El núcleo Linux

Josep Jorba Esteve

PID_00238612

Universitat Oberta de Catalunya

Índice

Introducción

El núcleo (en inglés *kernel*) del sistema operativo GNU/Linux (al que habitualmente denominamos Linux) [Vasb], es la parte central del sistema: se encarga de ponerlo en funcionamiento y, una vez este es ya utilizable por las aplicaciones y los usuarios, se encarga de gestionar los recursos de la máquina, controlando la gestión de la memoria, los sistemas de ficheros, las operaciones de entrada/salida, los procesos y su intercomunicación.

Su origen se remonta al año 1991, cuando en agosto, un estudiante finlandés llamado **Linus Torvalds** anunció en un *newsgroup* de la época, 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 ofreció para el uso 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 de código fuente, podemos recompilar el núcleo, para adaptarlo mejor a nuestro sistema, y como veremos posteriormente, configurarlo para dar un mejor rendimiento al sistema.

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

Además, examinaremos 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 se nos ofrecen con soporte desde el propio núcleo.

Objetivos

En este módulo se muestran los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Conocer el funcionamiento del kernel y de los procesos de configuración asociados.
2. Poder configurar y crear 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, para sintonizar sus sistemas.

1. El núcleo del sistema GNU/Linux

El núcleo o *kernel* es la parte básica de cualquier sistema operativo [Tan87, Tan06], y sobre él descansa el código de los servicios fundamentales para controlar el sistema completo. Básicamente, su estructura puede separarse típicamente en una serie de componentes, o módulos de gestión orientados a:

- **Gestión de procesos**: qué tareas se van a ejecutar, 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?

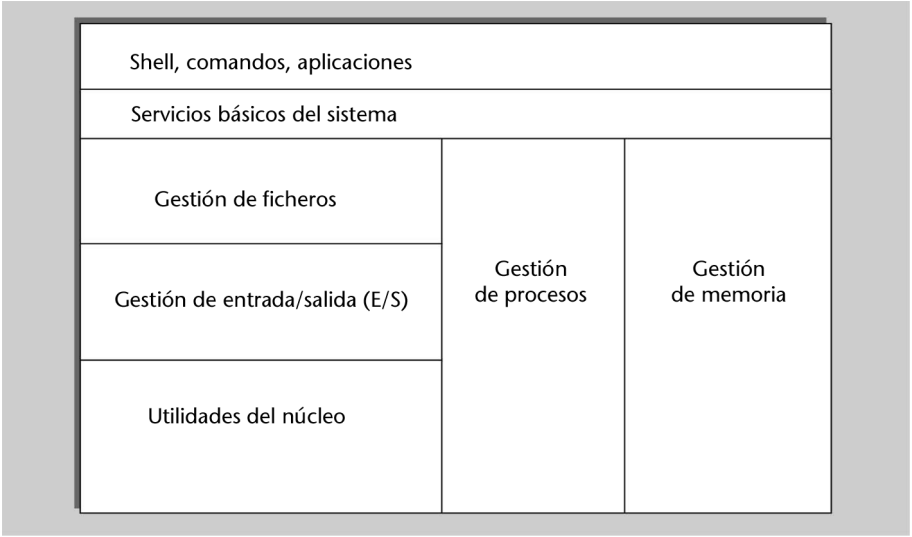
- **Intercomunicación de procesos y sincronización**: ¿cómo se comunican tareas entre sí, con qué diferentes mecanismos y cómo pueden sincronizarse grupos de tareas?

- **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* o *updates*).

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 con necesidades críticas controlar mejor sus aplicaciones y el propio sistema, y poder elaborar sistemas con el propio sistema operativo "a la carta", personalizado al gusto de cada usuario final. 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 (para dispositivos móviles u otra electrónica de consumo).

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, Tan06] (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

MINIX

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

soportaba 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.

En julio de 2011, Linus Torvalds anunció la versión Linux 3.0, no por sus grandes cambios tecnológicos, sino por iniciar una serie de cambios posteriores y para conmemorar el 20 aniversario de Linux. Algunos hitos más en la rama 3.x:

- 3.3 (marzo de 2012): se realiza una mezcla de los fuentes de Linux y se incorpora Android, que básicamente es un kernel Linux con ciertas modificaciones; tras varios intentos en esta versión, se realiza la unión de ambos. Se introduce también la capacidad de poder arrancar directamente a traves de las nuevas EFI, sustitución de las BIOS.

- 3.6 (septiembre de 2012): mejoras en el soporte del sistema de ficheros Btrfs (cuotas, *snapshots*), pensado para sustituir en el futuro a ext4.

- 3.7 (diciembre de 2012): se incorpora el soporte para diversas arquitecturas ARM.

- 3.8 (febrero de 2013): como curiosidad se elimina el soporte a procesadores 386, y se realizan importantes mejoras en varios sistemas de ficheros.

- 3.13 (enero de 2014): soporte de NFtables, la siguiente generación de reglas de firewall (sustituyendo a iptables).

- 3.15 (junio de 2014): se mejora el soporte de EFI, la escritura en sistemas de ficheros FUSE, y el soporte para repertorios vectoriales de nuevas CPUs Intel.

En abril de 2015 se anunció la versión 4.0, inaugurando una nueva rama donde se realizaron algunas mejoras de la infraestructura de aplicación de parches en el kernel. La idea era aplicar algunos cambios que permitiesen mejorar el

desarrollo de futuras versiones del kernel. Las siguientes son algunas de las nuevas prestaciones más destacadas:

- **4.0 (abril 2015).** Integración preliminar de tecnologías basadas en kGraft (de SUSE) y kpatch (procedente de RedHat) que permite aplicar parches al kernel sin reiniciar. Se añade soporte para nuevas CPUs de Intel (las Skylake).

- **4.1 (junio 2015).** Soporte de encriptación en ext4.

- **4.2 (agosto 2015).** Drivers gráficos básicos de AMD incluidos en el kernel, y driver virtual para GPU para la mejora, soporte, y prestaciones en virtualización para la GPU.

- **4.5 (marzo 2016).** Varias mejoras en sistemas de ficheros Btrfs. Y estabilización de la jerarquía cgroups (los grupos de control son utilizados para asignar, o limitar, recursos a grupos de procesos).

- **4.7 (julio 2016).** Se añade soporte para nuevas GPUs AMD y soporte para dispositivos virtuales USB compartidos en red. Se habilita una caché paralela de directorios (para búsquedas más eficientes sin/minimizando el acceso a disco). Y módulo de seguridad (LoadPin) para restringir el origen de los módulos del kernel (evitar intrusiones de módulos no válidos).

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 (rama de desarrollo).
- 2.6 Greg Kroah-Hartman (versiones estables, entre otros) / Linus Torvalds, Andrew Morton (*releases* de desarrollo).
- 3.x Greg Kroah-Hartman, Sasha Levin / Linus Torvalds (*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 iniciales y en el tamaño respectivo del código fuente (no es un indicador absoluto de la com-

plejidad, pero da indicios de su evolución), confrontando con una versión de las ultimas ramas del kernel. 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	1.800
2.4	01-2001	3.378
2.6	12-2003	5.930
3.10	06-2013	15.803
4.1	06-2015	19.500

Como podemos comprobar, se ha evolucionado de unas diez mil líneas a seis millones en las primeras versiones de la rama 2.6; las últimas versiones de la rama 4.x tienen ya más de diecinueve millones de líneas.

En estos momentos el desarrollo continúa en la rama 4.x del núcleo, la última versión estable, que incluye la mayoría de distribuciones como versión principal (algunas todavía incluyen algunas 2.6.x, pero 3.x o 4.x suele ser la opción por defecto en la instalación).

Aunque ahora la rama principal sea la 4.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 mantuvieron 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 aquel momento 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`. Después 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 catorce mil programadores en las últimas revisiones del kernel) 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, con diferentes prestaciones, para un mismo componente.

Respecto a la numeración de las versiones del núcleo de Linux [Ker, Ces06, Lov10], 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 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.

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.

Los distribuidores no suelen incluir la última versión del núcleo, sino la que ellos han probado con más frecuencia y pueden verificar que es estable para el software y componentes que 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.

3) En la rama 3.x, cuando después del 2.6.39 Linus decidió renombrar a 3.x, el segundo número se ha usado como número de revisión por secuencia temporal a medida que salían los nuevos kernels, y puede añadirse un tercer número para designar correcciones de fallos respecto a *bugs* o seguridad. Se espera que, de manera semejante, en un determinado momento se progrese de la misma manera en las ramas 4.x, 5.x. etc. En cuanto a las versiones diferentes del kernel, ahora se suele denominar *mainline* a la versión de desarrollo que suele mantener Linus, y que aparece periódicamente cada 2 o 3 meses. Cuando aparece liberada la versión pasa al estado de *Stable*, en la cual se realizan varias iteraciones (indicadas por el tercer número) por *bugs*, normalmente de 2 a 3 por mes, aunque hay solo unas pocas revisiones del estable, porque mientras tanto el *mainline* siguiente se vuelve estable. Y como caso excepcional, hay algunos kernels denominados *longterm*, para propósitos de mantenimiento de larga duración, en los que se incluye el soporte de parches que se hagan en otras versiones, para portarlos a estas. Pueden ser, por ejemplo, kernels que hayan sido escogidos por distribuciones de tipo LTS (*Long Term Support*) o interesantes por contener alguna prestación especial. Hay que diferenciar que

los kernels de distribución suelen ser mantenidos por las propias distribuciones, normalmente como paquetes en sus repositorios, y pueden incluir niveles de parche adicionales propios de la distribución. Estos kernels especiales de las distribuciones pueden detectarse con el comando `uname -r`, que nos da la versión del kernel actual, por los números adicionales a los predefinidos en la numeración del kernel, incluidos en la salida del comando.

4) Para obtener el último núcleo publicado (que normalmente se denomina *stable* o *vanilla*), hay que acudir al archivo de núcleos Linux (disponible en https://www.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: https://www.kernel.org

Algunas de las características técnicas [Ces06, Kan][Lov10] 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. Normal-

mente, 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 en las ramas 2.6 y 3. Del rendimiento SMP y multicore 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), ZFS (Oracle), Btrfs, 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 y en procesadores o arquitecturas obsoletas).

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 (formato antiguo de) 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 32 o 64 bits o bien virtualizar máquinas completas, pudiendo disponer de imágenes virtuales de distribuciones GNU/Linux ya preparadas para actuar de sistemas invitados en gestores de virtualización.

8) Linux dispone de un completísimo 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. Una deficiencia importante para la adopción de Linux en ciertos ámbitos 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, AMD ATI) e impresoras (Epson, HP) comienzan ya a proporcionar los controladores para sus dispositivos, bien sean de código abierto, o binarios utilizables 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 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 [Cor05], 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 3.x/4.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 por el contrario, una vieja) 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 características del 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. 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 manualmente 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.

