El núcleo Linux

Josep Jorba Esteve

PID_00174426

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

El núcleo (en inglés *kernel*) del sistema GNU/Linux (al que habitualmente denominaremos Linux) [Vasb], es el corazón del sistema: se encarga de arrancarlo y, una vez este es ya utilizable por las aplicaciones y los usuarios, se encarga de gestionar los recursos de la máquina, en forma de gestión de la memoria, del sistema de ficheros, de las operaciones de entrada/salida y de los procesos y su intercomunicación.

Su origen se remonta al año 1991, cuando en agosto, un estudiante finlandés llamado **Linus Torvalds** anunció en una lista de noticias, que había creado su propio núcleo de sistema operativo, funcionando conjuntamente con software GNU, y lo ofrecía a la comunidad de desarrolladores para que lo probara y sugiriera mejoras para hacerlo más utilizable. Este es el origen del núcleo del sistema operativo que más tarde se llamaría **GNU/Linux**.

Origen de Linux

El núcleo Linux se remonta al año 1991, cuando Linus Torvalds lo puso a disposición de la comunidad. Es de los pocos sistemas operativos, que siendo ampliamente usados, se dispone de su código fuente.

Una de las particularidades de Linux es que, siguiendo la filosofía de software libre, se nos ofrece el código fuente del núcleo del propio sistema operativo (del *kernel*), de manera que es una herramienta perfecta para la educación, en temas de análisis y diseño de sistemas operativos.

La otra ventaja principal es que, disponiendo de los archivos fuente, podemos recompilarlos, para adaptarlos mejor a nuestro sistema, y como veremos en el módulo "Sintonización, optimización y alta disponibilidad", configurarlos para dar un mejor rendimiento al sistema.

En este módulo veremos cómo manejar este proceso de preparación de un núcleo para nuestro sistema: cómo, partiendo de los archivos fuente, podemos obtener una nueva versión del núcleo adaptada a nuestro sistema. Veremos cómo se desarrolla la configuración, la posterior compilación y la realización de pruebas con el nuevo núcleo obtenido.

Además, veremos cómo el núcleo ha ido añadiendo toda una serie de características a lo largo de su evolución, que lo han convertido en competitivo frente a otros sistemas. En especial, observaremos algunas características de la virtualización que nos ofrecen con soporte desde el núcleo.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Conocer el funcionamiento del núcleo (en inglés, *kernel*) y de los procesos de configuración asociados.
2. Poder configurar el núcleo del sistema en las distribuciones más habituales.
3. Entender el uso de los módulos del núcleo y decidir su integración (o no) dentro de la parte estática del núcleo.
4. Conocer las técnicas de virtualización y, en particular, de las incluidas en el núcleo.
5. Saber adaptar el núcleo a las necesidades particulares del usuario.

1. El núcleo del sistema GNU/Linux

El núcleo o *kernel* es la parte básica de cualquier sistema operativo [Tan87], y sobre él descansa el código de los servicios fundamentales para controlar el sistema completo. Básicamente, su estructura puede separarse en una serie de componentes, o módulos de gestión orientados a:

- **Gestión de procesos**: qué tareas se van a ejecutar y en qué orden y con qué prioridad. Un aspecto importante es la planificación de la CPU: ¿cómo se optimiza el tiempo de la CPU para ejecutar las tareas con el mayor rendimiento o interactividad posible con los usuarios?

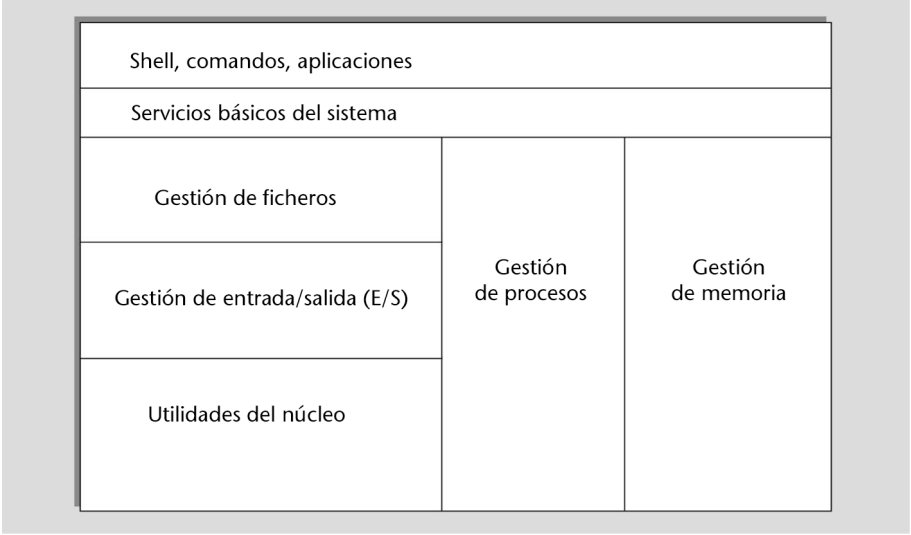
- **Intercomunicación de procesos y sincronización**: ¿cómo se comunican tareas entre sí, con qué diferentes mecanismos y cómo pueden sincronizarse grupos de tareas?

- **Gestión de entrada/salida (E/S)**: control de periféricos y gestión de recursos asociados.

- **Gestión de memoria**: optimización del uso de la memoria, sistema de paginación y memoria virtual.

- **Gestión de ficheros**: cómo el sistema controla y organiza los ficheros presentes en el sistema, y accede a los mismos.

Figura 1. Funciones básicas de un núcleo respecto a las aplicaciones y comandos ejecutados



En los sistemas privativos, el núcleo, o *kernel*, está perfectamente "oculto" bajo las capas del software del sistema operativo, y el usuario final no tiene una perspectiva clara de qué es ese núcleo ni tiene tampoco ninguna posibilidad de cambiarlo u optimizarlo, si no es por el uso de esotéricos editores de "registros" internos, o programas especializados de terceros (normalmente de alto coste). Además, el núcleo suele ser único, es el que proporciona el fabricante, el cual se reserva el derecho de introducir las modificaciones que quiera y cuando quiera, así como tratar los errores que aparezcan en plazos no estipulados, mediante actualizaciones que nos ofrece como "parches" de errores (o grupos de ellos denominados comúnmente *service packs*).

Uno de los principales problemas de esta aproximación es precisamente la disponibilidad de estos parches: disponer de las actualizaciones de los errores a su debido tiempo y, si se trata de problemas de seguridad, todavía con más razón, ya que hasta que no estén corregidos no podemos garantizar la seguridad del sistema para problemas ya conocidos. Muchas organizaciones, grandes empresas, gobiernos, instituciones científicas y militares no pueden depender de los caprichos de un fabricante para solucionar los problemas de sus aplicaciones críticas.

En este caso, el núcleo Linux ofrece una solución de código abierto, con los consecuentes permisos de modificación, corrección, posibilidad de generación de nuevas versiones y actualizaciones de forma rápida, por parte de cualquiera que quiera y tenga los conocimientos adecuados para realizarlo. Esto permite a los usuarios críticos controlar mejor sus aplicaciones y el propio sistema, y poder montar sistemas con el propio sistema operativo "a la carta", personalizado al gusto de cada uno. También permite disponer, a su vez, de un sistema operativo con código abierto, desarrollado por una comunidad de programadores coordinados mediante Internet y accesible ya sea para educación, por disponer del código fuente y abundante documentación, o para la producción final de los sistemas GNU/Linux adaptados a necesidades individuales o de un determinado colectivo.

Al disponer del código fuente, se pueden aplicar mejoras y soluciones de forma inmediata, a diferencia del software privativo, donde debemos esperar a las actualizaciones del fabricante. Podemos, además, personalizar el núcleo tanto como necesitemos, requisito esencial, por ejemplo, en aplicaciones de alto rendimiento, críticas en el tiempo o en soluciones con sistemas empotrados (como dispositivos móviles).

A continuación repasamos un poco la historia del núcleo [Kera, Kerb]. El núcleo Linux lo comenzó a desarrollar un estudiante finlandés llamado Linus Torvalds, en 1991, con la intención de realizar una versión parecida a MINIX [Tan87] (versión para PC de UNIX [Bac86]) para el procesador 386 de Intel. La primera versión publicada oficialmente fue la de Linux 1.0 en marzo de 1994, en la cual se incluía sólo la ejecución para la arquitectura i386 y soportaba

MINIX

El núcleo tiene sus orígenes en el sistema MINIX, desarrollado por Andrew Tanenbaum, como un clon de UNIX para PC.

máquinas de un solo procesador. Linux 1.2 fue publicado en marzo de 1995 y fue la primera versión en dar cobertura a diferentes arquitecturas, como Alpha, Sparc y Mips. Linux 2.0, en junio de 1996, añadió más arquitecturas y fue la primera versión en incorporar soporte multiprocesador (SMP) [Tum]. En Linux 2.2, de enero de 1999, se incrementaron las prestaciones de soporte SMP de manera significativa, y se añadieron controladores para gran cantidad de hardware. En la 2.4, en enero del 2001, se mejoró el soporte SMP, se incorporaron nuevas arquitecturas y se integraron controladores para dispositivos USB, PC Card (PCMCIA de los portátiles), parte de PnP (*plug and play*), soporte de RAID y volúmenes, etc. En la rama 2.6 del núcleo (diciembre de 2003), se mejoró sensiblemente el soporte SMP, se introdujo una mejor respuesta del sistema de planificación de CPU, el uso de hilos (*threads*) en el núcleo, mejor soporte de arquitecturas de 64 bits, soporte de virtualización y una mejor adaptación a dispositivos móviles.

Respecto al proceso de desarrollo, desde su creación por Linus Torvalds en 1991 (versión 0.01), el núcleo lo ha seguido manteniendo él mismo, pero a medida que su trabajo se lo permitía y a medida que el núcleo maduraba (y crecía), se ha visto obligado a mantener las diferentes versiones estables del núcleo gracias a diferentes colaboradores, mientras que Linus continua (en la medida de lo posible) desarrollando y recopilando aportaciones para la última versión de desarrollo del núcleo. Los colaboradores principales en estas versiones han sido [Lkm]:

- 2.0 David Weinehall.
- 2.2 Alan Cox (también desarrolla y publica parches para la mayoría de versiones).
- 2.4 Marcelo Tosatti.
- 2.5 Linus Torvalds.
- 2.6 Greg Kroah-Hartman (versiones estables) / Linus Torvalds, Andrew Morton (*releases* de desarrollo).

Complejidad del núcleo

El núcleo hoy en día ha alcanzado unos grados de madurez y complejidad significativos

Para ver un poco la complejidad del núcleo de Linux, veamos una tabla con un poco de su historia resumida en las diferentes versiones y en el tamaño respectivo del código fuente. En la tabla solo se indican las versiones de producción; el tamaño esta especificado en miles de líneas del código de los paquetes fuentes del núcleo:

Versión	Fecha de publicación	Líneas de código (en miles)
0.01	09-1991	10
1.0	03-1994	176
1.2	03-1995	311
2.0	06-1996	649
2.2	01-1999	1800
2.4	01-2001	3378
2.6	12-2003	5930

Como podemos comprobar, hemos pasado de unas diez mil líneas a seis millones en las primeras versiones de la rama 2.6; las últimas versiones de esta rama se mueven entre los diez a quince millones de líneas.

En estos momentos el desarrollo continúa en la rama 2.6.x del núcleo, la última versión estable, que incluyen la mayoría de distribuciones como versión principal (algunas todavía incluyen algunas 2.4.x, pero 2.6.x suele ser la opción por defecto en la instalación).

Aunque ahora la rama principal sea la 2.6.x, cierto conocimiento de las anteriores versiones es imprescindible, ya que con facilidad podemos encontrar máquinas con distribuciones antiguas que no se hayan actualizado, que es posible que debamos mantener o realizar un proceso de migración a versiones más actuales.

En la rama del 2.6, durante su desarrollo se aceleraron de forma significativa los trabajos del núcleo, ya que tanto Linus Torvalds como Andrew Morton (que mantienen varias de las ramas de Linux 2.6 en desarrollo) se incorporaron (durante 2003) al Open Source Developement Laboratory (OSDL), un consorcio de empresas cuyo fin es promocionar el uso de Open Source y GNU/Linux en la empresa (en el consorcio se encuentran, entre otras muchas empresas con intereses en GNU/Linux: HP, IBM, Sun, Intel, Fujitsu, Hitachi, Toshiba, Red Hat, Suse, Transmeta, etc.). En esos momentos se dio una situación interesante, ya que el consorcio OSDL hizo de patrocinador de los trabajos, tanto para el mantenedor de la versión estable del núcleo (Andrew) como para el de la de desarrollo (Linus), trabajando a tiempo completo en las versiones y en los temas relacionados. Linus se mantiene independiente, trabajando en el núcleo, mientras Andrew se fue a trabajar a Google, donde continuaba a tiempo completo sus desarrollos, realizando parches con diferentes y nuevas aportaciones al núcleo, en la que se conoce como rama de desarrollo `-mm`. Despues de cierto tiempo, OSDL se reconvirtió en la fundación The Linux Foundation.

Hay que tener en cuenta que con las versiones actuales del núcleo se ha alcanzado ya un alto grado de desarrollo y madurez, lo que hará que cada vez se amplíe más el tiempo entre la publicación de las versiones estables, no así de las revisiones parciales o de desarrollo, aspecto en el que los mantenedores esperan una nueva versión cada 2 o 3 meses.

Enlace de interés

Linux Foundation: http://www.linuxfoundation.org

Además, otro factor a considerar es el tamaño y el número de personas que están trabajando en el desarrollo actual. En un principio había unas pocas personas que tenían un conocimiento global del núcleo entero, mientras que hoy en día tenemos un importante número de personas que lo desarrollan (se cree que cerca de varios miles) con diferentes contribuciones, aunque el grupo duro se estima en unas pocas docenas de desarrolladores.

También cabe tener en cuenta que la mayoría de desarrolladores (de los miles) solo tienen unos conocimientos parciales del núcleo y, ni todos trabajan simultáneamente, ni su aportación es igual de relevante (algunas aportaciones solo corrigen errores sencillos). En el otro extremo, son unas pocas personas (como los mantenedores) las que disponen de un conocimiento total del núcleo. Esto supone que se puedan alargar los desarrollos y que se tengan que depurar las aportaciones, para comprobar que no entren en conflicto entre ellas, o que se deba escoger entre posibles alternativas de prestaciones.

Respecto a la numeración de las versiones del núcleo de Linux [Ker, Ces06], cabe tener en cuenta los aspectos siguientes:

1) Hasta la rama del núcleo 2.6.x, las versiones del núcleo Linux se regían por una división en dos series: una era la denominada "experimental" (con numeración impar en la segunda cifra, como 1.3.xx, 2.1.x o 2.5.x) y la otra era la de producción (serie par, como 1.2.xx, 2.0.xx, 2.2.x, 2.4.x y más). La serie experimental eran versiones que se movían rápidamente y se utilizaban para probar nuevas prestaciones, algoritmos, controladores de dispositivo, etc. Por la propia naturaleza de los núcleos experimentales, podían tener comportamientos impredecibles, como pérdidas de datos, bloqueos aleatorios de la máquina, etc. Por lo tanto, no estaban destinadas a utilizarse en máquinas para la producción, a no ser que se quisiese probar una característica determinada (con los consecuentes peligros).

Los núcleos de producción (serie par) o estables eran los núcleos con un conjunto de prestaciones bien definido, con un número bajo de errores conocidos y controladores de dispositivos probados. Se publicaban con menos frecuencia que los experimentales y existían variedad de versiones, unas de más o menos calidad que otras. Las distribuciones GNU/Linux se suelen basar en una determinada versión del núcleo estable, no necesariamente el último núcleo de producción publicado.

2) En la numeración actual del núcleo Linux (utilizada en la rama 2.6.x), se siguen conservando algunos aspectos básicos: la versión viene indicada por unos números X.Y.Z, donde normalmente X es la versión principal, que representa los cambios importantes del núcleo; Y es la versión secundaria, y habitualmente implica mejoras en las prestaciones del núcleo: Y es par en los núcleos estables e impar en los desarrollos o pruebas; Z es la versión de construcción, que indica el número de la revisión de X.Y, en cuanto a parches o correcciones hechas.

Los distribuidores no suelen incluir la última versión del núcleo, sino la que ellos hayan probado con más frecuencia y puedan verificar que es estable para el software y componentes que ellos incluyen. Partiendo de este esquema de numeración clásico (que se siguió durante las ramas 2.4.x hasta los inicios de la 2.6), hubo algunas modificaciones para adaptarse al hecho de que el núcleo (rama 2.6.x) se vuelve más estable (fijando X.Y a 2.6) y cada vez las

revisiones son menores (por significar un salto de versión de los primeros números), pero el desarrollo continuo y frenético sigue.

En los últimos esquemas se llega a introducir cuartos números, para especificar Z cambios menores, o diferentes posibilidades de la revisión (con diferentes parches añadidos que corrigen fallos). La versión así definida con cuatro números es la que se considera estable (*stable*). También se usan otros esquemas para las diversas versiones de prueba (normalmente no recomendables para entornos de producción), como sufijos `-rc` (*release candidate*), `-mm` que son núcleos experimentales con gran introducción de parches que suponen nuevas prestaciones adicionales como pruebas de diferentes técnicas novedosas, o los `-git` que son una especie de "foto" diaria del desarrollo del núcleo. Estos esquemas de numeración están en constante cambio para adaptarse a la forma de trabajar de la comunidad del núcleo y a sus necesidades para acelerar el desarrollo.

3) Para obtener el último núcleo publicado (que normalmente se denomina *vanilla* o *pristine*), hay que acudir al archivo de núcleos Linux (disponible en http://www.kernel.org) o al *mirror* local (en España http://www.es.kernel.org). También podrán encontrarse aquí algunos parches al núcleo original, que corrigen errores detectados *a posteriori* de la publicación del núcleo.

Enlace de interés

Repositorio de núcleos Linux:
http://www.kernel.org

Algunas de las características técnicas [Ces06, Kan] del núcleo Linux que podríamos destacar son:

- Núcleo de tipo monolítico: básicamente es un gran programa creado como una unidad, pero conceptualmente dividido en varios componentes lógicos.

- Tiene soporte para carga y descarga de porciones del núcleo bajo demanda; estas porciones se llaman *módulos* y suelen ser características del núcleo o controladores de dispositivo.

