo, correspondiente al pendrive que hemos conectado.

3. Compilando el kernel

Una vez examinado el hardware, pasamos a la compilación de un nuevo kernel. Para ello, vamos a descargar los fuentes del kernel de la pagina oficial kernel.org. Para no tener problemas posteriormente, debemos buscar una versión cercana a la que tenemos instalada.

Para saber que versión tenemos instalada, usamos el comando *uname -a*, que nos dará todos los datos sobre la versión del kernel que tenemos instalada.

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.32-696.13.2.el6.i686 #1 SMP Thu Oct 5 20:42:25 U
TC 2017 i686 i686 i386 GNU/Linux
[root@localhost ~]#
```

Tras ejecutar el comando, obtenemos que nuestra versión es la 2.6.32
Para descargar el kernel de esta versión, nos vamos a https://www.kernel.org/pub/linux/kernel/v2.6/.

En nuestro caso, seleccionamos el último parche de esta, el 2.6.32.9
Los fuentes de este kernel se obtienen desde:
https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.9.tar.gz

Para comprobar la autenticidad de estos fuentes, debemos verificar su firma, para lo cual tenemos que descargar el fichero de firma de esta versión desde:
https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.9.tar.sign

Una vez obtenidos los enlaces, debemos descargar los ficheros. Para eso usaremos el comando *wget*, el cual no viene en la instalación por defecto.
En este caso, *wget* ya había sido sido instalado previamente, por lo que no necesitamos instalarlo.

```
[root@localhost ~]# yum install wget
Complementos cargados:fastestmirror
Configurando el proceso de instalación
Loading mirror speeds from cached hostfile
 * base: centos.cadt.com
 * extras: mirror.airenetworks.es
 * updates: mirror.airenetworks.es
base                                                     | 3.7 kB     00:00
extras                                                   | 3.3 kB     00:00
updates                                                  | 3.4 kB     00:00
El paquete wget-1.12-10.el6.i686 ya se encuentra instalado con su versión más re
ciente
Nada para hacer
[root@localhost ~]#
```

Procedemos pues a descargar los fuentes del kernel y la firma, usando el comando *wget* seguido del enlace. Nos situamos en el directorio */usr/src* y descargamos.

```
[root@localhost ~]# cd /usr/src
[root@localhost src]# wget https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.9.tar.gz
--2017-10-25 20:52:27--  https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.9.tar.gz
Resolviendo www.kernel.org... 147.75.205.195, 2604:1380:2000:f000::7
Connecting to www.kernel.org|147.75.205.195|:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 81922503 (78M) [application/x-gzip]
Saving to: `linux-2.6.32.9.tar.gz.1'

100%[======================================>] 81.922.503  1,30M/s   in 94s

2017-10-25 20:54:01 (852 KB/s) - `linux-2.6.32.9.tar.gz.1' saved [81922503/81922503]

[root@localhost src]# wget https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.9.tar.sign
--2017-10-25 20:54:10--  https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.9.tar.sign
Resolviendo www.kernel.org... 147.75.205.195, 2604:1380:2000:f000::7
Connecting to www.kernel.org|147.75.205.195|:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 665 [application/pgp-signature]
Saving to: `linux-2.6.32.9.tar.sign.1'

100%[======================================>] 665         --.-K/s   in 0,001s

2017-10-25 20:54:10 (743 KB/s) - `linux-2.6.32.9.tar.sign.1' saved [665/665]

[root@localhost src]#
```

Una vez descargados, descomprimimos el archivador usando el comando *gzip*, y los verificamos usando gpg. Para validar la firma usaremos *gpg --verify* seguido del nombre del fichero firma (terminado en tar.sign)

En el primer intento, nos dirá que no encuentra la clave pública, así que la importamos con *gpg --recv-keys* seguido del RSA ID devuelto por el anterior comando. Este comando nos descargará la firma del servidor, tras lo cual podremos comprobar la validez del fichero ejecutando de nuevo *gpg --verify*

```
[root@localhost src]# ls
debug  kernels  linux-2.6.32.9.tar.gz  linux-2.6.32.9.tar.sign
[root@localhost src]# gzip -d linux-2.6.32.9.tar.gz
[root@localhost src]# gpg --verify linux-2.6.32.9.tar.sign
gpg: Firmado el jue 08 ago 2013 21:37:10 CEST usando clave RSA ID C4790F9D
gpg: Imposible comprobar la firma: No public key
[root@localhost src]# gpg --recv-keys C4790F9D
gpg: anillo `/root/.gnupg/secring.gpg' creado
gpg: solicitando clave C4790F9D de hkp servidor keys.gnupg.net
gpg: /root/.gnupg/trustdb.gpg: se ha creado base de datos de confianza
gpg: clave C4790F9D: clave pública "Linux Kernel Archives Verification Key (One-off resigning of old releases) <ftpadmin@kernel.org>" importada
gpg: no se encuentran claves absolutamente fiables
gpg: Cantidad total procesada: 1
gpg:               importadas: 1  (RSA: 1)
[root@localhost src]# gpg --verify linux-2.6.32.9.tar.sign
gpg: Firmado el jue 08 ago 2013 21:37:10 CEST usando clave RSA ID C4790F9D
gpg: Firma correcta de "Linux Kernel Archives Verification Key (One-off resigning of old releases) <ftpadmin@kernel.org>"
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: BFA7 DD3E 0D42 1C9D B6AB  6527 0D3B 3537 C479 0F9D
[root@localhost src]#
```

El comando nos indica que la firma es correcta, así que pasamos a extraer los fuentes usando el comando *tar*.

```
[root@localhost src]# tar xvf linux-2.6.32.9.tar
```

```
linux-2.6.32.9/tools/perf/util/svghelper.h
linux-2.6.32.9/tools/perf/util/symbol.c
linux-2.6.32.9/tools/perf/util/symbol.h
linux-2.6.32.9/tools/perf/util/thread.c
linux-2.6.32.9/tools/perf/util/thread.h
linux-2.6.32.9/tools/perf/util/trace-event-info.c
linux-2.6.32.9/tools/perf/util/trace-event-parse.c
linux-2.6.32.9/tools/perf/util/trace-event-read.c
linux-2.6.32.9/tools/perf/util/trace-event.h
linux-2.6.32.9/tools/perf/util/types.h
linux-2.6.32.9/tools/perf/util/usage.c
linux-2.6.32.9/tools/perf/util/util.h
linux-2.6.32.9/tools/perf/util/values.c
linux-2.6.32.9/tools/perf/util/values.h
linux-2.6.32.9/tools/perf/util/wrapper.c
linux-2.6.32.9/usr/
linux-2.6.32.9/usr/.gitignore
linux-2.6.32.9/usr/Kconfig
linux-2.6.32.9/usr/Makefile
linux-2.6.32.9/usr/gen_init_cpio.c
linux-2.6.32.9/usr/initramfs_data.S
linux-2.6.32.9/usr/initramfs_data.bz2.S
linux-2.6.32.9/usr/initramfs_data.gz.S
linux-2.6.32.9/usr/initramfs_data.lzma.S
linux-2.6.32.9/virt/
linux-2.6.32.9/virt/kvm/
linux-2.6.32.9/virt/kvm/Kconfig
linux-2.6.32.9/virt/kvm/coalesced_mmio.c
linux-2.6.32.9/virt/kvm/coalesced_mmio.h
linux-2.6.32.9/virt/kvm/eventfd.c
linux-2.6.32.9/virt/kvm/ioapic.c
linux-2.6.32.9/virt/kvm/ioapic.h
linux-2.6.32.9/virt/kvm/iodev.h
linux-2.6.32.9/virt/kvm/iommu.c
linux-2.6.32.9/virt/kvm/irq_comm.c
linux-2.6.32.9/virt/kvm/kvm_main.c
[root@localhost src]#
```

Para que el código pueda ser compilado, es necesario que exista un enlace simbólico con el nombre *linux* que apunte al directorio de los fuentes, así que lo creamos con *ln -s*

```
[root@localhost src]# ln -s linux-2.6.32.9 linux
[root@localhost src]# ls
debug    linux            linux-2.6.32.9.tar
kernels  linux-2.6.32.9   linux-2.6.32.9.tar.sign
[root@localhost src]#
```

3.1. Configurando el kernel

Para compilar el kernel, antes debemos configurarlo. El proceso de configuración puede ser muy tedioso, por lo que se recomienda partir de un fichero de configuración de un kernel ya funcional.

Esta configuración está guardada en el directorio */boot* , en un fichero con nombre *.config-[version].[arquitectura].*

```
[root@localhost src]# ls /boot/
config-2.6.32-696.13.2.el6.i686          symvers-2.6.32-696.13.2.el6.i686.gz
config-2.6.32-696.el6.i686               symvers-2.6.32-696.el6.i686.gz
efi                                      System.map-2.6.32-696.13.2.el6.i686
grub                                     System.map-2.6.32-696.el6.i686
initramfs-2.6.32-696.13.2.el6.i686.img   vmlinuz-2.6.32-696.13.2.el6.i686
initramfs-2.6.32-696.el6.i686.img        vmlinuz-2.6.32-696.el6.i686
lost+found
```

En este caso, vemos que tenemos 2 ficheros de configuración en el directorio */boot*, uno de la versión **2.6.32-696.13.2**, y otro de la **2.6.32-696.** Revisando el anterior *uname -a,* vemos que la versión en uso es la primera, así que tomaremos esa como base.

Para ello, copiamos el fichero *config-2.6.32-696.13.2.el6.i686* en */usr/src/linux* con el nombre *.config*