Ved también

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, para el sistema operativo, 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 embargo, ante sistemas en producción, siempre es importante tomar las medidas de precaución y hacer las copias de seguridad necesarias. O bien trabajar con un sistema equivalente, en estado de test o preproducción, donde podamos previamente probar los efectos de un nuevo kernel.

Examinemos el proceso necesario para instalar y configurar un núcleo Linux. En los subapartados siguientes, trataremos:

1) El proceso de compilación para las ramas 2.6.x y 3.x/4.x.

2) Detalles específicos de las versiones 3.x/4.x.

3) 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 2.6.x, en este módulo mantenemos 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 mantenerlo en las antiguas debido a incompatibilidades o a la existencia de hardware antiguo no soportado por las distribuciones actuales.

Los conceptos generales del proceso de compilación y configuración se explicarán en el primer subapartado (2.6.x), ya que la mayoría de ellos son genéricos, y observaremos posteriormente las diferencias respecto de las nuevas versiones.

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 compilación, 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 a una nueva *release* de esta.

- 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, por ejemplo 6 meses) 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 compilación de las diferentes ramas de desarrollo del núcleo Linux.

3.1. Compilación de las ramas 2.6.x y 3.x/4.x del kernel Linux

Las instrucciones son específicas para la arquitectura x86/x86_64(o amd64) de Intel, mediante usuario no-root la mayor parte del proceso (puede hacerse como usuario normal y, de hecho, es altamente aconsejable por seguridad), y únicamente el proceso final de instalación es necesario root para integrar el kernel final en el sistema. La compilación del kernel es un proceso costoso en tiempo y disco para los kernels actuales, se recomienda disponer de espacio de disco abundante (100 GB, o más, para kernels actuales), y realizar el proceso con el máximo de CPU y memoria disponible para acelerar el proceso. Las diferentes fases que están involucradas son:

1) Obtener el núcleo. Podemos acudir a http://www.kernel.org (o a su servidor ftp) y descargar la versión *stable* o *longterm* que queramos probar. Por otro lado, en la mayoría de las distribuciones de GNU/Linux, como Fedora/Red Hat o Debian, también ofrecen como paquete el código fuente del núcleo (normalmente con algunas modificaciones incluidas); si se trata de la versión del núcleo que necesitamos, quizá sea preferible usar estas (mediante los paquetes `kernel-source`, `kernel-version`, `linux-source`, o similares). Si queremos los últimos núcleos, quizá 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-2.6.32.63.tar.bz2 | tar xvf -
```

Si los paquetes fuente venían en un fichero `gz` (o `tar.gz`), reemplazamos bzip2 por gzip, o en los recientes (usados por kernel.org) `.tar.xz` o `.xz` reemplazamos por `unxz` (paquete xz-utils):

```
xz -cd linux-4.X.tar.xz | tar xvf -
```

Al descomprimir los paquetes fuente se habrá generado un directorio llamado `linux-version_kernel`, donde entraremos para establecer la configuración del núcleo. Es recomendable disponer de abundante espacio para la compilación del kernel. Dependiendo de la versión pueden ser necesarios de 20 a 100 GB de espacio, dependiendo de las opciones escogidas. Así mismo se necesita disponer de espacio suficiente para la instalación posterior del kernel creado en la partición raíz `/`. En este casos son recomendables de 10 a 20 GB libres adicionales.

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.

Ved también

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 proporcionadas 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.

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 *module-init-tools*, que ofrecen 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).

Por ejemplo, en una distribución Debian estable (en otras distribuciones u versiones, consultar los paquetes similares), entre otros son necesarios los paquetes: *build-essential libncurses5-dev libqt4-dev libqt4-qt3support qt4-qmake libgtk2.0-dev libglib2.0-dev libglade2-dev modules-init-tools gcc g++*, en el caso de Debian, necesitaremos una instalación previa de estos paquetes, por ejemplo, realizada con la lista anterior, y un *apt-get install* de los paquetes mencionados.

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`, que cita los requisitos mínimos de versiones de los programas prerrequisitos de la compilación, 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:

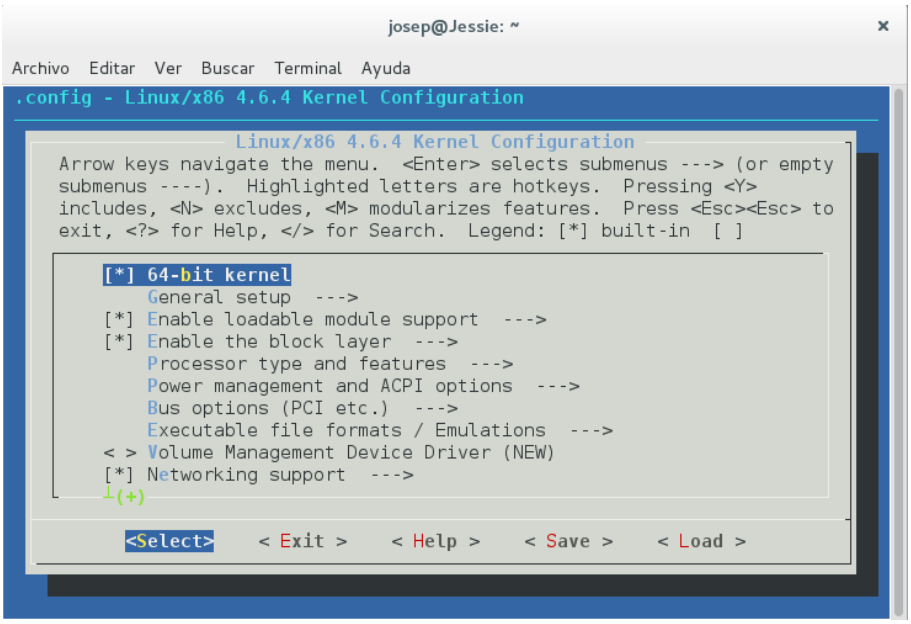
- `make config`: desde la línea de comandos se nos pregunta por cada opción y se nos pide confirmación (y/n), si deseamos o no la opción, o se nos piden los valores necesarios, o seleccionar una de las posibilidades de la opción. Es la configuración larga, en la que se nos piden muchas respuestas; podemos tener que responder a varios centenares 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 previas que hayamos compilado con anterioridad); 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. Pero en las últimas versiones del núcleo, esta opción detecta la configuración anterior, y solamente nos pregunta por las opciones nuevas disponibles en la nueva versión, ahorrando un tiempo considerable en las elecciones. Por otra parte, en general la configuración de un kernel disponible en la distribución está también guardada en el fichero `/boot/config-version` correspondiente (recordar que con `uname -r` tenemos la versión del kernel actual), el cual podemos copiar directamente como fichero `.config` en la raíz de los fuentes del kernel, y con `make oldconfig` configurar únicamente las nuevas opciones.

- `make menuconfig`: configuración basada en menús textuales, bastante cómoda (figura 2); podemos habilitar o inhabilitar lo que queramos, y es más rápida y versátil para la navegación entre opciones que el `make config`.

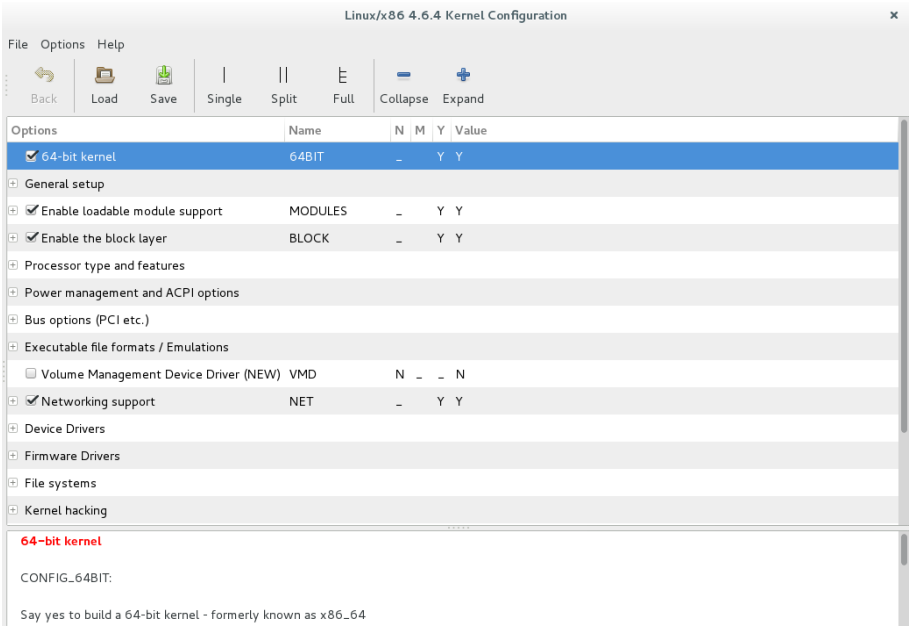
- `make xconfig`: la más cómoda, basada en diálogos gráficos en X Window. Es necesario tener instaladas las bibliotecas de desarrollo de qt3 o qt4, ya que esta configuración tiene estos requerimientos al estar basada en la librería Qt3/Qt4 (KDE). La configuración se basa en cuadros de diálogo, botones y casillas de activación. Es bastante rápida y dispone de ayuda con comentarios de muchas de las opciones.

Figura 2. Configuración del núcleo 4.x desde la interfaz textual de menuconfig



- `make gconfig`: similar al caso de xconfig, pero con la interfaz basada en gtk (Gnome) (figura 3). Como prerrequisito necesita los paquetes de desarrollo de las librerías Gnome (entre otros *libgtk2.0-dev*, *libglib2.0-dev* y *libglade2-dev*, los nombres son dependientes de la distribución y versión).

Figura 3. Configuración basada en Gnome de un kernel 4.x



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 se está planeando hacerla posteriormente 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. Esta es una decisión más o menos importante, ya que el tamaño y el rendimiento del núcleo (y, por tanto, del sistema entero) en algunos casos puede depender de nuestra elección.

Ved también

La gestión de módulos se trata en el apartado 5.

El núcleo de Linux, como imagen binaria una vez compilado, ha comenzado a ser de gran tamaño, tanto por complejidad como por los controladores de dispositivo [Cor05] que incluye. Si integrásemos todas sus posibilidades, 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 pequeñas porciones, que se cargarán dinámicamente bajo demanda cuando, por carga explícita o por uso de sus características, sean necesarias.

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 sí, o está funcionando uno o el otro (esto pasaba antiguamente, por ejemplo, con impresoras con cable paralelo y algunos dispositivos hardware que se conectaban al puerto paralelo). Por lo tanto, en este caso tenemos que colocar como módulos los controladores y cargar o descargar el que sea necesario en cada momento. Aunque también se han solucionado actualmente estos problemas, con soportes de daemons que controlan casos de *(*hotplug) (conexión en caliente) de dispositivos, como por ejemplo *udev*, que aporta soluciones en este sentido, gestionando dinámicamente las entradas del directorio `/dev` a medida que los dispositivos se conectan o desconectan en el sistema.

- 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 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, 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 (también en estos casos es posible utilizar otras alternativas, como FUSE, para dar soporte a nivel usuario, en lugar de en el kernel).

vfat

vfat (FAT32) es el formato propio de antiguas versiones de Windows y bastante usado en dispositivos de almacenamiento USB.

4) Compilación del núcleo. Una vez disponible una configuración de la versión del núcleo, mediante `make` comenzaremos el proceso de compilación:

```
$ make
```

Atajos Debian

Algunas distribuciones permiten modos más simplificados, por ejemplo Debian, permite mediante *make deb-pkg* realizar todo el proceso de compilación y preparar la instalación mediante la creación de los paquetes `.deb` para la instalación del kernel. Finalmente solo quedará instalar con *dpkg -i* los paquetes resultantes.

Este proceso puede acelerarse si disponemos de una máquina multiprocesador o multicore; `make` dispone de la opción -jn, con la que podemos especificar, con n, el número de procesadores o cores que emplearemos para la compilación.

Cuando este proceso acabe, tendremos la parte integral del núcleo y nos faltarán las partes que se pidieron colocar como módulos (en las últimas ramas de 3.x/4.x esta opción no es necesaria, porque se generan ya con `make` directamente, aunque por otra parte sí que habrá que instalar los módulos):

```
$ make modules
```

Hasta este momento hemos hecho la configuración y compilación del núcleo. Esta parte podía hacerse desde un usuario normal (altamente recomendable) o bien el root, pero ahora necesitaremos forzosamente usuario root, porque pasaremos a la parte de la instalación del nuevo kernel compilado en el sistema.

5) Instalación. Comenzamos instalando los módulos:

```
# make modules_install
```

esto nos instalará los módulos construidos en el sistema de fichero para que el nuevo kernel los pueda encontrar en el lugar adecuado. Normalmente los módulos están disponibles en el directorio `/lib/modules/version_kernel`, donde `version_kernel` será la numeración del kernel que acabamos de construir.

Y para la instalación del nuevo núcleo (desde el directorio de compilación de las fuentes, siendo `version` la versión usada), simplemente con:

```
# make install
```

Para este último paso la mayoría de distribuciones incluyen un soporte externo con un script denominado *installkernel* (normalmente proporcionado por el paquete *mkinitrd*), que es usado por el sistema de compilación del kernel para colocarlo en el lugar adecuado, modificar el *bootloader* (lilo o Grub), de manera que no se deban realizar pasos extras. Hay que señalar que algunas distribuciones de tipo *from scratch*, como *Gentoo*, no incluyen el script, por tanto consultar la documentación específica de kernel para estas distribuciones.

En el proceso de esta última fase *install*, se desarrollan las siguientes fases:

- Se verifica que el kernel haya estado bien construido.

- Se instala la porción estática del kernel en el directorio `/boot`, y se nombra el fichero con la versión del kernel que se ha construido (típicamente `/boot/vmlinuz-version`).

- Las imágenes de ramdisk que puedan ser necesarias son creadas usando aquellos módulos que se hayan creado previamente y sean necesarios para tales imágenes en arranque de la màquina. Típicamente esta imagen aparecerá como `/boot/initrd.img-version`. Respecto a la creación de *initrd*, hace falta disponer de las herramientas adecuadas; en Debian, por ejemplo, en el proceso de compilación es necesario como prerrequisito el paquete *initramfs-tools* (para kernels superiores a 2.6.12, antiguamente cumplían esta función las *mkinitrd-tools*, ya obsoletas). También durante estas primeras fases se suelen generar los ficheros `/boot/System.map-version` y `/boot/config-version`, que contienen, respectivamente, información sobre los símbolos disponibles en el kernel y la configuración empleada en el kernel en su compilación.

- El *bootloader* (lilo/Grub) correspondiente del sistema es notificado de la existencia del nuevo kernel, creándose la entrada necesaria del menú, de manera que el usuario pueda seleccionarlo en el siguiente arranque de la máquina.

Finalmente, después de estas fases, el kernel está instalado correctamente y podemos proceder al reinicio, y probar nuestra imagen nueva del kernel. Si el proceso se ha realizado correctamente, dispondremos de una entrada de *bootloader* correspondiente, y en caso de que surgieran problemas con la nueva imagen, dispondríamos de las entradas de los kernels antiguos para volver a la situación estable anterior.

Como procesos alternativos, ya sea para complementar la instalación, o bien para realizarla si aparecen problemas desde el `make install`, cabe tener en cuenta lo siguiente. En la instalación dependiendo del kernel, y de su uso posterior, puede también ser necesaria la instalación de los *headers* de desarrollo del kernel, y de los *firmwares* asociados (los últimos kernels han incluido

algunos *firmwares* de dispositivos directamente en el kernel), estos últimos procesos los desarrollaremos con:

```
# make headers_install
# make firmware_install
```

finalizando de esta manera el proceso de instalación del kernel completo. En este último caso los *firmwares* son instalados en `/lib/firmware`, y los *headers* necesarios en el directorio mismo de las fuentes, en el subdirectorio `usr/include/linux`.

Este último proceso, de instalación automática a través de *make install* (y añadidos), también puede realizarse de forma manual, si no se dispone del script antes comentado (*installkernel*), con el siguiente proceso (algunos detalles dependen de la arquitectura):

```
# make modules_install
# make kernelversion
```

(El último comando obtiene un número KERNEL_VERSION de la versión construida, en lo siguiente colocar el número obtenido donde aparezca la expresión KERNEL_VERSION.)

En algunas últimas ramas 3.X/4.X hay que sustituir el directorio i386 de la siguiente orden por x86, donde encontraremos la versión binaria comprimida del kernel compilado.

```
# cp arch/i386/boot/bzImage /boot/bzImage-KERNEL_VERSION
# cp System.map /boot/System.map-KERNEL_VERSION
```

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`, un fichero que contiene los símbolos disponibles en el núcleo, necesario para el proceso de arranque del núcleo, también se coloca en el mismo directorio `/boot`.

También puede ser necesaria la creación de la imagen de disco ramdisk `initrd` con las utilidades necesarias según la distribución (o su versión).

Respecto a la creación del `initrd`, en Fedora/Red Hat este se creará automáticamente con la opción `make install`. En Debian deberemos usar las técnicas conocidas como *Debian way* o bien crearlo explícitamente con `mkinitrd`

(versiones de núcleo $\leq$ 2.6.12) o, posteriormente con kernels actuales usar, bien `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'
```

aunque cabe señalar que en las distribuciones actuales, y versiones del kernel, es habitual que el initramfs se cree solo en el proceso de compilación y posterior instalación (en el *make install* igualmente). Aún así se recomienda la remodelación de este (vía previa configuración en */etc/initramfs-tools/initramfs.conf*), debido a que suele presentar un tamaño respetable, que puede ser ineficiente en algunas máquinas por incluir más módulos de los imprescindibles, para lo que puede jugarse (en el susodicho initramfs.conf) con la configuración *MODULES=most* o *dep* (que seguramente minimizará el tamaño de forma significativa). Después podemos recrear el initrd mediante el comando anterior, o por ejemplo substituyendo los parámetros por:

```
#update-initramfs -t -u -k version
```

que en este caso nos actualizará el initrd si el proceso de instalación ya nos creó uno previo.

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

Este paso solo debe realizarse si la instalación final es manual; de usarse `make install` las últimas ramas del kernel ya permiten la inclusión del nuevo kernel en el *bootloader*.

- 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:

  Actualmente es casi omnipresente Grub 2 como sistema de arranque. Aquí mencionamos LiLo por razones históricas.

  ```
  image = /boot/bzImage-KERNEL_VERSION
      label = KERNEL_VERSION
  ```

  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 recuperarlo 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 estará el sistema de archivos principal (el /). Recordar que los discos tienen dispositivos como `/dev/sda` (primer disco) `/dev/sdb` (segundo), y

la partición se indicaría como "`root=/dev/sda2`" si el / de nuestro Linux estuviese en la segunda partición del primer disco. 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
```

- 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. Hay que señalar que la configuración de Grub (normalmente disponible en `/boot/grub/menu.lst` o `grub.cfg`) depende mucho de la versión, sea Grub-Legacy (Grub 1.x) o Grub 2, se recomienda examinar la configuración de kernels ya presentes, para adaptar el recién creado, y consultar la documentación GNU, y el manual disponible vía *info grub*.

Lectura recomendada

Sobre Grub podéis consultar *Grub Manual* accesible desde la web del proyecto GNU: https://www.gnu.org/software/grub/

https://www.gnu.org/software/grub/manual/.

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` (u logs equivalentes dependiendo de la distribución) 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.

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

3.2. 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`, `bzip2` y `unxz`.

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 diferentes de los de la versión *vanilla* o *pristine* obtenida como estable 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.xx o 3.xx, de manera que el núcleo suele quedarse (generalmente) en esta numeración para toda la vida de la versión concreta de Debian estable, y así, cuando se actualiza con revisiones menores la distribución, solo se actualizan parches 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 apartado 4). Esta versión del paquete de los fuentes del núcleo es la que usaremos para crear un núcleo personalizado, 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 reconstrucción del kernel actual Debian, podemos proceder de dos maneras: bien obtenemos el kernel directamente de los fuentes (con todos los parches incluidos) o bien reconstruimos los paquetes Debian oficiales del kernel.

En el primer caso, obtenemos las fuentes directas, y para la compilación del núcleo actual procedemos usando el paquete disponible como *linux-source-version*; en este caso de ejemplo version=3.2 (versión principal del kernel para una Debian estable concreta) como versión del kernel Debian de la distribución, pero dependiendo en cada momento del kernel actual, tendremos que

escoger los fuentes que se correspondan con la versión principal de nuestro kernel actual (ver *uname -r*):

```
# apt-get install linux-source-3.16
$ tar -Jxf /usr/src/linux-source-3.16.tar.xz
```

Nota

Para el segundo comando hemos supuesto un formato `xz`. En las últimas ramas se utiliza `xz`; en anteriores podía ser `bzip2`, con lo cual la opción del `tar` pasaria de J a j para el formato bzip2).

En este caso las descargará normalmente en `/usr/src` en forma comprimida, aunque podemos copiarlas a otro espacio o disco de trabajo, ya que el proceso necesita un espacio importante (cerca de 10 GB o más), si no disponemos de él en `/usr/src`. Una vez tengamos el fuente, con tar, descomprimirá los paquetes fuente dejándolos en un árbol de directorios a partir del directorio `linux-version`, donde versión ahora será la revisión del kernel disponible (3.16.xx). Para la configuración y compilación posterior a partir de este directorio, podemos realizarla por el método clásico (visto en el anterior subapartado) o por la *Debian Way* que examinamos en este apartado.

Para el segundo caso (la reconstrucción de los paquetes Debian oficiales), el objetivo es obtener unos paquetes *deb* finales equivalentes a los que ofrece la distribución, pero con la personalización que queramos realizar.

Para este proceso, comenzaremos por obtener algunas herramientas que necesitaremos:

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

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. Después de esta configuración de herramientas iniciales, podemos pasar a la descarga de los fuentes (puede hacerse desde cualquier usuario, no es necesario root para el proceso):

