Servicios y módulos del núcleo

1. Presentación

El núcleo es el corazón del sistema operativo Linux. Linux es el núcleo, el sistema operativo tal y como lo desarrolló originalmente Linus Torvalds. El sistema operativo Linux en su amplio contexto se compone del núcleo y de las herramientas operativas básicas, las herramientas GNU. De igual forma, varios son los que dicen que Linux debería llamarse GNU/Linux: GNU por las herramientas, Linux por el núcleo, GNU/Linux para el Sistema Operativo. Nosotros lo llamaremos Linux.

El núcleo de Linux es libre. Sus fuentes están disponibles. Por lo tanto, se puede volver a compilar para adaptarlo específicamente a sus necesidades, modificarlo y añadirle extensiones.

El núcleo de Linux forma parte de la familia de los núcleos monolíticos, es decir, que agrupan todas sus funcionalidades y componentes en un programa único y sus extensiones, llamadas módulos. Desde la versión estable 2.0 el núcleo es modular.

El núcleo se llama **kernel**. Está en `/boot` y su nombre, por convención, comienza a menudo por `vmlinuz-X.Y.Z.p-Vtxt` .

Se obtiene la versión del núcleo con el comando **uname**.

```
$ uname -r
5.8.15-301.fc33.x86_64
```

Las letras tienen un significado particular.

- `X`: versión mayor del núcleo. Entre la versión 1 y 2, el paso al funcionamiento modular ha sido determinante, así como la nueva implementación de la capa de red. La versión 3 vio la luz en verano de 2011, la versión 4 en abril de 2015, la versión 5 en marzo de 2019.
- `Y`: hasta el núcleo 2.5, un valor par representa una rama estable del núcleo. Una versión impar representaba una rama de desarrollo (¡cuidado!). Cada incremento

representa una evolución importante del núcleo.

Las versiones 2.6, 3.x, 4x y 5.x no disponen de ramas de desarrollo, ya que evolucionan demasiado rápido. Los desarrolladores decidieron implementar sus novedades directamente en la versión estable después de una fase importante de revisiones y de versiones candidatas (rc).

- **z**: versión menor del núcleo. Cuando un lote de modificaciones, con respecto a una versión anterior, necesita la difusión du nuevo núcleo, entonces se incrementa esta cifra. Por ejemplo, un lote que agrupa una modificación del sistema de sonido (Alsa, que pasa de 1.0.8 a 1.0.9), del sistema de archivos (se añade de ReiserFS 4), y así sucesivamente...
- **p**: versión corregida o intermedia presente desde la versión 2.6. Cuando el núcleo necesita una actualización menor (corrección de uno o dos bugs, etc.) pero pocas funcionalidades nuevas, se incrementa este valor. Este valor es por lo general propio del empaquetador. Es decir, del editor de la distribución.
- **v**: al igual que con los paquetes, versión propia del editor de la distribución.
- **txt**: a veces se añade un texto para dar precisiones sobre el núcleo. Por ejemplo, smp indica un núcleo multiprocesador : x86-64 una arquitectura de 64 bits, etc.

Las versiones 4.0 y 5.0 han sido objeto de grandes debates, titulares en la prensa y en los sitios web especializados. Es posiblemente la única ventaja de esta nueva numeración. ¡Qué revolución!, podría pensar. Pues no. Dejemos hablar a Linux Torvalds:

«En términos de funcionalidades, la versión 4.0 no tiene nada de especial. Hemos hablado mucho de la nueva infraestructura de corrección del nuevo núcleo, pero en realidad ésta no es la única razón del cambio de versión. Hemos efectuado cambios mucho mayores en otras versiones. Por lo que es más una versión de "sólida mejora en el código".»

"Me gustaría subrayar (una vez más) que no hacemos releases basadas en la funcionalidades, y que ´5.0´no significa nada más que las cifras 4.X empiezan a quedarse

demasiado cortas"

Por lo tanto, el gran cambio sólo concierne al salto en la numeración.

2. uname

El comando **uname** (unix name) permite obtener toda la información sobre la versión de Unix (de Linux en este caso) de manera precisa y completa.

Parámetro	Resultado
`-m` (machine)	Tipo de hardware de la máquina.
`-n` (nodename)	Nombre de la máquina.
`-r` (release)	Versión (número) del núcleo.
`-s` (system name)	Nombre del sistema operativo. Por defecto.
`-p` (processor)	Tipo de procesador.
`-i`	Plataforma física.
`-v` (version)	Versión del sistema.
`-a` (all)	Toda la información.

```
$ uname -r
```

```
5.8.15-301.fc33.x86_64
$ uname
Linux
$ uname -m
x86_64
$ uname -n
fedora
$ uname -p
x86_64
$ uname -i
x86_64
$ uname -o
GNU/Linux
$ uname -v
#1 SMP Thu Oct 15 16:58:06 UTC 2020
$ uname -a
Linux fedora 5.8.15-301.fc33.x86_64 #1 SMP Thu Oct 15 16:58:06 UTC 2020
x86_64 x86_64 x86_64 GNU/Linux
```

3. Gestión de los módulos

Los componentes básicos (gestor de tareas, gestión de la memoria, de los procesos, API, etc.) están siempre dentro de un programa único. Pero algunos drivers de periférico, sistemas de archivos, extensiones, protocolos de redes, etc., pueden estar presentes en forma de módulos. Los módulos se comunican con el núcleo a través de un API común. Se ejecutan en el espacio del núcleo. Se pueden configurar. Se pueden cargar y descargar a petición y así evitar un reinicio de la máquina. El añadido de un nuevo módulo (desde sus fuentes por ejemplo) no necesita reinicio.

Los módulos están en **`/lib/modules/$(uname -r)`** .

```
cd /lib/modules/$(uname -r)
[root@fedora 5.8.15-301.fc33.x86_64]# pwd
/lib/modules/5.8.15-301.fc33.x86_64
```

Los módulos tienen un nombre que termina por “ko" para kernel object. La terminación de origen era "o" para los núcleos 2.0 a 2.4. Efectivamente, es eso: archivos de objetos relacionados (linked) de manera dinámica al núcleo, que proponen así una API adicional.