```
[root@localhost src]# cp /boot/config-2.6.32-696.13.2.el6.i686 ./linux/.config
[root@localhost src]# ls linux
arch     crypto         fs       Kbuild       Makefile  REPORTING-BUGS  sound
block    Documentation  include  kernel       mm        samples         tools
COPYING  drivers        init     lib          net       scripts         usr
CREDITS  firmware       ipc      MAINTAINERS  README    security        virt
[root@localhost src]# ls -a linux
.        CREDITS        .gitignore  lib          README          tools
..       crypto         include     .mailmap     REPORTING-BUGS  usr
arch     Documentation  init        MAINTAINERS  samples         virt
block    drivers        ipc         Makefile     scripts
.config  firmware       Kbuild      mm           security
COPYING  fs             kernel      net          sound
[root@localhost src]#
```

Antes de ello, ejecutamos *make mrproper*, que se encarga de eliminar los ficheros objeto antiguos que pudieran quedar en el directorio.

Una vez copiado el fichero de configuración, actualizamos el fichero usando *make silentoldconfig*

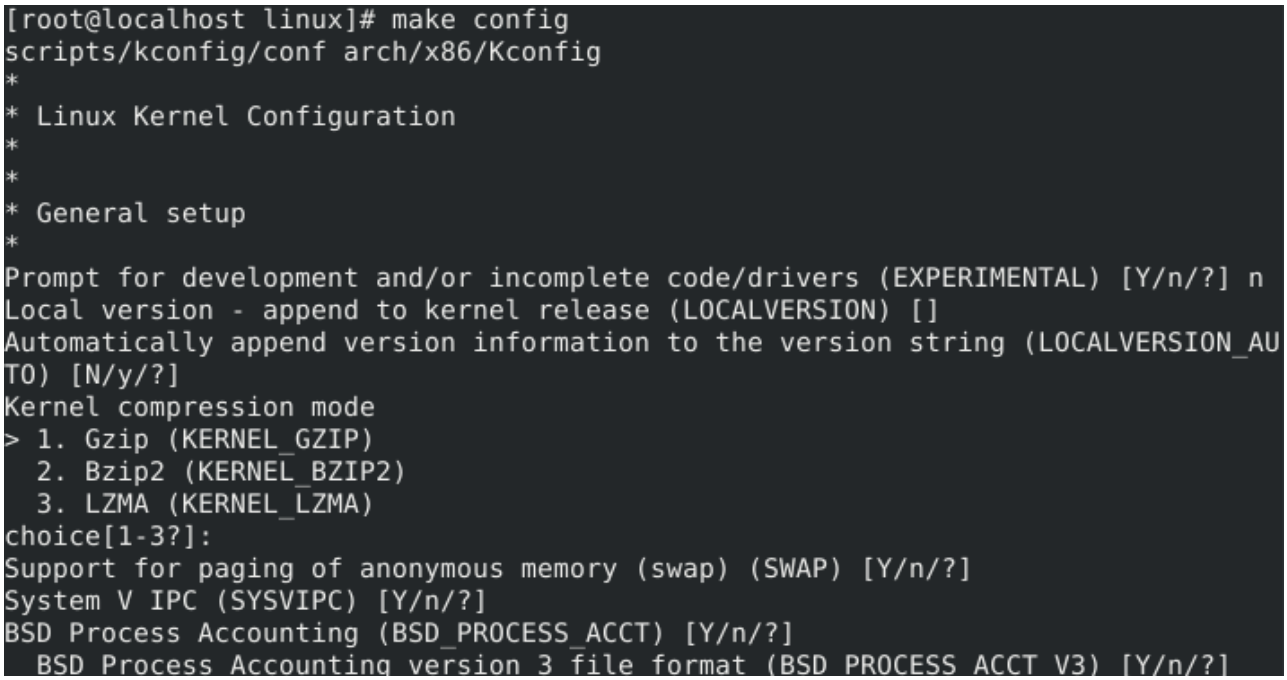
```
  Secure RPC: Kerberos V mechanism (EXPERIMENTAL) (RPCSEC_GSS_KRB5) [M/?] m
  Secure RPC: SPKM3 mechanism (EXPERIMENTAL) (RPCSEC_GSS_SPKM3) [M/n/?] m
  SMB file system support (OBSOLETE, please use CIFS) (SMB_FS) [N/m/y/?] n
  CIFS support (advanced network filesystem, SMBFS successor) (CIFS) [M/n/y/?]
m
    CIFS statistics (CIFS_STATS) [Y/n/?] y
      Extended statistics (CIFS_STATS2) [N/y/?] n
    Support legacy servers which use weaker LANMAN security (CIFS_WEAK_PW_HASH)
 [Y/n/?] y
    Kerberos/SPNEGO advanced session setup (CIFS_UPCALL) [Y/n/?] y
    CIFS extended attributes (CIFS_XATTR) [Y/n/?] y
      CIFS POSIX Extensions (CIFS_POSIX) [Y/n/?] y
    Enable additional CIFS debugging routines (CIFS_DEBUG2) [N/y/?] n
    DFS feature support (CIFS_DFS_UPCALL) [Y/n/?] y
    CIFS Experimental Features (EXPERIMENTAL) (CIFS_EXPERIMENTAL) [N/y/?] (NEW)

  NCP file system support (to mount NetWare volumes) (NCP_FS) [N/m/y/?] n
  Coda file system support (advanced network fs) (CODA_FS) [N/m/y/?] n
  Andrew File System support (AFS) (EXPERIMENTAL) (AFS_FS) [N/m/y/?] n
  Plan 9 Resource Sharing Support (9P2000) (Experimental) (9P_FS) [N/m/?] n
#
# configuration written to .config
```

Aceptamos todas las sugerencias, lo cual actualizará nuestro fichero de configuración a la nueva versión del kernel.

Si queremos modificar el fichero de configuración, podemos usar las herramientas *config, menuconfig o xconfig.* Estas herramientas nos permiten activar o desactivar opciones en el kernel y, dado el caso, seleccionar si queremos integrar los componentes o ponerlos como módulos.

- *config* ofrece una interfaz en modo texto, la cual va preguntando sobre cada opción y componente del kernel, dándonos a elegir si queremos activarla, si no, y si queremos incluirlo como módulo o integrado en el kernel.

  Para ejecutarlo, usamos *make config,* lo cual nos mostrará algo similar a esto:

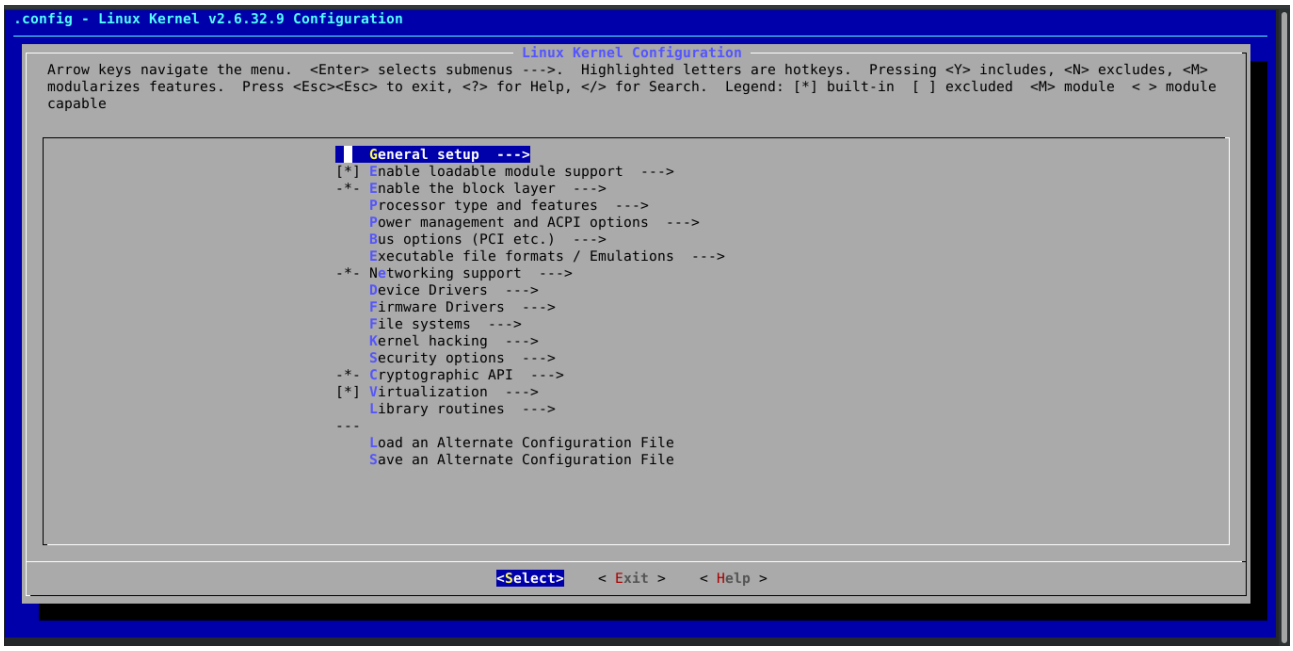


- *menuconfig* es otra herramienta de configuración, con una interfaz de menús basada en ncurses, la cual nos permite realizar la configuración de forma mucho mas cómoda, mostrándonos la configuración organizada en diferentes secciones y grupos.

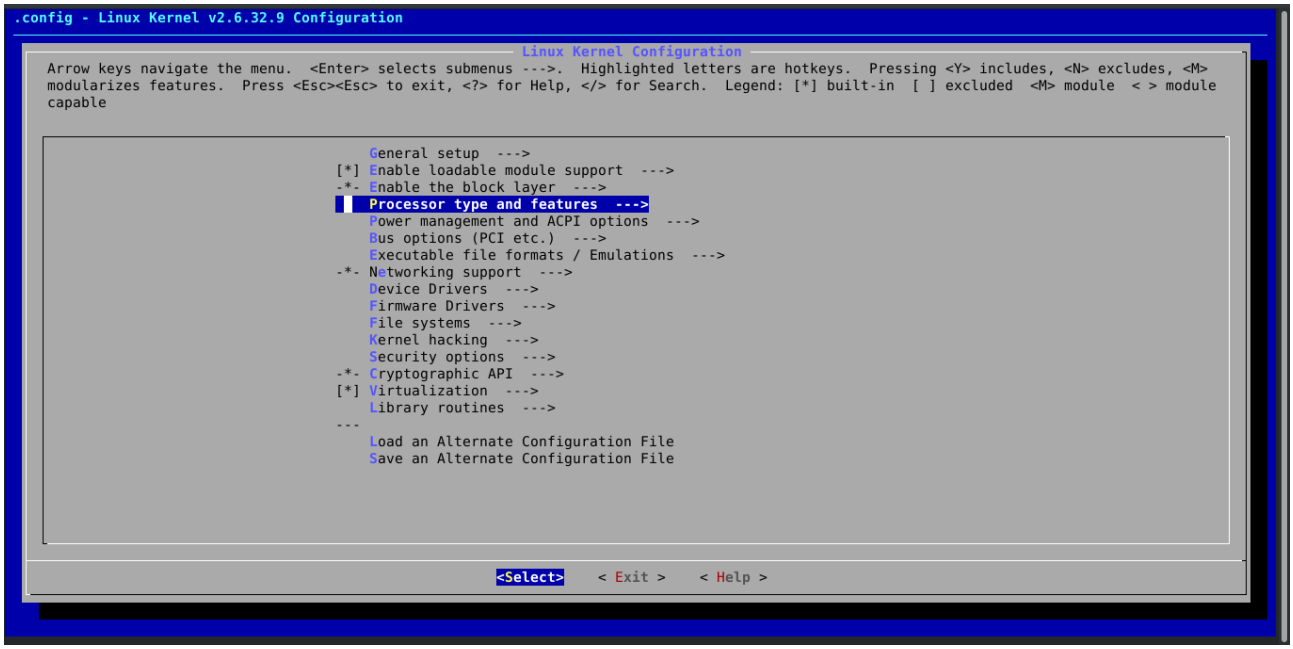
  Para poder ejecutarlo, debemos instalar *ncurses-devel*, así que lo instalamos con *yum.*