```
$ apt-get source linux
```

que nos descargará los fuentes y los descomprimirá en el directorio actual, además de otros paquetes que incluyen parches respecto a la versión original del kernel. El directorio necesario donde comenzar el proceso lo encontraremos como `linux-version`. En este momento es cuando podremos personalizar la configuración del kernel por los métodos examinados en la sección previa, ya que la configuración es la que trae el kernel Debian por defecto, con lo

que podremos cambiar aquellos componentes u opciones del kernel que nos interesen antes de volver a reconstruir los paquetes.

Para realizar la construcción de los paquetes kernel para la arquitectura (se recomiendan como mínimo 20 GB disponibles de disco), según la configuración preestablecida del paquete (semejante a la incluida en los paquetes oficiales del núcleo `linux-image` en Debian):

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

Con esto dispondremos de los paquetes binarios (y otros de soporte independientes de la arquitectura) del núcleo (ficheros `*.deb`) disponibles para la instalación (vía `dpkg -i`).

Existen algunos procedimientos extra para la creación del 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 a propósito del *Debian way*, al inicio de este subapartado, 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 (a la que daremos una numeración de revisión propia).

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` o `oldconfig`). 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*). En este caso sea 3.x la versión de kernel disponible de fuentes en el repositorio, o también comenzando en el segundo paso, el kernel estable obtenido de *kernel.org* (este último quizás esté en formato *xz* en lugar de *bzip2*):

```
# apt-get install linux-source-3.16
$ tar -xvJf /usr/src/linux-source-3.16.tar.xz
$ cd linux-source-3.16
```

de donde obtenemos los paquetes fuente y los descomprimimos (la instalación del paquete deja el archivo de fuentes en `/usr/src`; en el caso de que proceda de kernel.org ya depende de donde hayamos realizado la descarga inicial). Es recomendable que antes de proceder al paso de descompresión con

tar, lo realicemos en un espacio de disco con capacidad suficiente (recomendados 10-20 GB).

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, con *make oldconfig*, también puede usarse cualquiera de los otros métodos, `xconfig`, `gconfig`, etc.):

```
$ make menuconfig
```

El kernel preparado de esta manera, a no ser que indiquemos lo contrario, tiene la depuración activada, lo que resulta muy útil para la depuración en caso de errores (utilizando herramientas como *kdump, crash* o *SystemTap*), aunque, por otra parte, nos aumenta el espacio necesario del kernel para su construcción; cuando esto no sea necesario, antes de pasar a la construcción podemos deshabilitar la opción de incluir información de depuración:

```
$ scripts/config --disable DEBUG_INFO
```

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 de los paquetes asociados a la imagen del núcleo, que podremos finalmente instalar (examinar de los pasos anteriores el nombre del paquete obtenido, que sera el que incluiremos aquí):

```
# dpkg -i ../linux-image-3.xx.yy_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).

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

Añadimos también otra peculiaridad a tener en cuenta en Debian, que puede ser útil con algún hardware: la existencia de utilidades para añadir módulos dinámicos de núcleo proporcionados por terceros; en particular, la utilidad `module-assistant`, que permite automatizar todo este proceso a partir de los paquetes fuente del módulo.

Enlace de interés

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

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 (también pueden obtenerse durante la instalación del núcleo mediante un `make headers_install`, o mediante el paquete deb obtenido con el *Debian way*). 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.

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

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

Así se prepara el sistema para posibles dependencias, descarga fuentes del módulo, compila y, si no hay problemas, instala para el presente núcleo. En la lista interactiva de `module-assistant` podemos observar el nombre de los módulos disponibles.

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*) de funcionalidad 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, o xz con extensión .xz), como por ejemplo podría ser:

```
patchxxxx-version-pversion.xz
```

donde xxxx suele ser algún mensaje sobre el tipo o finalidad del parche, versión 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-version, o directorio que usemos).
unxz patch-xxxxx-version-pversion.xz
patch -p1 < patch-xxxxx-version-pversion
```

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, Debian) 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, siempre que la compatibilidad de la distribución lo permita.

Por último, cabe comentar la incorporación de una tecnología comercial al uso de parches en Linux, Ksplice (mantenida por Oracle), 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 encuen-

Ksplice

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

tra 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 (de Oracle para su distribución Oracle Linux); otros productos semejantes están apareciendo en varias distribuciones empresariales comerciales. 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.

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.

SUSE también anuncio su idea de integrar una tecnología semejante, denominada *kGraft*, como mecanismo para aplicar parches al kernel sin reiniciar el sistema, de forma semejante a *Ksplice*. Pero en este caso la idea de los desarrolladores es intentar incluirla en la línea principal del kernel, con lo que estaría disponible como parte del código del kernel. En esta tecnología, de la que ya existen algunas versiones que se estan integrando en las versiones de desarrollo del kernel, básicamente el parche se integra como un módulo del kernel (.ko) el cual es cargado con *insmod*, y reemplaza a las funciones kernel afectadas por el parche, incluso durante el tiempo de ejecución de estas funciones. La idea de kGraft es permitir arreglar *bugs* críticos de kernel sin afectar a los sistemas que necesitan gran estabilidad y disponibilidad.

kGraft

Tecnología de SUSE para parches en ejecución del kernel:
https://www.suse.com/promo/kgraft.html

También existe la tecnología kpatch de Red Hat, que junto con kGraft de SUSE, se ha comenzado a introducir en las ramas 4.x del kernel, con lo que se espera que en futuras iteraciones de la rama 4.x se disponga de esta funcionalidad.

5. Módulos del núcleo

Flexibilidad del sistema

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

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`, aunque en las recientes distribuciones se ha migrado a una estructura de subdirectorios donde se incluye un fichero de opciones por módulo, que puede verse en el subdirectorio `/etc/modprobe.d`

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.

Existen 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`):

- `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` (que, mediante un disco RAM en memoria principal, supone una precarga en tiempo de arranque del sistema) o por medio de `udev` (en situaciones dinámicas de uso de dispositivos removibles, o de conexión en caliente).

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 comúnmente 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` (o un posterior *update-initramfs*) permite crearlo a partir de sus opciones genéricas, 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 o grub.cfg`, modificando la línea de `initrd` oportuna. En cualquier caso, siempre es interesante establecer en el *bootloader* una configuración alternativa 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 alterna-

tiva posible. En este caso se utilizan técnicas de listas negras de módulos (típicamente la lista negra se guarda en `/etc/modprobe.d/blacklist.conf`), aunque puede crearse en un fichero arbitrario en ese directorio, simplemente añadiendo al fichero creado una línea *blacklist nombremodulo*.

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, o si estaba configurado en arranque, una vez producido éste el nuevo módulo se utilizará.

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 asociadas a la versión del núcleo actual para la que se quiere instalar el módulo; normalmente es suficiente con los headers, pero dependiendo del módulo pueden ser necesarios más paquetes de fuentes).

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 dónde se instalará 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.

Por último mediante:

```
# dkms status
```

podemos conocer que módulos están activos en este momento y en qué kernels de los disponibles en el sistema.

6. Virtualización en el núcleo

Una de las áreas en expansión, en la administración de IT, es la virtualización de sistemas y su integración con entornos Cloud de tipo público/privado o híbrido, como veremos más adelante. Con el tiempo, GNU/Linux ha ido 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, para observar diferencias entre ellas o personalizar nuestras tareas o scripts para cada distribución.

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

- LXC
- OpenVZ
- VirtualBox
- VMware

VMware es uno de los líderes comerciales en soluciones de virtualización, y dispone de productos de virtualización para escritorio (VMware Workstation/Fusion), mientras que para servidor dispone de un producto VMware vSphere Hypervisor (ESXi) que está disponible para Linux como descarga gratuita. El caso de servidor permite gestionar varias máquinas virtuales con un interfaz de gestión simple. Otras líneas de productos VMware implementan necesidades mayores para centros de datos (*data centers*) y entornos de Cloud Computing, 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

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).

LXC (LinuX Containers) es un sistema de virtualización a nivel de operativo que permite correr múltiples sistemas Linux de forma aislada. En lugar de crear máquinas virtuales completas, se basa en el uso de una funcionalidad del kernel de Linux, denominada cgroups, juntamente con *chroot*, que permite encapsular un espacio de ejecución de procesos y gestión de red de forma aislada. La idea es similar a OpenVZ, pero en este caso no necesita actuaciones de módulos o parches de kernel, ya que funciona bajo el kernel *vanilla*, sin modificaciones. Algunos sistemas para desplegamiento de contenedores, como *Docker*, lo utilizaron como base para implementar los contenedores (aunque ahora dependen de otras soluciones como *libcontainer*).

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, o svm en procesadores AMD, 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 imple-

Enlace de interés

Enlace web de KVM en:
http://www.linux-kvm.org

menta 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.

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

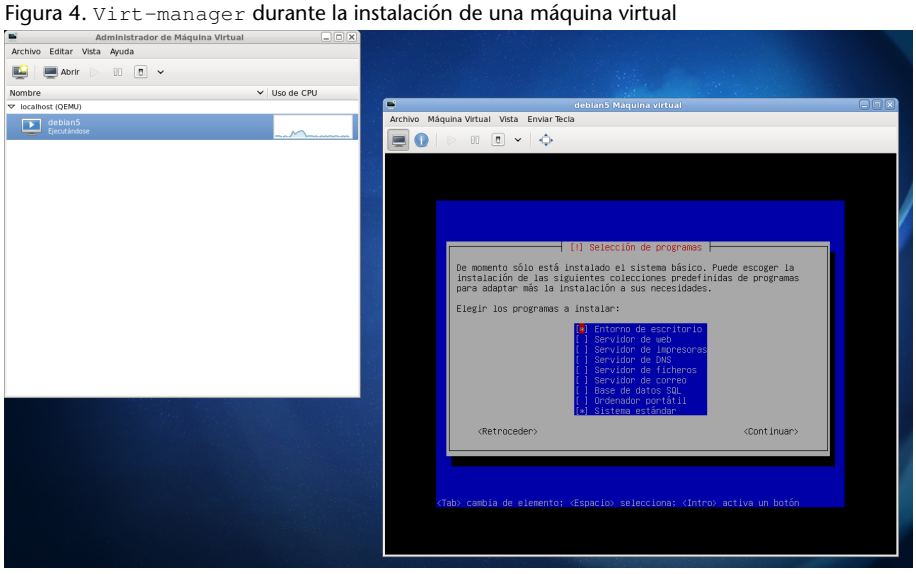
- El módulo de núcleo `kvm.ko`, que proporciona la infraestructura base de virtualización, y además un módulo adicional según el modelo de procesador `kvm-intel.ko` o `kvm-amd.ko`. Estos módulos deberán cargarse manualmente (`modprobe`) o habilitarlos durante el arranque para disponer de soporte de virtualización 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. Normalmente estas utilidades suelen necesitar un servicio de sistema, denominado `libvirtd` (en algunas distribuciones `libvirt-bin`). Tenemos que asegurarnos de poner en marcha el servicio (`/etc/init.d/libvirtd start`, `/etc/init.d/libvirt-bin start` o equivalentes con Systemd) 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.

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.

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
```

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-bin`, `virt-viewer` entre otros. 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, ya que la librería libvirt nos permite abstraer varios sistemas diferentes de virtualización.