- Hilos de núcleo: Para el funcionamiento interno se utilizan varios hilos (*threads* en inglés) de ejecución internos al núcleo, que pueden estar asociados a un programa de usuario o bien a una funcionalidad interna del núcleo. En Linux no se hacía un uso intensivo de este concepto en origen, pero ha pasado a ser un concepto fundamental para el rendimiento, en especial debido a la aparición de las CPU *multicore*. En las diferentes revisiones de la rama 2.6.x se ofreció un mejor soporte, y gran parte del núcleo se ejecuta usando diversos hilos de ejecución.

- Soporte de aplicaciones multihilo: soporte de aplicaciones de usuario de tipo multihilo (*multithread*), ya que muchos paradigmas de computación de tipo cliente/servidor necesitan servidores capaces de atender múltiples peticiones simultáneas dedicando un hilo de ejecución a cada petición o grupo de ellas. Linux tiene una biblioteca propia de hilos que puede usarse

para las aplicaciones multihilo, con las mejoras que se introdujeron en el núcleo, que también han permitido un mejor uso para implementar bibliotecas de hilos para el desarrollo de aplicaciones.

- El núcleo es de tipo no apropiativo (*nonpreemptive*): esto implica que dentro del núcleo no pueden pararse llamadas a sistema (en modo supervisor) mientras se está resolviendo la tarea de sistema, y cuando ésta acaba, se prosigue la ejecución de la tarea anterior. Por lo tanto, el núcleo dentro de una llamada no puede ser interrumpido para atender a otra tarea. Normalmente, los núcleos apropiativos están asociados a sistemas que trabajan en tiempo real, donde debe permitirse lo anterior para tratar eventos críticos. Hay algunas versiones especiales del núcleo de Linux para tiempo real (ramas `-rt`, de *realtime*), que permiten esto por medio de la introducción de unos puntos fijos donde las tareas del núcleo pueden interrumpirse entre si. También se ha mejorado especialmente este concepto en la rama 2.6.x del núcleo, que en algunos casos permite interrumpir algunas tareas del núcleo, reasumibles, para tratar otras, prosiguiendo posteriormente su ejecución. Este concepto de núcleo apropiativo también puede ser útil para mejorar tareas interactivas, ya que si se producen llamadas costosas al sistema, pueden provocar retardos en las aplicaciones interactivas.

- Soporte para multiprocesador, tanto lo que se denomina *multiprocesamiento simétrico* (SMP) como *multicore*. Este concepto suele englobar máquinas que van desde el caso simple de 2 hasta 64 CPU colocadas en diferentes zócalos físicos de la máquina. Este tema se ha puesto de especial actualidad con las arquitecturas de tipo *multicore*, que permiten de 2 a 8 o más núcleos de CPU en un mismo zócalo físico, en máquinas accesibles a los usuarios domésticos. Linux puede usar múltiples procesadores, donde cada procesador puede manejar una o más tareas. Pero originalmente había algunas partes del núcleo que disminuían el rendimiento, ya que están pensadas para una única CPU y obligan a parar el sistema entero en determinados bloqueos. SMP es una de las técnicas más estudiadas en la comunidad del núcleo de Linux, y se han obtenido mejoras importantes en la rama 2.6. Del rendimiento SMP depende en gran medida la adopción de Linux en los sistemas empresariales, en la faceta de sistema operativo para servidores.

- Sistemas de ficheros: el núcleo tiene una buena arquitectura de los sistemas de ficheros, ya que el trabajo interno se basa en una abstracción de un sistema virtual (VFS, *virtual file system*), que puede ser adaptada fácilmente a cualquier sistema real. Como resultado, Linux es quizás el sistema operativo que más sistemas de ficheros soporta, desde su propio ext2 inicial, hasta msdos, vfat, ntfs, sistemas con *journal* como ext3, ext4, ReiserFS, JFS(IBM), XFS(Silicon), NTFS, iso9660 (CD), udf, etc. y se van añadiendo más en las diferentes revisiones del núcleo.

Otras características menos técnicas (un poco de *marketing*) que podríamos destacar:

1) Linux es gratuito: junto con el software GNU, y el incluido en cualquier distribución, podemos tener un sistema tipo UNIX completo prácticamente por el coste del hardware; y por la parte de los costes de la distribución GNU/Linux, podemos obtenerla prácticamente gratis. Pero no está de más pagar por una distribución completa, con los manuales y apoyo técnico, a un coste menor comparado con lo que se paga por algunos sistemas privativos, o contribuir con la compra al desarrollo de las distribuciones que más nos gusten o nos sean prácticas.

2) Linux es personalizable: la licencia GPL nos permite leer y modificar el código fuente del núcleo (siempre que tengamos los conocimientos adecuados).

3) Linux se ejecuta en hardware antiguo bastante limitado; es posible, por ejemplo, crear un servidor de red con un 386 con 4 MB de RAM (hay distribuciones especializadas en bajos recursos).

4) Linux es un sistema de altas prestaciones: el objetivo principal en Linux es la eficiencia y se intenta aprovechar al máximo el hardware disponible.

5) Alta calidad: los sistemas GNU/Linux son muy estables, con una baja proporción de fallos, y reducen el tiempo dedicado a mantener los sistemas.

6) El núcleo es bastante reducido y compacto: es posible colocarlo, junto con algunos programas fundamentales en un solo disco de 1,44 MB (existen varias distribuciones de un solo disquete con programas básicos).

7) Linux es compatible con una gran parte de los sistemas operativos, puede leer ficheros de prácticamente cualquier sistema de ficheros y puede comunicarse por red para ofrecer y recibir servicios de cualquiera de estos sistemas. Además, también con ciertas bibliotecas puede ejecutar programas de otros sistemas (como MS-DOS, Windows, BSD, Xenix, etc.) en la arquitectura x86 o bien virtualizar máquinas completas.

8) Linux dispone de un amplísimo soporte: no hay ningún otro sistema que tenga la rapidez y cantidad de parches y actualizaciones que Linux, ni en los sistemas privativos. Para un problema determinado, hay infinidad de listas de correo y foros que en pocas horas pueden permitir solucionar cualquier problema. El único problema está en los controladores de hardware reciente, que muchos fabricantes todavía se resisten a proporcionar, si no es para sistemas privativos. Pero esto está cambiando poco a poco, y varios de los fabricantes más importantes de sectores como tarjetas de vídeo (NVIDIA, ATI) e impresoras (Epson, HP) comienzan ya a proporcionar los controladores para sus dispositivos, bien sean de código abierto, o binarios usables por el núcleo.

2. Personalización o actualización del núcleo

Como usuarios o administradores de sistemas GNU/Linux, debemos tener en cuenta las posibilidades que nos ofrece el núcleo para adaptarlo a nuestras necesidades y equipos.

Normalmente, construimos nuestros sistemas GNU/Linux a partir de la instalación en nuestros equipos de alguna de las distribuciones de GNU/Linux, ya sean comerciales como Red Hat, Mandriva o Suse, o "comunitarias" como Debian y Fedora.

Estas distribuciones aportan, en el momento de la instalación, una serie de núcleos Linux binarios ya preconfigurados y compilados, y normalmente tenemos que elegir qué núcleo del conjunto de los disponibles se adapta mejor a nuestro hardware. Hay núcleos genéricos para una arquitectura, para un modelo de procesador o bien orientados disponer de una serie de recursos de memoria, otros que ofrecen una mezcla de controladores de dispositivos [Ar05], posibilidades de virtualización, etc.

Otra opción de instalación suele ser la versión del núcleo. Normalmente las distribuciones usan una versión para instalación que consideran lo suficientemente estable y probada como para que no cause problemas a los usuarios. Por ejemplo, a día de hoy muchas distribuciones vienen con una versión de la rama 2.6.x del núcleo por defecto, que se consideraba la versión más estable del momento en que salió la distribución. En algunos casos en el momento de la instalación puede ofrecerse la posibilidad de usar como alternativa versiones más modernas, con mejor soporte para dispositivos más modernos (de última generación), pero quizás no tan probadas.

Los distribuidores suelen, además, modificar el núcleo para mejorar el comportamiento de su distribución o corregir errores que han detectado en el núcleo en el momento de las pruebas. Otra técnica bastante común en las distribuciones comerciales es deshabilitar prestaciones problemáticas, que pueden causar fallos o que necesitan una configuración especifica de la máquina, o bien una determinada prestación no se considera lo suficientemente estable para incluirla activada.

Personalización del núcleo

La posibilidad de actualizar y personalizar el núcleo a medida ofrece una buena adaptación a cualquier sistema, lo que permite así una optimización y sintonización del núcleo al sistema destino.

Esto nos lleva a considerar que, por muy bien que un distribuidor haga el trabajo de adaptar el núcleo a su distribución, siempre nos podemos encontrar con una serie de problemas u objetivos que no podemos realizar con la situación actual:

- El núcleo no está actualizado a la última versión estable disponible; no se dispone de soporte para algunos dispositivos modernos.
- El núcleo estándar no dispone de soporte para los dispositivos que tenemos, porque no han sido habilitados.
- Los controladores que nos ofrece un fabricante necesitan una nueva versión del núcleo o modificaciones.
- A la inversa, el núcleo es demasiado moderno, tenemos hardware antiguo que ya no tiene soporte en los últimos núcleos.
- El núcleo, tal como está, no obtiene las máximas prestaciones de nuestros dispositivos.
- Algunas aplicaciones que queremos usar requieren soporte de un núcleo nuevo o de algunas de sus prestaciones.
- Queremos estar a la última, nos arriesgamos, instalando últimas versiones del núcleo Linux.
- Nos gusta investigar o probar los nuevos avances del núcleo o bien queremos tocar o modificar el núcleo.
- Queremos programar un controlador para un dispositivo no soportado.
- Etc.

Por estos y otros motivos podemos no estar contentos con el núcleo que tenemos. Se nos plantean entonces dos posibilidades: actualizar el núcleo binario de la distribución o bien personalizarlo a partir de los paquetes fuente.

Vamos a ver algunas cuestiones relacionadas con las diferentes opciones y qué suponen:

1) Actualización del núcleo de la distribución. El distribuidor normalmente publica también las actualizaciones que van surgiendo del núcleo. Cuando la comunidad Linux crea una nueva versión del núcleo, cada distribuidor la une a su distribución y hace las pruebas pertinentes. Después del periodo de prueba, se identifican posibles errores, los corrige y produce la actualización del núcleo pertinente respecto a la que ofrecía en los CD de la distribución. Los usuarios pueden descargar la nueva revisión de la distribución del sitio web, o bien actualizarla mediante algún sistema automático de paquetes vía repositorio de paquetes. Normalmente, se verifica qué versión tiene el sistema, se descarga el núcleo nuevo y se hacen los cambios necesarios para que la siguiente vez el sistema funcione con el nuevo núcleo, y se mantiene la versión antigua por si hay problemas.

Este tipo de actualización nos simplifica mucho el proceso, pero no tiene porqué solucionar nuestros problemas, ya que puede ser que nuestro hardware no esté todavía soportado o la característica a probar del núcleo no esté todavía en la versión que tenemos de la distribución; cabe recordar que no tiene porqué usar la última versión disponible (por ejemplo en kernel.org), sino aquella que el distribuidor considere estable para su distribución.

Si nuestro hardware tampoco viene habilitado por defecto en la nueva versión, estamos en la misma situación. O sencillamente, si queremos la última versión, este proceso no nos sirve.

2) Personalización del núcleo. En este caso, iremos a los paquetes fuente del núcleo y adaptaremos "a mano" el hardware o las características deseadas. Pasaremos por un proceso de configuración y compilación de los paquetes fuente del núcleo para, finalmente, crear un núcleo binario que instalaremos en el sistema, y tenerlo, así, disponible en el siguiente arranque del sistema.

La personalización del núcleo es un proceso que se describe con detalle en los apartados siguientes.

También aquí podemos encontrarnos con dos opciones más: o bien por defecto obtenemos la versión "oficial" del núcleo (kernel.org) o bien podemos acudir a los paquetes fuente proporcionados por la propia distribución. Hay que tener en cuenta que distribuciones como Debian y Fedora hacen un trabajo importante de adecuación del núcleo y de corrección de errores del que afectan a su distribución, con lo cual podemos, en algunos casos, disponer de correcciones adicionales al código original del núcleo. Otra vez más los paquetes fuente ofrecidos por la distribución no tienen porqué corresponder a la última versión estable publicada.

Este sistema nos permite la máxima fiabilidad y control, pero a un coste de administración alto, ya que debemos disponer de conocimientos amplios de los dispositivos y de las características que estamos escogiendo (qué significan y qué implicaciones pueden tener), así como de las consecuencias que puedan tener las decisiones que tomemos.

3. Proceso de configuración y compilación

La personalización del núcleo [Vasb] es un proceso costoso, necesita amplios conocimientos del proceso a realizar y, además, es una de las tareas críticas, de la cual depende la estabilidad del sistema, por la propia naturaleza del núcleo, puesto que es su elemento central.

Obtención de un núcleo personalizado

El proceso de obtención de un nuevo núcleo personalizado pasa por obtener los paquetes fuente, adaptar la configuración, compilar e instalar el núcleo obtenido en el sistema.

Cualquier error de procedimiento puede comportar la inestabilidad o la pérdida del sistema. Por lo tanto, no está de más realizar cualquier tarea de copia de seguridad de los datos de usuarios, datos de configuraciones que hayamos personalizado o, si disponemos de dispositivos adecuados, una copia de seguridad completa del sistema. También es recomendable disponer de algún disquete de arranque (o distribución LiveCD con herramientas de rescate) que nos sirva de ayuda por si surgen problemas, o bien un disquete/CD/archivo de rescate (*rescue disk*) que la mayoría de distribuciones permiten crear desde los CD de la distribución (o directamente proporcionan alguno como CD de rescate para la distribución). Actualmente muchos de los LiveCD de las distribuciones ya proporcionan herramientas de rescate suficientes para estas tareas, aunque también existen algunas distribuciones especializadas para ello.

Sin ánimo de exagerar, casi nunca aparecen problemas si se siguen los pasos adecuadamente, se tiene conciencia de los pasos realizados y se toman algunas precauciones. Evidentemente, ante sistemas en producción, siempre es importante tomar las medidas de precaución y hacer las copias de seguridad necesarias.

Vamos a ver el proceso necesario para instalar y configurar un núcleo Linux. En los subapartados siguientes, examinamos:

1) El caso de las versiones antiguas 2.4.x.

2) Algunas consideraciones sobre la migración a las 2.6.x partiendo de 2.4.x

3) Detalles específicos de las versiones 2.6.x.

4) Un caso particular para la distribución Debian, que dispone de un sistema propio (*Debian way*) de compilación más flexible.

Enlace de interés

Para Fedora recomendamos consultar el siguiente enlace: http://fedoraproject.org/wiki/Building_a_custom_kernel.

Respecto las versiones 2.4.x, mantenemos en este módulo la explicación por razones históricas, ya que las distribuciones actuales prácticamente ya no las ofrecen, pero debemos considerar que en más de una ocasión nos veremos obligados a migrar un determinado sistema a nuevas versiones, o bien a man-

tenerlo en las antiguas, debido a incompatibilidades o existencia de hardware antiguo no soportado.

Los conceptos generales del proceso de compilación y configuración se explicarán en el primer subapartado (2.4.x), ya que la mayoría de ellos son genéricos, y observaremos posteriormente las diferencias respecto de las nuevas versiones. Aun así, cada subapartado puede examinarse de manera autosuficiente.

También hay que añadir que, con las últimas distribuciones, cada vez es más casual la necesidad de reconstruir o recompilar el propio núcleo, debido, entre otras consideraciones, a que:

- Antiguamente la mayoría de los controladores estaban integrados en el núcleo y había que recompilarlo por completo si queríamos incluir o excluir un controlador determinado. Hoy en día, como veremos en el apartado 5, pueden recompilarse los controladores o módulos concretos, no el núcleo en si mismo.

- Para sintonizar el núcleo, antiguamente había que recompilarlo. En muchos casos (no todos) puede realizarse la sintonización de algunos elementos del núcleo mediante el acceso al sistema de ficheros `/proc` o `/sys`.

- En algunos casos de distribuciones comerciales (versiones empresariales para las que se paga soporte), los núcleos y el sistema completo están soportados por el equipo de la distribución, y a veces pueden perderse las licencias de soporte o las garantías por realizar cambios de este estilo.

- Por otro lado, las distribuciones tienen una velocidad bastante rápida en cuanto a integrar parches y nuevos núcleos a medida que se generan.

Por contra, una personalización del núcleo, mediante recompilación, nos puede permitir:

- Escoger qué partes incluir o excluir del núcleo, dando un soporte concreto a una plataforma o un hardware muy concreto. Esto es imprescindible si estamos, por ejemplo, en situaciones de hardware empotrado (*embedded*).

- Sintonizar, de esta última manera, el consumo de memoria u otros recursos para adaptarse mejor a recursos limitados (CPU, memoria, disco, etc.).

- Versiones concretas de las distribuciones implican usar ciertas versiones específicas del núcleo. En muchos casos no podemos obtener nuevas actualizaciones del núcleo si no actualizamos la distribución concreta completamente.

- Probar versiones de núcleo todavía no disponibles en las distribuciones. Aunque hay cierta rapidez de integración de los nuevos núcleos, se puede tardar semanas o meses en disponer de los nuevos núcleos vía distribución. Por otra parte, algunas distribuciones son muy conservadoras en cuanto a la versión de núcleo utilizada, y hay que esperar varias versiones completas de la distribución (un periodo largo de tiempo) para llegar a una versión concreta de núcleo.

- Probar versiones beta, o parches de núcleo, con el objetivo de integrarlo rápidamente en algún sistema con problemas concretos, o bien sencillamente por cuestiones de evaluación de las nuevas posibilidades o de un mejor rendimiento.

- Participar directamente en el desarrollo del núcleo, estudiando posibles mejoras del código actual, proponiendo y probando propuestas concretas. Es típico de algunos componentes, así como estudiar diferentes estrategias de planificación de CPU, de gestión de memoria, mejorar parámetros del núcleo o colocar alguna prestación nueva a algún sistema de ficheros.

En los siguientes subapartados veremos las diferentes posibilidades en cuanto a la configuración y recompilación de las diferentes ramas de desarrollo del núcleo Linux.