```
[root@localhost linux]# yum install ncurses-devel
Complementos cargados:fastestmirror
Configurando el proceso de instalación
Loading mirror speeds from cached hostfile
 * base: mirror.airenetworks.es
 * extras: mirror.airenetworks.es
 * updates: mirror.airenetworks.es
Resolviendo dependencias
--> Ejecutando prueba de transacción
---> Package ncurses-devel.i686 0:5.7-4.20090207.el6 will be instalado
--> Resolución de dependencias finalizada

Dependencias resueltas

================================================================================
 Paquete              Arquitectura
                           Versión                   Repositorio Tamaño
================================================================================
Instalando:
 ncurses-devel        i686        5.7-4.20090207.el6         base        641 k

Resumen de la transacción
================================================================================
Instalar       1 Paquete(s)

Tamaño total de la descarga: 641 k
Tamaño instalado: 1.7 M
Está de acuerdo [s/N]:s
Descargando paquetes:
ncurses-devel-5.7-4.20090207.el6.i686.rpm                                                                | 641 kB     00:00
Ejecutando el rpm_check_debug
Ejecutando prueba de transacción
La prueba de transacción ha sido exitosa
Ejecutando transacción
  Instalando     : ncurses-devel-5.7-4.20090207.el6.i686                                                                  1/1
  Verifying      : ncurses-devel-5.7-4.20090207.el6.i686                                                                  1/1
```

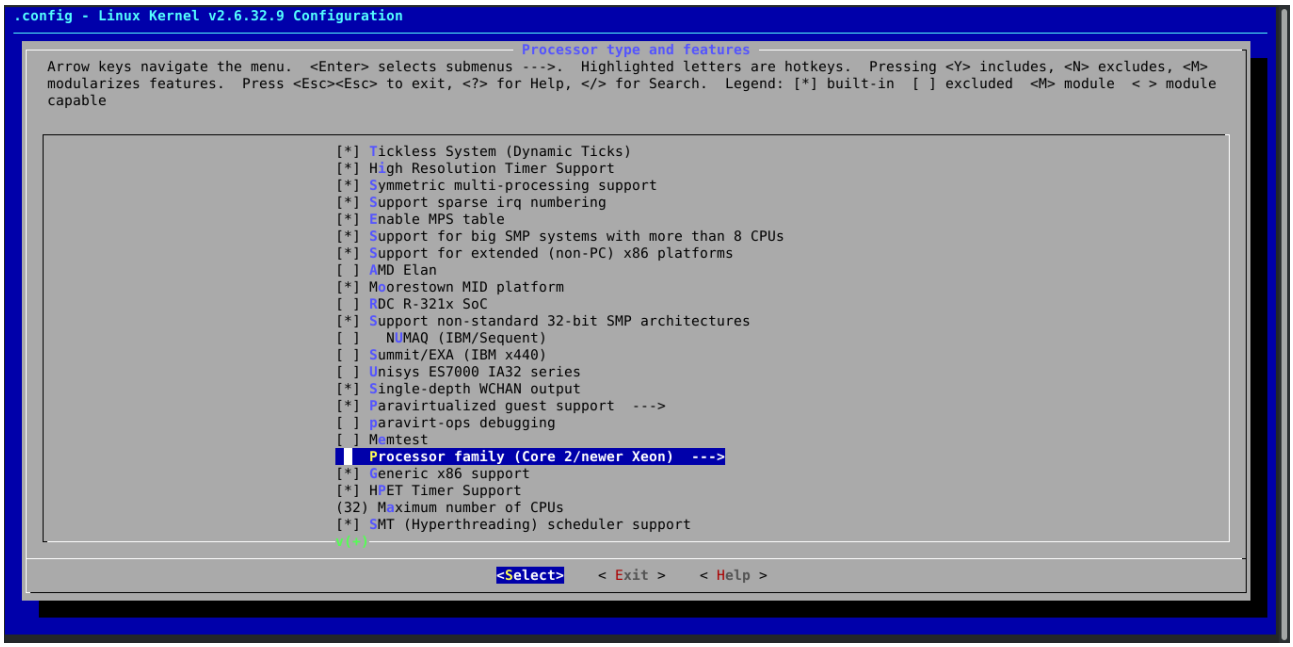
Una vez instalado ncurses-devel, abrimos menuconfig mediante *make menuconfig.* Nos aparecerá una interfaz similar a esta:



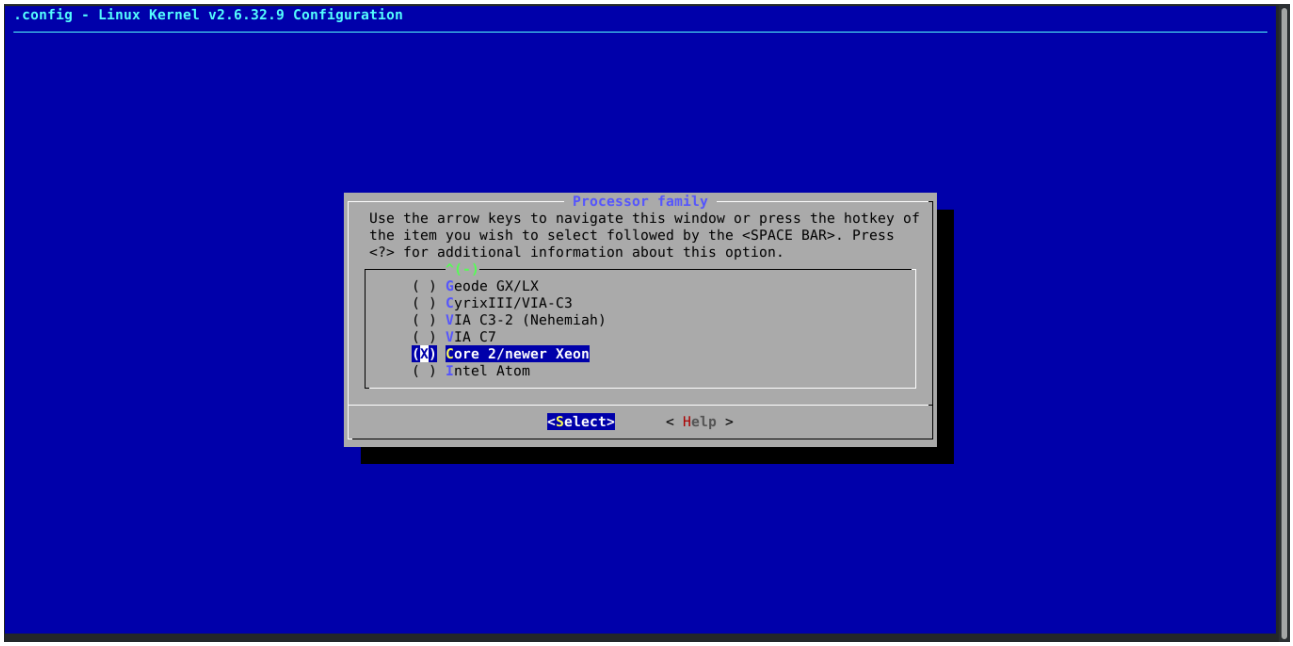
Si queremos cambiar la configuración del procesador, nos vamos a la sección “Processor type and features”, y pulsamos enter



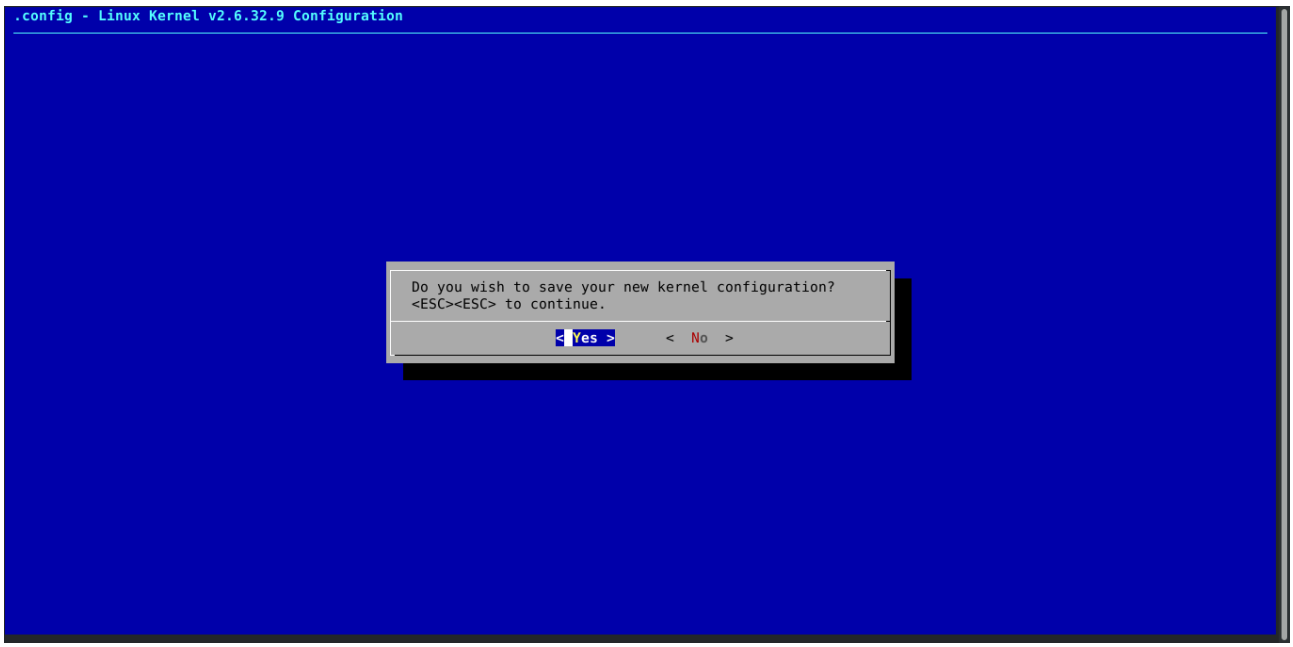
Podemos seleccionar otra familia de procesador, pulsando en “Processor family”. Esto nos permite usar un juego de instrucciones mas moderno y adecuado a nuestro procesador



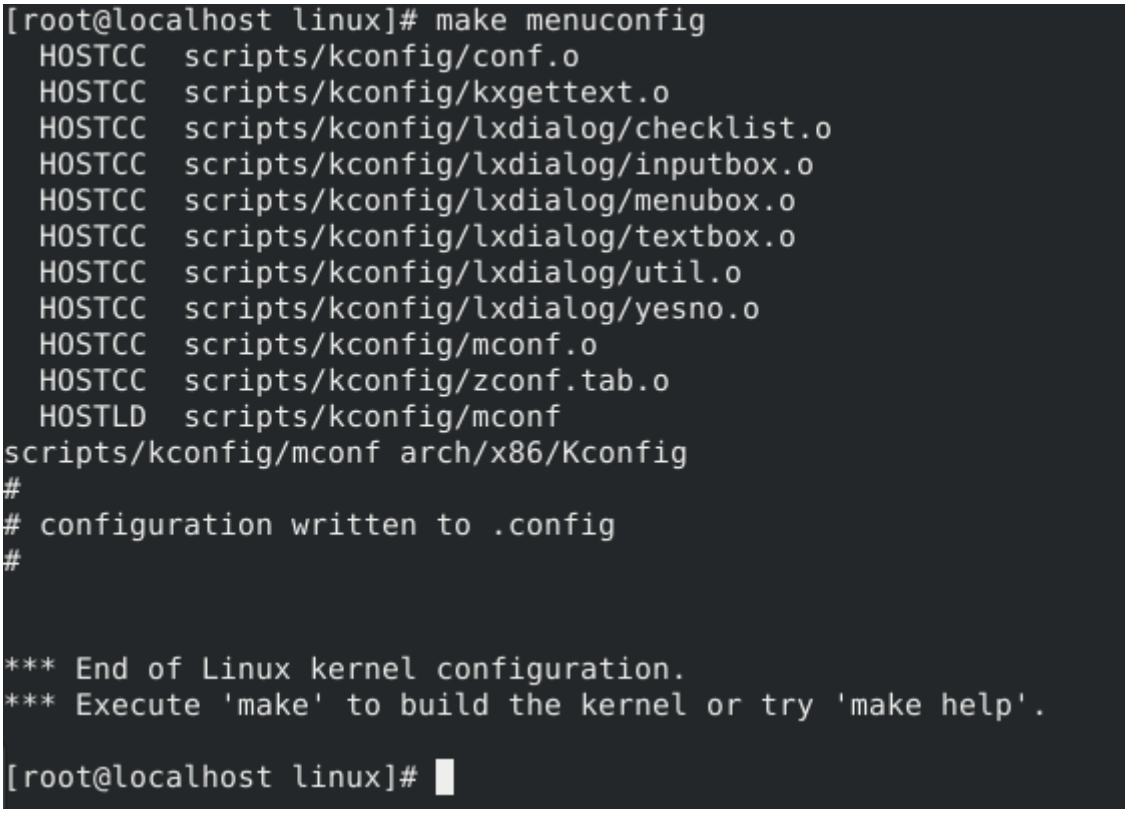
En éste caso, vamos a seleccionar “Core 2/newer Xeon”. Pulsamos enter para seleccionar la familia y volver al menú.



Pulsamos en exit hasta salir del menú. Antes de salir, menuconfig nos consultará si queremos guardar nuestra configuración. Pulsamos en “Yes” para guardar los cambios.



Una vez terminada la configuración, el configurador nos avisará de que los cambios se han aplicado, y que podemos proceder a la compilación.



3.2. Compilando el kernel ya configurado

Para compilar el kernel usaremos *make*. Para ahorrar tiempo, y dado que la máquina virtual tiene una CPU con varios núcleos, usaremos el argumento *-jX*, siendo X el número de núcleos del procesador + 1.

En nuestro caso, como tenemos 2 núcleos, usaremos *make -j3*