A veces los módulos están comprimidos en formato XZ, de esta manera puede encontrar archivos que terminan en ".ko.xz". Si es el caso en su distribución, piense en reemplazar en todos los ejemplos siguientes "ko" por ".ko.xz" donde sea necesario.

```
# cd /lib/modules/$(uname -r)/kernel/fs/vfat
# file msdos.ko
msdos.ko: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=a3033e5499669d40f91f25d7b99ced219454f162, not stripped
```

Si el módulo está comprimido (formato XZ), haga esto:

```
# xzcat vfat.ko.xz | file -
/dev/stdin: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=feaf8b2e31fe475d3ddcd016057354196c6e6f59, not stripped
```

La palabra "relocatable" indica que está ante un archivo de objeto. Los núcleos recientes adjuntan un valor suplementario, el BuildID.

a. lsmod

El comando **lsmod** lista los módulos actualmente cargados, con sus posibles dependencias.

La salida ha sido, por supuesto, truncada:

```
# lsmod

Module                  Size  Used by
nft_fib_inet           16384  1
nft_fib_ipv4           16384  1 nft_fib_inet
nft_fib_ipv6           16384  1 nft_fib_inet
nft_fib                16384  3 nft_fib_ipv6,nft_fib_ipv4,nft_fib_inet
```

```
nft_reject_inet      16384  4
nf_reject_ipv4       16384  1 nft_reject_inet
nf_reject_ipv6       20480  1 nft_reject_inet
nft_reject           16384  1 nft_reject_inet
nft_ct               20480  9
nft_chain_nat        16384  4
ip6table_nat         16384  0
ip6table_mangle      16384  0
ip6table_raw         16384  0
ip6table_security    16384  0
iptable_nat          16384  0
nf_nat               49152  3 ip6table_nat,nft_chain_nat,iptable_nat
nf_conntrack         163840  2 nf_nat,nft_ct
nf_defrag_ipv6       24576  1 nf_conntrack
nf_defrag_ipv4       16384  1 nf_conntrack
iptable_mangle       16384  0
iptable_raw          16384  0
iptable_security     16384  0
nf_tables            237568  169 nft_ct,nft_reject_inet,nft_fib_ipv6,
nft_fib_ipv4,nft_chain_nat,nft_reject,nft_fib,nft_fib_inet
...
# lsmod | wc -l
55
```

La primera columna indica el nombre del módulo cargado. Su nombre refleja a menudo para qué sirve. La segunda columna da el tamaño del módulo. La tercera columna suministra un contador de utilización (cuántos componentes del sistema acceden a los módulos). La última columna proporciona la lista de los módulos que utilizan (o, por lo tanto, dependen) del primero.

En el ejemplo anterior, el módulo snd_intel8x0 utiliza el módulo snd_ac97_codec.

lsmod sólo vuelve a dar forma al contenido del archivo virtual (aquí recortado) **`/proc/modules`**.

```
# cat /proc/modules

nft_fib_inet 16384 1 - Live 0xffffffffc0820000
```

```
nft_fib_ipv4 16384 1 nft_fib_inet, Live 0xffffffffc081b000
nft_fib_ipv6 16384 1 nft_fib_inet, Live 0xffffffffc0800000
nft_fib 16384 3 nft_fib_inet,nft_fib_ipv4,nft_fib_ipv6, Live 0xffffffffc07fb000
nft_reject_inet 16384 4 - Live 0xffffffffc07f6000
nf_reject_ipv4 16384 1 nft_reject_inet, Live 0xffffffffc07f1000
nf_reject_ipv6 20480 1 nft_reject_inet, Live 0xffffffffc07eb000
nft_reject 16384 1 nft_reject_inet, Live 0xffffffffc07e6000
nft_ct 20480 9 - Live 0xffffffffc07e0000
nft_chain_nat 16384 4 - Live 0xffffffffc07db000
ip6table_nat 16384 0 - Live 0xffffffffc07d6000
ip6table_mangle 16384 0 - Live 0xffffffffc07d1000
ip6table_raw 16384 0 - Live 0xffffffffc07cc000
ip6table_security 16384 0 - Live 0xffffffffc0750000
iptable_nat 16384 0 - Live 0xffffffffc074b000
nf_nat 49152 3 nft_chain_nat,ip6table_nat,iptable_nat, Live 0xffffffffc073e000
nf_conntrack 163840 2 nft_ct,nf_nat, Live 0xffffffffc07a3000
nf_defrag_ipv6 24576 1 nf_conntrack, Live 0xffffffffc0737000
 nf_defrag_ipv4 16384 1 nf_conntrack, Live 0xffffffffc0732000
iptable_mangle 16384 0 - Live 0xffffffffc072d000
iptable_raw 16384 0 - Live 0xffffffffc0728000
iptable_security 16384 0 - Live 0xffffffffc0723000
...
```

b. depmod

El comando **depmod** actualiza el árbol de las dependencias entre los módulos modificando el archivo **`modules.dep`**.

El archivo **`/lib/modules/$(uname -r)/modules.dep`** contiene dos columnas. La primera columna es la ruta del módulo, la segunda es la lista de dependencias: los módulos que se deben cargar también para que el primero funcione. Aquí, tenemos el ejemplo de la línea que corresponde al módulo `lp`:

```
$ grep "/lp.ko:" modules.dep
kernel/drivers/char/lp.ko: kernel /drivers/parport/parport.ko
```

El módulo `lp.ko` depende del módulo `parport.ko`. Por lo tanto se debe cargar el módulo `parport.ko` primero para que `lp.ko` funcione.