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):

```
# adduser <usuario> libvirt
```

En este momento podemos ejecutar el comando de shell para la gestión de máquinas KVM, `virsh`), que nos permitiría ver las disponibles:

```
$ virsh --connect qemu:///system list
```

esto permite conectarse al gestor local, y preguntarle por la lista de máquinas disponibles.

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:

```
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
```

Enlace de interés

Por otra parte, un tema importante es la gestión de red, que es uno de los temas complejos en virtualización; en el caso de KVM se recomienda examinar: http://www.linux-kvm.org/page/Networking.

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 o de archivos de imágenes ISO de la distribución Debian).

Creamos el espacio de disco (en el directorio actual) 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 CD/DVD con:

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

Hay que vigilar que el dispositivo, `/dev/cdrom` en este caso, sea el correcto, ya que según la distribución puede cambiar.

que nos genera así la imagen del CD/DVD o por contra directamente descargar la imagen ISO de la distribución.

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` (en algunas distribuciones, el comando kvm ya no se utiliza o solo existe por compatibilidad):

```
$ qemu-system-x86_64 --enable-kvm -m 512 -cdrom debian-install.iso \
-boot d -hda debianVM.img
```

En este caso crearía una máquina virtual (arquitectura x86_64) 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` o `/dev/sr0` 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 debian -r 512 \
--vcpus=2 -f debianVM.img -s 8 -c debianinstall.iso --vnc \
--noautoconsole --os-type linux --os-variant debianwheezy
```

que entre otros parámetros, coloca el método de conexión a la máquina virtual, el nombre de la VM `debian`, 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 el comando anterior de `virt-install` se ha utilizado la configuración de red virtual por defecto, la denominada *default*. Si aparece algún error por no estar correctamente inicializada, podemos realizarlo con:

```
virsh net-start default
```

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. Con `virt-manager` podemos añadir la conexión a la máquina de la que hemos creado la imagen, por ejemplo al *localhost*, lo que nos permitirá después tenerla accesible mediante conexión a *qemu:///system*, y también arrancarla al momento.

Después podemos visualizar la consola (de texto o gráfica) de la máquina recién creada mediante:

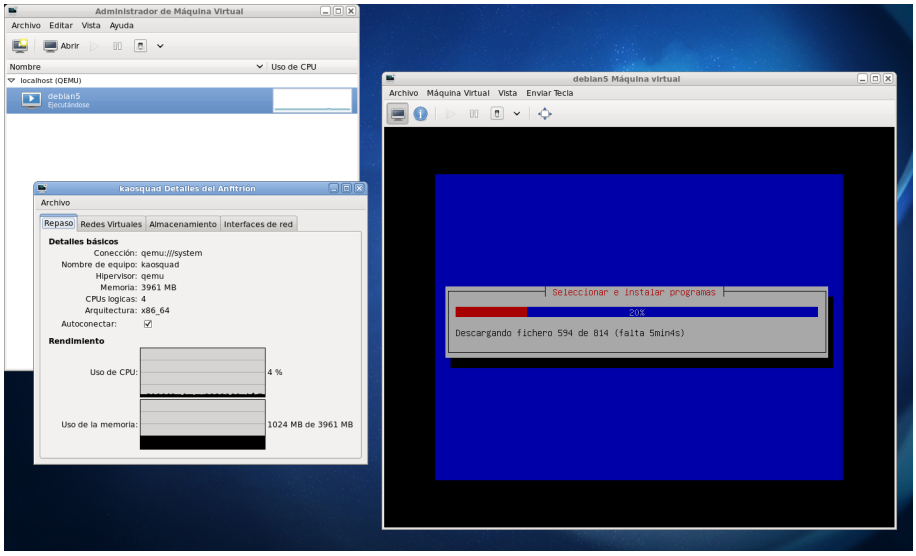
```
virt-viewer -c qemu:///system nombreVM
```

si está en el mismo sistema *localhost*, 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



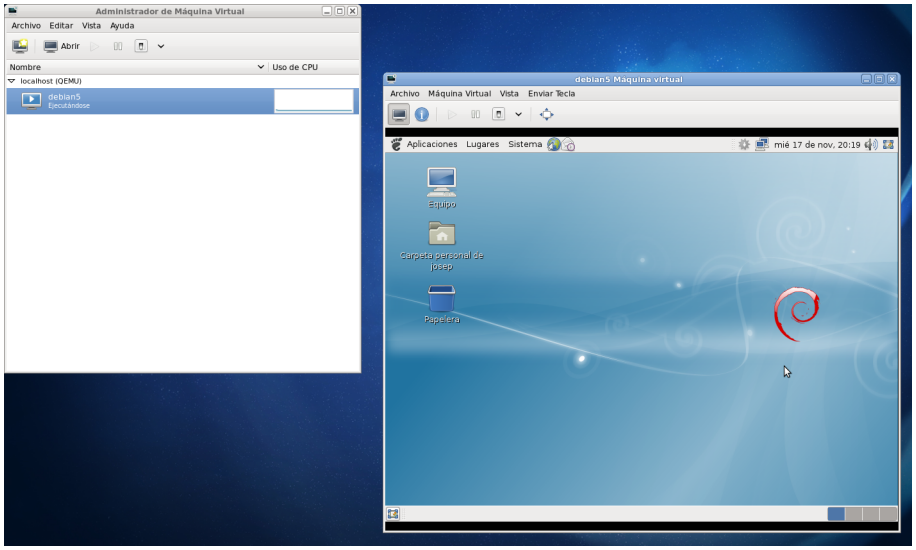
Esto nos permite conectar gráficamente con la máquina virtual creada y, por ejemplo, continuar con la instalación del sistema operativo (se habrá iniciado con el arranque anterior con `virt-install` o `qemu-system-x86_64`). Una vez instalado el sistema, ya podemos usar cualquier sistema, ya sea gráfico (`virt-manager`) o de línea de comandos (`virtsh`) para gestionar las máquinas *guest* virtuales y poder arrancar y pararlas.

Por ejemplo, con `virsh`:

```
virsh --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



6.2. VirtualBox

VirtualBox es una solución de virtualización de escritorio que está obteniendo resultados importantes, un proyecto en estos momentos mantenido por Oracle.

En el caso de Debian, que usaremos, está disponible en el repositorio oficial. Pero para otras distribuciones, pueden encontrarse en https://www.virtualbox.org/wiki/Linux_Downloads paquetes preparados para gran numero de distribuciones, así como repositorios propios que añadir a las distribuciones.

Esta solución de virtualización incluye tanto herramientas de tipo gráfico como utilidades a nivel de línea de comandos para virtualizar máquinas de 32bits y 64bits de arquitectura Intel (x86 y x86_64). VirtualBox se ofrece mayoritariamente con licencia GPL; aún así existe una parte adicional propietaria, denominada *Extension Pack*, que ofrece libre de coste Oracle para uso personal. Este paquete adicional ofrece algunas facilidades extra relacionadas con emulación de Hardware extra (USB2 por ejemplo, por defecto solo se ofrece USB1 sin pack), y ofrece acceso gráfico a las máquinas huésped a través de RDP (*Remote Desktop Protocol*).

Para la instalación en Debian se debe instalar el paquete *virtualbox* y es recomendado disponer de los headers correspondientes al kernel actual (paquete *linux-headers-version* siendo la versión la correspondiente al kernel actual). Los headers se van a utilizar para la compilación de una serie de módulos que virtualbox instala dinámicamente en el kernel. Normalmente, la instalación proporciona también una configuración de *dkms* para estos módulos que permite que se recompilen después de cambios de kernel en la distribución. En algunas distribuciones a veces pasa un tiempo hasta que están disponibles los módulos actualizados para una versión concreta de kernel; por tanto, antes de actualizar el kernel en un sistema con virtualización virtualbox, es recomendable comprobar que la versión de kernel ya tiene el soporte adecuado.

```
# apt-get install linux-headers-version virtualbox
```

VirtualBox puede arrancarse ya a través del comando `virtualbox`. Los módulos, si teníamos los headers adecuados, se habrán compilado y cargado en el sistema, y se cargarán en cada arranque. Si no deseamos que se arranquen automáticamente, podemos cambiar en `/etc/default/virtualbox` el parámetro LOAD_VBOXDRV_MODULE y ponerlo a 0.

Una vez arrancado el entorno gráfico con `virtualbox`, podemos crear una nueva VM mediante la opción del icono "Nueva", que nos arrancará el asistente de creación de máquinas virtuales. El proceso pasa por determinar:

- El operativo de la máquina huésped y su nombre.

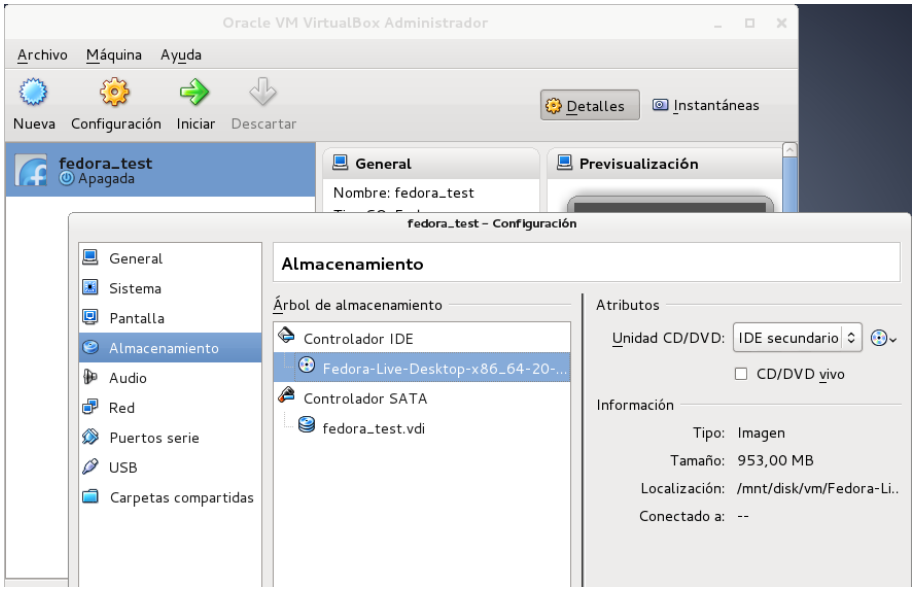
- La memoria de nuestro sistema, que proporcionaremos a la máquina huésped; generalmente se recomienda que el total de máquinas VM no consuman más del 50 % de la memoria. Dependiendo del uso, es frecuente entre 512 MB a 2 GB por máquina virtual.