```
[root@localhost linux]# make -j3
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/x86/Kconfig
  CHK     include/linux/version.h
  UPD     include/linux/version.h
  CHK     include/linux/utsrelease.h
  UPD     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-x86
  HOSTCC  scripts/genksyms/genksyms.o
  HOSTCC  scripts/selinux/mdp/mdp
  CC      scripts/mod/empty.o
  HOSTCC  scripts/mod/mk_elfconfig
  SHIPPED scripts/genksyms/lex.c
```

```
  CC      arch/x86/kernel/apic/io_apic.o
  LD      arch/x86/kernel/acpi/realmode/wakeup.elf
  OBJCOPY arch/x86/kernel/acpi/realmode/wakeup.bin
  CC      arch/x86/kernel/acpi/cstate.o
  CC      arch/x86/kernel/apic/bigsmp_32.o
  CC      arch/x86/kernel/acpi/processor.o
  AS      arch/x86/kernel/acpi/wakeup_rm.o
  CC      arch/x86/kernel/cpu/intel_cacheinfo.o
  LD      arch/x86/kernel/acpi/built-in.o
  CC      arch/x86/kernel/sfi.o
  CC      arch/x86/kernel/cpu/addon_cpuid_features.o
  CC      arch/x86/kernel/reboot.o
  CC      arch/x86/kernel/cpu/proc.o
  LD      arch/x86/kernel/apic/built-in.o
  MKCAP   arch/x86/kernel/cpu/capflags.c
/bin/sh: perl: no se encontró la orden
make[3]: *** [arch/x86/kernel/cpu/capflags.c] Error 127
make[3]: *** Se espera a que terminen otras tareas....
  CC      arch/x86/kernel/msr.o
make[3]: *** wait: No hay ningún proceso hijo.  Alto.
make[2]: *** [arch/x86/kernel/cpu] Error 2
make[2]: *** Se espera a que terminen otras tareas....
make[1]: *** [arch/x86/kernel] Error 2
make: *** [arch/x86] Error 2
```

make nos avisa de que el interprete de perl no esta instalado, así que lo instalamos con *yum install perl*

```
--> Procesando dependencias: perl(version) para el paquete: 4:perl-5.10.1-144.el6.i686
--> Procesando dependencias: perl(Pod::Simple) para el paquete: 4:perl-5.10.1-144.el6.i686
--> Procesando dependencias: perl(Module::Pluggable) para el paquete: 4:perl-5.10.1-144.el6.i686
--> Procesando dependencias: libperl.so para el paquete: 4:perl-5.10.1-144.el6.i686
--> Ejecutando prueba de transacción
---> Package perl-Module-Pluggable.i686 1:3.90-144.el6 will be instalado
---> Package perl-Pod-Simple.i686 1:3.13-144.el6 will be instalado
--> Procesando dependencias: perl(Pod::Escapes) >= 1.04 para el paquete: 1:perl-Pod-Simple-3.13-144.el6.i686
---> Package perl-libs.i686 4:5.10.1-144.el6 will be instalado
---> Package perl-version.i686 3:0.77-144.el6 will be instalado
--> Ejecutando prueba de transacción
---> Package perl-Pod-Escapes.i686 1:1.04-144.el6 will be instalado
--> Resolución de dependencias finalizada

Dependencias resueltas

================================================================================
 Paquete                    Arquitectura
                                  Versión               Repositorio
                                                                      Tamaño
================================================================================
Instalando:
 perl                       i686       4:5.10.1-144.el6       base       9.7 M
Instalando para las dependencias:
 perl-Module-Pluggable      i686       1:3.90-144.el6         base        41 k
 perl-Pod-Escapes           i686       1:1.04-144.el6         base        33 k
 perl-Pod-Simple            i686       1:3.13-144.el6         base       213 k
 perl-libs                  i686       4:5.10.1-144.el6       base       594 k
 perl-version               i686       3:0.77-144.el6         base        52 k

Resumen de la transacción
================================================================================
Instalar       6 Paquete(s)

Tamaño total de la descarga: 11 M
Tamaño instalado: 30 M
Está de acuerdo [s/N]:
```

Una vez instalado perl, la compilación comienza con normalidad

```
  CC      arch/x86/kernel/cpu/powerflags.o
  CC      arch/x86/kernel/cpu/common.o
  CC      kernel/resource.o
  CC      kernel/sysctl.o
  CC      arch/x86/kernel/cpu/vmware.o
  CC      arch/x86/kernel/cpu/hypervisor.o
  CC      mm/mempool.o
  CC      arch/x86/kernel/cpu/sched.o
  CC      arch/x86/kernel/cpu/bugs.o
  CC      kernel/capability.o
  CC      arch/x86/kernel/cpu/cmpxchg.o
  CC      arch/x86/kernel/cpu/intel.o
  CC      mm/oom_kill.o
  CC      kernel/ptrace.o
  CC      arch/x86/kernel/cpu/amd.o
  CC      mm/fadvise.o
  CC      mm/maccess.o
  CC      kernel/timer.o
  CC      arch/x86/kernel/cpu/cyrix.o
  CC      mm/page_alloc.o
  CC      arch/x86/kernel/cpu/centaur.o
  CC      arch/x86/kernel/cpu/transmeta.o
  CC      arch/x86/kernel/cpu/umc.o
  CC      arch/x86/kernel/cpu/perf_event.o
  CC      kernel/user.o
  CC      kernel/signal.o
  CC      arch/x86/kernel/cpu/cpufreq/powernow-k7.o
  CC      mm/page-writeback.o
  CC      arch/x86/kernel/cpu/cpufreq/longrun.o
  CC      arch/x86/kernel/cpu/cpufreq/speedstep-ich.o
  CC      arch/x86/kernel/cpu/cpufreq/speedstep-lib.o
  CC      mm/readahead.o
  CC      arch/x86/kernel/cpu/cpufreq/speedstep-smi.o
  CC      kernel/sys.o
  CC [M]  arch/x86/kernel/cpu/cpufreq/powernow-k8.o
  CC      mm/swap.o
```

Vemos como, junto al resto de componentes, se van compilando los módulos (indicados con [M])

