

Recompilación del kernel para no expertos

ASUNTOS DEL KERNEL

¿Preocupado por un agujero de seguridad recién descubierto? ¿Quieres sacar ventaja de alguna nueva funcionalidad hardware? No necesitamos ser un experto en Linux para parchear y compilar un kernel. Te mostramos el camino. **POR PETER KREUSSEL**

El kernel administra la memoria de las aplicaciones, controla el acceso al sistema de archivos y ejecuta otras importantes tareas. Pero para la mayoría de los usuarios está totalmente escondido tras un shell o una interfaz gráfica, es decir, el usuario medio no tiene generalmente que lidiar con el kernel.

El kernel de Linux, sin embargo, es bastante accesible. Se ubica en el archivo `/boot/vmlinuz-kernelversion`. En función de nuestra distribución, puede ocupar de 1 a 2MB en disco. Pero este archivo único es sólo parte de la película. Los archivos ubicados en `/lib/modules/[kernelversion]` son también igualmente parte del kernel. Este directorio ocupa más de 70MB en Suse 10.0, pero no todo su contenido requiere espacio en la memoria. De hecho, los módulos de este directorio se cargan bajo demanda.

La práctica totalidad de las distribuciones incorporan binarios ya preparados

para el núcleo y los módulos, sin embargo, es posible compilarlo exactamente igual que cualquier otro programa. Para ello, necesitamos las fuentes del kernel y el GNU C Compiler.

Es Hora De Algo Nuevo

Los kernels específicos de cada distribución funcionan habitualmente sin demasiados problemas, pero en algunas situaciones, puede que nos encontremos con la necesidad de compilar uno nuevo. Por ejemplo, si tenemos aplicaciones críticas en cuanto a seguridad, puede que queramos recompilar el kernel si se han descu-

bierto vulnerabilidades. Y en algunos casos las nuevas versiones mejoran el soporte para hardware novedoso.

Puede que tengamos la ocasión de trabajar con parches del kernel. Los parches nos dan la oportunidad de añadirle nuevas funcionalidades. Un ejemplo de esto es SoftwareSuspend2 [1]. Este parche, que es realmente útil para los usuarios de portátiles, consigue que funcione la suspensión en sistemas donde el núcleo normal falla.

Llevar el seguimiento del desarrollo del kernel no es una tarea sencilla debido a los numerosos cambios entre las versiones. El changelog de Kernel-newbies.org [2] nos ofrece una visión general. Si estamos recompilando el kernel para solucionar un problema de hardware, es más probable que encontremos información del problema buscando por el dispositivo hardware que estamos intentando arreglar. Si en el foro encontramos que nuestra tarjeta DVB funciona con el kernel 2.6.13, pero tenemos el 2.6.12 en nuestro sistema,

Tabla 1: Funcionalidades del Kernel

Administrador de recursos (memoria, ciclos de CPU, etc.)
Acceso al sistema de archivos
Acceso a redes
Acceso a componentes hardware

Menú Principal de Configuración del Kernel

Entrada del menú	Función	Configuración por defecto
Opciones de nivel de madurez del código	Permiten la selección de drivers experimentales	Habilitado
Tipo de procesador y funcionalidades	Configuración de la arquitectura del procesador	Normalmente, sólo <i>Processor family</i> necesitará modificarse.
Opciones de administración de energía	Útil en portátiles	Puede ser habilitado sin ningún peligro
Redes	Sólo para expertos (p.ej: soporte para Bluetooth e infrarrojos)	Dejar los valores por defecto
Drivers de dispositivos	Configuración de drivers para dispositivos hardware	Modificar según se requiera
Sistemas de archivos	Soporte para tipos de sistemas de archivos	Puede necesitar modificaciones si estamos funcionando sin un disco Ram inicial

puede ser la señal para preparar el compilador.

Compilar un nuevo kernel no es un trabajo arriesgado. El bootloader nos permite elegir entre diferentes. Tras instalarlo, nada nos impide arrancar la versión antigua. La Figura 1 nos muestra los pasos para conseguir una compilación del kernel hecha por uno mismo.