3.1. Compilación antigua de la rama 2.4.x del núcleo

Las instrucciones son específicas para la arquitectura x86 de Intel, mediante usuario root (aunque la mayor parte del proceso puede hacerse como usuario normal y, de hecho, es aconsejable por seguridad):

1) Obtener el núcleo. Por ejemplo, podemos acudir a http://www.kernel.org (o a su servidor ftp) y descargar la versión *vanilla* que queramos probar. Hay mirrors para diferentes países; podemos, por ejemplo, acudir a la siguiente web: http://www.es.kernel.org. Por otro lado, en la mayoría de las distribuciones de GNU/Linux, como Fedora/Red Hat o Debian, también se ofrece como paquete el código fuente del núcleo (normalmente con algunas modificaciones incluidas); si se trata de la versión del núcleo que necesitamos, quizás sea preferible usar estas (mediante los paquetes `kernel-source`, `linux-source`, o similares). Si queremos los últimos núcleo, quizás no estén disponibles en la distribución y tendremos que acudir a kernel.org.

2) Desempaquetar el núcleo. Los paquetes fuente del núcleo solían colocarse y desempaquetarse sobre el directorio `/usr/src`, aunque se recomienda utilizar algún directorio aparte para no mezclarlos con ficheros fuente que pueda traer la distribución. Por ejemplo, si los paquetes fuente venían en un fichero comprimido de tipo bzip2:

```
bzip2 -dc linux-2.4.0.tar.bz2 | tar xvf -
```

Si los paquetes fuente venían en un fichero gz, reemplazamos bzip2 por gzip. Al descomprimir los paquetes fuente se habrá generado un directorio llamado `linux-version-kernel`, donde entraremos para establecer la configuración del núcleo.

Herramientas de configuración y compilación

Antes de comenzar los pasos previos a la compilación, debemos asegurarnos de disponer de las herramientas correctas, en especial del compilador gcc, make y otras utilidades gnu complementarias en el proceso. Un ejemplo son las modutils, que disponen las diferentes utilidades para el uso y gestión de los módulos de núcleo dinámicos. Asimismo, para las diferentes opciones de configuración hay que tener en cuenta una serie de prerrequisitos en forma de bibliotecas asociadas a la interfaz de configuración usada (por ejemplo las ncurses para la interfaz menuconfig).

Se recomienda, en general, consultar la documentación del núcleo (ya sea vía paquete o en el directorio raíz de las fuentes del núcleo) para conocer qué prerrequisitos, así como versiones de estos, son necesarios para el proceso. Se recomienda examinar los ficheros `README` en el directorio "raíz", y el `Documentation/Changes`, o el índice de documentación del núcleo en `Documentation/00-INDEX`.

Si hemos realizado anteriores compilaciones en el mismo directorio, deberemos garantizar que el directorio utilizado esté limpio de compilaciones anteriores; podemos limpiarlo con `make mrproper` (realizado desde el directorio "raíz" de las fuentes).

3) Configuración del núcleo. Para el proceso de configuración del núcleo [Vasb], tenemos varios métodos alternativos, que nos presentan interfaces diferentes para ajustar los múltiples parámetros del núcleo (que suelen almacenarse en un fichero de configuración, normalmente `.config` en el directorio "raíz" de las fuentes). Las diferentes alternativas son:

- `make config`: desde la línea de comandos se nos pregunta por cada opción y se nos pide confirmación (y/n), si deseamos o no la opción, o se nos piden los valores necesarios. Es la configuración larga, en la que se nos piden muchas respuestas, podemos tener que responder a casi un centenar de preguntas (o más dependiendo de la versión).

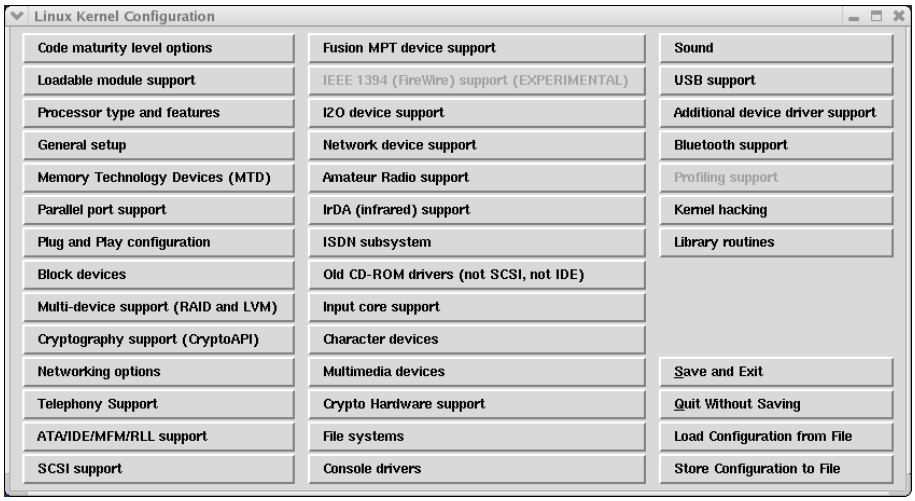
- `make oldconfig`: sirve por si queremos reutilizar una configuración ya usada (normalmente almacenada en un fichero .config, en el directorio "raíz" de las fuentes); hay que tener en cuenta que solo es válida si estamos compilando la misma versión del núcleo, ya que diferentes versiones del núcleo pueden variar en sus opciones.

- `make menu config`: configuración basada en menús textuales, bastante cómoda; podemos habilitar o inhabilitar lo que queramos y es más rápida que el `make config`.

- `make xconfig`: la más cómoda, basada en diálogos gráficos en X Window (figura 2). Es necesario tener instaladas las bibliotecas de tcl/tk, ya que esta configuración está programada en este lenguaje. La configuración se basa en cuadros de diálogo, botones y casillas de activación. Es bastante rápida y dispone de ayuda con comentarios de muchas de las opciones. Pero hay un defecto: puede ser que algunas de las opciones no aparezcan (depende de que el programa de configuración esté actualizado, y a veces no lo es-

tá). En este último caso, el `make config` (o el `menuconfig`) es el único que garantiza disponer de todas las opciones elegibles; en los otros tipos de configuración depende de que se hayan podido adaptar a tiempo los programas a las nuevas opciones cuando se libera el núcleo. Aunque en general se intentan mantener de forma equivalente.

Figura 2. Configuración del núcleo 2.4.x desde la interfaz gráfica en X Window



Una vez se haya hecho el proceso de configuración, hay que guardar el fichero `.config`, ya que la configuración consume un tiempo importante. Además, puede ser de utilidad disponer de la configuración realizada (`.config`) si está planeado hacerla en varias máquinas parecidas o idénticas.

Otro tema importante en las opciones de configuración es que en muchos casos se nos va a preguntar si una determinada característica la queremos integrada en el núcleo o como módulo. Ésta es una decisión más o menos importante, ya que el rendimiento del núcleo (y, por lo tanto, del sistema entero) en algunos casos puede depender de nuestra elección.

Los módulos es tratan en el apartado 5.

El núcleo de Linux ha comenzado a ser de gran tamaño, tanto por complejidad como por los controladores de dispositivo [Ar05] que incluye. Si lo integrásemos todo, se podría crear un fichero del núcleo bastante grande y ocupar mucha memoria, lo que ralentizaría algunos aspectos de funcionamiento. Los módulos del núcleo [Hen] son un método que permite separar parte del núcleo en pequeños trozos, que se cargarán dinámicamente bajo demanda cuando, por carga explícita o por uso de la característica, sean necesarios.

La elección más normal es integrar dentro del núcleo lo que se considere básico para el funcionamiento o crítico en rendimiento, y dejar como módulos aquellas partes o controladores de los que se vaya a hacer un uso esporádico o que se necesite conservar por si se producen futuras ampliaciones del equipo.

- Un caso claro son los controladores de dispositivo: si estamos actualizando la máquina, puede ser que a la hora de crear el núcleo no conozcamos con seguridad qué hardware va a tener: por ejemplo, qué tarjeta de red, pero

sí que sabemos que estará conectada a red; en este caso el soporte de red estará integrado en el núcleo, pero por lo que respecta a los controladores de las tarjetas, podremos seleccionar unos cuantos (o todos) y ponerlos como módulos. Así, cuando tengamos la tarjeta podremos cargar el módulo necesario, o si después tenemos que cambiar una tarjeta por otra, solo tendremos que cambiar el módulo que se va a cargar. Si hubiese solo un controlador integrado en el núcleo y cambiamos la tarjeta, esto obligaría a reconfigurar y recompilar el núcleo con el controlador de la tarjeta nueva.

- Otro caso que suele aparecer (aunque no es muy común) es cuando necesitamos dos dispositivos que son incompatibles entre si, o está funcionando uno o el otro (esto podría pasar, por ejemplo, con impresoras con cable paralelo y hardware que se conecta al puerto paralelo). Por lo tanto, en este caso tenemos que colocar como módulos los controladores y cargar o descargar el que sea necesario.

- Otro ejemplo podrían formarlo los sistemas de ficheros (*filesystems*). Normalmente esperaremos que nuestro sistema tenga acceso a algunos de ellos, por ejemplo ext2, ext3 o ext4 (propios de Linux), vfat (de los Windows 95/98/ME) y los daremos de alta en la configuración del núcleo. Si en otro momento tuviéramos que leer otro tipo no esperado, por ejemplo datos guardados en un disco o partición de sistema NTFS de Windows NT/XP, no podríamos: el núcleo no sabría o no tendría soporte para hacerlo. Si tenemos previsto que en algún momento (pero no habitualmente) se tenga que acceder a estos sistemas, podemos dejar los demás sistemas de ficheros como módulos.

4) Compilación del núcleo. Mediante `make` comenzaremos la compilación, primero hay que generar las posibles dependencias entre el código y luego el tipo de imagen de núcleo que se quiere (en este caso una imagen comprimida, que suele ser la normal):

```
make dep
make bzImage
```

Cuando este proceso acabe, tendremos la parte integrada del núcleo y nos faltarán las partes que hayamos puesto como módulos:

```
make modules
```

Hasta este momento hemos hecho la configuración y compilación del núcleo. Esta parte podía hacerse desde un usuario normal o bien el root, pero ahora necesitaremos forzosamente usuario root, porque pasaremos a la parte de la instalación.

5) Instalación. Comenzamos instalando los módulos:

```
make modules_install
```

y la instalación del nuevo núcleo (desde `/usr/src/linux-version`, siendo `xx` la versión usada):

```
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.xx
cp System.map /boot/System.map-2.4.xx
```

El archivo `bzImage` es el núcleo recién compilado, que se coloca en el directorio `/boot`. Normalmente el núcleo antiguo se encontrará en el mismo directorio boot con el nombre `vmlinuz` o bien `vmlinuz-versión-anterior` y `vmlinuz` como un enlace simbólico al núcleo antiguo. Una vez tengamos nuestro núcleo, es mejor conservar el antiguo, por si se producen fallos o un mal funcionamiento del nuevo y así poder recuperar el viejo. El fichero `System.map` es un fichero que contiene los símbolos disponibles en el núcleo y es necesario para el proceso de arranque del núcleo, también se coloca en el mismo directorio.

6) Configuración del arranque. El siguiente paso es decirle al sistema con que núcleo tiene que arrancar. Este paso depende del sistema de arranque de Linux, y del *bootloader* usado:

- Desde arranque con LiLo [Skoa], ya sea en el Master Boot Record (MBR) o desde partición propia, hay que añadir al fichero de configuración (en `/etc/lilo.conf`), por ejemplo, las líneas:

  ```
  image = /boot/vmlinuz-2.4.0
      label = 2.4.0
  ```

  donde `image` es el núcleo que se va arrancar y `label` será el nombre con el que aparecerá la opción en el arranque. Podemos añadir estas líneas o modificar las que hubiera del núcleo antiguo. Se recomienda añadirlas y dejar el núcleo antiguo, para poder recuperar el núcleo antiguo si aparecen problemas. En el fichero `/etc/lilo.conf` puede haber una o más configuraciones de arranque tanto de Linux como de otros sistemas (como Windows); cada arranque se identifica por su línea `image` y el `label` que aparece en el menú de arranque. Hay una línea `"default=label"` donde se indica el label por defecto que se arrancará. También podemos añadirle a las líneas anteriores un `"root=/dev/..."` para indicar la partición de disco donde está el sistema de archivos principal (el /). Recordar que los discos tienen dispositivos como `/dev/hda` (primer disco IDE) `/dev/hdb` (segundo IDE), y la partición se indicaría como `"root=/dev/hda2"` si el / de nuestro Linux estuviese en la segunda partición del primer disco ide. Además, con `append=` podemos añadir parámetros al arranque del núcleo. Después de cambiar la configuración del LiLo hay que escribirla físicamente en el disco (se modifica el sector de arranque) para que esté disponible en el siguiente arranque:

  ```
  /sbin/lilo -v
  ```

  A continuación reiniciamos y arrancamos con el nuevo núcleo.

Si tuviésemos problemas, podemos recuperar el antiguo núcleo, escogiendo la opción del viejo núcleo, y luego retocar el `lilo.conf` para volver a la antigua configuración o estudiar el problema y reconfigurar y recompilar el núcleo de nuevo.

- Arranque con Grub: La gestión en este caso es bastante simple, cabe añadir una nueva configuración formada por el núcleo nuevo y añadirla como una opción más al fichero de configuración del Grub, y rearrancar procediendo de forma parecida a la del LiLo, pero recordando que en Grub basta con editar el fichero y rearrancar. También es mejor dejar la antigua configuración para poder recuperarse de posibles errores o problemas con el núcleo recién compilado.

Lectura recomendada

Sobre Grub podéis consultar *Grub bootloader* y *Grub Manual* accesibles desde la web del proyecto GNU: http://www.gnu.org

3.2. Migración de 2.4 a la rama 2.6.x del núcleo

En el caso de tener que actualizar versiones de distribuciones antiguas, o bien realizar el cambio de generación del núcleo mediante los paquetes fuente, habrá que tener en cuenta una serie de consideraciones, debido a las novedades introducidas en la rama 2.6.x del núcleo.

Presentamos a continuación algunos puntos concretos que cabe observar:

- Algunos de los módulos del núcleo han cambiado de nombre, y algunos han podido desaparecer; hay que comprobar la situación de los módulos dinámicos que se cargan (por ejemplo, examinar `/etc/modules` y/o `/etc/modules.conf`), y editarlos para reflejar los cambios.

- Se han añadido nuevas opciones para la configuración inicial del núcleo: como `make gconfig`, una interfaz basada en gtk (Gnome). Habrá que observar, como prerrequisito en este caso, las bibliotecas de Gnome. La opción de `make xconfig` se ha implementado ahora con las bibliotecas de qt (KDE).

- Se incrementan las versiones mínimas necesarias de varias utilidades necesarias para el proceso de compilación (consultar `Documentation/Changes` en los paquetes fuente del núcleo). En especial la versión del compilador gcc mínima. Ha cambiado el paquete por defecto para las utilidades de módulos, que pasa a ser `module-init-tools` (en lugar de `modutils` usado en las 2.4.x). Este paquete es un prerrequisito para la compilación de núcleos 2.6.x, ya que el cargador de módulos dinámicos está basado en esta nueva versión.

- El sistema `devfs` queda obsoleto en favor de `udev`, el sistema que controla el arranque (conexión) en caliente (*hotplug*) de dispositivos (y su reconocimiento inicial, de hecho simulando un arranque en caliente al iniciar el

sistema) y crea dinámicamente las entradas en el directorio `/dev`, solo para los dispositivos que estén actualmente presentes.

- En Debian a partir de ciertas versiones del 2.6.x, para las imágenes binarias de núcleo, *headers* de desarrolllo y el código fuente del núcleo, los nombres de paquetes cambian de `kernel-images/source/headers` a `linux-image/source/headers`.

- En algunos casos, los dispositivos de tecnologías nuevas (como SATA) pueden haber pasado de `/dev/hdX` a `/dev/sdX`. En tales casos habrá que editar las configuraciones de `/etc/fstab` y el *bootloader* (LiLo o Grub) para reflejar los cambios.

- Pueden existir algunos problemas con dispositivos de entrada/salida concretos. El cambio de nombres de módulos de núcleo ha afectado entre otros a los dispositivos de ratón, lo que puede afectar asimismo a la ejecución de X-Window, a la verificación de qué modelos son necesarios y cargar los módulos correctos (por ejemplo el `psmouse`). Por otra parte, en el núcleo se integran los controladores de sonido Alsa. Si disponemos de los antiguos OSS, habrá que eliminarlos de la carga de módulos, ya que Alsa ya se encarga de la emulación de estos últimos.

- Respecto a las arquitecturas que soporta el núcleo, hay que tener en cuenta que con los 2.6.x, en las diferentes revisiones, se han ido incrementando las arquitecturas soportadas, lo que nos permitirá disponer de imágenes binarias del núcleo en las distribuciones (o las opciones de compilación del núcleo) más adecuadas para el soporte de nuestros procesadores. En concreto, podemos encontrarnos con arquitecturas como i386 (para Intel y AMD), que soporta la compatibilidad de Intel en 32 bits para toda la familia de procesadores (en algunas distribuciones se usa 486 como arquitectura general); en algunas distribuciones se integran versiones diferenciadas para i686 (Intel a partir de Pentium Pro en adelante), para k7 (AMD Athlon en adelante) y las específicas de 64 bits, para AMD de 64 bits (x86_64) e Intel con extensiones em64t de 64 bits como en los Xeon y algunas familias de multicores. Por otra parte, también existe la arquitectura IA64 para los modelos de 64 bits Intel Itanium. En la mayoría de los casos, las arquitecturas disponen de las capacidades SMP activadas en la imagen del núcleo (a no ser que la distribución soporte versiones con SMP o sin, creadas independientemente; en este caso, suele añadirse el sufijo *-smp* a la imagen binaria del núcleo que lo soporta).

- En Debian, para la generación de imágenes `initrd`, a partir de ciertas versiones del núcleo ($\geq$ 2.6.12) se consideran obsoletas las `mkinitrd-tools`, que son sustituidas por nuevas utilidades como `initramfs-tools` o como `yaird`. Ambas permiten la construcción de la imagen `initrd`, siendo la primera la recomendada por Debian.

3.3. Compilación de la rama 2.6.x del núcleo

En las versiones 2.6.x, teniendo en cuenta las consideraciones comentadas anteriormente, la compilación se desarrolla de forma análoga a la expuesta anteriormente.

Una vez descargados de *kernel.org* o proporcionados por la distribución, los paquetes fuente del núcleo 2.6.xx (donde xx es el número de revisión del núcleo estable), se procede a su extracción en el directorio que se usará para la compilación. Si el paquete obtenido es de tipo gzip (`tar.gz`):

```
gzip -cd linux-2.6.xx.tar.gz | tar xvf -
```

o si es de tipo bzip2 (`tar.bz2`):

```
bzip2 -dc linux-2.6.xx.tar.bz2 | tar xvf -
```

Una vez extraídos los archivos fuente en el directorio de compilación, se comprueban los paquetes del entorno de compilación necesario. Normalmente es necesario instalar algunos paquetes como por ejemplo `build-essentials`, `libncurses-dev`, `libqt3-dev`, `libgtk2-dev` (los nombres dependen de la distribución), que incorporan el entorno básico de compilación (gcc) y las necesidades para la construcción de los menús posteriores de compilación de los archivos fuente (`make menuconfig` o `xconfig`).