- Creación del disco duro virtual de la máquina. Podemos crear un disco nuevo o usar una imagen de disco previamente creada para la nueva máquina.

- Formato de la imagen del disco. En virtualbox, por defecto, es VDI. Pero podemos crear otros, que quizá podamos compartir con otros sistemas de virtualización.

- Asignamos el tamaño del disco virtual de forma dinámica o reservamos el total del espacio. Esto nos permitirá reservar espacio de disco a medida que se vaya usando o, por contra, tener un disco más rápido, lo que nos permitirá mejores prestaciones del disco virtual.

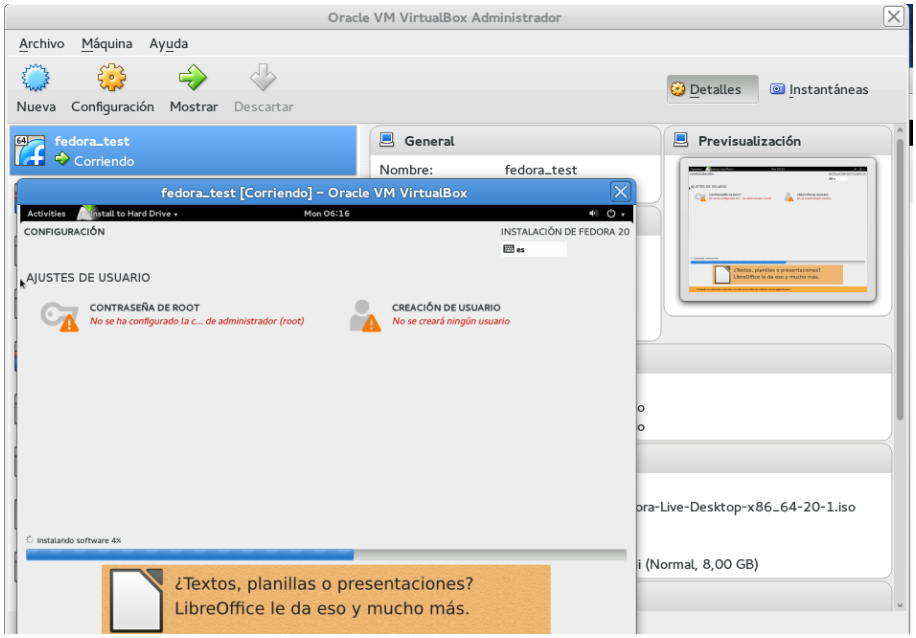
Figura 7. Máquina virtual creada en el gestor de VirtualBox añadiendo ISO para instalar.



Con este proceso ya disponemos de una máquina creada y podemos retocar su configuración seleccionándola; en especial, podemos conectar en almacenamiento una imagen ISO de una distribución a su CDROM virtual. Esto nos permitirá arrancar la máquina desde CD, arrancando así la imagen de instalación. En el proceso inicial también podemos cambiar la configuración de red; hay diversos modos disponibles: desde el NAT, por defecto, a modos que permiten redes privadas para las máquinas VM, o incluir una IP pública para nuestra máquina VM (ver documentación VirtualBox para la administración de red http://www.virtualbox.org/manual/UserManual.html). Véase en la figura 7 una máquina creada, con su sección de configuración de almacena-

miento a la que se le acaba de conectar una imagen ISO (de una distribución Fedora), si procedemos al arranque de la máquina con *Iniciar*. En la figura 8 siguiente podemos ver la máquina huésped durante el proceso de instalación.

Figura 8. Máquina virtual en el proceso de instalación de una distribución GNU/Linux.



Una vez tengamos la VM con sistema operativo instalado, ya la tendremos funcional. Para mejorar el rendimiento de las máquinas huésped, VirtualBox también proporciona una serie de controladores para el sistema huésped, conocidos como *VirtualBox Guest Additions*, que permiten mejoras de la gestión de la red virtual, de la aceleración de los gráficos y la posibilidad de compartir carpetas con la máquina host (anfitrión). Una vez arrancada e instalada la VM, desde el menú Dispositivos ->Insertar imagen CD Guest Additions, pueden instalarse. Cabe señalar que para realizar este proceso, que necesita instalar en el sistema algunos *drivers* adicionales, es necesario disponer del paquete de los *headers* (`linux-headers` o `kernel-devel` según la distribución). Además es recomendable tener instalado el paquete `dkms`, que nos permitirá *a posteriori* que estos *drivers* se recompilen bajo cambios de kernel del sistema.

Cabe señalar que además de todo este proceso, determinados proveedores proporcionan imágenes VDI de máquinas VirtualBox ya preparadas para su funcionamiento, para que no tengamos que pasar por todo el proceso de creación. Normalmente son imágenes denominadas Cloud, y deberemos buscar imágenes VDI ya preparadas u otros formatos compatibles con VirtualBox. Sin embargo, se puede incurrir en ciertos riesgos de seguridad. Conviene evitar el uso de VM ya customizadas si el proveedor no es de confianza o no es directamente la misma distribuidora; además, algunas máquinas VM arbitrarias podrían contener problemas de seguridad o introducir malware o herramientas destinadas al ataque de nuestros sistemas. Por eso se recomienda usar solo imágenes de proveedores adecuados.

Por ultimo, cabe señalar el tema del control mediante interfaz de usuario. VirtualBox proporciona los comandos `vboxmanage` y `VBoxHeadless` para permitirnos la mayor parte de la gestión de las máquinas desde línea de comandos, e incluso si ya disponemos de imágenes VDI, podremos realizar toda la gestión sin ningún uso de interfaz gráfica.

Un resumen de algunos comandos interesantes:

- Arrancar la máquina en modo servidor sin interfaz VirtualBox; al arrancar nos dará el puerto de conexión al servidor gráfico (VRDE). También con vboxmanage, con interfaz GUI y sin interfaz:

```
$ VBoxHeadless -startvm "nombre_maquina"
$ vboxmanage startvm "nombre_maquina"
$ vboxmanage startvm "nombre_maquina" --type headless
```

- Información de la máquina virtual, los detalles de su configuración, discos, red, memoria, cpu y otros:

```
$ vboxmanage showvminfo "nombre_maquina"
```

- Apagar la máquina:

```
$ vboxmanage controlvm "nombre_maquina" poweroff
```

- Modificar el servicio de pantalla:

```
$ vboxmanage modifyvm "nombre_maquina" --vrde on --vrdeport
puerto_VRDE --vrdeaddress ip_maquina
```

- Conexión gráfica a la máquina ejecutándose. Si la máquina está en NAT sin ip asignada, la ip=0.0.0.0 para la conexión, si no la ip correspondiente. Para pleno funcionamiento se necesitan instaladas las Guest Additions, las cuales tienen como requisito el paquete de headers del kernel correspondiente en la máquina huésped:

```
$ rdesktop -a 16 -N ip_maquina:puerto_VRDE
```

- En particular, con el siguiente veremos múltiples opciones de control de las VMs que tenemos disponibles:

```
$ vboxmanage controlvm
```

6.3. Xen

En el caso de Xen, tratamos con un entorno de hypervisor a nivel de máquina física que permite correr múltiples instancias del sistema operativo o de diferentes operativos en paralelo en una sola máquina.

Algunas de las características de Xen que cabe destacar son las siguientes:

Enlace de interés

Se recomienda leer esta introducción: http://wiki.xen.org/wiki/Xen_Overview, para comprender algunos conceptos de virtualización asociados a Xen.

- El hypervisor, o componente monitor de máquinas virtuales, es relativamente pequeño y consume pocos recursos de la máquina.

- Xen es agnóstico del sistema operativo. La mayoría de instalaciones usan Linux como sistema base; Xen aprovecha los servicios básicos, el denominado dominio 0 de Xen. Dom0 es una VM especializada con privilegios para el acceso directo al hardware. Como base para Xen, pueden usarse otros operativos, como algunos de la familia BSD.

- Xen es capaz de aislar el funcionamiento de un driver de sistema en una máquina virtual. Si este falla o es comprometido, la VM que contiene el driver puede rearrancarse sin afectar al resto del sistema.

- Pueden distinguirse dos modos de funcionamiento, por un lado a) el denominado Xen Full Virtualization (HVM), que usando las extensiones hardware de virtualización (las HVM), utiliza emulación de PC (Xen se basa en Qemu para ello). En este caso no se requiere soporte de kernel, pero por contra suele ser una solución de menor rendimiento debido a la emulación necesaria. Por otra parte, el modo b) Xen Paravirtualization (PV), es una virtualización eficiente y ligera en uso de recursos, que no requiere extensiones HVM, pero si disponer de kernels y drivers con soporte de PV. De manera que el hypervisor puede ejecutarse eficientemente sin emulación de máquina, o emulación de componentes virtuales de hardware.

- El soporte de Paravirtualización. Los huéspedes de este tipo son optimizados para poder ejecutarse como máquina virtual (algunos estudios han proporcionado solo sobrecargas en torno al 2-8 %, mientras que en emulación las penalizaciones se suelen acercar a un 20 %), dando mejor rendimiento que otros sistemas que se basan en componentes adicionales. Además, Xen puede incluso ejecutarse en hardware que no soporta extensiones de virtualización.

  Linux dispone del soporte de paravirtualización desde la versión 2.6.23, llamado *paravirt_ops* (o simplemente *pvops*). Se trata de un kernel Linux de Dominio 0 (disponible para arquitecturas x86, x86_64 y ia64). Este dom0 es el sistema que tienen los controladores de dispositivo para comunicarse con el hardware subyacente, ejecuta las herramientas de administración de Xen y proporciona los discos virtuales y el subsistema de red virtual al resto de sistemas huésped (denominados domU's).

También hay que mencionar que en las últimas generaciones de procesadores ha mejorado sensiblemente el soporte y las posibilidades de las extensiones HVM, lo que ha hecho viable nuevas aproximaciones híbridas para pasar de virtualización PV a PVHVM; básicamente consiste en huésped HVM pero con drivers especiales para los discos y red.

Enlace de interés

Una referencia para explicar el concepto de PVHVM: http://www.slideshare.net/fullscreen/xen_com_mgr/linux-pv-on-hvm/

- Como necesitamos de base el mencionado Dominio 0, Xen necesitará portar los sistemas operativos para adaptarse a la API de Xen, a diferencia de los VM tradicionales, que proporcionan entornos basados en software para simular el hardware. En estos momentos hay ports (de los kernels pvops) para NetBSD, FreeBSD y Linux entre otros.

- Gracias al código aportado por Intel y AMD a Xen, se han adaptado diferentes extensiones de virtualización hardware que permiten que los sistemas operativos sin modificaciones se ejecuten en máquinas virtuales Xen, lo que ha aportado mejoras de rendimiento y ofrecido la posibilidad de ejecutar sistemas GNU/Linux y Windows sin modificación alguna.