Aquí, de nuevo, como en la mayoría de las ubicaciones donde se utilizan los archivos de módulos, podría encontrar el sufijo xz que indicaría una compresión.

```
kernel/drivers/char/lp.ko.xz: kernel/drivers/parport/parport.ko.xz
```

El uso más corriente de **depmod** incluye el parámetro `-a`, que vuelve a construir las dependencias de todos los módulos que corresponden al núcleo actual. Se ejecuta esta acción a cada inicio del sistema, pero si compila o instala nuevos módulos, debe reiniciar manualmente este comando para tener en cuenta las nuevas dependencias.

```
# depmod -a
```

c. modinfo

El comando **modinfo** ofrece toda la información necesaria sobre un módulo:

- nombre del archivo correspondiente;
- descripción del módulo;
- autor;
- licencia;
- dependencias;
- parámetros;
- alias.

Los módulos no proporcionan toda esta información. Algunos módulos no tienen parámetros. En el primer ejemplo, el módulo `parport` no tiene parámetros. Éstos se instalan como opciones de montaje.

```
# modinfo parport
```

```
filename:       /lib/modules/5.8.15-301.fc33.x86_64/kernel/drivers/parport/
parport.ko.xz
license:        GPL
depends:
retpoline:      Y
intree:         Y
name:           parport
vermagic:       5.8.15-301.fc33.x86_64 SMP mod_unload
```

Dependiendo de la distribución, podrá encontrar información sobre la firma que certifica el origen del módulo:

```
sig_id:         PKCS#7
signer:         Fedora kernel signing key
sig_key:        36:D0:EA:49:3A:ED:79:59:72:4B:8B:4B:AD:42:7E:B7:9C:CA:40:F3
sig_hashalgo:   sha256
```

En este segundo ejemplo, el módulo dispone de parámetros. Este módulo permite utilizar una webcam que contiene un chip compatible con el módulo uvcvideo. Los parámetros están truncados de manera voluntaria.

```
# modinfo uvcvideo

filename:       /lib/modules/5.8.15-301.fc33.x86_64/kernel/drivers/media/
usb/uvc/uvcvideo.ko.xz
version:        1.1.1
license:        GPL
description:    USB Video Class driver
author:         Laurent Pinchart <laurent.pinchart@ideasonboard.com>
srcversion:     704B02C38DD542CFE0FA57E
alias:          usb:v*p*d*dc*dsc*dp*ic0Eisc01ip01in*
alias:          usb:v*p*d*dc*dsc*dp*ic0Eisc01ip00in*
alias:          usb:v8086p0B03d*dc*dsc*dp*ic0Eisc01ip00in*
alias:          usb:v29FEp4D53d*dc*dsc*dp*ic0Eisc01ip00in*
...

parm:           clock:Video buffers timestamp clock
```

```
parm:       hwtimestamps:Use hardware timestamps (uint)
parm:       nodrop:Don't drop incomplete frames (uint)
parm:       quirks:Forced device quirks (uint)
parm:       trace:Trace level bitmask (uint)
parm:       timeout:Streaming control requests timeout (uint)
```

Un mismo módulo puede gestionar varios tipos de componentes físicos. Existen varias webcams que disponen de un chip ov511 o similar que se pueden gestionar con el módulo uvcvideo. Para que el núcleo sepa qué módulo cargar durante la detección de la webcam, o para que el módulo sepa qué tipo de componente físico debe gestionar durante su carga, dispone de alias de hardware, usb, pci, scsi, etc., que permiten reconocer los periféricos que debe gestionar.

Se pasan los parámetros al módulo mediante los comandos **insmod** o **modprobe**, con el archivo `/etc/modprobe.conf` o los que están en `/etc/modprobe.d`.

d. insmod

El comando **insmod** carga un módulo dado sin gestionar las dependencias. Coge como parámetro un nombre de módulo con su ruta. Le corresponde a usted gestionar el orden de carga de los módulos para evitar errores relacionados con símbolos no resueltos causados por un problema de dependencia.

En el ejemplo siguiente, como los módulos parport y lp no están cargados, se intenta cargar el módulo `lp.ko` solo. Se produce un error, ya que este módulo depende sólo de `parport.ko`. El retorno de **dmesg** es elocuente al respecto.

```
# lsmod|egrep '^(parport|lp)'
# cd /lib/modules/$(uname -r)//kernel/drivers/char ; ls
lp.ko
# insmod lp.ko # o insmode lp.ko.xz
insmod: ERROR: could not insert module lp.ko: Unknown symbol in module
# dmesg | tail -20
...
[ 5788.876288] lp: Unknown symbol __parport_register_driver (err 0)
[ 5788.876297] lp: Unknown symbol parport_write (err 0)
```

```
[ 5788.876305] lp: Unknown symbol parport_set_timeout (err 0)
[ 5788.876311] lp: Unknown symbol parport_read (err 0)
[ 5788.876316] lp: Unknown symbol parport_register_device (err 0)
[ 5788.876320] lp: Unknown symbol parport_claim_or_block (err 0)
[ 5788.876324] lp: Unknown symbol parport_release (err 0)
[ 5788.876334] lp: Unknown symbol parport_unregister_device (err 0)
[ 5788.876342] lp: Unknown symbol parport_unregister_driver (err 0)
[ 5788.876351] lp: Unknown symbol parport_negotiate (err 0)
```

En este segundo ejemplo, primero se carga el módulo fat, luego el módulo vfat. No hay error, ya que todas las dependencias están presentes.

```
# pwd
/lib/modules/5.8.15-301.fc33.x86_64/kernel/drivers/parport
# insmod parport.ko # o insmod parport.ko.xz
# cd ../char
# insmod lp.ko
# lsmod|egrep '^(parport|lp)'
lp 20480 0
parport 61440 1 lp
```

Debería considerar el parámetro `-k`, que permite la limpieza automática de los módulos. Esta opción solicita al sistema que descargue automáticamente un módulo (contador a cero) si ya no se utiliza. Esto permite ganar algunos recursos y obtener un sistema más limpio.