```
  CC      arch/x86/power/cpu.o
  CC [M]  drivers/gpu/drm/radeon/r600_blit.o
  CC [M]  drivers/hwmon/lm75.o
  CC      arch/x86/power/hibernate_32.o
  AS      arch/x86/power/hibernate_asm_32.o
  LD      arch/x86/power/built-in.o
  CC [M]  drivers/hwmon/lm77.o
  CC      arch/x86/video/fbdev.o
  CC [M]  drivers/hwmon/lm78.o
  CC [M]  drivers/gpu/drm/radeon/r600_blit_shaders.o
  LD      arch/x86/video/built-in.o
  CC      drivers/gpu/vga/vgaarb.o
  CC [M]  drivers/gpu/drm/radeon/r600_blit_kms.o
  CC [M]  drivers/hwmon/lm80.o
  CC [M]  drivers/hwmon/lm83.o
  CC [M]  drivers/hwmon/lm85.o
  LD      drivers/gpu/vga/built-in.o
  CC [M]  drivers/gpu/drm/radeon/radeon_pm.o
  CC [M]  drivers/hwmon/lm87.o
  LD      drivers/gpu/drm/savage/built-in.o
  CC [M]  drivers/gpu/drm/savage/savage_drv.o
  MKREGTABLE drivers/gpu/drm/radeon/r100_reg_safe.h
  MKREGTABLE drivers/gpu/drm/radeon/rn50_reg_safe.h
  MKREGTABLE drivers/gpu/drm/radeon/r300_reg_safe.h
  CC [M]  drivers/gpu/drm/radeon/rs600.o
  CC [M]  drivers/gpu/drm/savage/savage_bci.o
  CC [M]  drivers/hwmon/lm90.o
  CC [M]  drivers/gpu/drm/radeon/rv515.o
  CC [M]  drivers/gpu/drm/savage/savage_state.o
  CC [M]  drivers/hwmon/lm92.o
  CC [M]  drivers/hwmon/lm93.o
  CC [M]  drivers/hwmon/lm95241.o
  LD [M]  drivers/gpu/drm/savage/savage.o
  CC [M]  drivers/hwmon/ltc4215.o
  CC [M]  drivers/gpu/drm/radeon/r200.o
  CC [M]  drivers/hwmon/ltc4245.o
```

Después de un rato, la compilación finaliza:

```
  IHEX    firmware/ositech/Xilinx7OD.bin
  IHEX    firmware/korg/k1212.dsp
  IHEX    firmware/ess/maestro3_assp_kernel.fw
  IHEX    firmware/ess/maestro3_assp_minisrc.fw
  IHEX    firmware/tehuti/bdx.bin
  IHEX    firmware/tigon/tg3.bin
  IHEX    firmware/tigon/tg3_tso.bin
  IHEX    firmware/tigon/tg3_tso5.bin
  IHEX    firmware/3com/typhoon.bin
  IHEX2FW firmware/emi26/loader.fw
  IHEX2FW firmware/emi26/firmware.fw
  IHEX2FW firmware/emi26/bitstream.fw
  IHEX2FW firmware/emi62/loader.fw
  IHEX2FW firmware/emi62/bitstream.fw
  IHEX2FW firmware/emi62/spdif.fw
  IHEX2FW firmware/emi62/midi.fw
  IHEX    firmware/kaweth/trigger_code.bin
  IHEX    firmware/kaweth/new_code.bin
  IHEX    firmware/kaweth/new_code_fix.bin
  IHEX    firmware/kaweth/trigger_code_fix.bin
  IHEX    firmware/ti_3410.fw
  IHEX    firmware/ti_5052.fw
  IHEX    firmware/mts_cdma.fw
  IHEX    firmware/mts_gsm.fw
  IHEX    firmware/mts_edge.fw
  H16TOFW firmware/edgeport/boot.fw
  H16TOFW firmware/edgeport/boot2.fw
  H16TOFW firmware/edgeport/down.fw
  H16TOFW firmware/edgeport/down2.fw
  IHEX    firmware/edgeport/down3.bin
  IHEX2FW firmware/whiteheat_loader.fw
  IHEX2FW firmware/whiteheat.fw
  IHEX2FW firmware/keyspan_pda/keyspan_pda.fw
  H16TOFW firmware/matrox/g200_warp.fw
  IHEX2FW firmware/keyspan_pda/xircom_pgs.fw
  H16TOFW firmware/matrox/g400_warp.fw
[root@localhost linux]#
```

3.3. Instalando el kernel

Una vez compilado el kernel, debemos instalar los ficheros, copiándolos en su ubicación final.

Para instalar el kernel, podemos usar *make install*. Este comando copiará el fichero principal del kernel al directorio /boot

```
[root@localhost linux]# make install
sh /usr/src/linux-2.6.32.9/arch/x86/boot/install.sh 2.6.32.9 arch/x86/boot/bzImage \
                System.map "/boot"
ERROR: modinfo: could not find module nf_defrag_ipv6
[root@localhost linux]#
```

Hacemos un ls de /boot para verificar que el kernel se ha instalado correctamente.

```
[root@localhost linux]# ls /boot
config-2.6.32-696.13.2.el6.i686          initramfs-2.6.32-696.el6.i686.img    System.map                          vmlinuz-2.6.32-696.13.2.el6.i686
config-2.6.32-696.el6.i686               initramfs-2.6.32.9.img               System.map-2.6.32-696.13.2.el6.i686 vmlinuz-2.6.32-696.el6.i686
efi                                      lost+found                           System.map-2.6.32-696.el6.i686      vmlinuz-2.6.32.9
grub                                     symvers-2.6.32-696.13.2.el6.i686.gz  System.map-2.6.32.9
initramfs-2.6.32-696.13.2.el6.i686.img   symvers-2.6.32-696.el6.i686.gz       vmlinuz
[root@localhost linux]#
```

Vemos un fichero de nombre *vmlinuz-2.6.32.9*, correspondiente al kernel recién compilado.

Una vez instalado el kernel, debemos instalar sus módulos. Para ello, usamos la orden *make modules_install*. Los módulos se copiarán en */lib/modules*, en un directorio con nombre similar a la versión del kernel que acabamos de compilar.

```
[root@localhost linux]# make modules_install
  INSTALL Documentation/DocBook/procfs_example.ko
  INSTALL Documentation/connector/cn_test.ko
  INSTALL Documentation/filesystems/configfs/configfs_example_explicit.ko
  INSTALL Documentation/filesystems/configfs/configfs_example_macros.ko
  INSTALL arch/x86/crypto/aes-i586.ko
  INSTALL arch/x86/crypto/crc32c-intel.ko
  INSTALL arch/x86/crypto/salsa20-i586.ko
  INSTALL arch/x86/crypto/twofish-i586.ko
  INSTALL arch/x86/kernel/cpu/cpufreq/acpi-cpufreq.ko
  INSTALL arch/x86/kernel/cpu/cpufreq/p4-clockmod.ko
  INSTALL arch/x86/kernel/cpu/cpufreq/powernow-k8.ko
  INSTALL arch/x86/kernel/cpu/mcheck/mce-inject.ko
  INSTALL arch/x86/kernel/microcode.ko
  INSTALL arch/x86/kernel/test_nx.ko
  INSTALL arch/x86/oprofile/oprofile.ko
  INSTALL crypto/aes_generic.ko
  INSTALL crypto/ansi_cprng.ko
  INSTALL crypto/anubis.ko
  INSTALL crypto/arc4.ko
  INSTALL crypto/async_tx/async_memcpy.ko
  INSTALL crypto/async_tx/async_pq.ko
  INSTALL crypto/async_tx/async_raid6_recov.ko
  INSTALL crypto/async_tx/async_tx.ko
  INSTALL crypto/async_tx/async_xor.ko
  INSTALL crypto/async_tx/raid6test.ko
  INSTALL crypto/authenc.ko
  INSTALL crypto/blowfish.ko
  INSTALL crypto/camellia.ko
  INSTALL crypto/cast5.ko
  INSTALL crypto/cast6.ko
  INSTALL crypto/cbc.ko
  INSTALL crypto/ccm.ko
  INSTALL crypto/cryptd.ko
  INSTALL crypto/crypto_null.ko
  INSTALL crypto/ctr.ko
  INSTALL crypto/cts.ko
```

Comprobamos que los módulos se han instalado, haciendo un ls en /lib/modules

```
[root@localhost linux]# ls /lib/modules/
2.6.32-696.13.2.el6.i686  2.6.32-696.el6.i686  2.6.32.9
[root@localhost linux]#
```

Vemos que hay un directorio con nombre 2.6.32.9, correspondiente a la versión recién compilada

3.3.1. Anexo: Instalar el kernel manualmente

Aunque en muchas versiones del kernel, se puede usar *make install* para instalar el kernel, habitualmente este proceso ha de realizarse manualmente.

El fichero principal del kernel se aloja en el directorio */usr/src/linux/arch.*
Si nuestra arquitectura es i386, el kernel estará en *arch/x86/boot/bzImage*

Copiamos el fichero en /boot con el nombre de nuestra versión.

```
[root@localhost linux]# ls arch/x86/boot/
a20.c       bitops.h     compressed  cpu.o     install.sh  memory.c        pmjump.o  regs.o     tools      video-bios.c  video.o       voffset.h
a20.o       boot.h       copy.o      cpustr.h  main.c      memory.o        pmjump.S  setup.bin  tty.c      video-bios.o  video-vesa.c  zoffset.h
apm.c       bzImage      copy.S      edd.c     main.o      mkcpustr        pm.o      setup.elf  tty.o      video.c       video-vesa.o
apm.o       cmdline.c    cpu.c       edd.o     Makefile    mkcpustr.c      printf.c  setup.ld   version.c  video.h       video-vga.c
bioscall.o  cmdline.o    cpucheck.c  header.o  mca.c       mtools.conf.in  printf.o  string.c   version.o  video-mode.c  video-vga.o
bioscall.S  code16gcc.h  cpucheck.o  header.S  mca.o       pm.c            regs.c    string.o   vesa.h     video-mode.o  vmlinux.bin
[root@localhost linux]# ls arch/x86/boot/bzImage
arch/x86/boot/bzImage
[root@localhost linux]# file arch/x86/boot/bzImage
arch/x86/boot/bzImage: Linux kernel x86 boot executable bzImage, version 2.6.32.9 (root@localhost.locald, RO-rootFS, swap_dev 0x3, Normal VGA
[root@localhost linux]# ^C
[root@localhost linux]# cp arch/x86/boot/bzImage /boot/bzImage-2.6.32.9
[root@localhost linux]#
```