- Hay soporte para la migración de máquinas VM en caliente entre equipos físicos. En este sentido, una máquina virtual en ejecución puede ser copiada y trasladada de equipo físico sin detener la ejecución, tan solo para pequeñas sincronizaciones.

A continuación veremos algunos usos simples de Xen sobre una distribución Debian, que dispone de la particularidad de que es fácil construir VM huéspedes basadas en Debian misma de forma bastante flexible a partir de los repositorios de Debian.

Comenzaremos por la instalación del dominio 0 (`dom0`). Básicamente se realiza con una instalación Debian típica, donde únicamente se tendrán en cuenta las particiones a emplear para reservar espacio posterior a los huéspedes. En las versiones actuales de Xen se suele reservar 4 GB de disco para el `dom0` (su /), y 1 GB de memoria para la RAM que necesitará. El espacio extra puede dedicarse a almacenar las VM huéspedes, o en el caso de *storage* podemos por ejemplo usar volúmenes LVM que nos permitirán crecer los *filesystems* a medida que los necesitemos por parte del `dom0` o las domU's (huéspedes).

Por ejemplo, podríamos realizar 3 particiones (sda1,2,3), y colocar `/ (root)` y `swap` en las dos primeras y un LVM en la tercera, con un volumen físico y un grupo (*vg0*, por ejemplo). Instalamos el sistema Debian.

Pasamos a instalar el hypervisor mediante un metapaquete disponible en Debian que nos hace todo el trabajo de dependencias y utilidades necesarias:

```
# apt-get install xen-linux-system
```

Podemos comprobar también si nuestra máquina soporta extensiones HVM, ya sean de Intel o AMD (en algunos casos pueden estar desactivadas en la BIOS de la máquina, y entonces deberemos habilitarlas previamente), con:

```
egrep '(vmx|svm)' /proc/cpuinfo
```

Si están disponibles, Xen podrá utilizar más eficientemente la paravirtualización, basándose en el soporte de virtualización de los procesadores modernos. En varios estudios (de *benchmarking*) se ha comprobado que PV+HVM (sobre PV pura) en diferentes *bencharmaks* obtiene beneficios que van del 15 % hasta el 200 o 300 %.

***Benchmarking* PVHVM**

Algunas referencias:

https://developer.rackspace.com/blog/welcome-to-performance-cloud-servers-have-some-benchmarks/

https://xen-orchestra.com/debian-pvhvm-vs-pv/

http://wiki.xen.org/wiki/PV_on_HVM

Una vez realizada la instalación previa del sistema Xen, observaremos que la distribución Debian, en su arranque `Grub2` (en este caso), tiene una nueva entrada denominada `Xen4`. Es la que nos permitirá arrancar la máquina, bajo el hypervisor Xen gestionando el `dom0`. La entrada Xen no está puesta por defecto, con lo cual se seguirá cargando por defecto el kernel del sistema. Con estos comandos la pondremos por defecto:

```
dpkg-divert --divert /etc/grub.d/08_linux_xen --rename /etc/grub.d/20_linux_xen
update-grub
```

El siguiente paso será configurar la red para el dominio 0. La más común es usar un *bridge* por software (necesitamos disponer del paquete bridge-utils); un ejemplo sencillo para `/etc/network/interfaces`:

```
#The loopback network interface
auto lo
iface lo inet loopback

iface eth0 inet manual

auto xenbr0
iface xenbr0 inet dhcp
   bridge_ports eth0

#other possibly useful options in a virtualized environment
  #bridge_stp off       # disable Spanning Tree Protocol
  #bridge_waitport 0    # no delay before a port becomes available
  #bridge_fd 0          # no forwarding delay

## configure a (separate) bridge for the DomUs without
## giving Dom0 an IP on it
#auto xenbr1
#iface xenbr1 inet manual
#   bridge_ports eth1
```

Otras técnicas en la configuración son opcionales, como por ejemplo: ajustar el uso de memoria para `dom0`, añadir a `/etc/default/grub` la siguiente línea para reservar memoria para el `dom0` (1 GB en este caso), y posteriormente un *update-grub*:

```
GRUB_CMDLINE_XEN="dom0_mem=1024M"
```

En Xen se utiliza una técnica denominada *ballooned* que lo que hace es asignar inicialmente la mayoría de la memoria al `dom0`, y a medida de que van apareciendo domU's se la va reduciendo. Con esto la fijamos de manera estática. Hay que comprobar que sea suficiente; si sufrimos algún crash del kernel Xen hay que aumentarla. También podemos obviar esta personalización de memoria y dejar la técnica de *balloning* por defecto. También hay que hacer el proceso equivalente en `/etc/xen/xend-config.sxp` con:

```
(dom0-min-mem 1024)
(enable-dom0-ballooning no)
```

Con estos cambios ya podremos rearrancar la máquina de nuevo con el `dom0` de Xen disponible con opciones básicas. La configuración de Xen en producción escapa a los términos de espacio de esta sección, y recomendamos consultar los manuales Xen para configurar, en especial: a) las CPUs disponibles a las máquinas Guest y `dom0`; b) comportamiento de Guests en rearranque; c) también es posible, para depuración, activar la consola por puerto serie para mensajes de depuración.

Con la configuración actual, pasaremos a detallar una instalación básica de domU (huésped) de tipo Debian mismo, ya que se nos ofrecen ciertas facilidades para el proceso automático:

```
apt-get install xen-tools
```

Así instalamos scripts de utilidades que nos provee Xen y después modificaremos `/etc/xen-tools/xen-tools.conf` para definir `dir`=/mnt/disk y `passwd`=1, con la dirección del parche donde podremos nuestras máquinas creadas (en /mnt/disk, en nuestro caso un volumen especial para guardar las máquinas huésped -*guest*-).

Con el comando siguiente, y la configuración previa, podremos construir una imagen de Xen, una VM ya preparada con la imagen de Debian correspondiente (en este caso una wheezy):

```
# xen-create-image --hostname nombre_maquina --ip ip_maquina
--vcpus 1 --pygrub --dist wheezy --genpass=0
```

Nota

Por defecto, sin la opción `--genpass=0`, en la orden `xen-create-image`, se genera un *password* de root aleatorio, que se mostraría durante la salida siguiente, aunque puede obtenerse *a posteriori* en el *log* de la máquina `/var/log/xen-tools/deb.log` en este caso.

Coloquemos en el comando anterior un nombre de VM y una ip disponible (ya sea pública o privada), y comenzará la construcción de la imagen. En este caso se utilizan los repositorios de Debian para descargar los paquetes adecuados para construir la imagen.

Observaremos que nos da las características por defecto de la máquina, y que nos comienza a generar las imágenes de las particiones de *root* y *swap* en el directorio (a partir de nuestro caso /mnt/disk), *domains/nombre_maquina* donde residirán las imágenes de la VM guest que se está creando. Finalmente nos pedirá el *password* de *root* de la máquina creada y finalizará:

```
General Information
--------------------
Hostname       :  deb
Distribution   :  wheezy
Mirror         :  http://mirror.switch.ch/ftp/mirror/debian/
Partitions     :  swap            128Mb (swap)
                  /               4Gb   (ext3)
Image type     :  sparse
Memory size    :  128Mb
Kernel path    :  /boot/vmlinuz-3.2.0-4-amd64
Initrd path    :  /boot/initrd.img-3.2.0-4-amd64

Networking Information
----------------------
IP Address 1   : 10.0.0.2 [MAC: 00:16:3E:2D:D7:A2]

Creating partition image: /mnt/disk/domains/deb/swap.img
Creating swap on /mnt/disk/domains/deb/swap.img
Creating partition image: /mnt/disk/domains/deb/disk.img
Creating ext3 filesystem on /mnt/disk/domains/deb/disk.img
Installation method: debootstrap
Running hooks
Creating Xen configuration file

Setting up root password
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
All done

Logfile produced at:
```

```
 /var/log/xen-tools/deb.log

Installation Summary
--------------------
Hostname        :  deb
Distribution    :  wheezy
IP-Address(es)  :  10.0.0.2
RSA Fingerprint :  4e:f3:0a:47:c1:15:14:71:15:28:da:e2:9f:df:42:2b
Root Password   :  N/A
```

En los siguientes ejemplos, los comandos utilizados dependen del *toolstack* o conjunto de utilidades xen que se esté utilizando. En muchos de los casos es posible cambiar los comandos `xm` existentes por `xl`, dependiendo de la versión de xen y/o distribución. `xl` es un nuevo *toolstack* para xen (que mantiene compatibilidad con `xm`).

Para ejecutar la máquina creada (*nombre_maquina=deb* en este ejemplo):

```
$ xm create /etc/xen/deb.cfg
```

Enlaces de interés

Sobre Toolstacks Xen podéis consultar la siguiente página web: http://wiki.xen.org/wiki/Choice_of_Toolstacks. Para comparar xm y xl podéis ver la siguiente página web: http://wiki.xen.org/wiki/XL_vs_Xend_Feature_Comparison

Y para borrar una imagen de VM:

```
$ xen-delete-image VMs_name
```

Para listar los dominios activos:

```
$xm list
Name            ID   Mem VCPUs      State   Time(s)
Domain-0         0  1023     1     r-----    247.9
deb              2   128     1     -b----      5.4
```

Y para conectar una consola a la máquina disponible (al dominio con ID=2 en este caso):

```
$xm console 2
```

Lo que nos permite realizar la conexión a la máquina virtual y realizar el proceso de login.

7. Presente 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 y las ramas posteriormente derivadas 3.x/4.x, cada vez pasa más tiempo entre las versiones principales 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 antigua rama del núcleo 2.6.x [Ces06, Pra03][Lov10], 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 en su código fuente, 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 (http://kernelnewbies.org/LinuxVersions).

En las versiones 3.x/4.x del kernel, básicamente se han mejorado diferentes aspectos de la rama previa 2.6.x, aportando nuevas prestaciones, que también son el futuro inmediato de futuras versiones en el Kernel, que se centrarán en:

- 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.

- Mejoras en sistemas de ficheros estándar Linux para adaptarlos a nuevas necesidades, como el caso de Btrfs y diversas mejoras introducidas en ext4, así como las mejoras introducidas y mejor soporte en el kernel para XFS y ZFS.

- Por el contrario, la mejora en núcleos más optimizados para dispositivos móviles (*smartphones*, *tablets*, etc.). Por ejemplo, se ha incorporado el nú-

cleo de Android, como plataforma, al código fuente del kernel. Y existen diversas iniciativas como Ubuntu Phone o Tizen, para proporcionar nuevas plataformas basándose en kernel Linux para entornos móviles. En especial, el soporte del kernel para arquitecturas ARM (de las más utilizadas en entornos móviles), ha traído la posibilidad de usar Linux en multitud de nuevos dispositivos.

- 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 se hicieron muchos avances en este aspecto, todavía hay un bajo rendimiento en algunas situaciones. En particular, en el uso de aplicaciones interactivas de escritorio se están estudiando diferentes alternativas, para mejorar este y otros aspectos relacionados con el escritorio y el uso del rendimiento gráfico.

