Software libre

Josep Jorba Esteve
Remo Suppi Boldrito

XP04/90785/00019

Administración avanzada de GNU/Linux

UOC
Formación de Posgrado

David Megías Jiménez

Coordinador

Ingeniero en Informática por la UAB.

Magíster en Técnicas Avanzadas de Automatización de Procesos por la UAB.

Doctor en Informática por la UAB.

Profesor de los Estudios de Informática y Multimedia de la UOC.

Jordi Mas

Coordinador

Ingeniero de software en la empresa de código abierto Ximian, donde trabaja en la implementación del proyecto libre Mono. Como voluntario, colabora en el desarrollo del procesador de textos Abiword y en la ingeniería de las versiones en catalán del proyecto Mozilla y Gnome. Es también coordinador general de Softcatalà. Como consultor ha trabajado para empresas como Menta, Telépolis, Vodafone, Lotus, eresMas, Amena y Terra España.

Josep Jorba Esteve

Autor

Ingeniero superior en Informática por la UAB.

Magíster en Arquitectura y Procesamiento Paralelo por la UAB.

Profesor de Arquitectura y Sistemas operativos del departamento de Informática UAB.

Consultor de Estudios de Informática y Multimedia de la UOC.

Profesor ayudante en el Departamento de Informática ETSE-UAB.

Remo Suppi Boldrito

Autor

Doctor en Informática.

Profesor del Departamento de Informática de la UAB.

Director de la Escuela Universitaria de Informática (Universidad Autónoma de Barcelona, España).

Primera edición: marzo 2004

Av. Tibidabo, 39-43, 08035 Barcelona
Material realizado por Eureca Media, SL

Depósito legal: B-7.597-2004
ISBN: 84-9788-116-8

Índice

ANOTACIONES

ANOTACIONES

ANOTACIONES

ANOTACIONES

Agradecimientos

Los autores agradecen a la Fundación para la Universitat Oberta de Catalunya (http://www.uoc.edu) la financiación de la primera edición de esta obra, enmarcada en el Máster Internacional en Software Libre ofrecido por la citada institución.

ANOTACIONES

Presentación

Los sistemas GNU/Linux han llegado a un grado de madurez importante, que los hacen válidos para integrarlos en cualquier ambiente de trabajo, ya sea desde el escritorio del PC personal, hasta el servidor de una gran empresa.

El objetivo principal de este curso es introducirnos en el mundo de la administración de los sistemas GNU/Linux.

Aprenderemos cómo proporcionar desde GNU/Linux los servicios necesarios a diferentes ambientes de usuarios y máquinas. El campo de la administración de sistemas es enorme, hay muchas tareas, muchos problemas por tratar, hay que tener grandes conocimientos de hardware y software, y no está de más un poco de psicología para tratar con los usuarios finales de los sistemas.

El curso no pretende abordar una distribución GNU/Linux particular, pero se han escogido un par de ellas para tratar los ejemplos: Debian y Red Hat. Respecto al campo de la administración, ésta se intentará gestionar desde el nivel más bajo posible, normalmente la línea de comandos y los ficheros de configuración. Se comentarán, en su caso, herramientas de más alto nivel, pero hay que tener cuidado con estas últimas, ya que suelen ser fuertemente dependientes de la distribución utilizada e incluso de la versión de ésta; además, estas herramientas suelen variar mucho entre versiones. La administración de bajo nivel suele ser mucho más dura, pero sabemos qué estamos haciendo y dónde podemos ver los resultados, además de que nos aporta muchos conocimientos extra sobre las diferentes tecnologías utilizadas.

Las distribuciones escogidas han sido: Debian Woody 3.0, y Red Hat 9.0 (o compatibles como Fedora), utilizadas en el momento de confeccionar este curso (a finales del 2003). La distribución Debian es un paradigma dentro del movimiento Open Source, por no pertenecer a ninguna empresa y estar confeccionada sólo por las apor-

ANOTACIONES

taciones de los voluntarios distribuidos por todo el mundo. Debian, además, integra exclusivamente software libre (pueden añadirse otros aparte).

Red Hat, por otra parte, es la distribución de una de las empresas más solventes en el panorama comercial, y por eso sea quizás la que otorgue más soporte a nivel empresarial (mediante servicios de pago). En Debian el soporte depende de los voluntarios y del conocimiento compartido de los usuarios.