Para transmitir parámetros al módulo, indíquelos a continuación del comando. Por ejemplo, para el módulo uvcvideo, desea que el driver muestre más información:

```
# insmod uvcvideo.ko trace=5
```

e. rmmod

El comando **rmmod** descarga el módulo dado. Se trata del contrario de **insmod** y, al igual que éste, rmmod no gestiona las dependencias:

- No es posible descargar un módulo en curso de utilización.
- No es posible descargar un módulo si otro módulo lo utiliza, incluso si este último no es utilizado (problema de dependencia).

```
# rmmod parport
rmmod: ERROR: Module parport is in use by: lp
```

En este segundo ejemplo, se descarga el módulo lp, luego el módulo parport, en este orden.

```
# rmmod lp
# rmmod parport
```

f. modprobe

El comando **modprobe** carga el módulo dado, así como todas sus dependencias y parámetros contenidos en `/etc/modprobe.conf` o los archivos `/etc/modprobe.d/*` . El parámetro `-r` permite descargar un módulo y los que dependen de él (si no están en uso).

Cargar el módulo lp mediante **modprobe** significa en la práctica cargar también el módulo parport.

```
# lsmod|egrep '^(parport|lp)'
# modprobe lp
# lsmod|egrep '^(parport|lp)'
lp 20480 0
parport 61440 1 lp
```

Ahora, como ve que sólo lp utiliza parport (contador a 1), pero que nada utiliza lp (contador a cero), puede intentar descargar lp y los módulos de los cuales depende, si no se usan.

```
# modprobe -r lp
# lsmod|egrep '^(parport|lp)'
```

Puede pasar opciones al módulo; en ese caso, debe comprobar si una línea "install" que corresponde al módulo existe en el archivo modprobe.conf y modificarla para que acepte los parámetros, o utilizar una línea “opciones".

g. modprobe.d

¡Ojo! El uso del archivo **modprobe.conf** ha caído en desuso en las versiones recientes. Cuando está presente, aparece un aviso. En algunas distribuciones recientes, este archivo no se encuentra y se ignorará por los comandos. La configuración se traslada a los archivos independientes albergados en `/etc/modprobe.d`.

La configuración de los módulos está en los archivos albergados en `/etc/modprobe.d/*.conf` y en `/etc/modprobe.conf`. Aunque las distribuciones actuales han dejado de utilizar `modprobe.conf`, sigue siendo importante, ya que todavía se utiliza en distribuciones empresariales, especialmente en las distribuciones anteriores de SLES, RHEL y Debian. En estos archivos la sintaxis es idéntica. Puede definir alias de módulos (muy práctico para las tarjetas de red), pasar opciones a los módulos así como añadir acciones en la carga y descarga de un módulo.

Alias y opciones

Pongamos un ejemplo: tiene dos tarjetas de red. El driver de la primera tarjeta está en el módulo **e1000**. Quiere que se cargue utilizando el nombre **eth0**. El driver de la segunda tarjeta está en el módulo **airo** y desea cargarlo con el nombre **eth1**. También va a

especificar parámetros en el módulo uvcvideo, cuya información ha recuperado con **modinfo**.

```
# cat /etc/modprobe.d/network-interfaces.conf
...
alias eth0 e1000
alias eth1 airo
# cat /etc/modprobe.d/webcam.conf
options uvcvideo led=2 compress=1
...
```

Luego carga los módulos por nombre o por alias. En todos los casos, se tienen en cuenta los parámetros automáticamente.

```
# modprobe eth0
# modprobe eth1
# modprobe uvcvideo
```

Se pueden utilizar las opciones de módulos con los alias.

```
alias webcam uvcvideo
options webcam led=2 compress=1
...
```

install y remove

Los comandos **install** y **remove** del archivo `modprobe.conf` son los más potentes. Durante la carga de un módulo, si **modprobe** encuentra un comando **install** asociado, no carga este módulo, sino que ejecuta los comandos indicados. El comando puede ser lo que desee, como por ejemplo la carga de un módulo más adaptado, la instalación de una configuración específica, la ejecución de un script, etc.

El ejemplo siguiente intenta cargar el módulo `uvcvideo_new` si existe, y si no, conmuta a uvcvideo cuando se invoca modprobe webcam.

```
install webcam /sbin/modprobe uvcvideo_new || /sbin/modprobe uvcvideo
```

modprobe puede coger un parámetro práctico que le permita ignorar las líneas install de modprobe.conf: **--ignore-install**. El ejemplo siguiente carga el módulo ahci, luego el módulo `ata_piix` cuando se intenta cargar este último. Sin el parámetro, **modprobe** se ejecutaría en bucle, leyendo e interpretando la línea install a cada intento de carga de `ata_piix`.

```
install ata_piix /sbin/modprobe ahci 2>&1 |:; /sbin/modprobe
--ignore-install ata_piix
```

El comando **remove** hace lo mismo, pero en el momento de descargar el módulo con **modprobe -r**.

CMDLINE_OPTS

Si hay una línea install en modprobe.conf e intenta cargar el módulo correspondiente con modprobe y unos parámetros, no se tendrán en cuenta estos últimos excepto si ha añadido detrás del módulo la cadena $CMDLINE_OPTS.

```
install webcam /sbin/modprobe ov511_new $CMDLINE_OPTS ||
/sbin/modprobe ov511 $CMDLINE_OPTS
```

4. Carga de los módulos al inicio

a. initrd e initramfs

Algunos módulos pueden ser necesarios en el inicio de la máquina, en particular para montar un sistema de archivos. ¿Cómo montar la partición raíz en ext4 cuando el módulo que gestiona este tipo de sistema de archivos está en esta partición (y no en el núcleo)? Se colocan estos módulos en forma de imagen de disco virtual llamada **initrd** (inicial ramdisk) o **initramfs** (initial ram filesystem). Se cargan estos archivos comprimidos al inicio en memoria y son ramdisks. Contienen instrucciones y módulos que se cargan al

inicio. Generalmente hay un initrd por núcleo presente. Poseen el mismo formato, solamente cambian los nombres, así los comandos siguientes funcionarán de la misma manera, sean los que sean.

```
# cd /boot
# ls -l initrd.img*

lrwxrwxrwx 1 root root       27 abr 17 14:39 initrd.img -> initrd.img-5.4.0-72-generic
-rw-r--r-- 1 root root 82813867 abr 22 21:40 initrd.img-5.4.0-72-generic
lrwxrwxrwx 1 root root       27 abr 17 14:39 initrd.img.old -> initrd.img-5.4.0-72-generic
```

Se puede extraer el contenido de **`initrd`** e incluso modificarlo y volver a construirlo para adaptarlo. Hay que comprobar el formato de este archivo, porque puede estar comprimido. Una vez que se haya descomprimido, en el caso de que fuera necesario, este archivo es una carpeta cpio comprimida.