- Soporte para las nuevas EFI/UEFI como sustitutos de las antiguas BIOS en entornos de PC x86/x86_64.

- Soporte para un nuevo sistema de filtrado IP NFtables, que substituirá progresivamente a los firewalls mediante iptables.

También, aunque se aparta de los sistemas Linux, la Free Software Foundation (FSF), y su proyecto GNU, siguen trabajando para 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/DVD de la instalación, el disquete (o CD) de rescate (actualmente suele utilizarse el primer CD/DVD 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, por ejemplo).

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, x86 o i386 para Intel o amd o, en particular para 64 bits, versiones amd64 para intel y amd). El código de la versión indica la versión del núcleo, la revisión de Debian del núcleo y la arquitectura. Por ejemplo, 3.16.0-4-amd64 es un núcleo para x86_64 AMD/Intel, revisión Debian 4 del núcleo 3.16.

4) Comprobar, para la versión elegida, que existan los módulos accesorios extras. 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 si los módulos están soportados. En el caso en que los módulos necesarios no estuvieran 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` (en este caso 3.16, es decir, el número principal) 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
```

(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 o Grub (en las últimas versiones encontraremos este 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 o grub.cfg`).

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 (ahora Grub2 en particular), 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 en 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 del subsistema udisks `udisksctl info -b /dev/particion`, comando `blkid`, o también con el comando `dumpe2fs /dev/particion | grep UUID` o, si la partición se encuentra montada en el sistema, consultando el archivo `/etc/fstab`.

8.2. Configuración del núcleo en Fedora/Red Hat

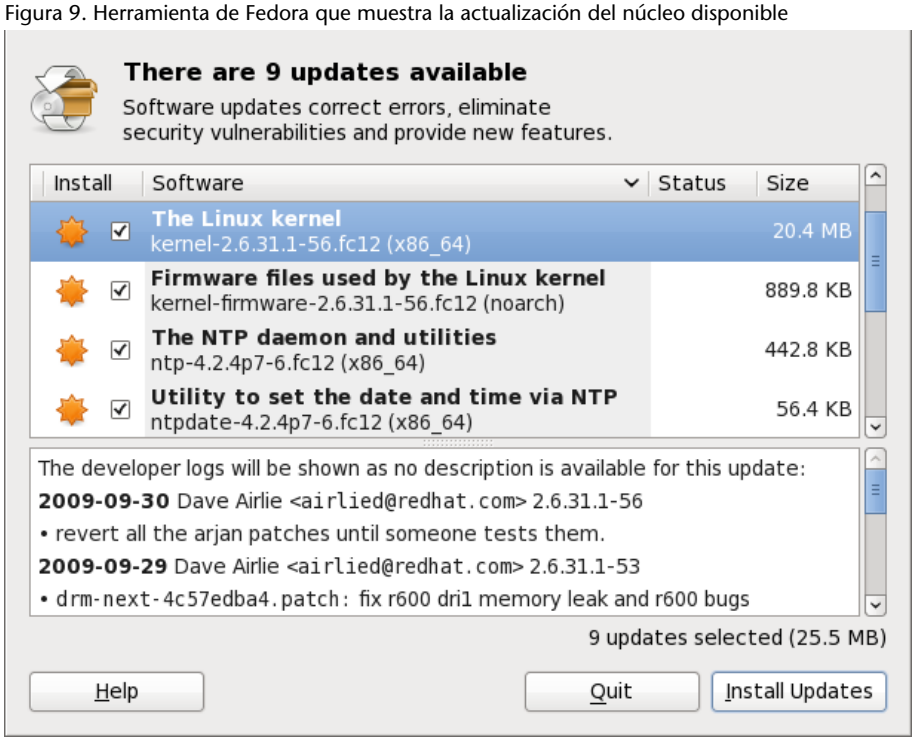
La actualización del núcleo en la distribución Fedora/Red Hat es totalmente automática por medio de su servicio de gestión de paquetes (`yum` en línea de comandos, por ejemplo), o bien mediante los programas gráficos que incluye

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 9, 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 9. Herramienta de Fedora que muestra la actualización del núcleo disponible



En Fedora podemos utilizar las herramientas gráficas equivalentes o usar directamente `yum`/`dnf`, si conocemos la disponibilidad de nuevos núcleos (obtenemos kernel y ficheros de desarrollo para compilación de módulos):

```
# dnf install kernel kernel-devel kernel-headers
```

Una vez descargada, se procederá a su instalación, normalmente también de forma automática, ya dispongamos de Grub o LiLo como gestores de arran-

que. En el caso de Grub (legacy, véanse más adelante los comentarios para el actual Grub 2), 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.cfg` 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
```

Notad que la configuración actual es Grub-Legacy (Grub <2.x); para Grub2 los campos son parecidos pero hace falta consultar las *menuentry* correspondientes a cada opción.

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 en Fedora 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 *descargado de* 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 *descargar* 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 *comentado* 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 3.1, donde se comenta el caso de configuración de un núcleo *vanilla*, o bien los comentarios en el caso Debian, en el subapartado 3.2, o la referencia [Fedk] para el caso Fedora.

Ved también

Sería conveniente releer el apartado 3.

Con las consideraciones comentadas podemos seguir también el proceso siguiente:

Recordad que hace falta disponer de espacio suficiente para la compilación. Es recomendable espacio libre en disco superior a 20 GB.

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

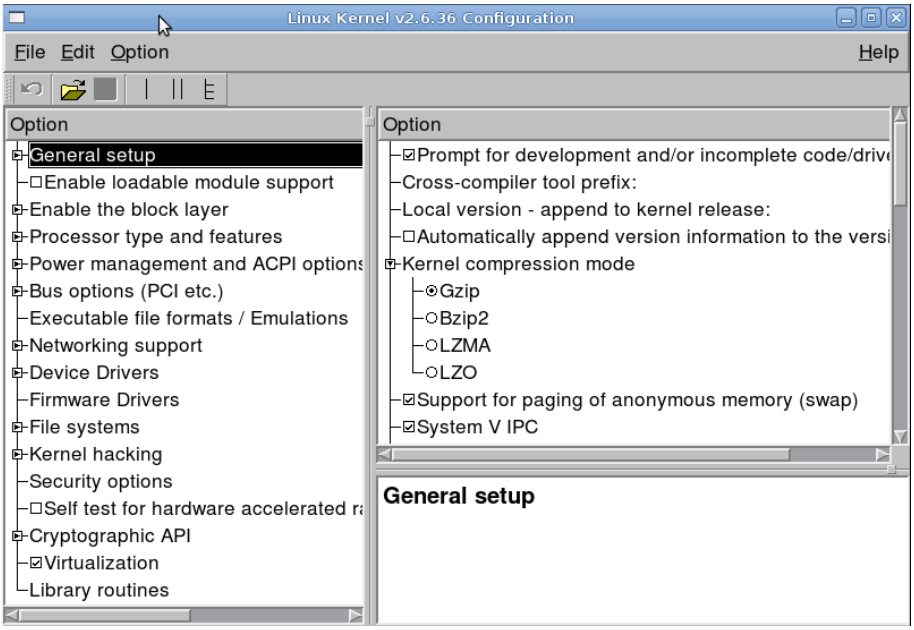
```
make mrproper
```

2) Configurar el núcleo con, por ejemplo, `make menuconfig` (o `xconfig`, figura 10, `gconfig` o `oldconfig`). Lo vimos en el subapartado 3.1.

3) Compilación y creación de la imagen del núcleo: `make`. 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 32/64 bits); dependiendo de la versión el nombre del directorio i386 cambia por x86.

4) Ahora compilamos los módulos con `make modules` (en las últimas versiones esto es automático). Hasta este momento no hemos modificado nada en nuestro sistema. Ahora tendremos que proceder a la instalación.

Figura 10. Configuración (xconfig) del núcleo por menús gráficos



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
```

5) Ahora podemos pasar a la instalación del núcleo (se ha de substituir i386 por x86 dependiendo de la versión), por ejemplo (de manera manual) 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`.

6) Ya solo nos queda poner la configuración necesaria en el fichero de configuración del gestor de arranque, ya sea LiLo (`/etc/lilo.conf`), Grub-Legacy o Grub2 (`/boot/grub/menu.lst o grub.cfg`) según las configuraciones que ya vimos con Fedora o Debian. Y recordad que en el caso de LiLo, habrá que volver a actualizar la configuración con `/sbin/lilo` o `/sbin/lilo -v`.

Cabe recordar que como vimos, todo este proceso (puntos 5 y 6) en los kernels modernos puede hacerse simplemente con un:

```
# make install
```

7) Reiniciar la máquina y observar los resultados (si todo el proceso ha sido correcto dispondremos de una nueva entrada de kernel en el gestor de arranque).

Resumen

En este módulo se han examinado diferentes características del núcleo (en inglés, *kernel*) de Linux en sus ramas 2.6.x y 3.x/4.x. Asimismo, se han comentado algunos de sus procesos de actualización, configuración y sintonización para el sistema, útiles tanto para la optimización del uso de memoria como para maximizar las prestaciones en una arquitectura de CPU concreta, así como la adaptación al hardware (dispositivos) presente en el sistema.

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/dnf).

2. Efectuad una actualización automática de vuestra distribución. Comprobad posibles dependencias con otros módulos utilizados y con el *bootloader* (LiLo o Grub-Legacy o Grub2) 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 https://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

[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.

[Cor05] **Corbet, J.; Rubini, A.; Kroah-Hartman, G.** (2005). *Linux Device Drivers* (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.

[Grub1] **GNU**. *Grub bootloader*. <http://www.gnu.org/software/grub/grub-legacy.html>

[Grub2] **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*. <https://www.kernel.org/doc/htmldocs/kernel-hacking/index.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>

[Lov10] **Love, R.** *Linux Kernel Development*. 3rd Edition. 2010. Addison-Wesley.

[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>

[Pra11] **Pranevich, J.** (2011). *The Wonderful World of Linux 3.0*. <http://www.kniggit.net/wwol30/>

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

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

[Tan06] **Tanenbaum, A.; Woodhull, A. S.** (2006). *Operating Systems Design and Implementation*. 3rd Edition. 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.

Sobre estas 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. [Lov10] es un texto alternativo más actualizado. Para la rama 3.x estos libros siguen siendo útiles, ya que la mayoría de conceptos del kernel se mantienen inalterados.

[Pra03] Artículo que describe algunas de las principales novedades de la rama 2.6 del núcleo Linux. [Pra11] es el equivalente para 3.0+.

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

[Bac86] [Vah96] [Tan87] [Tan86] Algunos textos sobre los conceptos, diseño e implementación de los núcleos de diferentes versiones UNIX y Linux.

[Skoa][Zan01][Kan][Grub1][Grub2] Recursos para tener más información sobre los cargadores LiLo, Grub y Grub2.

[Gru2][Grub1] Sitios oficiales de Grub2 y el anterior original Grub (ahora conocido como Grub Legacy.)