Es recomendable leer el fichero README que está situado en el directorio raíz de los paquetes fuente.

En este punto, podría ser interesante parchear el núcleo, lo cual podríamos realizar, debido a que tenemos un código fuente adicional (en forma de parche) que mejora algún problema conocido de la versión o bien porque queremos proponer o probar un cambio de código en el núcleo. También podría darse el caso de que un fabricante de hardware ofreciese algún soporte o corrección de fallos para un controlador de dispositivo, como un parche para una versión de núcleo concreta.

El proceso de parchear el núcleo se trata en el apartado 4.

Una vez dispongamos de los paquetes necesarios, procedemos al proceso de compilación en el directorio utilizado para los paquetes fuente. Cabe recordar que todo el proceso puede realizarse como usuario normal, solo partes muy concretas, como la instalación final del núcleo o de módulos dinámicos, es necesario hacerlas usando el usuario root.

También pueden surgir problemas de seguridad por realizar la compilación en modo root, de fuentes de núcleo desconocidas o no fiables. Al respecto, tanto los paquetes fuente de *kernel.org* como las proporcionados por las distribuciones suelen contener firmas (o repositorios firmados) que pueden usarse para

verificar la integridad de los ficheros de fuentes. Hay que evitar, a toda costa, usar fuentes de núcleo o de módulos proporcionados por emisores no fiables.

Pasamos a iniciar el proceso en el directorio de los fuentes, comenzando con la limpieza de anteriores compilaciones:

```
# make mrproper
```

El siguiente paso es la configuración de parámetros. El fichero `.config` podemos usarlo bien a partir compilaciones previas que hayamos realizado o bien partir de la configuración del núcleo actual existente (recordad que con `uname -r` tenemos la versión de núcleo). Esta configuración podemos obtenerla, dependiendo de la distribución, de `/boot/config-version-kernel` y podemos copiar este fichero como `.config` en el directorio raíz de los paquetes fuente del núcleo. Así partimos de una serie de opciones de núcleo ya preparadas, en las que solo tendremos que realizar los cambios deseados o mirar las nuevas opciones no presentes en el núcleo antiguo. Si estamos en el directorio de los paquetes fuente, entonces un:

Recordad que si disponemos de un `.config` previo, nos permitirá no comenzar de cero la configuración del núcleo.

```
cp /boot/config-‘uname -r‘ ./.config
```

nos dejará estas opciones del núcleo actual preparadas. También podríamos usar un fichero de configuración de otras compilaciones de núcleo previas que hayamos realizado. Si no proponemos ningún fichero de configuración previo, entonces partimos de unas opciones por defecto en las que deberemos mirar y comprobar todas las opciones del núcleo.

La configuración la realizamos a través de la opción escogida de `make`, pasando por (entre otras) opciones:

- `make config`: interfaz plana de texto.
- `make menuconfig`: interfaz basada en menus textuales.
- `make xconfig`: interfaz gráfica basada en `toolkit Qt` de KDE.
- `make gconfig`: interfaz gráfica basada en `toolkit Gtk` de Gnome.
- `make oldconfig`: se basa en el fichero `.config` previo y pregunta textualmente por las opciones nuevas que no estaban previamente en la configuración antigua (de `.config`).

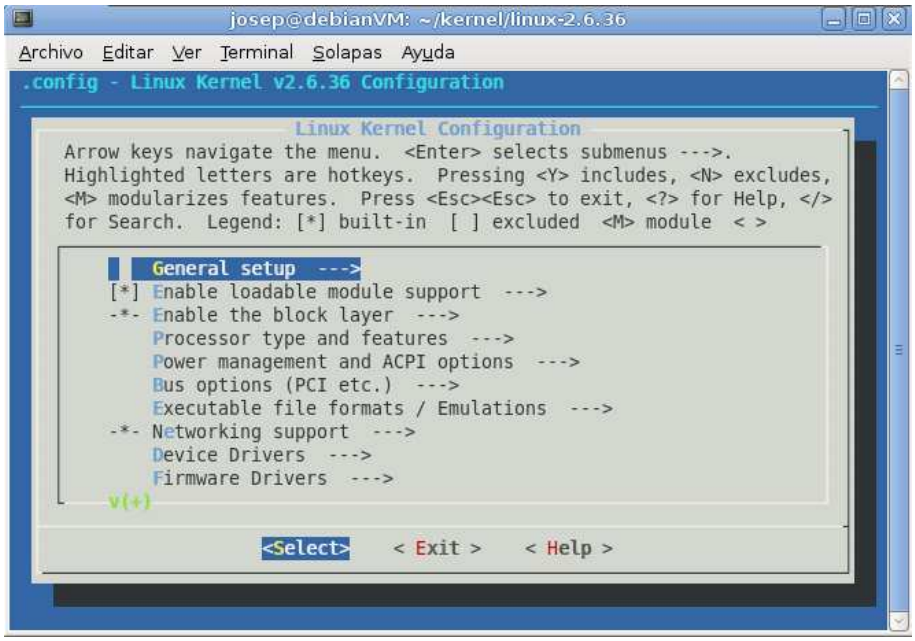
Por ejemplo, escogemos la interfaz textual (figura 3):

```
# make menuconfig
```

A continuación procedemos a la construcción de la imagen binaria del núcleo, con simplemente:

```
# make
```

Figura 3. Configuración del núcleo (`make menuconfig`) desde interfaz textual



Cabe mencionar que para estos núcleo genéricos (habitualmente denominados *vanilla* o *pristine*), también existen algunos atajos más rápidos para la construcción final del núcleo (saltando la mayoría de los pasos siguientes); por ejemplo, en Debian es posible sustituir el actual `make` por `make deb-pkg`, que nos obtendrá paquetes binarios DEB instalables en el sistema y actualizara *bootloaders* y los ficheros necesarios. En cualquier caso, siempre tenemos la alternativa de realizar los siguientes pasos:

1) Construcción de los módulos (aquellos especificados como tales):

```
# make modules
```

2) instalación de los módulos creados (en `/lib/modules/version-kernel`):

```
# make modules_install
```

3) copia de la imagen a su posición final, suponiendo i386 como arquitectura (hay que vigilar este punto, porque hay detalles en los nombres y directorios que cambian en función de la distribución GNU/Linux):

```
# cp arch/i386/boot/bzimage /boot/vmlinuz-2.6.xx.img
```

4) y finalmente, si es necesaria, creación de la imagen de disco RAM `initrd`, con las utilidades necesarias según la versión de la distribución*, y ajuste de las entradas en el *bootloader* LiLo o Grub según sea el que utilicemos en nuestra distribución.

***Véanse los comentarios posteriores en este apartado y en el apartado 5.**

Los últimos pasos (`vmlinuz`, `system.map` y `initrd`) de movimiento de archivos a `/boot` pueden realizarse también normalmente con el proceso (en root):

```
# make install
```

Pero hay que tener en cuenta que esta opción realiza todo el proceso y actualizará los *bootloaders* quitando antiguas configuraciones o modificándolas. Pueden alterarse también los enlaces por defecto en el directorio `/boot`. Antes de utilizar esta opción automática, sería recomendable un proceso de copia de seguridad de la configuración del *bootloader* (LiLo en `/etc/lilo.conf` o Grub en `/etc/grub/menu.lst`) y posibles enlaces de `/boot` a la hora de pensar en las configuraciones pasadas que queremos guardar.

Respecto a la creación del `initrd`, en Fedora/Red Hat este se creará automáticamente con la opción `make install`. En Debian deberemos, o bien usar las técnicas del siguiente apartado, o bien crearlo explicitamente con `mkinitrd` (versiones de núcleo $\leq$ 2.6.12) o, posteriormente, con `mkinitramfs` o una utilidad denominada `update-initramfs`, especificando la versión del núcleo (se asume que este se llama `vmlinuz-version` dentro del directorio `/boot`):

```
# update-initramfs -c -k 'version'
```

Una vez disponemos de los ficheros correctos en `/boot` y del *bootloader* actualizado, ya podemos proceder al rearranque con `shutdown -r now`, escoger nuestro núcleo y, si todo ha ido bien, podremos comprobar con `uname -r` que disponemos de la nueva versión del núcleo. También es particularmente interesante examinar algunos registros, como `/var/log/messages` y el comando `dmesg`, para examinar el registro de salida de mensajes producidos por el nuevo núcleo en el arranque y detectar si ha aparecido algún problema de funcionalidad o con algún dispositivo concreto.

3.4. Compilación del núcleo en Debian (*Debian Way*)

En Debian, además de los métodos comentados en los subapartados previos, hay que añadir la configuración por el método denominado *Debian Way*. Es un método que nos permite construir el núcleo de una forma flexible y rápida, adaptada a la distribución.

Enlace de interés

Para Fedora recomendamos consultar el siguiente enlace: http://fedoraproject.org/wiki/Building_a_custom_kernel.

Para el proceso necesitaremos una serie de utilidades (hay que instalar los paquetes o similares): `kernel-package`, `ncurses-dev`, `fakeroot`, `wget` y `bzip2`.

Podemos observar el método [Debk] desde dos perspectivas: reconstruir un núcleo equivalente al proporcionado por la distribución como núcleo base (cambiando opciones) o bien crear un núcleo con una numeración de versión-revisión personalizada.

Enlace de interés

Puede verse el proceso *Debian Way* de forma detallada en: http://kernel-handbook.alioth.debian.org/

Respecto a los paquetes de fuentes del núcleo, Debian proporciona los paquetes fuente usados en su distribución, que pueden llegar a ser bastante diferen-

tes de los de la versión *vanilla* o *pristine* obtenida de *kernel.org*. Ello es debido a que en Debian producen múltiples revisiones con diversos parches que van añadiendo, muchos de ellos a partir de fallos que se detectan *a posteriori* en las siguientes versiones *vanilla* del núcleo.

En las versiones estables de Debian, la distribución suele escoger una revisión xx de la rama 2.6, de manera que el núcleo 2.6.xx suele quedarse (generalmente) en esta numeración para la versión de Debian estable y, así, cuando se actualiza con revisiones menores la distribución, solo se actualizan parches en revisiones menores del núcleo (sin cambiar el número principal). Cuando Debian produce la siguiente versión estable se salta a una nueva versión del núcleo. Durante la duración de una versión estable de la distribución, Debian suele producir diferentes modificaciones (*patchlevels*) o revisiones del núcleo que se escogió.

Debian ha cambiado varias veces la gestión de sus paquetes asociados a los paquetes fuente del núcleo. A partir de la versión 2.6.12 es habitual encontrar en el repositorio Debian una versión `linux-source-versión` que contiene la versión de los fuentes del núcleo con los últimos parches aplicados (véase el apartatdo 4). Esta versión del paquete de los paquetes fuente del núcleo es la que usaremos para crear un núcleo personalizado.

A partir de la versión mencionada (2.6.12) se incluyó en los respositorios *source* de Debian un nuevo paquete denominado simplemente *linux-2.6* que incluye los paquetes fuente y utilidades preparadas para generar el núcleo en la mencionada *Debian Way*. Este paquete fuente es usado para crear los paquetes binarios de la distribución asociados al núcleo y también es el indicado para usar en caso de querer aplicar parches al núcleo actual de la distribución, o por si queremos probar modificaciones del núcleo a nivel de código.

Examinemos primero esta opción y después, al final del subapartado, comentaremos la personalización.

Para la compilación del núcleo actual procedemos usando el segundo paquete:

```
# apt-get source linux-2.6
```

En este caso nos descargará y descomprimirá los paquetes fuente dejándolos en un árbol de directorios a partir del directorio *linux-2.6-version*.

Obtenemos después algunas herramientas que necesitaremos:

```
# apt-get install build-essential fakeroot
# apt-get build-dep linux-2.6
```

Estas líneas básicamente nos instalan el entorno de compilación para el núcleo (de hecho los paquetes del compilador gcc necesarios y herramientas propias de la *Debian Way*) y por último se comprueban las dependencias de los fuentes por si son necesarios paquetes fuente extra de desarrollo.

Para la construcción del binario para la arquitectura, según la configuración preestablecida del paquete (semejante a la incluida en los paquetes oficiales del núcleo `linux-image` en Debian):

```
$ cd linux-2.6-version
$ fakeroot debian/rules binary
```

Con esto dispondremos de los paquetes binarios de núcleo (ficheros `*.deb`) disponibles para la instalación (vía `dpkg`).

Existen algunos procedimientos extra para la creación de núcleo en base a diferentes niveles de parche (*patch*) proporcionados por la distribución, y posibilidades de generar diferentes configuraciones finales (puede verse la referencia de la nota para complementar estos aspectos).

Pasamos ahora a la otra opción que habíamos comentado al inicio, a los aspectos de personalización, cuando queremos cambiar ciertas opciones del núcleo y crear una versión personal del mismo.

En este caso, más habitual, cuando deseamos un núcleo personalizado, deberemos realizar un proceso semejante a través de un paso de personalización típico (por ejemplo, mediante `make menuconfig`). Los pasos son, en primer lugar, la obtención y preparación del directorio (aquí obtenemos los paquetes de la distribución, pero es equivalente obteniendo los paquetes fuente desde *kernel.org*):

```
# apt-get install linux-source-2.6.xx
$ tar -xvjf /usr/src/linux-source-2.6.xx.tar.bz2
$ cd linux-source-2.6.xx
```

donde obtenemos los paquetes fuente y los descomprimimos (la instalación del paquete deja el archivo de fuentes en `/usr/src`).

A continuación realizamos la configuración de parámetros. Como siempre podemos basarnos en ficheros `.config` que hayamos utilizado anteriormente, para partir de una configuración conocida (para la personalización, también puede usarse cualquiera de los otros métodos, `xconfig`, `gconfig`, etc.):

```
$ make menuconfig
```

A continuación, la construcción final del núcleo:

```
$ make clean
$ make KDEB_PKGVERSION=custom.1.0 deb-pkg
```

donde creamos un identificador para el núcleo construido (`custom.1.0`) que se añadirá al nombre del paquete binario del núcleo, posteriormente visible en el arranque con el comando `uname`.

Así, el proceso finalizará con la obtención del paquete asociado a la imagen del núcleo, que podremos finalmente instalar:

```
# dpkg -i ../linux-image-2.6.xx_custom.1.0_i386.deb
```

Esto nos descomprimirá e instalará el núcleo y generará una imagen `initrd` adicional si fuera necesario. Además, nos configura el *bootloader* con el nuevo núcleo por defecto (hay que vigilar con este paso, vale la pena haber obtenido antes una copia de seguridad del *bootloader*, para no perder ninguna configuración estable).

Enlace de interés

No será habitualmente necesario, pero si se necesita alguna reconfiguración del `initrd` generado, se recomienda leer el siguiente enlace donde se comentan múltiples herramientas Debian disponibles: http://kernel-handbook.alioth.debian.org/ch-initramfs.html.

Ahora directamente con un `shutdown -r now` podemos probar el arranque con el nuevo núcleo.

Añadimos, también en este final de este subapartado, otra peculiaridad a tener en cuenta en Debian, que es la existencia de utilidades para añadir módulos dinámicos de núcleo proporcionados por terceros. En particular la utilidad `module-assistant` permite automatizar todo este proceso a partir de los paquetes fuente del módulo.

Necesitamos disponer de los *headers* del núcleo instalado (disponible en el paquete `linux-headers-version`) o bien de los paquetes fuente que utilizamos en su compilación. A partir de aquí `module-assistant` puede utilizarse interactivamente y seleccionar entre una amplia lista de módulos registrados previamente en la aplicación, y puede encargarse de descargar el módulo, compilarlo e instalarlo en el núcleo existente.

También, en la utilización desde la línea de comandos, podemos simplemente especificar (`m-a` es equivalente a `module-assistant`):

```
# m-a prepare
# m-a auto-install nombre_modulo
```

Lo cual prepara el sistema para posibles dependencias, descarga fuentes del módulo, compila y, si no hay problemas, instala para el presente núcleo. De la lista interactiva de `module-assistant` podemos observar el nombre del módulo.

4. Aplicación de parches al núcleo

En algunos casos también puede ser habitual la aplicación de parches (*patches*) al núcleo [Lkm].

Un fichero de parche (*patch file*) respecto al núcleo de Linux es un fichero de texto ASCII que contiene las diferencias entre el código fuente original y el nuevo código, con información adicional de nombres de fichero y líneas de código. El programa *patch* (ver `man patch`) sirve para aplicarlo al árbol del código fuente del núcleo (normalmente, dependiendo de la distribución, en `/usr/src/linux`).

Los parches suelen necesitarse cuando un hardware especial necesita alguna modificación en el núcleo, o se han detectado algunos errores (*bugs*) posteriores a alguna distribución concreta de una versión del núcleo, o bien quiere añadirse una nueva prestación sin generar una versión de núcleo nueva. Para corregir el problema (o añadir la nueva prestación), se suele distribuir un parche en lugar de un nuevo núcleo completo. Cuando ya existen varios de estos parches, se unen con diversas mejoras del núcleo anterior para formar una nueva versión del mismo. En todo caso, si tenemos hardware problemático o el error afecta a la funcionalidad o a la estabilidad del sistema y no podemos esperar a la siguiente versión del núcleo, será necesario aplicar el/los parche(s).

El parche se suele distribuir en un fichero comprimido tipo bz2 (bunzip2, aunque también puede encontrarse en gzip con extensión .gz), como por ejemplo podría ser:

```
patchxxxx-2.6.xx-pversion.bz2
```

donde xxxx suele ser algún mensaje sobre el tipo o finalidad del parche. 2.6.xx sería la versión del núcleo al cual se le va a aplicar el parche, y `pversion` haría referencia a la versión del parche, del que también pueden existir varias. Hay que tener en cuenta que estamos hablando de aplicar parches a los paquetes fuente del núcleo (normalmente instalados, como vimos, en `/usr/src/linux` o directorio similar del usuario usado en la compilación del núcleo).