Siendo la administración de sistemas un campo tan amplio, este manual sólo pretende introducirnos en este apasionante (y cómo no, también a veces frustrante) mundo. Veremos algunas de las tareas típicas, y cómo tratar las problemáticas; pero la administración es un campo que se aprende día a día, con el trabajo diario. Y desde aquí advertimos de que este manual es un trabajo abierto, que con sus aciertos y los más que probables errores, se puede ver complementado con los comentarios de sus (sufridores) usuarios. De modo que son bienvenidos cualquier tipo de comentarios y sugerencias de mejora de los materiales.

Comentamos, por último, que el contenido del manual refleja el estado de las distribuciones y de las herramientas de administración en el momento de su confección (a finales del 2003).

ANOTACIONES

1. Introducción al sistema operativo GNU/Linux

Los sistemas GNU/Linux [Joh98] ya no son una novedad, cuentan con una amplia variedad de usuarios y de ámbitos de trabajo donde son utilizados.

Su origen se remonta al mes de agosto de 1991, cuando un estudiante finlandés llamado Linus Torvalds anunció en una lista de *news* que había creado su propio núcleo de sistema operativo y lo ofrecía a la comunidad de desarrolladores para que lo probara y sugiriera mejoras para hacerlo más utilizable. Éste sería el origen del núcleo (o *kernel*) del operativo que más tarde se llamaría Linux.

Por otra parte, la FSF (Free Software Foundation), mediante su proyecto GNU, producía software (desde 1984) que podía ser utilizado libremente. Debido a lo que Richard Stallman (miembro de la FSF) consideraba software libre, es decir, como aquél del que podíamos conseguir sus fuentes (código), estudiarlas y modificarlas, y redistribuirlo sin que nos obliguen a pagar por ello. En este modelo, el negocio no está en la ocultación del código, sino en el software complementario añadido, en la adecuación del software a los clientes y en los servicios añadidos, como el mantenimiento y la formación de usuarios (el soporte que les demos), ya sea en forma de material, libros y manuales, o en cursos de formación.

La combinación (o suma) del software GNU y del *kernel* Linux, es el que nos ha traído a los actuales sistemas GNU/Linux. Actualmente, los movimientos Open Source, desde diferentes organizaciones (como FSF) y empresas como las que generan las diferentes distribuciones Linux (Red Hat, Mandrake, SuSe, ...), pasando por grandes empresas como HP, IBM o Sun que proporcionan apoyo, han dado un empujón muy grande a los sistemas GNU/Linux hasta situarlos al nivel de poder competir, y superar, muchas de las soluciones propietarias cerradas existentes.

ANOTACIONES

Nota

Los sistemas GNU/Linux no son ya una novedad. El software GNU se inició a mediados de los ochenta, el *kernel* Linux, a principios de los noventa. Y Linux se apoya en tecnología probada de UNIX, con más de 30 años de historia.

En esta unidad introductoria repasaremos algunas ideas generales de los movimientos Open Source y Free software, así como un poco de historia de Linux, y de sus orígenes compartidos con UNIX, de donde ha heredado más de 30 años de investigación en sistemas operativos.

1.1. Software Libre y Open Source

Bajo la idea de los movimientos (o filosofías) de Software Libre y Open Source [OSI03c] [OSI03b] (también llamado de código abierto o software abierto), se encuentran varias formas de software, no todas del mismo tipo, pero sí compartiendo muchas ideas comunes.

La denominación de un producto de software como 'de código abierto' conlleva como idea más importante la posibilidad de acceder a su código fuente, y la posibilidad de modificarlo y redistribuirlo de la manera que se considere conveniente, estando sujeto a una determinada licencia de código abierto, que nos da el marco legal.

Frente a un código de tipo propietario, en el cual un fabricante (empresa de software) encierra su código, ocultándolo y restringiéndose los derechos a sí misma, sin dar posibilidad de realizar ninguna adaptación ni cambios que no haya realizado previamente la empresa fabricante, el código abierto ofrece, entre otras consideraciones:

a) Acceso al código fuente, ya sea para estudiarlo (ideal para educación) o modificarlo, sea para corregir errores, adaptarlo o añadir más prestaciones.