Tenga precaución, no obstante, ya que el initrd presente en las distribuciones recientes no se puede explotar fácilmente. He aquí dos ejemplos. El primero proviene de un initrd de "antigua generación": Ubuntu 16.04 LTS, comprimido en formato gzip:

```
# mkdir initrd
# cd initrd/
# zcat /boot/initrd.img-4.8.0-34-generic|cpio -id --no-absolute-filenames
216955 bloques
# ls -l
total 72
drwxr-xr-x 2 root root 20480 ene 25 21:36 bin
drwxr-xr-x 3 root root 4096 ene 25 21:36 conf
drwxr-xr-x 10 root root 4096 ene 25 21:36 etc
-rwxr-xr-x 1 root root 6647 ene 25 21:36 init
drwxr-xr-x 9 root root 4096 ene 25 21:36 lib
drwxr-xr-x 2 root root 4096 ene 25 21:36 lib64
drwxr-xr-x 2 root root 4096 ene 25 21:36 run
drwxr-xr-x 2 root root 4096 ene 25 21:36 sbin
drwxr-xr-x 10 root root 4096 ene 25 21:36 scripts
drwxr-xr-x 4 root root 4096 ene 25 21:36 usr
drwxr-xr-x 4 root root 4096 ene 25 21:36 var
```

Este ejemplo proviene de un initrd de "última generación": Fedora 31. Si aplicamos el mismo método nos damos cuenta rápidamente de que el archivo no parece que esté comprimido, y si extraemos el contenido, se obtiene:

```
# file /boot/initramfs-5.4.14-200.fc31.x86_64.img
/boot/initramfs-5.4.14-200.fc31.x86_64.img: ASCII cpio archive (SVR4 with no CRC)
# cat /boot/initramfs-5.4.14-200.fc31.x86_64.img | cpio -i
48 blocs
# tree
.
+-- early_cpio
+-- kernel
   +-- x86
      +-- microcode
         +-- GenuineIntel.bin
```

No es el resultado esperado ya que 48 bloques de 512 bytes corresponden 24576 bytes. Sin embargo, el archivo initrd es más voluminoso. El archivo comprimido está constituido, en las distribuciones modernas, por distintas partes: las primeras no están comprimidas y contienen a menudo firmwares. y después hay una última que contiene, efectivamente, el initrd que se esperaba. El núcleo reaccionará de la manera siguiente:

- Si uno o algunos archivos cpio no comprimidos existen al principio, extraerá y cargará el microcódigo presente si fuera necesario
- Lo que sigue se considerará como el initrd "normal"
- Extrae el archivo de la última parte en un ramdisk
- Monta el ramdisk como primer sistema de archivos raíz.

Por lo tanto, a partir del bloque 48, deberíamos encontrar algo. Observemos lo que pasará sí extraemos el contenido a partir de este bloque:

```
# dd if=/boot/initramfs-5.4.12-200.fc31.x86_64.img bs=512 skip=48 of=new_initrd.img
62596+1 registros leídos
62596+1 registros escritos
32049202 bytes (32 MB, 31 MiB) copied, 0,412198 s, 77,8 MB/s
```

```
[root@fedora initrd]# file new_initrd.img
new_initrd.img: gzip compressed data, max compression, from Unix, original size
modulo 2^32 73633280
```

Es un archivo gzip. Ahora lo podemos extraer:

```
# zcat new_initrd.img | cpio -i
143815 blocs
# ls -l
total 31312
lrwxrwxrwx.  1 root root       7  2 abr 17:52 bin -> usr/bin
drwxr-xr-x.  2 root root      74  2 abr 17:52 dev
-rw-r--r--.  1 root root       2  2 abr 17:48 early_cpio
drwxr-xr-x. 10 root root    4096  2 abr 17:52 etc
lrwxrwxrwx.  1 root root      23  2 abr 17:52 init -> usr/lib/systemd/systemd
drwxr-xr-x.  3 root root      17  2 abr 17:48 kernel
lrwxrwxrwx.  1 root root       7  2 abr 17:52 lib -> usr/lib
lrwxrwxrwx.  1 root root       9  2 abr 17:52 lib64 -> usr/lib64
-rw-r--r--.  1 root root 32049202  2 abr 17:51 new_initrd.img
drwxr-xr-x.  2 root root       6  2 abr 17:52 proc
drwxr-xr-x.  2 root root       6  2 abr 17:52 root
drwxr-xr-x.  2 root root       6  2 abr 17:52 run
lrwxrwxrwx.  1 root root       8  2 abr 17:52 sbin -> usr/sbin
-rwxr-xr-x.  1 root root    3121  2 abr 17:52 shutdown
drwxr-xr-x.  2 root root       6  2 abr 17:52 sys
drwxr-xr-x.  2 root root       6  2 abr 17:52 sysroot
drwxr-xr-x.  2 root root       6  2 abr 17:52 tmp
drwxr-xr-x.  8 root root      81  2 abr 17:52 usr
drwxr-xr-x.  3 root root      40  2 abr 17:52 var
```