Una vez dispongamos del parche, tendremos que aplicarlo. Veremos el proceso a seguir en algún fichero `Readme` que acompaña al parche, pero generalmente el proceso sigue los pasos (una vez comprobados los requisitos previos) de descomprimir el parche en el directorio de los ficheros fuente y aplicarlo sobre las fuentes del núcleo, como por ejemplo:

```
cd /usr/src/linux (o /usr/src/linux-2.6.xx o la versión que sea).
bunzip2 patch-xxxxx-2.6.xx-version.bz2
patch -p1 < patch-xxxxx-2.6.xx-version
```

También puede aplicarse previamente con la opción `patch -p1 -dry-run` que solo procede a realizar un primer test, para asegurarnos previamente que no haya alguna condición de error cuando se sustituya el código. Si no hay error, volvemos a aplicar entonces sin la opción de test.

Posteriormente, una vez aplicado el parche, tendremos que recompilar el núcleo para volverlo a generar.

Los parches pueden obtenerse de diferentes lugares. Lo más normal es encontrarlos en el sitio de almacén de los núcleos *vanilla* (http://www.kernel.org), que tiene un archivo completo de los mismos. Determinadas comunidades Linux (o usuarios individuales) también suelen ofrecer algunas correcciones, pero es mejor buscar en los sitios estándar para asegurar un mínimo de confianza en estos parches y evitar problemas de seguridad con posibles parches "piratas". Otra vía es el fabricante de hardware que puede ofrecer ciertas modificaciones del núcleo (o de controladores en forma de módulos dinámicos de núcleo) para que funcionen mejor sus dispositivos (un ejemplo conocido es NVIDIA y sus controladores Linux propietarios para sus tarjetas gráficas).

Por último, señalaremos que muchas distribuciones de GNU/Linux (Fedora/Red Hat, Mandriva) ya ofrecen núcleos parcheados por ellos mismos y sistemas para actualizarlos (algunos incluso de forma automática, como en el caso de Fedora/Red Hat y Debian). Normalmente, en sistemas de producción es más recomendable seguir las actualizaciones del fabricante, aunque éste no ofrecerá necesariamente el último núcleo publicado, sino el que crea más estable para su distribución, con el inconveniente de perder prestaciones de última generación o alguna novedad en las técnicas incluidas en el núcleo.

Núcleo actualizado

En sistemas que se quieran tener actualizados, por razones de test o de necesidad de las últimas prestaciones, siempre se puede acudir a http://www.kernel.org y obtener el núcleo más moderno publicado.

Un ejemplo, en el caso de distribución de parches, podría ser el caso de la distribución Debian para los paquetes fuente de núcleo proporcionados por la distribución. Debian produce revisiones del núcleo aplicando diferentes series de parches sobre los fuentes originales, así su núcleo ofrecido en repositorios no es el original, sino el resultado de aplicar una serie de parches. Un ejemplo de proceso, para obtener un núcleo con diferentes parches aplica-

dos, podría ser el siguiente (donde xx es la revisión de la rama 2.6 usada por Debian):

```
# apt-get install linux-source-2.6.xx
```

Para desempaquetar el núcleo (lo encontraremos en `/usr/src/`):

```
# apt-get install linux-patch-debian-2.6.xx
# cd linux-source-2.6.xx
# /usr/src/kernel-patches/all/2.6.xx/apply/debian 2.6.xx-1
```

Esta última parte nos instala el conjunto de parches que Debian ha generado para la versión fuente del núcleo usado y, en el ultimo comando, nos permite volver a una revisión Debian concreta del núcleo (lo que se suele denominar un *patchlevel* de los fuentes del núcleo). Así, por ejemplo, si la revisión del núcleo es 2.6.xx-2, el comando anterior volvería (proceso denominado *rollback*) los fuentes del núcleo a la anterior versión *patchlevel* 2.6.xx-1. Así podemos escoger una versión del núcleo según el subconjunto de parches aplicados.

Por último, cabe comentar la incorporación de una tecnología reciente al uso de parches en Linux, Ksplice, que permite a un sistema Linux añadir parches al núcleo sin necesidad de parar y rearrancar el sistema. Básicamente, Ksplice determina a partir de los paquetes fuente cuáles son los cambios introducidos por un parche o una serie de ellos, y comprueba en memoria cómo afectan a la imagen del núcleo en memoria que se encuentra ejecutándose. Se intenta entonces parar la ejecución en el momento en que no existan dependencias de tareas que necesiten las partes del núcleo a parchear. Entonces se procede a cambiar, en el código objeto del núcleo, las funciones afectadas, apuntando a las nuevas funciones con el parche aplicado y modificando datos y estructuras de memoria que tengan que reflejar los cambios. Actualmente es un producto comercial, pero algunas distribuciones de comunidad ya lo están incluyendo debido al soporte gratuito que se ofrece para algunas. En los casos de producción en empresas, con servidores en los que es importante no disminuir el tiempo de servicio, puede ser una tecnología crítica para usar, altamente recomendable tanto para disminuir el tiempo de pérdida de servicio como para minimizar incidentes de seguridad que afecten al núcleo.

Ksplice

Ksplice es una tecnología muy útil para servidores empresariales en producción. Podéis consultar su web: http://www.ksplice.com

Básicamente ofrecen un servicio denominado *Ksplice Uptrack* que es una especie de actualizador de parches para el núcleo en ejecución. La gente de Ksplice sigue el desarrollo de los parches fuente del núcleo, los prepara en forma de paquetes que puedan incorporarse a un núcleo en ejecución y los hace disponibles en este servicio *uptrack*. Una herramienta gráfica gestiona estos paquetes y los hace disponibles para la actualización durante la ejecución.

5. Módulos del núcleo

Flexibilidad del sistema

Los módulos aportan una flexibilidad importante al sistema, permitiendo que se adapte a situaciones dinamicas.

El núcleo es capaz de cargar dinámicamente porciones de código (módulos) bajo demanda [Hen], para complementar su funcionalidad (se dispone de esta posibilidad desde la versión 1.2 del núcleo). Por ejemplo, los módulos pueden añadir soporte para un sistema de ficheros o para dispositivos de hardware específicos. Cuando la funcionalidad proporcionada por el módulo no es necesaria, el módulo puede ser descargado y así liberar memoria.

Normalmente bajo demanda, el núcleo identifica una característica no presente en el núcleo en ese momento, contacta con un hilo (*thread*) del núcleo denominado `kmod` (en las versiones del núcleo 2.0.x el *daemon* era llamado `kerneld`) y este ejecuta un comando `modprobe` para intentar cargar el módulo asociado a partir de una cadena con el nombre de módulo o bien de un identificador genérico. Esta información en forma de alias entre el nombre y el identificador se consulta en el fichero `/etc/modules.conf`.

A continuación se busca en `/lib/modules/version-kernel/modules.dep` para saber si hay dependencias con otros módulos. Finalmente, con el comando `insmod` se carga el módulo desde `/lib/modules/version_kernel/` (el directorio estándar para los módulos), la `version-kernel` es la versión del núcleo actual y se utiliza el comando `uname -r` para determinarla. Por tanto, los módulos en forma binaria están relacionados con una versión concreta del núcleo, y suelen colocarse en `/lib/modules/version-kernel`. Los módulos se reconocen como archivos dentro de la estructura de este directorio, con `.ko` como extensión de archivo.

En general, el administrador debe conocer cómo se cargan los módulos en el sistema. La mayor parte de veces por el proceso anterior, los módulos de la mayoría del hardware y necesidades concretas son detectados automáticamente en arranque o por demanda de uso y cargados en el momento correspondiente. En muchos casos no deberemos realizar ningún proceso como administradores. Pero en algunos casos, habrá que preparar alguna sintonización del proceso o de los parámetros de los módulos, o en algunos casos añadir nuevos módulos ya en forma binaria o por compilación a partir de los fuentes.

Si hay que realizar alguna compilación de módulos a partir de sus fuentes, se tiene que disponer de los paquetes fuente y/o *headers* de la versión del núcleo al cual está destinado.

Hay unas cuantas utilidades que nos permiten trabajar con módulos (solían aparecer en un paquete de software llamado `modutils`, que se reemplazó por `module-init-tools` para la gestión de módulos de la rama 2.6.x):

- `lsmod`: Podemos ver los módulos cargados en el núcleo (la información se obtiene del pseudofichero `/proc/modules`). Se listan los nombres, las dependencias con otros (entre corchetes, [ ]), el tamaño del módulo en bytes y el contador de uso del módulo; esto permite descargarlo si la cuenta es cero.

Ejemplo

Algunos módulos en un Debian:

```
Module                  Size  Used by    Tainted: P
agpgart                37344   3   (autoclean)
apm                    10024   1   (autoclean)
parport_pc             23304   1   (autoclean)
lp                      6816   0   (autoclean)
parport                25992   1   (autoclean) [parport_pc lp]
snd                    30884   0
af_packet              13448   1   (autoclean)
nvidia               1539872  10
es1371                 27116   1
soundcore               3972   4   [snd es1371]
ac97_codec             10964   0   [es1371]
gameport                1676   0   [es1371]
3c59x                  26960   1
```

- `modprobe`: Intenta la carga a mano de un módulo y de sus dependencias.
- `insmod`: Carga un módulo determinado.
- `depmod`: Analiza dependencias entre módulos y crea un fichero de dependencias.
- `rmmod`: Saca un módulo del núcleo.
- `depmod`: Usado para generar el fichero de dependencias de los módulos, que se encuentra en `/lib/modules/version-kernel/modules.dep` y que incluye las dependencias de todos los módulos del sistema. Si se instalan nuevos módulos de núcleo, es interesante ejecutar manualmente este comando para actualizar las dependencias. También se suelen generar automáticamente al arrancar el sistema.
- Se pueden usar otros comandos para depuración o análisis de los módulos, como `modinfo`, que lista informaciones asociadas al módulo (como licencias, descripción, uso y dependencias), o ficheros `/proc/kallsyms`, que nos permiten examinar los símbolos exportados por los módulos.

Ya sea por el mismo núcleo o por el usuario manualmente con `insmod`, normalmente para la carga se especificará el nombre del módulo y, opcionalmente, determinados parámetros; por ejemplo, en el caso de dispositivos suele ser habitual especificar las direcciones de los puertos de E/S o bien los recursos de IRQ o DMA. Por ejemplo:

```
insmod soundx io=0x320 irq=5
```

La carga general de módulos, en función del momento y la forma, puede hacerse manualmente, como hemos comentado, mediante `initrd/initramfs` o por medio de `udev`.

En el caso de `initrd/initramfs`, cuando el sistema arranca, se necesitan inmediatamente algunos módulos, para acceder al dispositivo y al sistema de ficheros raíz del sistema, por ejemplo controladores específicos de disco o tipos de sistemas de ficheros. Estos módulos, necesarios se cargan mediante un sistema de ficheros especial en RAM denominado `initrd/initramfs`. Dependiendo de la distribución GNU/Linux se utilizan estos términos de forma indiferente, aunque en algunos casos se han producido cambios a lo largo de la vida de la distribución. Por convención, suele denominarse a este elemento como *filesystem* RAM inicial, y se le refiere comunmente como `initramfs`.

El sistema inicial de ficheros en RAM, `initramfs`, es cargado por el *bootloader* en la especificación de la entrada correspondiente a la carga del núcleo correspondiente (por ejemplo en la línea/opción `initrd` de la entrada correspondiente de Grub).

En la mayoría de distribuciones, ya sea con el núcleo distribuido originalmente o bien con nuestra configuración del núcleo, suele crearse un `initramfs` inicial. En todo caso, si dependiendo de la distribución no se produce (puede no ser necesario), entonces podemos crearlo y sintonizarlo manualmente. El comando `mkinitramfs` permite crearlo a partir de sus opciones genéricas, que no suelen cambiarse y que se pueden configurar en el archivo `/etc/initramfs-tools/initramfs.conf` y, específicamente, los módulos que se cargarán en inicio automáticamente, que podemos encontrarlos en */etc/initramfs-tools/modules*.

Un `mkinitramfs -o new_initrd_file` nos permitirá crearlo, y normalmente podemos proceder a copiarlo en el directorio `/boot` para hacerlo accesible al *bootloader* usado. Por ejemplo, mediante un cambio en Grub de su fichero de configuración `/boot/grub/menu.lst`, modificando la línea de `initrd` oportuna. En cualquier caso, siempre es interesante establecer en el *bootloader* una configuración alternativa de text durante estas pruebas, para poder reiniciar y probar la nueva configuración, pero de la misma manera mantener la configuración estable antigua.

Durante el proceso de arranque, además del `initramfs` necesario para el arranque inicial, se producirá también la carga del resto de módulos por detección automática. Si no se carga algún módulo deseado siempre puede forzarse su carga al incluir su nombre implícitamente en el fichero de configuración `/etc/modules`.

También puede darse o desearse el caso contrario: evitar la carga de un módulo que puede detectarse erróneamente o para el que existe más de una alternativa posible. En este caso se utilizan técnicas de listas negras de módulos (típicamente la lista negra se guarda en `/etc/modprobe.d/blacklist.conf`).

5.1. DKMS: módulos recompilados dinámicamente

Respecto a los módulos dinámicos, un problema clásico ha sido la recompilación de estos frente a nuevas versiones del núcleo. Los módulos dinámicos de terceros, no incluidos *a priori* en el núcleo, necesitan de código fuente para compilarse, proporcionado por la comunidad o por el fabricante del hardware del dispositivo.

Durante la compilación, normalmente es necesario disponer de los paquetes de desarrollo del núcleo, de los paquetes del código fuente del núcleo y de sus *headers* para desarrollo, con la misma numeración que el núcleo usado actualmente, para el que se quiere compilar el módulo. Este hecho obliga a una recompilación constante con el cambio de versiones del núcleo del sistema, en especial ahora que las distribuciones distribuyen revisiones del núcleo con un menor tiempo, ya sea para solventar potenciales problemas de seguridad o para corregir errores detectados.

Algunas distribuciones, para minimizar esta problemática, distribuyen un entorno denominado DKMS, que permite facilitar la recompilación automática de un módulo con los cambios de versión del núcleo. Esto normalmente se produce en arranque al detectar un numero de núcleo: todos los módulos de terceros registrados por el sistema DKMS se recompilan a partir de los archivos fuente de los módulos y de los archivos fuente o *headers* del núcleo nuevo. De esta manera el proceso total es transparente al usuario. Una vez realizado este proceso, el sistema o el usuario mediante comandos de manipulación de módulos (como `modprobe`), pueden utilizar directamente el nuevo módulo.

Algunas distribuciones ofrecen solo el paquete base (dkms) del sistema, en algunas se proporcionan paquetes dkms preparados para módulos concretos, o incluso el fabricante puede ofrecer su módulo con soporte dkms.

El proceso habitualmente pasa por los siguientes pasos:

1) Obtener los archivos fuente del módulo (un paquete o un archivo TGZ suele ser lo más normal).

2) Instalar los paquetes necesarios para el proceso: `dkms`, `kernel-source`, `kernel-headers` (los nombres dependen de la distribución, y en el caso de los paquetes fuente del núcleo, hay que tener en cuenta que sean las versiones asociados a la versión del núcleo actual para la que se quiere instalar el módulo).

3) Activar el servicio DKMS en arranque. Habitualmente el servicio es denominado `dkms_autoinstaller`

4) Crear un directorio en `/usr/src` para los paquetes fuente del módulo y colocarlos allí.

5) Crear un fichero `dkms.conf` en el directorio anterior, que le especifica como construir (compilar) e instalar el módulo.

6) Añadir el servicio a la base de datos DKMS, compilarlo e instalar, normalmente con unos comandos:

```
dkms add -m nombre-modulo -v numero-version-modulo
dkms build   -m nombre-modulo -v numero-version-modulo
dkms install -m nombre-modulo -v numero-version-modulo
```

Respecto al fichero `dkms.conf` mencionado, podría ser como sigue (donde `nombre-modulo` es el nombre del módulo y `version`, el código numérico de su versión):

```
#
# /usr/src/nombre-modulo/dkms.conf
#

PACKAGE_NAME="nombre-modulo"
PACKAGE_VERSION="version"
CLEAN="make clean"
MAKE[0]="make module"
BUILD_MODULE_NAME[0]="nombre-modulo"
DEST_MODULE_LOCATION[0]="/kernel/drivers/video"
AUTOINSTALL="yes"

# End Of File
```

En este caso tenemos ubicados los paquetes fuente del módulo en un directorio `/usr/src/nombre-modulo` donde ponemos este fichero `dkms.conf`. La opción `MAKE` da los pasos para compilar y construir el módulo, previamente limpiados con `CLEAN`. `BUILD_MODULE_NAME` establece el nombre del módulo construido (cuidado en este punto porque depende del sistema de compilación y puede coincidir con el nombre del módulo general o no, algunos paquetes fuente permiten construir varios controladores/módulos diferentes, con diferente denominación).

`DEST_MODULE_LOCATION` define donde se instalara el módulo final en el árbol asociado al núcleo, en este caso suponemos que es un controlador de vídeo, (recordad que la raíz está en `/lib/modules/version-kernel`, lo que se coloca aquí es a partir de esta raíz). `AUTOINSTALL` permite que se reconstruya automáticamente el módulo durante cambios del núcleo actual.

En los casos de

`CLEAN`, `MAKE`, `BUILD_MODULE_NAME` y `DEST_MODULE_LOCATION`

se recomienda consultar el fichero de explicación (normalmente un `README` o `INSTALL`) que acompaña a los paquetes fuentes de los módulos, ya que pueden ser necesarios comandos adicionales, o tener que modificarlos para que se compile y se instale correctamente el módulo.

6. Virtualización en el núcleo

Una de las áreas en expansión, en la administración de IT, es la virtualización de sistemas. GNU/Linux ha ido con el tiempo incorporando diferentes posibilidades provenientes tanto de soluciones comerciales, como de diferentes proyectos de código abierto.

La virtualización de sistemas es un recurso básico actual en las empresas y organizaciones para mejorar su administración de sistemas, disminuir costes y aprovechar los recursos de hardware de manera más eficiente.

En general, en el pasado si necesitábamos varias instancias de uno (o más) sistemas operativos, teníamos que adquirir un servidor para cada instancia a implantar. La corriente actual es comprar servidores mucho más potentes y utilizar virtualización en estos servidores para implantar los diferentes sistemas en desarrollo o producción.

Normalmente, en virtualización disponemos de un sistema operativo instalado (que habitualmente se denomina sistema *host*) y una serie de máquinas virtuales sobre este sistema (denominados sistemas *guest*). Aunque también hay soluciones que sustituyen al sistema operativo *host* por una capa denominada *hypervisor*.