ANOTACIONES

b) Gratuidad: normalmente, el software, ya sea en forma binaria o en la forma de código fuente, puede obtenerse libremente o por una módica cantidad en concepto de gastos de empaquetamiento, distribución y valores añadidos.

c) Evitar monopolios de software propietario: no depender de una única opción o único fabricante de nuestro software. Esto es más importante cuando se trata de una gran organización, ya sea una empresa o estado, los cuales no pueden (o no deberían) ponerse en manos de una determinada única solución y pasar a depender exclusivamente de ella.

d) Un modelo de avance, no basado en la ocultación de información, sino en la compartición del conocimiento (semejante al de la comunidad científica), para lograr progresos de forma más rápida, con mejor calidad, ya que las elecciones tomadas están basadas en el consenso de la comunidad, y no en los caprichos de empresas desarrolladoras de software propietario.

Crear programas y distribuirlos junto al código fuente no es nuevo. Ya desde los inicios de la informática y en los inicios de la red Internet se había hecho así. Sin embargo, el concepto de *código abierto* como tal, la definición y la redacción de las condiciones que tenía que cumplir datan de mediados de 1997.

Eric Raymond y Bruce Perens fueron los que divulgaron la idea. Raymond [Ray97] era autor del ensayo titulado "La catedral y el Bazar", que hablaba sobre las técnicas de desarrollo de software utilizadas por la comunidad Linux, encabezada por Linus Torvalds, y la comunidad GNU de la Free Software Foundation (FSF), encabezada por Richard Stallman. Por su parte, Bruce Perens era en aquel momento el jefe del proyecto Debian, que trabajaba en la creación de una distribución de GNU/Linux integrada únicamente con software libre.

Nota

Dos de las comunidades más importantes son la FSF, con su proyecto de software GNU, y la comunidad Open Source, cuyo máximo exponente de proyecto es Linux. GNU/Linux es el resultado de la unión de sus trabajos.

Nota

Ver versión española en:
http://es.tldp.org/Otros/catedral-bazar/cathedral-es-paper-00.html

ANOTACIONES

Una distinción importante entre estas comunidades son las definiciones de *código abierto* y *software libre*. [Deb03a] [PS02]

El Software Libre (*free software*) [FSF03] es un movimiento que parte de las ideas de Richard Stallman, que considera que hay que garantizar que los programas estuviesen al alcance de todo el mundo de forma gratuita, se tuviese acceso libre a éstos y pudieran utilizarse al antojo de cada uno. Una distinción importante, que causó ciertas reticencias a las empresas, es el término *free*. En inglés, este término tiene el doble significado de 'gratuito' y 'libre'. La gente de la FSF buscaba las dos cosas, pero era difícil vender ambas cosas a las empresas; la pregunta típica era: ¿cómo se podía ganar dinero con esto? La respuesta vino de la comunidad Linux (con Linus Torvalds en cabeza), cuando consiguieron tener una cosa que todavía no había logrado la comunidad GNU y la FSF: tener un sistema operativo libre con código fuente disponible. En este momento es cuando a la comunidad se le ocurrió juntar las diversas actividades que había en la filosofía del Software Libre bajo la nueva denominación de *código abierto* (*open source*).

Open Source se registró como una marca de certificación, a la que podían adherirse los productos software que respetasen sus especificaciones. Esto no gustó a todo el mundo y suele haber cierta separación y controversias entre los dos grupos del Open Source y la FSF (con GNU), pero son más las cosas que los unen que las que los separan.

En cierta manera, para los partidarios del software libre (como la FSF), el código abierto (u *open source*) representa un paso en falso, ya que representa una cierta "venta" al mercado de sus ideales, y deja la puerta abierta a que se vaya haciendo propietario el software que era libre. Los partidarios de *open source* ven la oportunidad de promocionar el software que de otra manera estaría en una utilización minoritaria, mientras que con la divulgación y la puesta en común para todo el mundo, incluidas empresas que quieran participar en código abierto, entramos con suficiente fuerza para plantar cara al software propietario.

Sin embargo, la idea que persiguen ambas filosofías es la de aumentar la utilidad del software libre, ofreciendo así una alternativa a las soluciones únicas que las gran-

ANOTACIONES

des empresas quieren imponer. Las diferencias son más filosóficas que prácticas.