Este ejemplo es uno de los más sencillos. El comando **binwalk** le dará muchos datos:

```
# binwalk /boot/initramfs-5.4.14-200.fc31.x86_64.img

DECIMAL       HEXADECIMAL     DESCRIPTION
```

```
-------------------------------------------------------------------------------
0           0x0         ASCII cpio archive (SVR4 with no CRC), file name:
".", file name length: "0x00000002", file size: "0x00000000"
112         0x70        ASCII cpio archive (SVR4 with no CRC), file name:
"early_cpio", file name length: "0x0000000B", file size: "0x00000002"
240         0xF0        ASCII cpio archive (SVR4 with no CRC), file name:
"kernel", file name length: "0x00000007", file size: "0x00000000"
360         0x168       ASCII cpio archive (SVR4 with no CRC), file name:
"kernel/x86", file name length: "0x0000000B", file size: "0x00000000"
484         0x1E4       ASCII cpio archive (SVR4 with no CRC), file name:
"kernel/x86/microcode", file name length: "0x00000015", file size:  "0x00000000"
616         0x268       ASCII cpio archive (SVR4 with no CRC), file name:
"kernel/x86/microcode/GenuineIntel.bin", file name length: "0x00000026",
file size: "0x00005C00"
24316       0x5EFC      ASCII cpio archive (SVR4 with no CRC), file name:
"TRAILER!!!", file name length: "0x0000000B", file size: "0x00000000"
24576       0x6000      gzip compressed data, maximum compression,
from Unix, NULL date (1970-01-01 00:00:00)
...
```

El **microcódigo** de un microprocesador permite, al cargarse, modificar el funcionamiento de algunas instrucciones del mismo. Es el firmware de un procesador. El objetivo del microcódigo es corregir errores y problemas de seguridad.

Cuando se carga y monta initrd, el núcleo intenta ejecutar el script o el binario **init** presente en la raíz del seudosistema de archivos. Es el encargado de cargar los módulos necesarios y establecer la primera configuración básica completa, antes de que el núcleo monte el sistema de archivos raíz y ejecute `/sbin/init`. En la práctica, este archivo (o uno invocado por este) es el encargado de montar el sistema de archivos raíz y de pasarle el control a `/sbin/init`. En las distribuciones modernas, se carga systemd, en lugar de init, y será este el que se encargará de la continuación de la carga.

Observe a continuación el contenido (truncado) del archivo init, de una distribución Ubuntu 16.10. La variable init contiene la ruta de /sbin/init. Al final del script, el comando se ejecuta y se convierte en el PID1. Si esta etapa no funciona, el sistema operativo entra en modo panic.

```
...
export init=/sbin/init
...
exec run-init ${drop_caps} ${rootmnt} ${init} "$@" ${recovery:+--startup-
event=recovery} <${rootmnt}/dev/console >${rootmnt}/dev/console 2>&1
echo "Something went badly wrong in the initramfs."
panic "Please file a bug on initramfs-tools."
```

En algunas distribuciones, en particular las antiguas, se invoca a un script llamado `/sbin/init`. Se ha suprimido esto para evitar confundir el script `/sbin/init` de initrd con el "verdadero" init del sistema. Si un archivo **linuxrc** está en la raíz del initrd, se ejecuta. En caso de duda en cuanto al nombre del script iniciado, se puede ver en las fuentes del núcleo `init/main.c`, variable `ram_execute_command`.

```
$ cd /usr/src/linux
$ grep -n ram_execute_command init/main.c
...
865:        ramdisk_execute_command = "/init";
...
```

El comando **mkinitrd** (Red Hat, SuSE), o **mkinitramfs** (Debian, Ubuntu) permite volver a construir un archivo initrd estándar, pero a menudo es posible, en función de la distribución, facilitarle una lista de módulos para colocarlos en él, y así cargarlos:

- En Red Hat, se puede utilizar varias veces la opción **--preload=module**, que específica cuáles son los módulos que se deben cargar.
- En OpenSUSE, le corresponde a la opción **-m "module1 module2 etc."** especificar esta lista. Veremos el método con la utilización de las variables de sysconfig.
- En Debian, hay que colocar los nombres de los módulos en el archivo

`/etc/mkinitrd/modules` .

Tenemos el resultado siguiente para un comando **mkinitrd** en Fedora 31. La sintaxis y la salida cambiarán según la distribución. Observe las dos líneas en negrita: el initrd albergará los módulos necesarios para el arranque, aunque también el famoso microcódigo insertado al principio del archivo.

```
# mkinitrd /boot/initramfs-5.4.12-200.fc31.x86_64.img 5.4.12-200.fc31.x86_64 --force -v
Creating: target|kernel|dracut args|basicmodules
dracut: Executing: /usr/bin/dracut -v -f /boot/initramfs-5.4.12-200.fc31.x86_64.img
5.4.12-200.fc31.x86_64
dracut: dracut module 'busybox' will not be installed, because command 'busybox'
could not be found!
...
dracut: *** Including module: lvm ***
dracut: *** Including module: systemd ***
dracut: *** Including module: systemd-initrd ***
dracut: *** Including module: nss-softokn ***
dracut: *** Including module: lvm ***
...
dracut: *** Installing kernel module dependencies ***
dracut: *** Installing kernel module dependencies done ***
dracut: *** Resolving executable dependencies ***
dracut: *** Resolving executable dependencies done ***
dracut: *** Hardlinking files ***
dracut: *** Hardlinking files done ***
dracut: *** Stripping files ***
dracut: *** Stripping files done ***
dracut: *** Generating early-microcode cpio image ***
dracut: *** Constructing GenuineIntel.bin ***
dracut: *** Store current command line parameters ***
dracut: *** Creating image file '/boot/initramfs-5.4.12-200.fc31.x86_64.img' ***
dracut: *** Creating initramfs image file '/boot/initramfs-5.4.12-200.fc31.x86_64.img' done ***
```

Un initrd construido correctamente (o más bien un comando **mkinitrd** correcto) recupera también la salida del comando **modprobe** y el archivo de configuración `modprobe.conf` , que puede (y debe) contener los parámetros de los módulos si es necesario.