La virtualización como solución nos permite optimizar el uso de nuestros servidores o bien, por ejemplo en el caso de escritorio, disponer de máquinas de test de otros sistemas operativos conviviendo en la misma máquina simultáneamente. En el caso de GNU/Linux disponemos de múltiples soluciones que permiten tanto un caso como el otro. También podemos disponer tanto de GNU/Linux como sistema *host* que aloja máquinas virtuales, como utilizarlo como máquina virtual sobre otro sistema diferente o bien sobre otro sistema *host* también GNU/Linux. Un esquema, este último, particularmente útil en el caso de administración porque nos permitirá, por ejemplo, examinar y ejecutar diferentes distribuciones GNU/Linux sobre un mismo sistema *host* base.

Existen muchas soluciones de virtualización, pero por mencionar algunas de las más populares en sistemas GNU/Linux, disponemos (ordenamos de más a menos en relación directa con el núcleo):

- KVM
- Xen
- OpenVZ

- VirtualBox
- VMware

VMware es uno de los líderes comerciales en soluciones de virtualización, y dispone de productos de virtualización para escritorio (VMware Workstation), mientras que para servidor dispone de un producto VMware Server que está disponible para Linux para descarga gratuita. El caso de servidor permite gestionar varias máquinas virtuales con un interfaz de gestión simple. Otra línea de producto VMware ESX implementa necesidades mayores para centros de datos (*data centers*), con gestión elaborada de máquinas, tolerancia a fallos, migraciones y otras necesidades explícitas para centros de datos.

Enlace de interés

Podéis visitar la web de VMware en: http://www.vmware.com

Sun/Oracle VirtualBox ofrece virtualización orientada a escritorio, que nos permite una opción bastante sencilla para probar máquinas con diferentes sistemas operativos. Dispone de versión de código libre utilizable en gran número de distribuciones GNU/Linux.

OpenVZ es una solución de virtualización que utiliza el concepto de contenedor de máquina virtual. Así el *host* arranca con un núcleo común a las máquinas virtuales (existen paquetes de imágenes de núcleo con soporte OpenVZ integrado en las distribuciones, como por ejemplo en Debian), que permite arrancar máquinas con el núcleo en común pero cada una dentro de un entorno aislado del resto. OpenVZ solo permite máquinas virtuales *guest* Linux (debido al núcleo compartido).

Xen usa el concepto de *hypervisor*, utilizando un tipo de virtualización denominada *paravirtualización*, en la que se elimina el concepto de *host-guest* y se delega a la capa de *hypervisor* la gestión de los recursos físicos de la máquina, de manera que permita el máximo acceso a los recursos de hardware por parte de las máquinas virtuales. En estos casos se necesitan núcleos sintonizados que puedan beneficiarse de las posibilidades de la paravirtualización. En el caso de GNU/Linux en la mayoría de distribuciones se ofrecen núcleos optimizados para Xen (véase el caso de Debian, para el que existen imágenes binarias para xen de los núcleos). En general, la paravirtualización y la capa de *hypervisor* para acceder al hardware aprovechan las facilidades de las CPU actuales, con recursos de hardware dedicados a facilitar la virtualización. Si se dispone de las características se pueden usar sistemas como Xen, sino, puede utilizarse un sistema más clásico de *host-guest* como por ejemplo VirtualBox.

En general, para ver si la CPU dispone de soporte de virtualización, hay que examinar sus datos en `/proc/cpuinfo`, en concreto el *flag* `VMX`, para procesadores Intel, que puede verse en la sección `flags`:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
```

```
cpu family      : 6
model           : 23
model name      : Intel(R) Xeon(R) CPU E5405  @ 2.00GHz
stepping        : 10
cpu MHz         : 1994.999
cache size      : 6144 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
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 dts acpi mmx fxsr sse sse2
ss ht tm syscall   nx lm constant_tsc pni monitor ds_cpl
vmx tm2 ssse3 cx16 xtpr sse4_1 lahf_lm
bogomips        : 3989.99
clflush size    : 64
cache_alignment : 64
address sizes   : 38 bits physical, 48 bits virtual
power management:
```

6.1. KVM

Finalmente, la solución en que nos centraremos en este subapartado, **KVM**, está presente desde el núcleo 2.6.20, como solución incluida para la virtualización de sistemas. Es una solución parecida a Xen, pero con diferencias en cuanto a la implementación. Xen es un producto complejo, con diferentes capas y, en especial, su diseño de capa hipervisora. Por contra, KVM se implementa como un módulo del núcleo existente, que se complementa con otras soluciones. Es la opción por defecto para virtualización en distribuciones como Debian y Fedora.

Enlace de interés

Podéis acceder a la web de KVM en: http://www.linux-kvm.org

Normalmente una solución de virtualización basada en KVM se compone de una mezcla de:

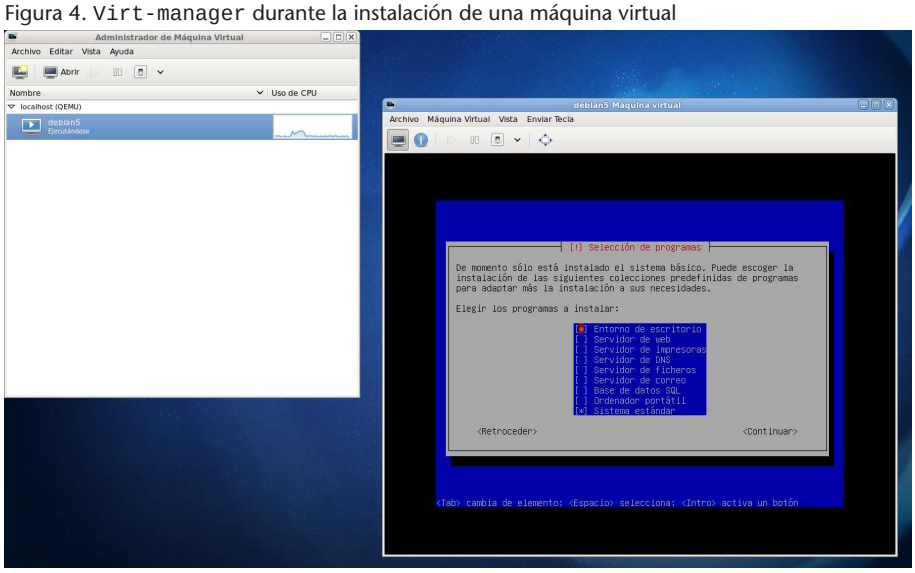
- El módulo de núcleo `kvm.ko`, que proporciona la infraestructura base de virtualización, y además un módulo adicional según el modelo de procesador `kvm-intel.ko` o `kvm-amd.ko`. Estos módulos deberán cargarse manualmente (`modprobe`) o habilitarlos durante el arranque para disponer de soporte de virtualización KVM.

- Qemu, un simulador cruzado de arquitecturas de CPU, que nos permite simular una CPU virtual sobre otra de igual o diferente arquitectura real. KVM utiliza una versión modificada de Qemu para la gestión y creación de las máquinas *guest* (este paquete suele aparecer como `qemu-kvm` en las distribuciones).

- La biblioteca `libvirt`, que es una API que permite una gestión de la virtualización independiente del sistema de virtualización usado (sea Xen, KVM u otros).

- Utilidades basadas en `libvirt` para la creación de las VM *guest* y su mantenimiento, como `virt-manager` una interfaz gráfica de gestión de máquinas virtuales (figura 4), `virt-install`, una utilidad de línea para la gestión de máquinas virtuales, o `virtsh`, un *shell* basado en comandos de gestión de las máquinas virtuales. Estas utilidades suelen necesitar un servicio de sistema, denominado `libvirtd`. Tenemos que asegurarnos de poner en marcha el servicio (`/etc/init.d/libvirtd start`) o bien de cargarlo al inicio.

Figura 4. `Virt-manager` durante la instalación de una máquina virtual



A continuación veremos los procesos básicos asociados a la utilización de KVM y describiremos algunas de las fases de la instalación de KVM y la puesta en marcha de algunas máquinas virtuales.

Para comenzar debemos examinar si disponemos de soporte de harwdare en la CPU: como hemos comentado buscamos el *flag* `vmx` en `/proc/cpuinfo` (para procesadores Intel) o el *flag* `svm` para procesadores AMD. Por ejemplo, con (sustituir `vmx` por `svm` en AMD):

```
grep vmx /proc/cpuinfo
```

Soporte de virtualización

Hay que tener cuidado con el hecho de que muchas de las máquinas actuales permiten desactivar/activar en BIOS el soporte de virtualización; comprobemos primero que no esté desactivado.

obtendremos como resultado la línea de *flags* (si existe el *flag* `vmx`, si no, ningún resultado). También podemos obtener varias líneas en CPU *multicore* y/o Intel con *hyperthreading*, donde obtenemos una línea de *flags* por cada elemento de cómputo (CPU con HT o múltiples núcleos con/sin HT).

Una vez determinado el correcto soporte de virtualización, procedemos a instalar los paquetes asociados a KVM; estos ya dependerán de la distribución, pero en general, el mismo "kvm" ya obtendrá la mayoría de paquetes requeridos como dependencias. En general, los paquetes recomendados a instalar (vía apt o yum) son `kvm, qemu-kvm, libvirt`. Dependiendo de la distribución pueden cambiar algunos nombres de paquetes, y en especial con el nombre `virt` existen varios paquetes de utilidades de generación y monitorización de máquinas virtuales tanto de KVM como de Xen.

Si se permite a los usuarios del sistema el uso de máquinas virtuales, hay que añadir los nombres de usuario a grupos específicos (vía comandos `adduser` o `usermod`), normalmente a los grupos `kvm` o `libvirt` (dependiendo de la distribución, Fedora o Debian, y versión de KVM).

El siguiente paso (opcional) es facilitar el uso de red a las máquinas virtuales. Por defecto KVM utiliza NAT, dando direcciones IP privadas de tipo 10.0.2.x, y accediendo mediante la red de la máquina *host*. En otro caso, si queremos una configuración diferente (por ejemplo que permita acceso externo a las máquinas virtuales) tendremos que permitir hacer de *bridge* a la red actual; en este caso es necesario instalar el paquete `bridge-utils` y configurar un dispositivo especial de red denominado `br0`, en Debian en la configuración de red presente en `/etc/network/interface` y en Fedora puede crearse un fichero asociado al dispositivo como `/etc/sysconfig/network-scripts/ifcfg-br0`. Por ejemplo, en Debian podría colocarse un ejemplo de configuración como:

Enlace de interés

La gestión de red es uno de los temas complejos en virtualización, en el caso de KVM se recomienda examinar: http://www.linux-kvm.org/page/Networking.

```
auto lo
iface lo inet loopback

auto br0
iface br0 inet static
       address 192.168.0.100
       network 192.168.0.0
       netmask 255.255.255.0
       broadcast 192.168.0.255
       gateway 192.168.0.1
       bridge_ports eth0
       bridge_fd 9
       bridge_hello 2
       bridge_maxage 12
       bridge_stp off
```

Esta configuración permite que se cree un dispositivo `br0` para reemplazar a `eth0`. Así las tarjetas de red virtuales redirigirán el tráfico asignadas a este dispositivo. `bridge_ports` especifica cuál será el dispositivo físico real que se utilizará.

Como comentamos, esta parte de configuración de red es opcional, y solo tiene sentido si queremos acceder desde el exterior a nuestras máquinas virtuales. Por el contrario, en un entorno de virtualización de escritorio puede ser suficiente con el modo NAT por defecto, ya que las máquinas dispondrán de salida de red a través de la red del *host*.

A partir de estas configuraciones, ya estaremos capacitados para crear las imágenes de las máquinas virtuales. Hay diferentes conjuntos de comandos para realizarlo, bien usando kvm directamente (comando `kvm`), utilidades asociadas a qemu para creación de estas imágenes (`qemu-img`) o utilidades asociadas a libvirt (`virt-install`). El proceso pasa por crear la imagen de disco (o espacio de disco) asociado a la máquina *guest* como un fichero y después realizar la instalación del sistema operativo en ese fichero (imagen de disco), bien desde el CD/DVD de instalación del sistema operativo o bien también desde una imagen `*.iso` del CD/DVD.

Por ejemplo, supongamos una instalación de un *guest* determinado (por ejemplo disponemos de unos CD/DVD de la distribución Debian).

Creamos el espacio de disco para la máquina virtual (en este caso 8 GB):

```
# dd if=/dev/zero of=~/debianVM.img bs=1M count=8192
```

Mediante el comando `dd` creamos así un fichero de salida de 8192 bloques de 1 MB, es decir, 8 GB, que nos formará la unidad de disco para nuestra máquina virtual (también existe un comando alternativo para crear imágenes, `qemu-img`, véase página man).

Lo siguiente es obtener el medio de instalación del sistema operativo a instalar, bien proporcionado como CD/DVD y, por tanto, accesible en el dispositivo `/dev/cdrom` (o equivalente si tenemos más unidades de CD/DVD) o, por contra, a partir de una imagen `iso` del soporte. En cualquier caso, este último siempre lo podemos obtener a partir de los discos con:

```
dd if=/dev/cdrom of=debian-install.iso
```

que nos genera así la imagen del CD/DVD.

Ahora que disponemos de la imagen binaria y el medio de instalación del sistema operativo, podemos proceder a instalar, con diferentes utilidades. En

general suele utilizarse `virt-install` como comando para crear la VM, pero también existe la posibilidad de usar el mencionado `qemu-kvm` directamente como opción más simple, que suele aparecer (dependiendo de la distribución) como comando denominado `qemu-kvm, qemu-system-x86_64` o simplemente como `kvm`:

```
kvm -m 512 -cdrom debian-install.iso -boot d -hda debianVM.img
```

En este caso crearía una máquina virtual básica de 512 MB de memoria principal usando nuestro disco de 8 GB creado previamente y arrancando la máquina a partir de nuestra imagen `iso` del medio de instalación del sistema operativo. Se puede sustituir el fichero `iso` por `/dev/cdrom` si usamos los discos de instalación.

La otra posible alternativa de creación se realiza mediante la utilidad de línea de comandos `virt-install`, mucho más completa, pero con la dificultad de tener que especificar muchos más parámetros. Por ejemplo, podríamos indicar la creación de la máquina anterior mediante:

```
virt-install --connect qemu:///system -n debian5 -r 512 \
--vcpus=2 -f debianVM.img -s 8 -c debianinstall.iso --vnc \
--noautoconsole --os-type linux --os-variant debianLenny
```

que entre otros parámetros, coloca el método de conexión a la máquina virtual, el nombre de la VM `debian5`, define 512 MB de memoria, hace disponibles 2 núcleos de la máquina física, utiliza el disco virtual `debianVM`, de 8 GB (`-s` permite que si no existe, lo cree previamente con ese tamaño de GB), utiliza la `iso` como medio de instalación y permitirá conexiones gráficas con vnc con la máquina virtual. Además definimos que el sistema que va a instalarse es Linux, en especial una versión de Debian. Los parámetros de tipo y variante del sistema operativo son altamente dependientes de la versión de `virt-install`, de manera que vale la pena consultar su página man (`man virt-install`) y la lista de sistemas operativos compatibles con la versión KVM del núcleo y el conjunto de utilidades `qemu` y `libvirt`.

Enlace de interés

La lista de compatibilidad de KVM puede encontrarse en: http://www.linux-kvm.org/page/Guest_Support_Status

En este punto solo hemos creado la máquina, pero no estamos conectados a ella, y hemos realizado una configuración muy básica de sus parámetros. Con otras utilidades, por ejemplo las basadas en `libvirt`, como la interfaz gráfica `virt-manager`, podemos personalizar más la VM creada, por ejemplo añadiendo o quitando hardware virtual al sistema.

Podemos conectar después con la máquina recién creada mediante:

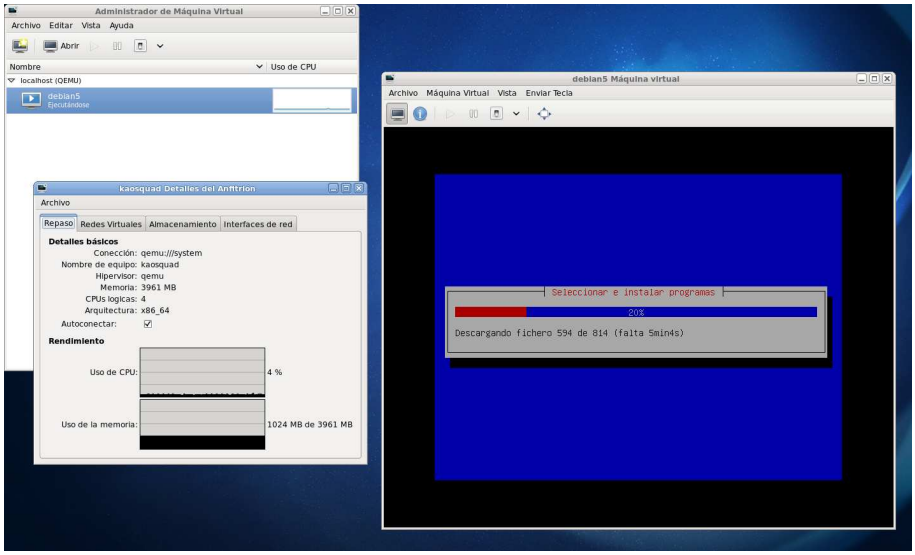
```
virt-viewer -c qemu:///system nombreVM
```

si está en el mismo sistema, o si estamos en una máquina remota con:

```
virt-viewer -c qemu+ssh://ip/system nombreVM
```

o usando directamente la interfaz `virt-manager` (figura 5).

Figura 5. `Virt-manager` conectado a la máquina virtual durante el proceso de instalación, observando los recursos usados por la máquina virtual



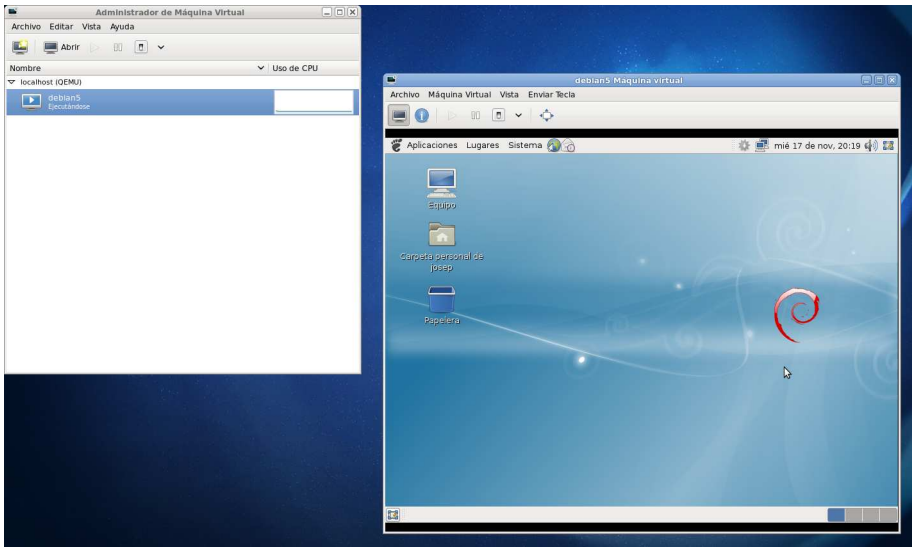
Esto nos permite conectar gráficamente con la máquina virtual creada y, por ejemplo, continuar con la instalación del sistema operativo (se habrá iniciado con el arranque anterior con `virt-install` o `kvm`). Una vez instalado el sistema, ya podemos usar cualquier sistema, ya sea gráfico (`virt-manager`) o de línea de comandos (`virtsh`) para gestionar las máquinas *guest* virtuales y poder arrancar y pararlas.

Por ejemplo, con `virtsh`:

```
virtsh --connect qemu:///system
```

conectamos con el *shell* de comandos, comandos como `list` nos dan las máquinas activas, `list -all`, todas las máquinas disponibles, y otros como `start`, `shutdown`, `destroy`, `suspend` o `resume` nos dan diferentes posibilidades de gestión de cada máquina virtual.

Figura 6. `Virt-manager` conectado a la máquina virtual recién instalada



7. Futuro del núcleo y alternativas

Los avances en el núcleo de Linux en determinados momentos fueron muy rápidos, pero actualmente, ya con una situación bastante estable con los núcleos de la rama 2.6.x, cada vez pasa más tiempo entre las versiones que van apareciendo. En cierta manera esto es bastante positivo: permite tener tiempo para corregir errores cometidos, ver aquellas ideas que no funcionaron bien y probar nuevas ideas, que, si resultan, se incluyen.

El núcleo, en evolución

El núcleo continúa evolucionando, incorporando las últimas novedades en soporte de hardware y mejoras en las prestaciones.

Comentaremos en este apartado algunas de las ideas de los últimos núcleos y algunas que están previstas, para dar indicaciones de lo que será el futuro próximo en el desarrollo del núcleo.

En la antigua rama 2.4.x del núcleo [Ces06] se realizaron algunas aportaciones en:

- Cumplimiento de los estándares IEEE POSIX, lo que permite que muchos de los programas existentes de UNIX pueden recompilarse y ejecutarse en Linux.
- Mejor soporte de dispositivos: PnP, USB, puerto paralelo, SCSI, etc.
- Soporte para nuevos sistemas de ficheros, como UDF (CD-ROM reescribibles como un disco). Otros sistemas con *journal*, como los Reiser de IBM o el ext3, que permiten tener un registro (*journal*) de las modificaciones de los sistemas de ficheros, y así poder recuperarse de errores o tratamientos incorrectos de los ficheros.
- Soporte de memoria hasta 4 GB. En su día surgieron algunos problemas (con núcleos 1.2.x) que no soportaban más de 128 MB de memoria (una cantidad que en aquel tiempo era mucha memoria).
- Se mejoró la interfaz `/proc`. Se trata de un pseudosistema de ficheros (el directorio `/proc`) que no existe realmente en disco, sino que es simplemente una forma de acceder a datos del núcleo y del hardware de una manera organizada.
- Soporte del sonido en el núcleo: se añadieron parcialmente los controladores Alsa que antes se configuraban por separado.
- Se incluyó soporte preliminar para el RAID software y el gestor de volúmenes dinámicos LVM1.

En la serie actual, la rama del núcleo 2.6.x [Ces06, Pra03], se dispuso de importantes avances respecto a la anterior (con las diferentes revisiones .x de la rama 2.6):

- Mejores prestaciones en SMP, importante para sistemas multiprocesadores muy utilizados en entornos empresariales y científicos.

- Mejoras en el planificador de CPU (*scheduler*). En particular, se introducen avances para mejorar el uso de tareas interactivas de usuario, imprescindibles para mejorar el uso de Linux en un ambiente de escritorio.

- Mejoras en el soporte *multithread* para las aplicaciones de usuario. Se incorporan nuevos modelos de hilos NGPT (IBM) y NPTL (Red Hat) (con el tiempo se consolidó finalmente la NPTL).

- Soporte para USB 2.0 y, posteriormente, para USB 3.0.

- Controladores Alsa de sonido incorporados en el núcleo.

- Nuevas arquitecturas de CPU de 64 bits, se soportan AMD x86_64 (también conocida como amd64) y PowerPC 64 y IA64 (arquitectura de los Intel Itanium).

- Sistemas de ficheros con *journal*: JFS, JFS2 (IBM) y XFS (Silicon Graphics).

- Mejoras con *journal* en los sistemas de ficheros propios, `ext3` y `ext4`, con mejoras del tamaño máximo de los archivos y de rendimiento general.

- Mejoras de prestaciones en E/S y nuevos modelos de controladores unificados.

- Mejoras en implementación de TCP/IP y el sistema NFSv4 (compartición de sistema de ficheros por red con otros sistemas).

- Mejoras significativas para núcleo apropiativo: permite que internamente el núcleo gestione varias tareas que se pueden interrumpir entre ellas, imprescindible para implementar eficazmente sistemas de tiempo real y también para aumentar el rendimiento de tareas interactivas.

- Suspensión del sistema y restauración después de reiniciar (por núcleo).

- UML, User Mode Linux, una especie de máquina virtual de Linux sobre Linux que permite ver un Linux (en modo usuario) ejecutándose sobre una máquina virtual. Esto es ideal para la propia depuración, ya que se puede desarrollar y probar una versión de Linux sobre otro sistema, y es útil tanto para el propio desarrollo del núcleo como para un análisis de seguridad del mismo. En versiones posteriores este concepto evolucionó hacia el módulo KVM.

- Técnicas de virtualización incluidas en el núcleo: en las distribuciones se han ido incorporando diferentes técnicas de virtualización, que necesitan extensiones en el núcleo. Cabe destacar, por ejemplo, núcleos modificados para Xen, Virtual Server (Vserver), OpenVZ o el propio módulo KVM.

- Nueva versión del soporte de volúmenes LVM2.

- Nuevo pseudosistema de ficheros `/sys`, destinado a incluir la información del sistema, y dispositivos que se irán migrando desde el sistema `/proc`, dejando este último con información relacionada con los procesos y su desarrollo en ejecución, así como la información dinámica del propio núcleo.

- Módulo FUSE para implementar sistemas de ficheros en espacio de usuario (en especial usado para el caso de NTFS).

Para conocer los cambios de las versiones más recientes de Linux, pueden examinarse los ficheros `ChangeLog` que acompañan a cada versión del núcleo, o consultar un registro histórico que se mantiene en *Kernelnewbies.org*, en especial http://kernelnewbies.org/LinuxChanges, que mantiene los cambios de la última versión y pueden consultarse los del resto de versiones (misma dirección con `/Linux26Changes`).

En el futuro se tiene pensado mejorar los aspectos siguientes:

- Incremento de la tecnología de virtualización en el núcleo, para soportar diferentes configuraciones de sistemas operativos y diferentes tecnologías de virtualización, así como un mejor soporte del hardware para virtualización incluido en los procesadores que surjan en las nuevas arquitecturas. Están bastante soportadas x86 y x86_64, con KVM, por ejemplo, pero otras no lo están o solamente parcialmente.

- El soporte de SMP (máquinas multiprocesador), de CPU de 64 bits (Xeon, nuevos multicore de Intel y Opteron de AMD), el soporte de CPU *multicore* y la escalabilidad de aplicaciones multihilo en estas CPU.

- La mejora de sistemas de ficheros para clusterización y grandes sistemas distribuidos.

- Por el contrario, la mejora en núcleos más optimizados para dispositivos móviles (PDA, *smartphones*, *tablets*, etc.).

- Mejora en el cumplimiento de los estándar POSIX.

Enlace de interés

Para saber más sobre POSIX podéis visitar la siguiente web:
http://www.unix.org/

- Mejora de la planificación de la CPU. Aunque en la serie inicial de la rama 2.6.x se hicieron muchos avances en este aspecto, todavía hay un bajo rendimiento en algunas situaciones. En particular en el uso de aplicacio-

nes interactivas de escritorio se están estudiando diferentes alternativas, para mejorar este y otros aspectos relacionas con el escritorio y el uso del rendimiento gráfico.

También, aunque se aparta de los sistemas Linux, la Free Software Foundation (FSF) y su proyecto GNU siguen trabajando en su proyecto de acabar un sistema operativo completo. Cabe recordar que el proyecto GNU tenía como principal objetivo conseguir un clon UNIX de software libre, y las utilidades GNU solo son el software de sistema necesario. A partir de 1991, cuando Linus consigue conjuntar su núcleo con algunas utilidades GNU, se dio un primer paso que ha acabado en los sistemas GNU/Linux actuales. Pero el proyecto GNU sigue trabajando en su idea de terminar el sistema completo. En este momento disponen ya de un núcleo en el que pueden correr sus utilidades GNU. A este núcleo se le denomina Hurd; y a un sistema construido con él se le conoce como GNU/Hurd. Ya existen algunas distribuciones de prueba, en concreto, una Debian GNU/Hurd.

Enlace de interés

Para saber más sobre el proyecto GNU podéis visitar la siguiente página web: http://www.gnu.org/gnu/thegnuproject.html

Hurd fue pensado como el núcleo para el sistema GNU hacia 1990, cuando comenzó su desarrollo, ya que entonces la mayor parte del software GNU estaba desarrollado y solo faltaba el núcleo. Fue en 1991 cuando Linus combinó GNU con su núcleo Linux y creó así el inicio de los sistemas GNU/Linux. Pero Hurd sigue desarrollándose. Las ideas de desarrollo en Hurd son más complejas, ya que Linux podría considerarse un diseño "conservador", que partía de ideas ya conocidas e implantadas.

Enlace de interés

Podéis leer las opiniones de Richard Stallman sobre GNU y Linux en: http://www.gnu.org/gnu/linux-and-gnu.html

En concreto, Hurd estaba pensada como una colección de servidores implementados sobre un micronúcleo Mach [Vah96], que es un diseño de núcleo tipo micronúcleo (a diferencia de Linux, que es de tipo monolítico) desarrollado por la Universidad Carnegie Mellon y posteriormente por la Universidad de Utah. La idea básica era modelar las funcionalidades del núcleo de UNIX como servidores que se implementarían sobre un núcleo básico Mach. El desarrollo de Hurd se retrasó mientras se estaba acabando el diseño de Mach, y este se publicó finalmente como software libre, que permitiría usarlo para desarrollar Hurd. En este punto debemos comentar la importancia de Mach, ya que muchos sistemas operativos se han basado en ideas extraídas de él, el más destacado de los cuales es el MacOS X de Apple.

El desarrollo de Hurd se retrasó más por la complejidad interna, ya que existían varios servidores con diferentes tareas de tipo *multithread* (de ejecución de múltiples hilos) y la depuración era extremadamente difícil. Pero hoy en día se dispone de algunas versiones de test, así como de versiones de prueba de distribución GNU/Hurd producidas por Debian. Con todo, el proyecto en si no es especialmente optimista de cara a obtener sistemas en producción, debido tanto a la complejidad como a la falta de soporte para dispositivos.

Puede que en un futuro no tan lejano pueda haber avances y coexistencia de sistemas GNU/Linux con GNU/Hurd, o incluso que sea sustituido el núcleo

Linux por el Hurd, si se hicieran avances importantes en su desarrollo. Esto sería una solución si en algún momento Linux se estanca (ya que su diseño monolítico puede causar problemas si se hace mucho más grande). En cualquier caso, tanto unos sistemas como otros tienen un prometedor futuro por delante. El tiempo dirá hacia dónde se inclina la balanza.

8. Taller de configuración del núcleo a las necesidades del usuario

En este apartado vamos a ver un pequeño taller interactivo para el proceso de actualización y configuración del núcleo en el par de distribuciones utilizadas: Debian y Fedora.

Una primera cosa imprescindible, antes de comenzar, es conocer la versión actual que tenemos del núcleo, mediante `uname -r`, para poder determinar cuál es la versión siguiente que queremos actualizar o personalizar. Y otra, es la de disponer de medios para arrancar nuestro sistema en caso de fallos: el conjunto de CD de la instalación, el disquete (o CD) de rescate (actualmente suele utilizarse el primer CD de la distribución) o alguna distribución en LiveCD que nos permita acceder al sistema de ficheros de la máquina, para rehacer configuraciones que hayan causado problemas. Además, deberíamos hacer copia de seguridad de nuestros datos o configuraciones importantes.

Veremos las siguientes posibilidades:

1) Actualización del núcleo de la distribución. Caso automático de Debian.

2) Actualización automática en Fedora.

3) Personalización de un núcleo genérico (tanto Debian como Fedora). En este último caso los pasos son básicamente los mismos que los que se presentan en el apartado de configuración, pero haremos algunos comentarios adicionales.

8.1. Configuración del núcleo en Debian

En el caso de la distribución Debian, la instalación puede hacerse también de forma automática, mediante el sistema de paquetes de APT. Puede hacerse tanto desde línea de comandos como con gestores APT gráficos (synaptic, gnome-apt, etc.).

Vamos a realizar la instalación por línea de comandos con `apt-get`, suponiendo que el acceso a los paquetes fuente apt (sobre todo a los Debian originales) está bien configurado en el fichero de `/etc/apt/sources.list`. Veamos los pasos:

1) Actualizar la lista de paquetes:

```
# apt-get update
```

2) Listar paquetes asociados a imágenes del núcleo:

```
# apt-cache search linux-image
```

3) Elegir una versión adecuada a nuestra arquitectura (genérica, 386/486/686 para Intel, k6 o k7 para amd o, en particular para 64 bits, versiones amd64 para intel y amd o ia64 para Intel Itanium). El código de la versión indica la versión del núcleo, la revisión de Debian del núcleo y la arquitectura. Por ejemplo, 2.6.xx-4-k7 es un núcleo para AMD Athlon, revisión Debian 4 del núcleo 2.6.xx.

4) Comprobar, para la versión elegida, que existan los módulos accesorios extras (con el mismo número de versión). Con `apt-cache` buscamos si existen otros módulos dinámicos que puedan ser interesantes para nuestro hardware, según la versión del núcleo a instalar. Recordad que, como vimos en la *Debian Way*, también existe la utilidad `module-assistant`, que nos permite automatizar este proceso después de la compilación del núcleo. En el caso en que los módulos necesarios no fueran soportados, esto nos podría impedir actualizar el núcleo si consideramos que el funcionamiento del hardware problemático es vital para el sistema.

5) Buscar, si queremos disponer también del código fuente del núcleo, los `linux-source-version` (solo el 2.6.xx, es decir, los números principales) y los `linux-headers` correspondientes, por si más tarde queremos hacer un núcleo personalizado (en este caso, el núcleo genérico correspondiente parcheado por Debian).

6) Instalar lo que hayamos decidido. Si queremos compilar desde los paquetes fuente o simplemente disponer del código:

```
# apt-get install linux-image-version
# apt-get install xxxx-modules-version
```

(si fueran necesarios algunos módulos) y

```
# apt-get install linux-source-version-generica
# apt-get install linux-headers-version
```

7) Instalar el nuevo núcleo, por ejemplo en el *bootloader* LiLo (deberemos comprobar el *bootloader* usado, ya que las últimas versiones de Debian usan Grub por defecto). Normalmente este paso se hace automáticamente, pero no estaría de más realizar alguna copia de seguridad previa de la configuración del *bootloader* (ya sea `/etc/lilo.conf` o `/boot/grub/menu.lst`).

Si se nos pregunta si tenemos el `initrd` activado, habrá que verificar el fichero de LiLo (`/etc/lilo.conf`) e incluir la nueva línea en la configuración LiLo de la imagen nueva:

```
initrd = /initrd.img-version (o /boot/initrd.img-version)
```

Una vez hecha esta configuración, tendríamos que tener un LiLo parecido al siguiente listado (fragmento del fichero), suponiendo que `initrd.img` y `vmlinuz` sean enlaces a la posición de los ficheros del nuevo núcleo:

```
default = Linux
image = /vmlinuz
  label = Linux
  initrd = /initrd.img
# restricted
# alias = 1
image = /vmlinuz.old
  label = LinuxOLD
  initrd = /initrd.img.old
# restricted
# alias = 2
```

Tenemos la primera imagen por defecto, la otra es el núcleo antiguo. Así, desde el menú LiLo podremos pedir una u otra o, simplemente cambiando el `default`, recuperar la antigua. Siempre que realicemos cambios en el archivo `/etc/lilo.conf` no debemos olvidarnos de reescribirlos en el sector correspondiente con el comando `/sbin/lilo` o `/sbin/lilo -v`.

En el caso de Grub, que suele ser la opción normal en las distribuciones, se habría creado una nueva entrada (hay que tener presente realizar la copia de seguridad previa, porque podemos haber perdido alguna entrada anterior dependiendo del funcionamiento o configuración de Grub; por ejemplo, puede limitarse el número máximo de entradas):