Una vez establecidas las ideas básicas de la comunidad del código abierto, llegamos al punto en que había que concretar de manera clara qué criterios tenía que cumplir un producto de software para considerarse de código abierto. Había que contar con una definición de código abierto [OSI03b], que inicialmente escribió Bruce Perens en junio de 1997 como resultado de comentarios de los desarrolladores de la distribución Debian Linux, y que posteriormente fue reeditada (con modificaciones menores) por la organización OSI (Open Source Initiative). Esta organización está encargada de regular la definición y controlar las licencias de código abierto.

Nota

Ver la definición original de Open Source en:

http://www.opensource.org/docs/definition.php

Y la reedición en:

http://www.opensource.org

Nota

El código abierto está regulado por una definición pública que se utiliza como base de la redacción de sus licencias de software.

Un pequeño resumen (interpretación) de la definición: Un Open Source Software [OSI03b], o software de código fuente abierto, debe cumplir los requisitos siguientes:

1) Se puede copiar, regalar o vender a terceros el software, sin tener que pagar a nadie por ello. Se permite copiar el programa.

2) El programa debe incluir el código fuente y tiene que permitir la distribución tanto en forma compilada, como en fuente. O, en todo caso, hay que facilitar algún modo de obtener los códigos fuente (por ejemplo, descarga desde Internet). No está permitido ocultar el código, o darlo en representaciones intermedias. Garantiza que se pueden hacer modificaciones.

3) La licencia del software tiene que permitir que se puedan realizar modificaciones y trabajos que se deriven, y que entonces se puedan distribuir bajo la misma licencia que la original. Permite reutilizar el código original.

4) Puede requerirse la integridad del código del autor, o sea, las modificaciones se pueden presentar en forma de parches al código

ANOTACIONES

original, o se puede pedir que tengan nombres o números distintos a los originales. Esto protege al autor de qué modificaciones puedan considerarse como suyas. Este punto depende de lo que diga la licencia del software.

5) La licencia no debe discriminar a ninguna persona o grupo. No se debe restringir el acceso al software. Un caso aparte son las restricciones por ley, como las de las exportaciones tecnológicas fuera de USA a terceros países. Si existen restricciones de este tipo, hay que mencionarlas.

6) No discriminar campos laborales. El software puede utilizarse en cualquier ambiente de trabajo, aunque no haya estado pensado para él. Otra lectura es permitir fines comerciales, nadie puede impedir que el software se utilice con fines comerciales.

7) La licencia es aplicable a todo el mundo que reciba el programa.

8) Si el software forma parte de producto mayor, debe permanecer con la misma licencia. Esto controla que no se separen partes para formar software propietario (de forma no controlada). En el caso de software propietario, hay que informar que hay partes (y cuáles) de software de código abierto.

9) La licencia no debe restringir ningún software incorporado o distribuido conjuntamente, o sea, incorporarlo no debe suponer ninguna barrera para otro producto de software distribuido conjuntamente. Éste es un punto "polémico", ya que parece contradecirse con el anterior, básicamente dice que cualquiera puede coger software de código abierto y añadirlo al suyo sin que afecte a las condiciones de su licencia (por ejemplo propietaria), aunque sí que, según el punto anterior, tendría que informar que existen partes de código abierto.

10) La licencia tiene que ser tecnológicamente neutra. No deben mencionarse medios de distribución únicos, o excluirse posibilidades. Por ejemplo, no puede limitarse (por licencia) que se haga la distribución en forma de CD, ftp o mediante web.

Esta definición de *código abierto* no es por sí misma una licencia de software, sino más bien una especificación de qué requisitos debería cumplir una licencia de software de código abierto.

ANOTACIONES

La licencia que traiga el programa tiene que cumplir las especificaciones anteriores para que el programa se considere de código abierto. La organización OSI se encarga de comprobar que las licencias cumplen las especificaciones. En la página web de Open Source Licenses se puede encontrar la lista de las licencias [OSI03a], siendo una de las más famosas y utilizadas, la GPL (GNU Public License).

Nota

Open Source Licences: http://www.opensource.org/licenses/index.html

Bajo GPL, el software puede ser copiado y modificado, pero las modificaciones deben hacerse públicas bajo la misma licencia. Y se impide que el código se mezcle con código propietario, para evitar así que el código propietario se haga con partes abiertas. Existe una licencia LGPL que es prácticamente igual, pero permite que software con esta licencia sea integrado en software propietario. Un ejemplo clásico es la biblioteca (*library*) C de Linux (con licencia LGPL); si ésta fuera GPL, sólo podría desarrollarse software libre, con la LGPL se permite usar para desarrollar software propietario.