```
[root@localhost linux]# ls /boot
bzImage-2.6.32.9                initramfs-2.6.32-696.13.2.el6.i686.img  symvers-2.6.32-696.el6.i686.gz       vmlinuz
config-2.6.32-696.13.2.el6.i686  initramfs-2.6.32-696.el6.i686.img       System.map                           vmlinuz-2.6.32-696.13.2.el6.i686
config-2.6.32-696.el6.i686       initramfs-2.6.32.9.img                  System.map-2.6.32-696.13.2.el6.i686  vmlinuz-2.6.32-696.el6.i686
efi                              lost+found                              System.map-2.6.32-696.el6.i686       vmlinuz-2.6.32.9
grub                             symvers-2.6.32-696.13.2.el6.i686.gz     System.map-2.6.32.9
[root@localhost linux]#
```

Vemos que el fichero se ha copiado correctamente, con el nombre *bzImage-2.6.32.9*

3.4. Configurando el bootloader

Antes de arrancar el nuevo kernel, vamos a copiar nuestros actuales kernel y ramdisk con el prefijo .mio.

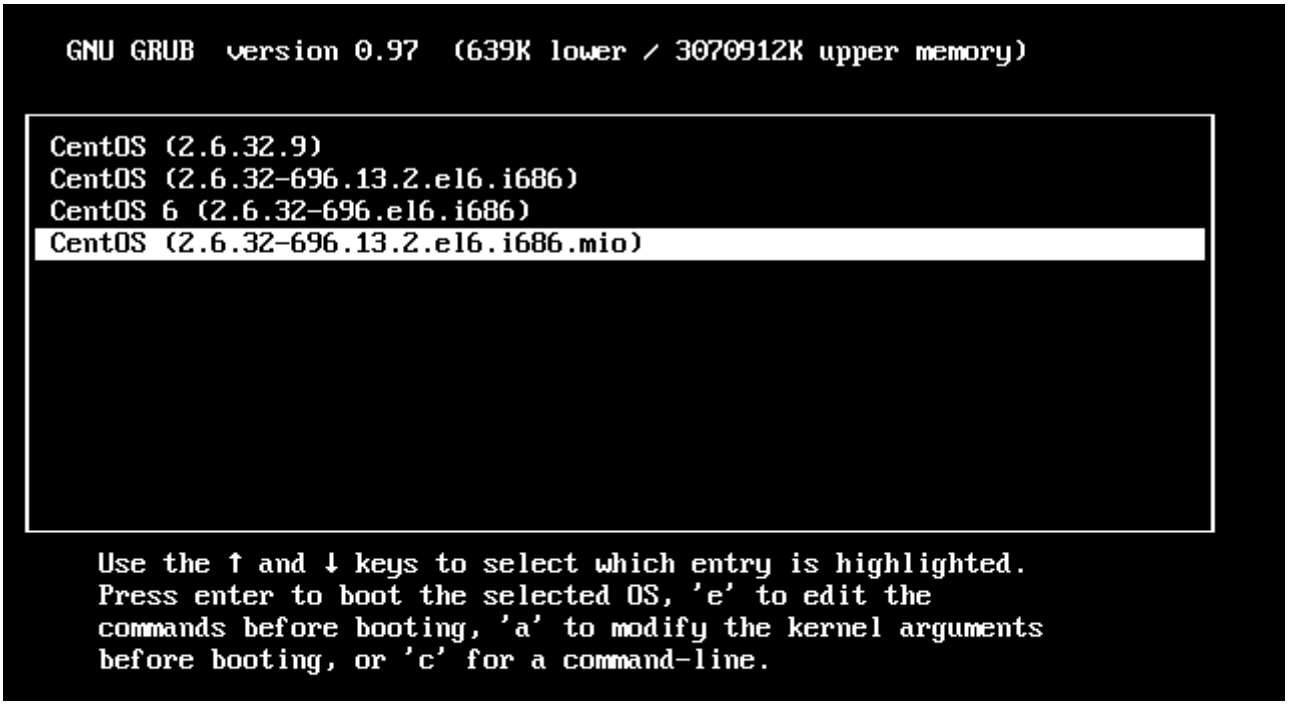
```
[root@localhost linux]# cd /boot
[root@localhost boot]# uname -a
Linux localhost.localdomain 2.6.32-696.13.2.el6.i686 #1 SMP Thu Oct 5 20:42:25 UTC 2017 i686 i686 i386 GNU/Linux
[root@localhost boot]# cp vmlinuz-2.6.32-696.13.2.el6.i686 vmlinuz-2.6.32-696.13.2.el6.i686.mio
[root@localhost boot]# cp initramfs-2.6.32-696.13.2.el6.i686.img initramfs-2.6.32-696.13.2.el6.i686.img
cp: «initramfs-2.6.32-696.13.2.el6.i686.img» y «initramfs-2.6.32-696.13.2.el6.i686.img» son el mismo fichero
[root@localhost boot]# cp initramfs-2.6.32-696.13.2.el6.i686.img initramfs-2.6.32-696.13.2.el6.i686.img.mio
[root@localhost boot]#
```

Una vez hecho esto, editamos el bootloader para que arranque el kernel que acabamos de copiar. En nuestro caso, el bootloader es GRUB Legacy, y su configuración está en el fichero */boot/grub/grub.conf*

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/mapper/VolGroup-lv_root
#          initrd /initrd-[generic-]version.img
#boot=/dev/sda
default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.32.9)
        root (hd0,0)
        kernel /vmlinuz-2.6.32.9 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-sun16 crashke
rnel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32.9.img
title CentOS (2.6.32-696.13.2.el6.i686)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.13.2.el6.i686 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrh
eb-sun16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.13.2.el6.i686.img
title CentOS 6 (2.6.32-696.el6.i686)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.el6.i686 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-su
n16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.el6.i686.img
title CentOS (2.6.32-696.13.2.el6.i686.mio)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.13.2.el6.i686.mio ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latar
cyrheb-sun16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.13.2.el6.i686.img.mio
~
~
-- INSERT --
```

Editamos el fichero añadiendo la entrada de nuestro kernel, y guardamos.

Una vez hecho esto, reiniciamos para comprobar que el kernel arranca correctamente.



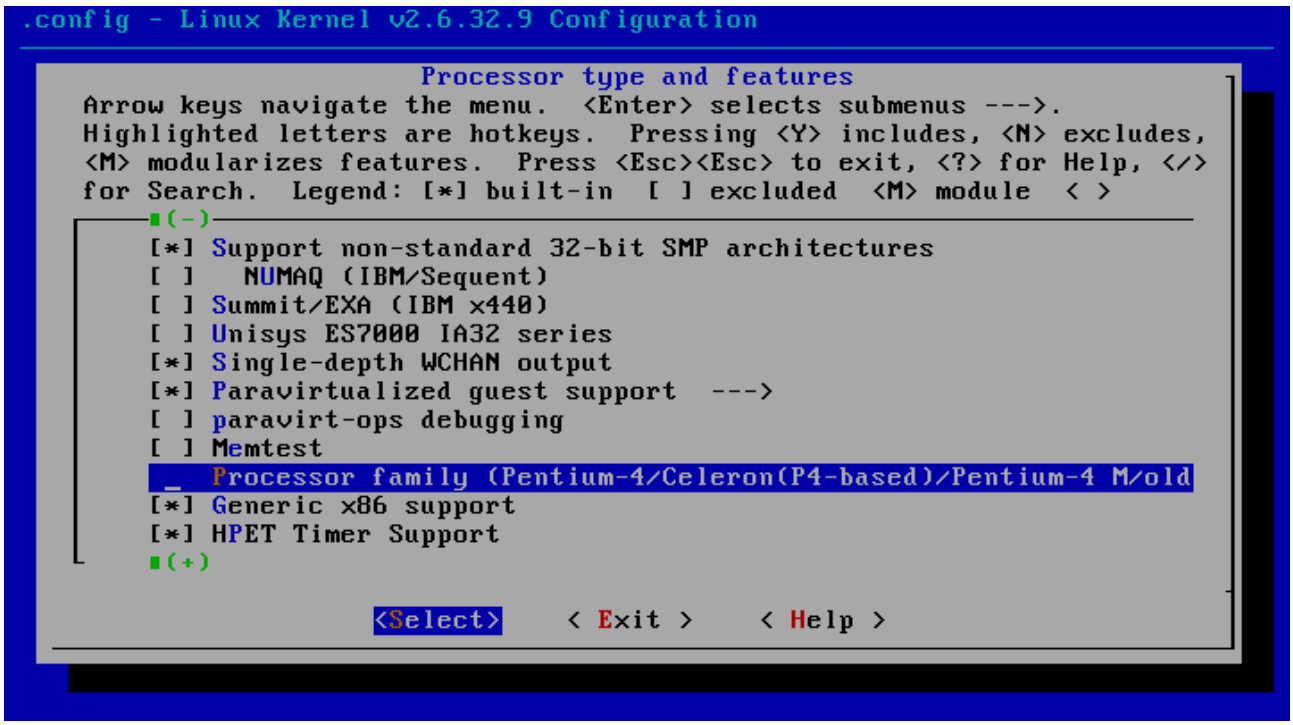
Vemos que la nueva entrada del kernel se ha añadido correctamente. También se ha añadido de forma automática (durante el *make install)* la entrada del nuevo kernel compilado.

Iniciamos el kernel copiado, y comprobamos que arranca correctamente.