```
title           Debian GNU/Linux, kernel 2.6.32-5-amd64
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.32-5-amd64 \
                root=UUID=4df6e0cd-1156-444e-bdfd-9a9392fc3f7e ro
initrd          /boot/initrd.img-2.6.32-5-amd64
```

donde aparecerían los ficheros binarios del núcleo y `initrd`. Con la etiqueta `root` en el núcleo aparece el identificador de la partición raíz del sistema donde está instalado el núcleo, identificador que es común a todas las entradas de núcleos del mismo sistema. Antes se utilizaba un esquema con `root=/dev/hda0` o `/dev/sda0`, pero este esquema ya no es útil, porque la detección de los discos puede cambiar el orden de estos en el sistema; así, se prefiere etiquetar las particiones. Estas etiquetas pueden obtenerse mediante el comando `e2label /dev/particion` o también con el comando `dumpe2fs /dev/particion | grep UUID` o, si la partición se encuentra montada en el sistema, consultando `/etc/fstab`.

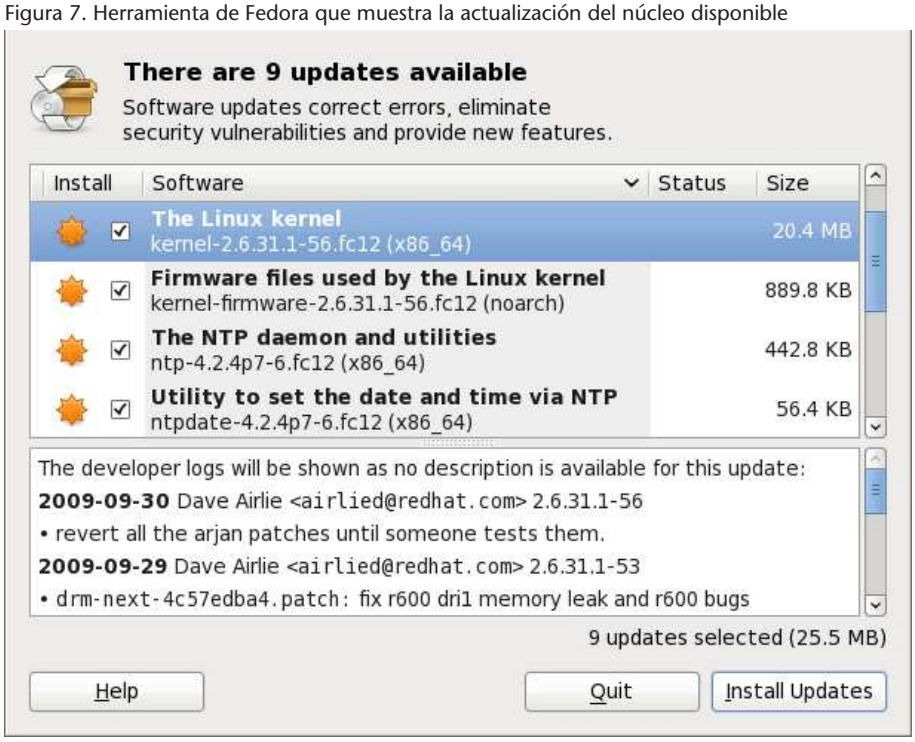
8.2. Configuración del núcleo en Fedora/Red Hat

La actualización del núcleo en la distribución Fedora/Red Hat es totalmente automática por medio de su servicio de gestión de paquetes (`yum` en línea de comandos, por ejemplo), o bien mediante los programas gráficos que incluye la distribución, dependiendo de su versión, para la actualización (PackageKit en Fedora o pup en Red Hat empresarial o equivalentes como CentOS). Normalmente, las encontraremos en la barra de tareas o en el menú de herramientas de sistema de Fedora/Red Hat.

Este programa de actualización básicamente verifica los paquetes de la distribución actual frente a una base de datos de Fedora/Red Hat, y ofrece la posibilidad de descargar los paquetes actualizados, entre ellos los del núcleo. Este servicio, en el caso de Red Hat empresarial, funciona por una cuenta de servicio y Red Hat lo ofrece por pago. Con este tipo de utilidades la actualización del núcleo es automática. Hay que comprobar las utilidades disponibles en los menús Herramientas/Administración, ya que las herramientas gráficas disponibles en una distribución son altamente dependientes de su versión, puesto que son actualizadas con frecuencia.

Por ejemplo, en la figura 7, observamos que una vez puesto en ejecución nos ha detectado una nueva versión del núcleo disponible y podemos seleccionarla para que nos la descargue.

Figura 7. Herramienta de Fedora que muestra la actualización del núcleo disponible



En Fedora podemos utilizar las herramientas gráficas equivalentes o usar directamente yum, si conocemos la disponibilidad de nuevos núcleos:

```
# yum install kernel kernel-source
```

Una vez descargada, se procederá a su instalación, normalmente también de forma automática, ya dispongamos de Grub o LiLo como gestores de arranque. En el caso de Grub, suele ser automático y deja un par de entradas en el menú, una para la versión más nueva y otra para la antigua. Por ejemplo, en esta configuración de Grub (el fichero está en `/boot/grub/grub.conf` o bien `/boot/grub/ menu.lst`), tenemos dos núcleos diferentes, con sus respectivos números de versión:

```
#fichero grub.conf
default = 1
timeout = 10
splashimage = (hd0,1)/boot/grub/splash.xpm.gz

title Linux (2.6.30-2945)
root (hd0,1)
kernel /boot/vmlinuz-2.6.30-2945 ro
       root = UUID=4df6e0cd-1156-444e-bdfd-9a9392fc345f
initrd /boot/initrd-2.6.30-18.9.img

title LinuxOLD (2.6.30-2933)
root (hd0,1)
kernel /boot/vmlinuz-2.4.30-2933 ro
       root = UUID=4df6e0cd-1156-444e-bdfd-9a9392fc345f
initrd /boot/initrd-2.4.30-2933.img
```

Cada configuración incluye un título, que aparecerá en el arranque; el root, o partición del disco desde donde arrancar; el directorio donde se encuentra el fichero correspondiente al núcleo y el fichero `initrd` correspondiente.

En el caso de que dispongamos de LiLo* como gestor en la Fedora/Red Hat, el sistema también lo actualiza (fichero `/etc/lilo.conf`), pero luego habrá que reescribir el arranque con el comando `/sbin/lilo -v` manualmente.

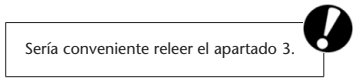
***Por defecto se usa Grub.**

Cabe señalar, asimismo, que con la instalación anterior teníamos posibilidades de descargar los paquetes fuente del núcleo; éstos, una vez instalados, están en `/usr/src/linux-version`, y pueden configurarse y compilarse por el procedimiento habitual, como si fuese un núcleo genérico. Hay que mencionar que la empresa Red Hat lleva a cabo un gran trabajo de parches y correcciones para el núcleo (usado después en Fedora), y que sus núcleos son modificaciones al estándar genérico con bastantes añadidos, por lo cual puede ser mejor utilizar los fuentes propios de Red Hat, a no ser que queramos un núcleo más nuevo o experimental que el que nos proporcionan.

8.3. Configuración de un núcleo genérico

Vamos a ver el caso general de instalación de un núcleo a partir de sus fuentes. Supongamos que tenemos unas fuentes ya instaladas en `/usr/src` (o el prefijo correspondiente).

Normalmente, tendremos un directorio `linux, linux-version` o sencillamente el número versión; este será el árbol de los paquetes fuente del núcleo. Estos pueden provenir de la misma distribución (o puede ser que los hayamos bajado en una actualización previa) y en primer lugar será interesante comprobar si son los últimas disponibles, como ya hemos hecho antes con Fedora o Debian. Por otro lado, si queremos tener las últimas y genéricas versiones, podemos ir a *kernel.org* y bajar la última versión disponible (mejor la estable que las experimentales, a no ser que estemos interesados en el desarrollo del núcleo). Descargamos el archivo y descomprimimos en `/usr/src` (u otro elegido, quizá mejor) los paquetes fuente del núcleo. También podríamos buscar si existen parches para el núcleo y aplicarlos (según hemos visto en el apartado 4).



A continuación comentaremos los pasos que habrá que realizar. El procedimiento que se indica en esta parte del taller es genérico, pero puede dar algún problema dependiendo de la distribución usada. Se recomienda seguir en lo posible el subapartado 3.3 donde se comenta el caso de configuración de un núcleo *vanilla* de la rama 2.6.xx o bien los comentarios en el caso Debian, en el subapartado 3.4, o la referencia [Fedk] para el caso Fedora.

Con las consideraciones comentadas podemos seguir también el proceso siguiente:

1) Limpiar el directorio de pruebas anteriores (si es el caso):

```
make mrproper
```

2) Configurar el núcleo con, por ejemplo, `make menuconfig` (o `xconfig`, figura 8, `gconfig` o `oldconfig`). Lo vimos en el subapartado 3.3.

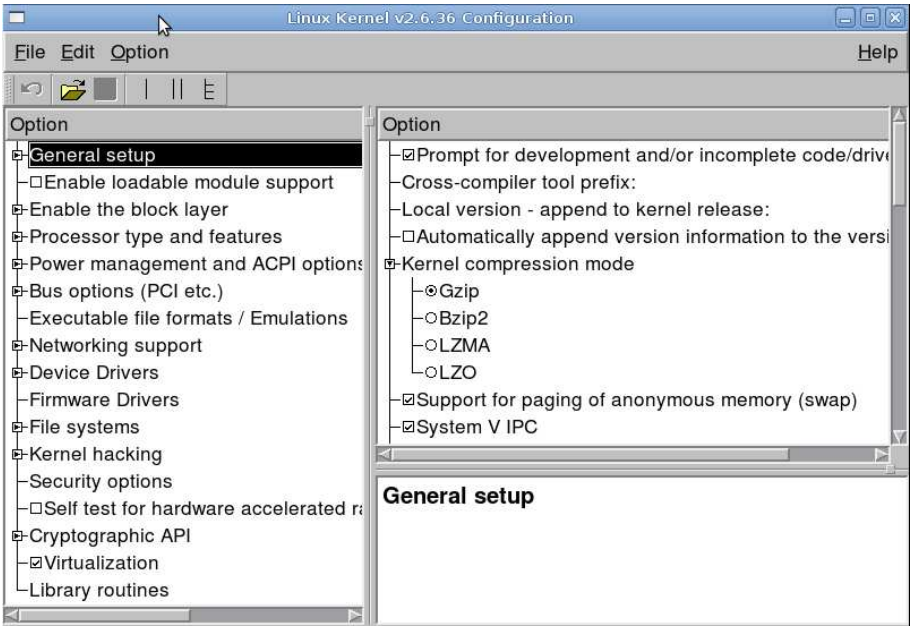
3) Dependencias y limpieza de compilaciones anteriores:

```
make dep
```

4) Compilación y creación de la imagen del núcleo: `make bzImage`. También sería posible `zImage` si la imagen fuera más pequeña, pero es más normal `bzImage`, que optimiza el proceso de carga y la compresión para núcleos más grandes. En algún hardware muy antiguo puede no funcionar, a causa del tamaño final del núcleo, y ser necesario usar `zImage`. El proceso puede durar desde algunas decenas de minutos a una hora en hardware moderno o varias horas en hardware muy antiguo. Cuando finaliza, la imagen se halla en:

`/usr/src/directorio-fuentes/arch/i386/boot` (en el caso de arquitectura Intel de 32 bits).

Figura 8. Configuración (xconfig) del núcleo por menús gráficos



5) Ahora compilamos los módulos con `make modules`. Hasta este momento no hemos modificado nada en nuestro sistema. Ahora tendremos que proceder a la instalación.

6) En el caso de los módulos, si probamos alguna versión antigua del núcleo (rama 2.2 o primeras de la 2.4), hay que tener cuidado, ya que en alguna se sobrescribían los antiguos (en las últimas versiones 2.4.x o la rama actual 2.6.x ya no es así).

Pero hay también que tener cuidado si estamos compilando una versión que es la misma (exacta numeración) que la que tenemos (los módulos se sobrescribirán); en este caso es mejor realizar una copia de seguridad de los módulos:

```
cd /lib/modules
tar -cvzf old_modules.tgz versionkernel-antigua/
```

Así, tenemos una versión en `.tgz` que podríamos recuperar después en caso de problemas. Y, finalmente, instalamos los módulos con:

```
make modules install
```

7) Ahora podemos pasar a la instalación del núcleo, por ejemplo con:

```
# cd /usr/src/directorio-Fuentes/arch/i386/boot
# cp bzImage /boot/vmlinuz-versionkernel
# cp System.map /boot/System.map-versionkernel
# ln -s /boot/vmlinuz-versionkernel /boot/vmlinuz
# ln -s /boot/System.map-versionkernel /boot/System.map
```

Así colocamos el fichero de símbolos del núcleo (`System.map`) y la imagen del núcleo. Cabe recordar que puede ser necesaria también una imagen de `initrd`.

8) Ya solo nos queda poner la configuración necesaria en el fichero de configuración del gestor de arranque, ya sea LiLo (`/etc/lilo.conf`) o Grub (`/boot/grub/menu.lst`) según las configuraciones que ya vimos con Fedora o Debian. Y recordad, en el caso de LiLo, que habrá que volver a actualizar la configuración con `/sbin/lilo` o `/sbin/lilo -v`.

9) Reiniciar la máquina y observar los resultados (si todo ha ido bien).

Resumen

En este módulo hemos examinado diferentes características del núcleo (en inglés, *kernel*) de Linux en su rama principal 2.6. Asimismo, hemos comentado algunos de sus procesos de actualización, configuración y sintonización para el sistema, útiles tanto para la optimización de su uso de memoria como para las prestaciones en una arquitectura dada, así como la adaptación al hardware de destino.

También hemos examinado las posibilidades que ofrecen los módulos dinámicos como mecanismo de extensión del núcleo y ampliación del hardware soportado.

La inclusión en el núcleo de técnicas de virtualización nos permite, a partir del núcleo y ciertas utilidades, construir un entorno de virtualización potente y flexible para generar diferentes combinaciones de máquinas virtuales que residen en un sistema *host* GNU/Linux.

Actividades

1. Determinad la versión actual del núcleo Linux incorporada en vuestra distribución. Comprobad las actualizaciones disponibles de forma automática, ya sea en Debian (apt) o en Fedora/Red Hat (vía yum o una herramienta gráfica de actualización).

2. Efectuad una actualización automática de vuestra distribución. Comprobad posibles dependencias con otros módulos utilizados y con el *bootloader* (LiLo o Grub) utilizado. Dependiendo del sistema puede ser recomendable una copia de seguridad de los datos importantes del sistema (cuentas de usuarios y ficheros de configuración modificados), o bien realizar el proceso en otro sistema del que se disponga para pruebas.

3. Para vuestra rama del núcleo, determinad la última versión disponible (consultad la página web http://www.kernel.org) y realizad una instalación manual con los pasos examinados en el módulo. La instalación final puede dejarse como opcional, o bien poner una entrada dentro del *bootloader* para las pruebas del nuevo núcleo. Observad también, dependiendo del sistema, la posibilidad de realizar una copia de seguridad previa, en especial de las configuraciones estables de los *bootloaders*.

4. En el caso de la distribución Debian, además de los pasos manuales, existe, como vimos, una forma especial (recomendada) de instalar el núcleo a partir de sus fuentes mediante el paquete `kernel-package`. Proceded con los pasos necesarios para crear una versión personalizada de un núcleo *vanilla*.

5. Elaborad una máquina virtual basada en KVM, que tenga como *guest* otra distribución GNU/Linux diferente a la del sistema *host*, a partir de su instalación vía cdrom o vía imagen ISO de la distribución.

Bibliografía

[Ar05] **Corbet, J.; Rubini, A.; Kroah-Hartman, G.** (2005). *Linux Device Drivers* (3.ª ed.). O'Reilly.

[Arc] **Arcomano, R.**. *Kernel Analysis-HOWTO*. The Linux Documentation Project.

[Bac86] **Bach, M. J.** (1986). *The Design of the UNIX Operating System*. Prentice Hall.

[Ces06] **Cesati, M.; Bovet, D.** (2006). *Understanding the Linux Kernel* (3.ª ed.). O'Reilly.

[Debk] **Debian Kernel Handbook Project**. *Debian Linux Kernel Handbook*. <http://kernel-handbook.alioth.debian.org>

[Fedk] **Fedora Project**. *Building a custom kernel*. <http://fedoraproject.org/wiki/Building_a_custom_kernel>

[Gor] **Gortmaker, P.** (2003). *The Linux BootPrompt HOWTO*. The Linux Documentation Project.

[Gru] **GNU**. *Grub bootloader*. <http://www.gnu.org/software/grub/>

[Grub] **GNU**. *Grub Manual*. <http://www.gnu.org/software/grub/manual/>

[Hen] **Henderson, B.** *Linux Loadable Kernel Module HOWTO*. The Linux Documentation Project.

[Kan] **Kanis, I.**. *Multiboot with GRUB Mini-HOWTO*. The Linux Documentation Project.

[Ker] **Rusty Russell**. *Unreliable Guide To Hacking The Linux Kernel*. <http://www.kernel.org/doc/htmldocs/kernel-hacking.html>

[Kera] **Kernelnewbies.org**. *Kernel Newbies*. <http://www.kernelnewbies.org>

[Kerb] **Kernel.org**. *Linux Kernel Archives*. <http://www.kernel.org>

[Lkm] **Lkm**. *Linux Kernel Mailing List*. <http://www.tux.org/lkml>

[Mur] **Murphy, G. L.** *Kernel Book Project*.
<http://kernelbook.sourceforge.net>

[OSDa] **OSDL**. *Open Source Developement Laboratories*. (Ahora *The Linux Foundation*)
<http://www.linuxfoundation.org>

[Pra03] **Pranevich, J.** (2003). *The Wonderful World of Linux 2.6*.
<http://www.kniggit.net/wwol26.html>

[Pro] **GNU Project**. *Grub Manual*.
<http://www.gnu.org/software/grub/manual/>

[Skoa] **Skoric, M.** *LILO mini-HOWTO*. The Linux Documentation Project.

[Tan87] **Tanenbaum, A.** (1987). *Sistemas operativos: Diseño e Implementación*. Prentice Hall.

[Tum] **Tumenbayar, E.** (2002). *Linux SMP HOWTO*. The Linux Documentation Project.

[Vah96] **Vahalia, U.** (1996). *UNIX Internals: The New Frontiers*. Prentice Hall.

[Vasb] **Vasudevan, A.** *The Linux Kernel HOWTO*. The Linux Documentation Project.

[Zan] **Zanelli, R.**. *Win95 + WinNT + Linux multiboot using LILOmini-HOWTO*. The Linux Documentation Project.

Otras fuentes de referencia e información

[Kerb] Sitio que proporciona un repositorio de las diversas versiones del núcleo Linux y sus parches.

[Kera] [lkm] Sitios web que recogen una parte de la comunidad del núcleo de Linux. Dispone de varios recursos de documentación y listas de correo de la evolución del núcleo, su estabilidad y las nuevas prestaciones que se desarrollan.

[Debk] Es un manual imprescindible sobre los procesos de compilación del núcleo en la distribución Debian. Suele actualizarse con los cambios producidos en la distribución. [Fedk] aporta una referencia similar para el caso Fedora.

[Ces06] Libro sobre el núcleo de Linux 2.4, que detalla los diferentes componentes y su implementación y diseño. Existe una primera edición sobre el núcleo 2.2 y una nueva actualización al núcleo 2.6.

[Pra03] Artículo que describe algunas de las principales novedades de la rama 2.6 del núcleo Linux.

[Ker] [Mur] Proyectos de documentación del núcleo, incompletos pero con material útil.

[Bac86] [Vah96] [Tan87] Algunos textos sobre los conceptos, diseño e implementación de los núcleos de diferentes versiones UNIX.

[Skoa][Zan01][Kan][Pro] Recursos para tener más información sobre los cargadores LiLo y Grub.

[Gru][Grub] Sitios oficiales de Grub2 y el anterior original Grub (ahora conocido como Grub Legacy).