Fuentes del Kernel

Nuestra distribución puede tener un paquete con las fuentes del kernel actualizadas, pero el último disponible estará siempre en Kernel.org. Como esta página sufre habitualmente problemas de capacidad, debemos asegurarnos que usamos un mirror [3]. Descargamos el que posea el número de versión más alto, y desempaquetamos el archivo en */usr/src*.

Debemos tener cuidado, pues el kernel a secas puede diferir bastante del kernel de nuestra distribución. Las distribuciones normalmente incluyen un buen número de parches que añaden funcionalidades, mejoran el soporte de los drivers y resuelven fallos conocidos.

La pregunta de si es el kernel a secas de Kernel.org o la versión específica de la distribución la que proporciona el mejor rendimiento en cuanto a estabilidad es casi una cuestión de fe. Ninguna de las dos debería causarnos problemas

Parchear el Kernel

Copiamos el parche a */usr/src* y desempaquetamos los parches acabados en *.gz* con *gunzip*. A continuación nos ubicamos en el directorio del kernel y usamos la misma herramienta de parcheo para aplicarlo al kernel:

```
$ cd linux-2.6.[Version]
$ patch -p1 <../[Patchfile]
```

importantes. Dicho esto, si queremos la versión más reciente, puede que no tengamos alternativa a la versión oficial.

En este artículo hemos trabajado con la versión oficial del equipo de desarrollo del kernel. Una alternativa es el llamado árbol *mm*, que contiene las mejoras actuales del proceso de desarrollo. Estas mejoras pueden ser tanto arreglos de fallos como extensiones que necesitan pasar por un testeo final antes de ser adoptadas por el kernel estándar. Andrew Morton lanza el árbol *mm* como un parche del kernel estándar. El cuadro "Parcheo del Kernel" describe cómo aplicar un parche al kernel de Linux.

Problemas para Elegir

El kernel de Linux tiene una estructura modular: varias funcionalidades (por ejemplo el soporte para un sistema de archivos específico o una tarjeta de sonido concreta) pueden ser habilitadas o deshabilitadas cuando lo compilamos. Cuantas más funcionalidades habilitemos, mayor se hará, al menos en teoría. El tiempo de compilación crecerá si usamos la carga de funcionalidades bajo demanda.

Muchas características del kernel pueden serle añadidas como módulos o compiladas en su interior. Sólo los componentes compilados quedarán residentes en memoria, mientras que el kernel puede cargar los módulos cuando se necesiten. Esta es la razón por la que no hay mucha diferencia a efectos prácticos si habilitamos las funcionalidades del kernel que puede que no necesitemos (véase Figura 2).

Preparación

Como las distribuciones de Linux se diseñan para ejecutarse en hardware de todo tipo, es habitual incluir casi cual-

quier cosa que pueda ofrecer el kernel en forma de módulos, con la excepción de algunos componentes extremadamente raros o problemáticos.

La mayoría de las distribuciones guardan la configuración del kernel estándar en */boot/config-kernelversion*. El administrador puede verificar la versión del kernel actual tecleando *uname -r*. Los usuarios de Suse no encontrarán la configuración en */boot*. En su lugar, el kernel actual genera dinámicamente un archivo *config.gz* en */proc*. Copiamos este archivo al directorio de kernel, ejecutamos *gunzip config.gz* para desempaquetar el archivo y lo renombramos como *.config*.