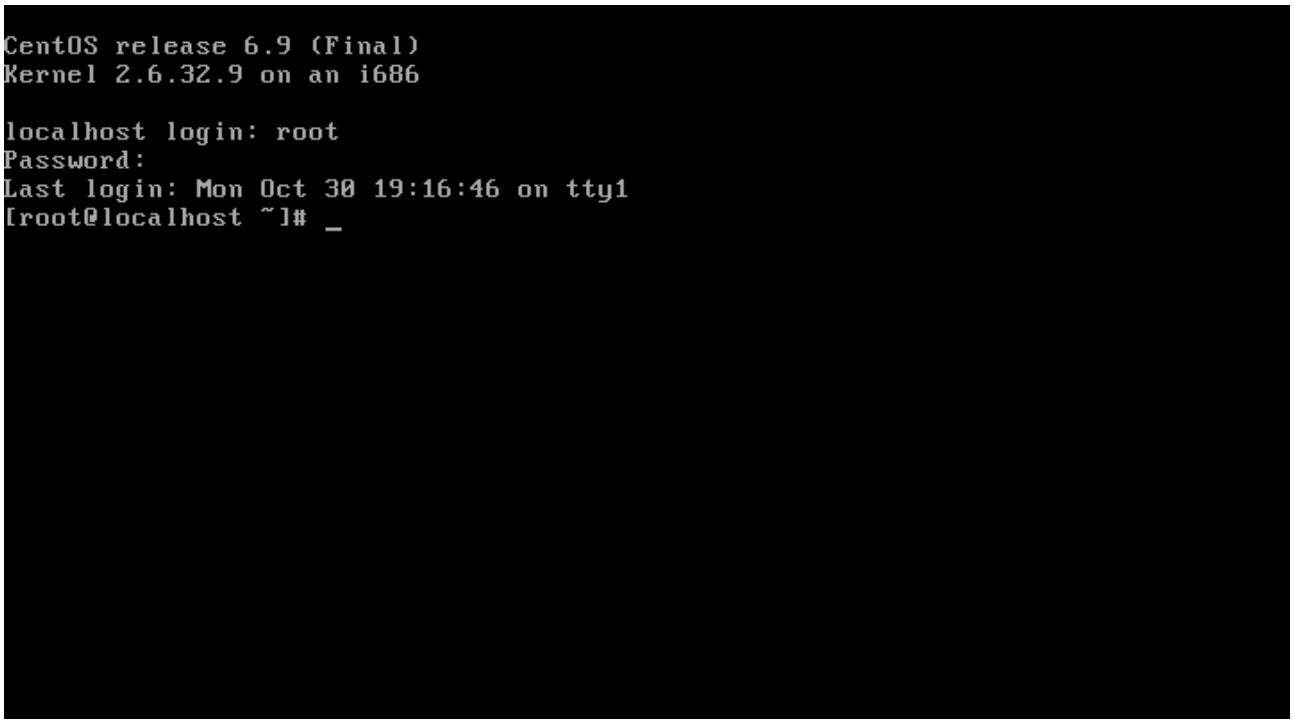
En el caso del nuevo kernel, no hay tanto éxito, así que toca revisar la configuración.

```
[<c042d56f>] ? bad_area_nosemaphore+0xf/0x20
[<c07e2d6b>] ? error_code+0x73/0x78
[<c05cd328>] ? ioread32_rep+0x38/0x50
[<c06aba89>] ? ata_sff_data_xfer32+0x69/0x120
[<c06ab546>] ? ata_pio_sector+0x126/0x160
[<c06ab5df>] ? ata_pio_sectors+0x5f/0xa0
[<c06ac9c1>] ? ata_sff_hsm_move+0x1f1/0x790
[<c069b2ce>] ? __ata_qc_complete+0x6e/0x110
[<c06ad008>] ? ata_sff_host_intr+0xa8/0x170
[<c06ad216>] ? ata_sff_interrupt+0x96/0xc0
[<c04a3635>] ? handle_IRQ_event+0x45/0x140
[<c04a6781>] ? move_native_irq+0x11/0x50
[<c04a55a3>] ? handle_edge_irq+0xa3/0x130
[<c040ba82>] ? handle_irq+0x32/0x60
[<c040afe7>] ? do_IRQ+0x47/0xc0
[<c040ba82>] ? handle_irq+0x32/0x60
[<c0409db0>] ? common_interrupt+0x30/0x38
[<c042bd63>] ? native_flush_tlb_single+0x3/0x10
[<c04334f7>] ? kunmap_atomic+0x67/0x80
[<c04e3ecc>] ? unmap_vmas+0x3cc/0x870
[<c04eabb3>] ? unmap_region+0x93/0x120
[<c04eae17>] ? do_munmap+0x1d7/0x270
[<c04eaeec>] ? sys_munmap+0x3c/0x60
[<c040969b>] ? sysenter_do_call+0x12/0x28
```

Abrimos menuconfig, y nos vamos a la sección del procesador. Allí, cambiamos la familia del procesador que definimos anteriormente, por otra algo más antigua, y recompilamos.



Finalmente, tras indicar como familia de procesador “586” y recompilar, la máquina virtual inicia correctamente con el nuevo kernel.



3.5. Generando un RAM disk

Una vez compilado el kernel, puede ser necesario crear un RAM disk, que permita al kernel acceder a los módulos necesarios para iniciar el sistema.

En nuestro caso, vemos que el initramfs ya ha sido generado previamente por *make install,* y se halla ubicado en el directorio /boot.

```
[root@localhost boot]# ls /boot/
bzImage-2.6.32.9                          initramfs-2.6.32-696.el6.i686.img    System.map-2.6.32-696.el6.i686
config-2.6.32-696.13.2.el6.i686           initramfs-2.6.32.9.img               System.map-2.6.32.9
config-2.6.32-696.el6.i686                lost+found                           vmlinuz
efi                                       symvers-2.6.32-696.13.2.el6.i686.gz  vmlinuz-2.6.32-696.13.2.el6.i686
grub                                      symvers-2.6.32-696.el6.i686.gz       vmlinuz-2.6.32-696.13.2.el6.i686.mio
initramfs-2.6.32-696.13.2.el6.i686.img    System.map                           vmlinuz-2.6.32-696.el6.i686
initramfs-2.6.32-696.13.2.el6.i686.img.mio System.map-2.6.32-696.13.2.el6.i686 vmlinuz-2.6.32.9
```

En caso de no haber sido generado, habría que generarlo manualmente.
Para realizar esto, usaremos el comando *mkinitrd*

La sintaxis de este comando es*: mkinitrd [ruta_ramdisk] [version_kernel],* siendo *ruta_ramdisk* la ruta y el nombre del nuevo RAM disk, y *versión_kernel* la versión del kernel para el cual queremos generarlo

En nuestro caso, vamos a generar un RAM disk para el kernel que tenemos en ejecución (2.6.32.9), con el nombre *miRAMDisk.img,* y lo vamos a guardar en el directorio */tmp* .
Para ello usaremos la orden *mkinitrd /tmp/miRAMDisk.img 2.6.32.9*

Ejecutamos el comando, y vemos como el RAM Disk se ha generado correctamente

```
[root@localhost ~]# mkinitrd /tmp/miRAMDisk.img 2.6.32.9
[root@localhost ~]# ls /tmp
miRAMDisk.img  yum.log
[root@localhost ~]#
```

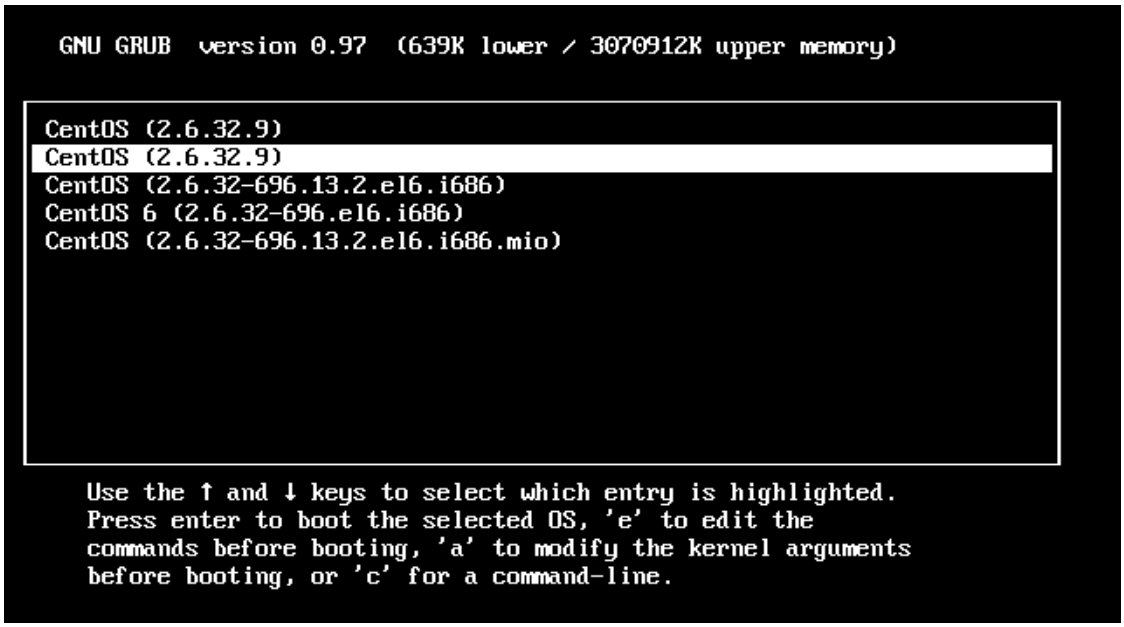
3.5.1. Probando el nuevo RAMDisk

Para comprobar que el nuevo RAMDisk funciona, debemos reconfigurar el bootloader (en este caso, GRUB Legacy) para que use el nuevo RAMDisk.

Abrimos el grub.conf, y añadimos una nueva linea para arrancar el kernel 2.6.32.9 con el nuevo RAMDisk