Muchos proyectos de software libre, o con parte de código abierto y parte propietario, tienen su propia licencia: Apache (basada en BSD), Mozilla (MPL y NPL de Netscape), etc. Básicamente, a la hora de poner el software como *open source* podemos poner nuestra propia licencia que cumpla la definición anterior (de código abierto), o podemos escoger licenciar bajo una licencia ya establecida, o como en el caso de la GPL, nos obliga a que nuestra licencia también sea GPL.

Una vez vistos los conceptos de *código abierto* y sus licencias, nos queda por tratar hasta qué punto es rentable para una empresa trabajar o producir código abierto. Si no fuera atrayente para las empresas, perderíamos a la vez tanto un potencial cliente como uno de los principales productores de software.

En el código abierto existen diferentes rentabilidades atrayentes de cara a las empresas:

Nota

El código abierto es también atrayente para las empresas, con un modelo de negocio donde se prima el valor añadido al producto.

ANOTACIONES

a) Para las empresas desarrolladoras de software, se crea un problema, ¿cómo es posible ganar dinero sin vender un producto? Hay mucho dinero gastado en desarrollar un programa y después es necesario obtener beneficios. Bien, la respuesta no es simple, no se puede conseguir con cualquier software, la rentabilidad se encuentra en el tipo de software que puede generar beneficios más allá de la simple venta. Normalmente, hay que hacer un estudio de si la aplicación se tornará rentable al desarrollarla como software abierto (la mayoría sí que lo hará), basándose en las premisas de que tendremos un descenso de gasto en desarrollo (la comunidad nos ayudará), reducción de mantenimiento o corrección de errores (la comunidad puede ofrecer esto muy rápido), y tener en cuenta el aumento de número de usuarios que nos proporcionará el código abierto, así como las necesidades que tendrán de nuestros servicios de apoyo o documentación. Si la balanza es positiva, entonces será viable prescindir de los ingresos generados por las ventas.

b) Aumentar la cuota de usuarios.

c) Obtener mayor flexibilidad de desarrollo, cuantas más personas intervienen, más gente habrá para detectar errores.

d) Los ingresos en su mayor parte vendrán por el lado del apoyo, formación de usuarios y mantenimiento.

e) En empresas que utilizan software, hay que considerar muchos parámetros a la hora de escoger el software para el desarrollo de las tareas, hay que tener en cuenta cosas como: rendimiento, fiabilidad, seguridad, escalabilidad y coste monetario. Y aunque parece que el código abierto ya supone de por sí una elección por el coste económico, hay que decir que existe software abierto que puede competir con (o incluso superar) el propietario en cualquiera de los otros parámetros. Además, hay que vigilar mucho con las opciones o sistemas propietarios de un único fabricante, no podemos depender únicamente de ellos (podemos recordar casos, en otros ámbitos, como los vídeos beta de Sony frente a VHS, o en los PC la arquitectura MicroChannel de IBM). Tenemos que evitar el uso de monopolios con lo que éstos suponen: falta de competencia en los precios, servicios caros, mantenimiento caro, poca (o nula) variedad de opciones, etc.

ANOTACIONES

f) Para los usuarios particulares ofrece gran variedad de software adaptado a tareas comunes, ya que mucho del software ha sido pensado e implementado por personas que querían hacer esas mismas tareas pero no encontraban el software adecuado. Normalmente, en el caso del usuario particular un parámetro muy importante es el coste del software, pero la paradoja es que en el usuario doméstico es donde se hace más uso de software propietario. Normalmente, los usuarios domésticos hacen uso de productos de software con copias ilegales, algunas estadísticas recientes indican índices del 60-70% de copias ilegales domésticas. El usuario siente que sólo por tener el ordenador doméstico PC ya tiene "derecho" a disponer de software para usarlo. En estos casos estamos bajo situaciones "ilegales" que, aunque no han sido perseguidas, pueden serlo en su día, o bien se intentan controlar por sistemas de licencias (o activaciones de productos). Además, esto tiene unos efectos perjudiciales indirectos sobre el software libre, debido a que si los usuarios hacen un uso amplio de software propietario, esto obliga a quien se quiera comunicar con ellos, ya sean bancos, empresas o administraciones públicas, a hacer uso del mismo software propietario, y ellos sí que abonan las licencias a los productos. Una de las "batallas" más importantes para el software libre es la posibilidad de captar a los usuarios domésticos.