Debian y Ubuntu ya no utilizan **mkinitrd**, sino **mkinitramfs**. El principio es el mismo. **initramfs** dispone de su propia configuración situada en **/etc/initramfs-tools/***. Los módulos que se cargarán en el arranque se especifican en **/etc/initramfs-tools/modules**. A partir de Ubuntu 19.10, el formato de compresión pasa de XZ (gzip) a LZ4. Para crear un **initrd** para el núcleo actual, utilice el comando siguiente:

```
# mkinitramfs -o /tmp/initrd-$(uname -r) -v
Adding module /lib/modules/5.3.0-26-generic/kernel/drivers/usb/host/fsl-mph-dr-of.ko
Adding module /lib/modules/5.3.0-26-generic/kernel/drivers/usb/host/oxu210hp-hcd.ko
Adding module /lib/modules/5.3.0-26-generic/kernel/drivers/ssb/ssb.ko
...
Calling hook intel_microcode
intel-microcode: adding microcode for either all or selected Intel
processor models
intel-microcode: using early initramfs microcode update mode...
...
```

b. Módulos persistentes

Caso general con systemd

Con systemd, cualquiera que sea la distribución, usted crea los archivos de configuración en el directorio **/etc/modules-load.d**, bajo el nombre `<archivo>.conf` . Estos archivos contienen la lista de módulos adicionales que se han de cargar durante el arranque por el servicio systemd-modules-load.service.

```
$ cat /etc/modules-load.d/drbd.conf
drbd
```

Red Hat

Hasta Hat 5, el método preferido en Red Hat consiste en cargar los módulos desde el **initrd**. Sin embargo, tiene la posibilidad crear un script `/etc/rc.modules` que contendrá los comandos necesarios para la carga de los módulos, principalmente con **modprobe**, como vimos antes. Se ejecuta este archivo con rc.sysinit al iniciar el sistema,

mediante **inittab**. Debe ser ejecutable.

En Red Hat 6, usted puede añadir los módulos creando scripts shell llamados `<nombre>.modules` en el directorio **/etc/sysconfig/modules**. El formato es el mismo que para rc.modules.

OpenSUSE y SLES

La distribución OpenSUSE y las distribuciones SLES previas a systemd ubican su configuración en un directorio `/etc/sysconfig`. El archivo `/etc/sysconfig/kernel` contiene unos elementos de configuración del núcleo, pero, sobre todo dos variables encargadas de especificar los módulos que se deben cargar:

- **INITRD_MODULES** es la lista de los módulos cargados por el initrd.
- **MODULES_LOADED_ON_BOOT** es la lista de módulos cargados por init (por lo tanto, después de initrd).

Verá una diferencia entre la lista de los módulos indicada para el initrd y la lista real cargada durante su creación. El comando **mkinitrd** de OpenSUSE permite especificar funcionalidades preestablecidas durante la creación del **initrd** (opción **-f "feature1 feature2 etc"**). Por ejemplo, la funcionalidad "usb" cargará todos los módulos relacionados con el soporte USB, lo que resulta vital si dispone de un teclado USB o si inicia el sistema desde un dispositivo USB o un disco externo.

Debian y Ubuntu

En las versiones Debian y Ubuntu anteriores a systemd, bastaba con añadir los nombres de los módulos que se deben cargar en el archivo `/etc/modules`. Después del nombre de los módulos, puede indicar parámetros. Para cada línea se ejecuta un modprobe. Puede añadir un comentario después de una #. Es el archivo `/etc/init.d/modutils` o el archivo de configuración upstart `/etc/init/kmod.conf` (Ubuntu) quien carga los módulos en el momento del inicio mediante init System V, o `/etc/init/module-init-tools.conf` (Debian).

```
# /etc/modules: kernel modules to load at boot time.
#
```

```
# This file contains the names of kernel modules that should
# be loaded at boot time, one per line. Lines beginning with
#  "#" are ignored.
# Parameters can be specified after the module name.

lp
rtc
```

5. Parámetros dinámicos

a. /proc y /sys

`/proc` y **`/sys`** son sistemas de archivos virtuales que contienen información relativa al núcleo en curso de ejecución. La versión 2.4 del núcleo sólo conoce **`/proc`**, donde se agrupa toda la información. La versión 2.6 del núcleo ha modificado la función de **`/proc`** para delegar una parte de sus tareas a **`/sys`**.

Al tratarse de sistemas de archivos virtuales, no ocupan espacio ni en la memoria, ni en cualquier disco. Uno no debe dejarse engañar por el tamaño de los seudoarchivos que contiene. ¡No intente suprimir **`/proc/kcore`** para ganar espacio! Se puede leer y visualizar directamente todos estos archivos (o casi).

Los archivos de **`/proc`** le proporcionarán mucha información sobre el sistema:

- **`interrupts`**: los parámetros IRQ.
- **`cpuinfo`**: detalles sobre sus procesadores.
- **`dma`**: los parámetros DMA.
- **`ioports`**: los puertos de memoria de E/S.
- **`devices`**: los periféricos presentes.
- **`meminfo`**: el estado global de la memoria.
- **`loadavg`**: la carga del sistema.
- **`uptime`**: tiempo transcurrido desde el arranque en segundos.

- **`version`**: detalles de la versión de Linux.
- **`modules`**: idéntico al resultado de lsmod.
- **`swaps`**: lista y estado de las particiones de intercambio.
- **`partitions`**: lista y estado de las particiones conocidas del sistema.
- **`disktats`**: estadísticas de entrada/salida de los periféricos de tipo bloque.
- **`mounts`**: montaje de los sistemas de archivos.
- **`swaps`**: lista de particiones de intercambio, con su tamaño y prioridad.
- **`pci`**: detalles del bus PCI.

```
# cat /proc/cpuinfo

processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 6
model name      : Common KVM processor
stepping        : 1
microcode       : 0x1
cpu MHz         : 2993.198
cache size      : 16384 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 syscall nx lm constant_tsc nopl xtopology cpuid
tsc_known_freq pni cx16 x2apic hypervisor lahf_lm cpuid_fault pti
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs itlb_multihit
bogomips        : 5986.39
```

```
clflush size    : 64
cache_alignment : 128
address sizes   : 40 bits physical, 48 bits virtual
power management:
$ cat /proc/version
Linux version 4.19.0-16-amd64 (debian-kernel@lists.debian.org) (gcc version 8.3.0
(Debian 8.3.0-6)) #1 SMP Debian 4.19.181-1 (2021-03-19)
```

/proc contiene subdirectorios que agrupan información por tema.