Ejecutamos todos los comandos *make* como root y desde dentro del directorio */usr/src/linux-kernelversion*. Comenzamos borrando los rastros del proceso de desarrollo del árbol del kernel que acabamos de desempaquetar tecleando *make mrproper*. Copiamos el archivo de configuración al árbol del kernel con *cp /boot/config-kernelversion*

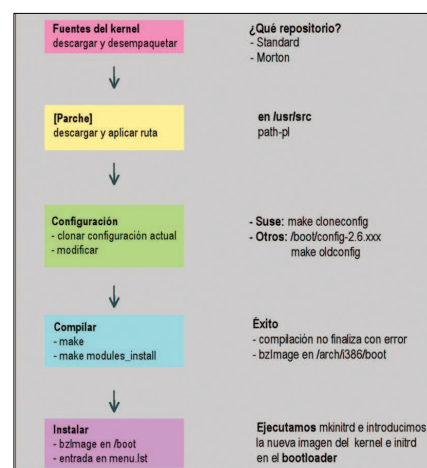


Figura 1: Paso a paso hasta llegar al nuevo kernel. No tenemos que ser unos expertos para compilar nuestro propio kernel.

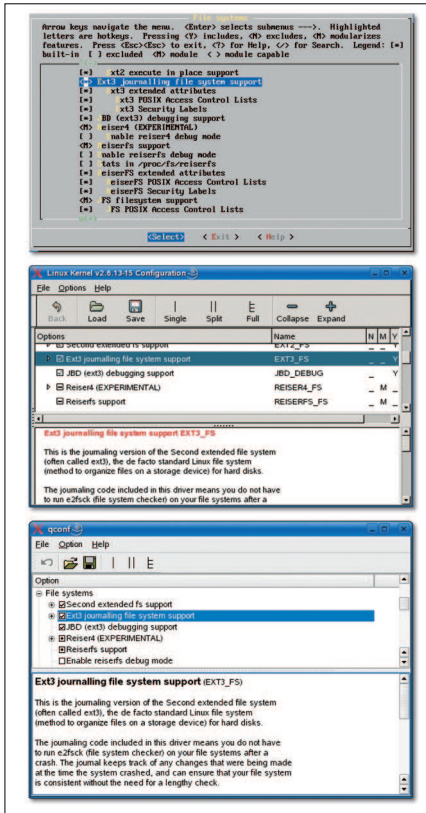


Figura 2: La misma configuración del kernel con varias herramientas de configuración. El driver para el sistema de archivos Ext3 se ha compilado dentro del kernel, mientras que Reiserfs se usa como módulo.

`/usr/src/linux-kernelversion/.config`, donde `kernelversion` es la versión del kernel de Linux que estamos compilando.

El kernel incluye varios front-end de configuración basados en Qt, en Gtk y en consola. Estas herramientas nos ayudan a seleccionar las funcionalidades “hágalo usted mismo” que ofrece la compilación del kernel. El menú de configuración, que tiene las mismas entradas sin importar qué front-end hayamos elegido, contiene varios cientos de configuraciones. Una perspectiva desalentadora a primera vista.

Compilar el Kernel en Debian y Ubuntu

Los usuarios de Debian y Ubuntu no deberían usar el comando `make` para compilar el kernel. Deberían usar `make-kpkg` en su lugar. Este comando supone que tenemos el `kernel-package`. Si necesitamos un disco Ram inicial, debemos añadir el parámetro `—initrd`.

Acabamos de copiar los archivos de configuración, y tendremos un kernel preconfigurado tras ejecutar la herramienta de configuración tecleando `make menuconfig` (en la versión consola), `make gconfig` (versión Gtk) o `make xconfig` (versión Qt) (véase Figura 2). La versión de consola requiere `ncurses-devel` y es considerada generalmente como la más fiable.

Si copiamos un archivo `.config` de una versión anterior del kernel a nuestro árbol de fuentes, debemos asegurarnos de que ejecutamos Menuconfig u otra herramienta de configuración al menos una vez antes de lanzarnos a compilar. Esto añade las nuevas opciones de configuración de la última versión del kernel y fija las opciones por defecto.