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/mapper/VolGroup-lv_root
#          initrd /initrd-[generic-]version.img
#boot=/dev/sda
default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.32.9)
        root (hd0,0)
        kernel /vmlinuz-2.6.32.9 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-sun16 crashke
rnel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32.9.img
title CentOS (2.6.32.9)
        root (hd0,0)
        kernel /vmlinuz-2.6.32.9 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-sun16 crashke
rnel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /../tmp/miRAMDisk.img
title CentOS (2.6.32-696.13.2.el6.i686)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.13.2.el6.i686 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrh
eb-sun16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.13.2.el6.i686.img
title CentOS 6 (2.6.32-696.el6.i686)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.el6.i686 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-su
n16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.el6.i686.img
title CentOS (2.6.32-696.13.2.el6.i686.mio)
        root (hd0,0)
@
```

Hecho esto, reiniciamos la máquina para probar el nuevo RAMDisk.

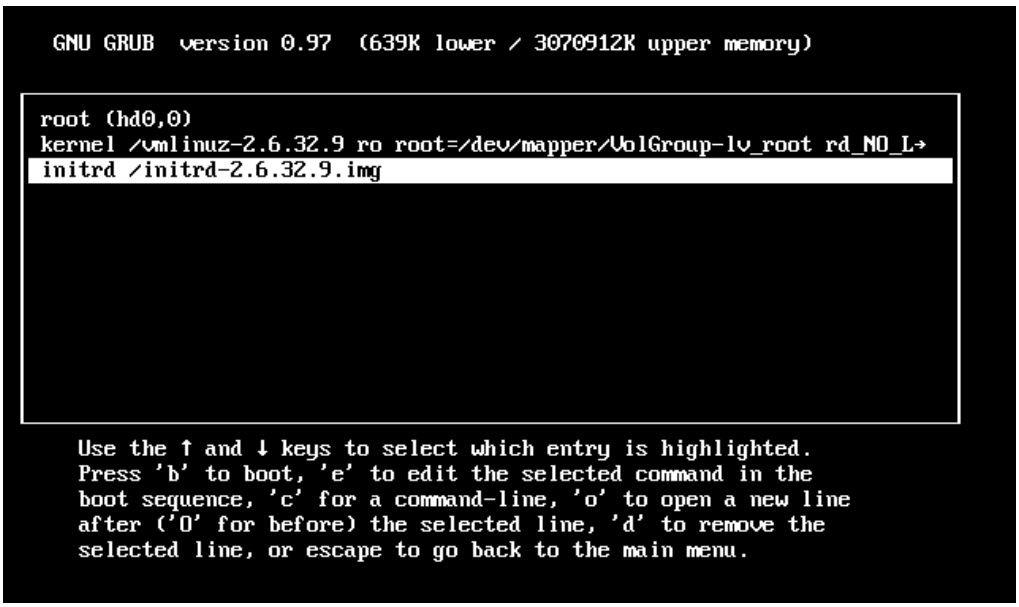


Tras reiniciar, vemos que el nuevo kernel es incapaz de iniciarse con el RAMDisk que hemos generado.

Dado que el RAMDisk se encuentra fuera del directorio del kernel, y el kernel no puede acceder al sistema de ficheros ext3, este es incapaz de encontrarlo; con lo cual el kernel no puede iniciarse y el sistema no arranca.

Renombramos el RAMDisk a *initrd-2.6.32.9.img* y lo movemos al directorio /boot.

Editamos la línea de arranque del grub para que busque el RAMDisk en la nueva ubicación. Para ahorrar tiempo, lo editamos directamente desde el menú del GRUB



Tras esto, comprobamos que el sistema ya arranca correctamente.

```
CentOS release 6.9 (Final)
Kernel 2.6.32.9 on an i686

localhost login: root
Password:
Last login: Mon Oct 30 21:31:54 from 10.0.2.2
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.32.9 #5 SMP Mon Oct 30 20:30:30 CET 2017 i686 i6
86 i386 GNU/Linux
[root@localhost ~]# _
```

Hacemos una última comprobación renombrando el RAMDisk alojado en /boot a *miRAMDisk.img,* y cambiamos la entrada del GRUB correspondiente.

```
[root@localhost tmp]# mv miRAMDisk.img initrd-2.6.32.9.img
[root@localhost tmp]# vi /boot/grub/grub.conf
[root@localhost tmp]#
```

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/mapper/VolGroup-lv_root
#          initrd /initrd-[generic-]version.img
#boot=/dev/sda
default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.32.9)
        root (hd0,0)
        kernel /vmlinuz-2.6.32.9 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-sun16 crashke
rnel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32.9.img
title CentOS (2.6.32.9)
        root (hd0,0)
        kernel /vmlinuz-2.6.32.9 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-sun16 crashke
rnel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /miRAMDisk.img
title CentOS (2.6.32-696.13.2.el6.i686)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.13.2.el6.i686 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrh
eb-sun16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.13.2.el6.i686.img
title CentOS 6 (2.6.32-696.el6.i686)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-696.el6.i686 ro root=/dev/mapper/VolGroup-lv_root rd_NO_LUKS rd_NO_MD rd_LVM_LV=VolGroup/lv_swap SYSFONT=latarcyrheb-su
n16 crashkernel=auto LANG=es_ES.UTF-8 rd_LVM_LV=VolGroup/lv_root  KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
        initrd /initramfs-2.6.32-696.el6.i686.img
title CentOS (2.6.32-696.13.2.el6.i686.mio)
        root (hd0,0)
@
"grub/grub.conf" 34L, 2221C written
```

Tras el cambio de nombre, el sistema continúa arrancando normalmente.
Esto demuestra que el único requisito para que el núcleo pueda encontrar el RAMDisk es que éste esté ubicado en */boot*, siendo el nombre del fichero un aspecto irrelevante para el proceso.

3.6. Gestionando módulos del kernel

Una vez con el sistema iniciado, pasamos a gestionar los módulos del kernel, haciendo diferentes pruebas sobre los mismos.

En este caso, vamos a usar como prueba el módulo de cdrom.
En primer lugar, buscamos el nombre del módulo de cdrom. Para ello, listamos los módulos cargados actualmente usando *lsmod* y filtramos el resultado usando *grep.*
Encontramos un módulo llamado *cdrom,* y que esta siendo usando por otro módulo llamado *sr_mod.*

Nos interesa conocer los parámetros que admite éste módulo, así que ejecutamos *modinfo*, que nos devuelve información sobre el módulo.
Vemos que el módulo admite 6 parámetros: *debug, autoclose, autoeject, lockdoor, check_media_type* y *mrw_format_restart.*

Tras esto, pasamos a montar un cdrom en nuestro árbol de ficheros, usando el comando *mount.* Vemos que el cdrom se ha montado correctamente, y un *ls* nos muestra su contenido.

Ya de paso, obtenemos la información del módulo sr_mod, correspondiente al driver para lectores CDROM con puerto SCSI (SATA también es considerado SCSI dentro del kernel)

```
[root@localhost ~]# modinfo sr_mod
filename:       /lib/modules/2.6.32.9/kernel/drivers/scsi/sr_mod.ko
license:        GPL
alias:          scsi:t-0x04*
alias:          scsi:t-0x05*
alias:          block-major-11-*
license:        GPL
description:    SCSI cdrom (sr) driver
srcversion:     8B1930CD4BFDA75A28E6F7F
depends:        cdrom
vermagic:       2.6.32.9 SMP mod_unload modversions 586
parm:           xa_test:int
[root@localhost ~]#
```

Ya vemos que el módulo funciona, pero queremos probar el efecto que provocaría la descarga de ese módulo.

Para ello, desmontamos el cdrom y retiramos el módulo. Como el módulo *cdrom* esta en uso por *sr_mod,* debemos retirar previamente éste.

Hecho esto, intentamos volver a montar el cdrom usando *mount.*

```
[root@localhost ~]# umount /mnt
[root@localhost ~]# rmmod -f sr_mod
[root@localhost ~]# rmmod cdrom
[root@localhost ~]# mount /dev/cdrom /mnt
mount: debe especificar el tipo de sistema de ficheros
[root@localhost ~]#
```

Al intentar montar el CDROM, vemos que *mount* nos pide especificar el sistema de ficheros.

Especificamos el sistema de ficheros (iso9660), y probamos de nuevo.

```
[root@localhost ~]# mount -t iso9660 /dev/cdrom /mnt
mount: el dispositivo especial /dev/cdrom no existe
[root@localhost ~]#
```

Ésta vez, *mount* nos dice que el dispositivo */dev/cdrom* no existe. Corroboramos eso listando los dispositivos del directorio /dev y filtrando.

```
[root@localhost ~]# ls /dev | grep cdrom
[root@localhost ~]#
```

Vemos que, efectivamente, ya no existe ningún dispositivo cdrom en el directorio. Al ejecutar *lsblk,* volvemos a comprobar que ya el sistema no detecta ningún CDROM.

```
[root@localhost ~]# lsblk
NAME                          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                             8:0    0   20G  0 disk
├─sda1                          8:1    0  500M  0 part /boot
└─sda2                          8:2    0 19,5G  0 part
  ├─VolGroup-lv_root (dm-0) 253:0    0 17,6G  0 lvm  /
  └─VolGroup-lv_swap (dm-1) 253:1    0    2G  0 lvm  [SWAP]
[root@localhost ~]#
```

Una vez comprobado el efecto de descargar ese módulo, deshacemos los cambios volviendo a cargarlo para que el sistema vuelva a leer CDROM.

Para ello, debemos cargar los dos módulos que hemos retirado: *cdrom* y *sr_mod.* Usamos el comando *modprobe* para cargar ambos módulos.

```
[root@localhost ~]# modprobe cdrom
[root@localhost ~]# modprobe sr_mod
[root@localhost ~]# lsblk
NAME                          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                             8:0    0   20G  0 disk
├─sda1                          8:1    0  500M  0 part /boot
└─sda2                          8:2    0 19,5G  0 part
  ├─VolGroup-lv_root (dm-0) 253:0    0 17,6G  0 lvm  /
  └─VolGroup-lv_swap (dm-1) 253:1    0    2G  0 lvm  [SWAP]
sr0                            11:0    1  358M  0 rom
[root@localhost ~]#
```

Una vez cargados ambos módulos, vemos que el sistema vuelve a detectar el CDROM, y se puede montar con normalidad.

```
[root@localhost ~]# mount /dev/cdrom /mnt
mount: dispositivo de bloques /dev/sr0 está protegido contra escritura; se monta como sólo lectura
[root@localhost ~]# ls /mnt
CentOS_BuildTag  GPL     isolinux  RELEASE-NOTES-en-US.html  RPM-GPG-KEY-CentOS-6        RPM-GPG-KEY-CentOS-Security-6  TRANS.TBL
EULA             images  Packages  repodata                  RPM-GPG-KEY-CentOS-Debug-6  RPM-GPG-KEY-CentOS-Testing-6
[root@localhost ~]#
```

4. Conclusión

La configuración y compilación del kernel nos permite adaptar el núcleo a nuestras necesidades, permitiendo ajustar aquellas opciones que nos permiten obtener un mayor rendimiento en nuestro sistema, o simplemente activar o desactivar cierto soporte hardware.

Ésta es una tarea complicada, que requiere amplios conocimientos sobre el hardware de la máquina. Cualquier error puede suponer, desde la falta de soporte hardware, hasta que simplemente el sistema no inicie.

Los desarrolladores del kernel nos ofrecen ciertas herramientas para facilitar su configuración, aunque siempre se requieren ciertos conocimientos para poder aplicar las configuraciones correctas.

Asimismo, el kernel instalado no arrancará si el bootloader no tiene correctamente añadida la entrada referente al mismo, o si no es capaz de encontrar los ficheros necesarios para que éste pueda arrancar. La tarea de añadirlo puede ser tediosa y, de nuevo, puede estar sujeta a muchos posibles errores.
Pero, bien realizado, nos permite ajustar el arranque del kernel con mucha precisión, permitiendo activar determinadas opciones durante el arranque

Finalmente, el soporte de módulos del kernel Linux nos puede ayudar a insertar determinados componentes o retirarlos a voluntad, aunque esto pueda suponer determinados problemas en algunos casos, que hay que saber controlar.