Nota

Las copias ilegales domésticas son también denominadas a veces ***copias piratas***.

g) Por último, los estados, como caso particular, pueden obtener beneficios importantes del software de código abierto, ya que pueden disponer de software de calidad a precios "ridículos" comparados con el enorme gasto de licencias de software propietario (miles o decenas de miles). Además de que el software de código abierto permite integrar fácilmente a las aplicaciones, cuestiones culturales (de cada país) como, por ejemplo, su lengua. Este último caso es bastante problemático, ya que en determinadas regiones, estados pequeños con lengua propia, los fabricantes de software propietario se niegan a adaptar sus aplicaciones, o instan a que se les pague por hacerlo.

1.2. UNIX. Un poco de historia

Como antecesor de nuestros sistemas GNU/Linux [Sta02], vamos a recordar un poco la historia de UNIX [Sal94] [Lev03]. En origen, Linux

ANOTACIONES

se pensó como un clon de Minix (una implementación académica de UNIX para PC) y de algunas ideas desarrolladas en los UNIX propietarios; pero, a su vez, se desarrolló en código abierto, y con orientación a los PC domésticos. Veremos, en este apartado dedicado a UNIX y el siguiente dedicado a GNU/Linux, cómo esta evolución nos ha llevado hasta los sistemas GNU/Linux actuales que pueden competir con cualquier UNIX propietario, y que están disponibles para un amplio número de arquitecturas hardware, desde el simple PC hasta los supercomputadores.

Nota

Linux puede ser utilizado en un amplio rango de máquinas. En la lista TOP500, pueden encontrarse varios supercomputadores con Linux (ver lista en sitio web top500.org): por ejemplo, el MCR Linux Cluster del departamento de energía de USA en los laboratorios Lawrence en Livermore, un cluster de 2304 CPUs Intel Xeon con sistema operativo Linux.

UNIX se inició hacia el año 1969 (en el 2003 tiene más de 30 años de historia) en los laboratorios BTL (Bell Telephone Labs) de AT&T. Éstos se acababan de retirar de la participación de un proyecto llamado MULTICS, cuyo objetivo era crear un sistema operativo con el cual un gran ordenador pudiera dar cabida a un millar de usuarios simultáneos. En este proyecto participaban los BTL, General Electric, y el MIT. Pero falló, en parte, por ser demasiado ambicioso para su época.

Mientras se desarrollaba este proyecto, dos ingenieros de los BTL que participaban en MULTICS: Ken Thompson y Dennis Ritchie, encontraron un ordenador que no estaba utilizando nadie, un DEC PDP7, que sólo tenía un ensamblador y un programa cargador. Thompson y Ritchie desarrollaron como pruebas (y a menudo en su tiempo libre) partes de UNIX, un programa ensamblador (del código máquina) y el núcleo rudimentario del sistema operativo.

Ese mismo año, 1969, Thompson tuvo la idea de escribir un sistema de ficheros para el núcleo creado, de manera que se pudiesen almacenar ficheros de forma ordenada en un sistema de directorios jerárquicos. Después de unas cuantas discusiones teóricas (que se

ANOTACIONES

alargaron unos dos meses) se implementó el sistema en un par de días. A medida que se avanzaba en el diseño del sistema, en el cual se incorporaron algunos ingenieros más de los BTL, la máquina original se les quedó pequeña, y pensaron en pedir una nueva (en aquellos días costaban cerca de 100.000 dólares, era una buena inversión). Tuvieron que inventarse una excusa (ya que el sistema UNIX era un desarrollo en tiempo libre) y dijeron que la querían para crear un nuevo procesador de texto (aplicación que daba dinero en aquellos tiempos), y se les aprobó la compra de una PDP11.

Nota

UNIX se remonta al año 1969, cuenta con más de 30 años de tecnologías desarrolladas y utilizadas en todo tipo de sistemas.