Ahora salimos de la configuración sin cambiar ningún parámetro. Para salir de la configuración, seleccionamos `Exit` en Menuconfig o Xconfig y la herramienta nos preguntará si queremos guardar la configuración. Decimos que sí. Si esta-

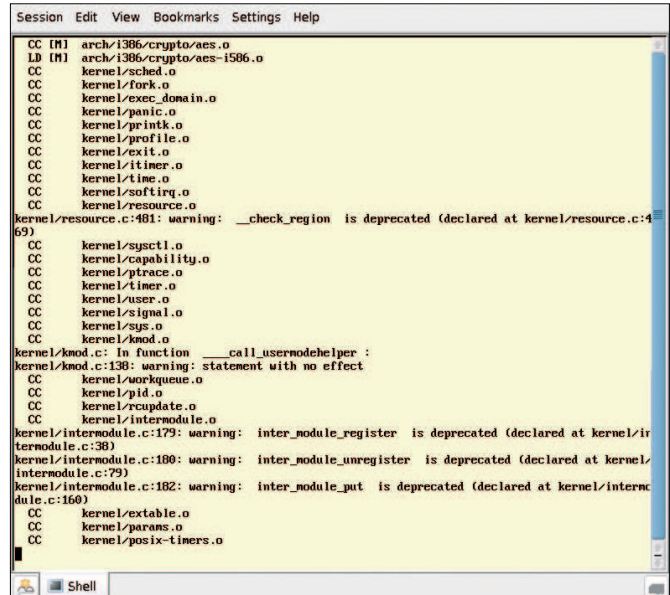


Figura 3: Generalmente podemos ignorar los avisos durante la compilación, pero la palabra “error” indica que algo ha ido mal.

mos trabajando con Gconfig, tendremos que guardar la configuración nosotros mismos. Si no lo hacemos así, los cambios serán ignorados.

Tras crear el nuevo kernel y arrancar por primera vez desde éste, ejecutamos de nuevo la herramienta de configuración y modificamos la configuración del kernel una vez más en caso necesario.

Derecho a Casa

Llegados a este punto, lo único que nos separa de un kernel listo para usar es hacer un `make` (y esperar de 5 a 30

Listado 1: Entradas de menu.lst para Ubuntu y Suse

```

01 title Ubuntu,
   kernel 2.6.12-9-386
02 root (hd0,0)
03 kernel
   /boot/vmlinuz-2.6.12-9-386
04 root=/dev/hdc1 ro quiet splash
05 initrd /boot/
   initrd.img-2.6.12-9-386
06 title SUSE LINUX 10.0
07 root (hd0,8)
08 kernel /boot/vmlinuz
   root=/dev/
09 hdc9 vga=0x31a selinux=0
   resume=/
10 dev/hdc3 splash=silent showopts
11 initrd /boot/initrd
    
```

Recursos

- [1] Parche Software Suspend2 (parche del kernel con funcionalidad alternativa de suspensión): <http://www.suspend2.net/>
- [2] Changelog del Kernel en formato “para humanos”: <http://kernelnewbies.org/LinuxChanges>
- [3] Mirrors for downloading the Kernel sources: <http://kernel.org/mirrors/countries/html/DE.html>
- [4] Parche Bootsplash: <http://bootsplash.org/> y <http://www.bootsplash.de/>
- [5] HOWTO detallado sobre compilación del kernel de Linux: <http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html>
- [6] Consejos e información del kernel de Linux, no sólo para geeks: <http://kernelnewbies.org/>

minutos). Durante la compilación, veremos muchos mensajes en pantalla. Se suelen ignorar los warnings (véase Figura 3). Hay algunas excepciones para los usuarios de Debian y Ubuntu (véase el cuadro).

Si la compilación se completa sin que aparezca la DREADED palabra *error* por pantalla, encontraremos un nuevo kernel en el árbol del código fuente de este, en `/arch/processor-architecture/boot` bajo la forma de un archivo *bzImage*. Para un PC, esto es `/arch/i386/boot`. Esto también se aplica para los kernels *amd_64*. Copiamos el archivo *bzImage* al directorio `/boot`. Renombramos el archivo como *vmlinuz-kernelversion*. Copiamos también el archivo *system.map* en `/boot/system.map-kernelversion`. Ahora ejecutamos *make modules_install* para instalar estas partes del kernel que hemos configurado como módulos.