- **`/proc/scsi`**: información sobre el bus SCSI.
- **`/proc/ide`**: información sobre el bus IDE.
- **`/proc/net`**: información sobre la red.
- **`/proc/sys`**: parámetros y configuración dinámica del núcleo.
- **`/proc/<PID>`**: información sobre el proceso PID.

Algunas entradas de los sistemas de archivos **`/proc/sys`** y **`/sys`** son diferentes de los demás, ya que se puede modificar su contenido y el núcleo tiene en cuenta las modificaciones directamente, sin que sea preciso iniciar de nuevo la máquina.

Por ejemplo, veamos cómo activar el forwarding IP y pasar el número de manejadores de archivos (número máximo de archivos abiertos en un instante dado) en 200000.

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
# echo "200000" > /proc/sys/fs/file-max
```

He aquí otro ejemplo relacionado con el capítulo anterior y la gestión de la memoria alta y baja. En los antiguos núcleos (por ejemplo 2.6.18) se podía modificar el ratio de protección de la memoria baja lowmem modificando **lower_zone_protection**. Se indica que el sistema debe intentar proteger lo mejor posible una zona de 300 MB de la memoria lowmem.

```
# echo 300>/proc/sys/vm/lower_zone_protection
```

Los núcleos recientes usan **lowmem_reserve_ratio**. Los tres valores se expresan en ratio 1/n de páginas de memoria que se deben proteger por zona de memoria. Para reservar más páginas, basta con reducir los valores:

```
# echo "128 128 32" >/proc/sys/vm/lowmem_reserve_ratio
```

El sistema de archivos `/sys` permitirá acceder o modificar la información sobre la vista del "núcleo" del sistema.

```
# ls -1
block
bus
class
dev
devices
firmware
fs
hypervisor
kernel
module
power
```

Los valores hablan por sí solos. El repertorio `block` contiene la lista de los periféricos de tipo bloque. Pero como los periféricos, sean los que sean, están listados en la carpeta devices, el repertorio `block` contendrá archivos de enlaces simbólicos hacia estos últimos.

```
lrwxrwxrwx 1 root root 0 abr 26 22:10 sda -> ../devices/pci0000:
00/0000:00:05.0/virtio1/host2/target2:0:0/2:0:0:0/block/sda
```

En `class` se encuentran los distintos periféricos ordenados por tipo o clase. De esta manera, si mira la línea host0, encontrará el periférico asociado a sda:

```
# ls -l scsi_host/
```

```
total 0
lrwxrwxrwx 1 root root 0 abr 26 22:10 host0 -> ../../devices/pci0000:
00/0000:00:01.1/ata1/host0/scsi_host/host0
lrwxrwxrwx 1 root root 0 abr 26 22:10 host1 -> ../../devices/pci0000:
00/0000:00:01.1/ata2/host1/scsi_host/host1
lrwxrwxrwx 1 root root 0 abr 26 22:10 host2 -> ../../devices/pci0000:
00/0000:00:05.0/virtio1/host2/scsi_host/host2
```

Finalmente, para hacer el vínculo con la secuencia de arranque y especialmente EFI, observe el contenido de `firmware`. Si un directorio `efi` se encuentra presente, entonces ha arrancado desde un firmware UEFI, en modo efi. Es en ese momento que el comando **efibootmgr** leerá y modificará las entradas:

```
root@ubuntuuefi:/sys/firmware/efi# ls -ld efivars/Boot*
-rw-r--r-- 1 root root 66 abr 2 17:41 efivars/Boot0000-8be4df61-
93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root 118 abr 2 17:41 efivars/Boot0001-8be4df61-
93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root 138 abr 2 17:41 efivars/Boot0002-8be4df61-
93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root  92 abr 2 17:41 efivars/Boot0003-8be4df61-
93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root 122 abr 2 17:41 efivars/Boot0004-8be4df61-
93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root 6 abr 2 17:41 efivars/BootCurrent-8be4df61-
93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root 8 abr 2 17:41 efivars/BootOptionSupport-
8be4df61-93ca-11d2-aa0d-00e098032b8c
-rw-r--r-- 1 root root 14 abr 2 17:41 efivars/BootOrder-8be4df61-
93ca-11d2-aa0d-00e098032b8c
```

b. sysctl

Los valores modificados en caliente no se registran. En caso de un nuevo inicio, hay que empezar de nuevo. El archivo `rc.sysinit` o, con systemd, el servicio `systemd-sysctl`, llamando al comando **sysctl**, que actúa sobre estos parámetros. Para que los valores se queden permanentes (nueva instalación a cada inicio), hay que modificar el

archivo **`/etc/sysctl.conf`** o crear/modificar uno de los archivos ubicados en **`/etc/sysctl.d/*.conf`** . Puede buscar las modificaciones manualmente.

```
# sysctl -e -p /etc/sysctl.conf

# sysctl -a
...
dev.raid.speed_limit_max = 100000
dev.raid.speed_limit_min = 100
net.token-ring.rif_timeout = 60000
net.ipv4.conf.eth1.arp_filter = 0
net.ipv4.conf.eth1.tag = 0
net.ipv4.conf.eth1.log_martians = 0
net.ipv4.conf.eth1.bootp_relay = 0
net.ipv4.conf.eth1.proxy_arp = 0
net.ipv4.conf.eth1.accept_source_route = 1
net.ipv4.conf.eth1.send_redirects = 1
net.ipv4.conf.eth1.rp_filter = 1
net.ipv4.conf.eth1.shared_media = 1
net.ipv4.conf.eth1.secure_redirects = 1
...
```