Todo lo que tenemos que hacer es añadir una entrada para el nuevo kernel en el menú de inicio del bootloader. Si nuestra distro usa Grub, como la mayoría de las distros hoy día, podemos editar el archivo `/boot/grub/menu.lst` para ello. Si tenemos Lilo, las entradas están en `/etc/lilo.conf`.

Activo y Corriendo

A la hora de arrancar encontramos el clásico problema del huevo y la gallina: para leer el sistema de archivos, el kernel necesita los módulos que residen dentro del sistema de archivos del disco. Muchas distros optan por el llamado disco Ram inicial para solventar el problema. El kernel monta temporalmente el disco Ram inicial como archivo de sistema raíz y el bootloader pone el contenido del disco virtual a disposición del kernel. Éste encuentra

los módulos en el disco Ram. Una alternativa sería compilar el sistema de archivo y los drivers del disco duro dentro del kernel.

Todo lo que necesitamos para generar nuevos discos Ram para todas las imágenes de kernel de `/boot` es ejecutar *mkinitrd* sin más parámetros. Los detalles de *mkinitrd* pueden variar, por lo que es una buena idea leerse las páginas man. Si todo ha ido bien, tendremos una nueva imagen *initrd-[kernelversion]* en el directorio `/boot`.

El Listado 1 muestra un archivo *menu.lst* con entradas para un sistema Ubuntu y otro Suse. La línea *root* especifica la partición con la imagen del kernel. La sintaxis *hd(0,0)* refiere al disco duro, primera partición, es decir: *hda*. Las siguientes dos líneas designan la imagen del kernel y la imagen del disco Ram inicial. Tras la imagen del kernel tenemos los parámetros a pasar al kernel en el momento del arranque.

A pesar de que *menu.lst* pueda parecer compleja, es realmente fácil crear un menú de entradas de Grub para un nuevo kernel. Simplemente copiamos la entrada estándar de nuestra distribución, y modificamos los nombres de los archivos del kernel y el disco Ram inicial, junto con el *title*. No debemos borrar las entradas ya existentes.

Ahora reiniciamos nuestra máquina, seleccionamos el nuevo kernel desde la pantalla de arranque y vigilamos algún posible mensaje de error. Con el kernel a secas no veremos una pantalla gráfica bootsplash, pero siempre podremos añadir el parche [4] más tarde.

Ajuste Fino

Si el nuevo kernel consigue arrancar con la configuración que ha heredado

del kernel anterior, sólo tendremos que modificar algunas configuraciones (por ejemplo habilitar los drivers añadidos en la nueva versión). Si recompilamos el kernel, sólo las partes afectadas por los cambios serán recompiladas.

La tabla “Menú Principal de Configuración del Kernel” nos ofrece una visión general de las entradas críticas en el nivel superior del menú de configuración del kernel. Los puntos que faltan en la tabla están reservados normalmente para expertos. Por ejemplo, no suele ser una buena idea cambiar el *Networking Support*, a menos que realmente sepamos mucho acerca del soporte de red para el kernel. Eliminar los drivers que no necesitamos ha dejado de ser un problema, generalmente, con los kernels 2.6. Por contra, los 2.4 compilados solían fallar debido a interdependencias de drivers que la herramienta de configuración etiquetaba como no resueltas.

No debemos habilitar las opciones de depuración, ya que suelen provocar problemas serios de rendimiento, y debemos crear siempre una copia de seguridad del archivo `.config` antes de intentar modificar la configuración de un kernel que funciona.

A Toda Máquina

Si el sistema funciona como se esperaba, podemos modificar el valor de la entrada por defecto de *menu.lst* para hacer de nuestro nuevo kernel la opción de arranque por defecto. El HOWTO de compilación del kernel [5] tiene más detalles sobre cómo lidiar con el kernel de Linux. Y la página Kernelnewbies.org [6] tiene una enorme colección de información referida a asuntos relacionados con el núcleo. ■

Asociación Linux Español