Cuando les llegó la máquina, sólo les llegó la CPU y la memoria, pero no el disco ni el sistema operativo. Thompson, sin poder esperarse, diseñó un disco RAM en memoria y utilizó la mitad de la memoria como disco, y la otra para el sistema operativo que estaba diseñando. Una vez llegó el disco, se siguió trabajando tanto en UNIX como en el procesador de textos prometido (la excusa). El procesador de textos fue un éxito (se trataba de Troff, un lenguaje de edición, que posteriormente fue utilizado para crear las páginas *man* de UNIX), y los BTL comenzaron a utilizar el rudimentario UNIX con el nuevo procesador de texto, convirtiéndose así los BTL en el primer usuario de UNIX.

En aquellos momentos comenzaron a presentarse varios principios filosóficos de UNIX [Ray02a]:

- Escribir programas para hacer una cosa y hacerla bien.
- Escribir programas para que trabajaran juntos.
- Escribir programas para que manejaran flujos de texto.

Otra idea muy importante fue que UNIX fue uno de los primeros sistemas pensados para ser independiente de la arquitectura hardware, y que ha permitido portarlo con éxito a un gran número de arquitecturas hardware diferentes.

La necesidad de documentar lo que se estaba haciendo, ya que había usuarios externos, dio lugar en noviembre de 1971 al *UNIX*

ANOTACIONES

Programmer's Manual, que firmaron Thompson y Richie. En la segunda edición (junio 1972), denominada V2 (se hacía corresponder la edición de los manuales con el número de versión UNIX), se decía que el número de instalaciones de UNIX ya llegaba a las 10. Y el número siguió creciendo hasta unas 50 en la V5.

Entonces se decidió (finales de 1973) presentar los resultados en un congreso de sistemas operativos. Y como resultado, varios centros informáticos y universidades pidieron copias de UNIX. AT&T no daba apoyo ni mantenimiento de UNIX, lo que hizo que los usuarios necesitaran unirse y compartir sus conocimientos para formar comunidades de usuarios de UNIX. AT&T decidió ceder UNIX a las universidades, pero tampoco les daba apoyo, ni corrección de errores. Los usuarios comenzaron a compartir sus ideas, información programas, *bugs*, etc. Se creó una asociación denominada USENIX como agrupación de usuarios de UNIX. Su primera reunión (mayo de 1974) tuvo una docena de asistentes.

Nota

Ver: http://www.usenix.org

Una de las universidades que había obtenido una licencia de UNIX fue la universidad de California en Berkeley, donde había estudiado Ken Thompson. En 1975, Thompson volvió como profesor a Berkeley, y trajo consigo la última versión de UNIX. Dos estudiantes graduados recién incorporados, Chuck Haley y Bill Joy (hoy en día uno de los vicepresidentes de SUN Microsystems) comenzaron a trabajar en una implementación de UNIX.

Una de las primeras cosas que les decepcionó eran los editores; Joy perfeccionó un editor llamado EX, hasta transformarlo en el VI, un editor visual a pantalla completa. Y los dos escribieron un compilador de lenguaje Pascal, que añadieron a UNIX. Hubo cierta demanda de esta implementación de UNIX, y Joy lo comenzó a producir como el BSD, *Berkeley Software Distribution* (o UNIX BSD).

BSD (en 1978) tenía una licencia particular sobre su precio: decía que estaba acorde con el coste de los medios y la distribución que se tenía en ese momento. Así, los nuevos usuarios acababan haciendo algunos cambios o incorporando cosas, vendiendo sus copias "rehechas" y, al cabo de un tiempo, los cambios se incorporaban en la siguiente versión de BSD.

ANOTACIONES

Joy también realizó en su trabajo del editor VI algunas aportaciones más, como el tratamiento de los terminales de texto, de manera que el editor fuera independiente del terminal en que se utilizase; creó el sistema TERMCAP como interfaz genérica de terminales con controladores para cada terminal concreto, de manera que en la realización de los programas ya nos podíamos olvidar de los terminales utilizando la interfaz.

Un siguiente paso fue adaptarlo a diferentes arquitecturas. Hasta el año 1977 sólo se podía ejecutar en máquinas PDP; en ese año se comenzaron a hacer adaptaciones para máquinas del momento como las Interdata e IBM. La versión 7 (V7 en junio 1979) de UNIX fue la primera portable. Esta versión trajo muchos avances, ya que contenía: *awk*, *lint*, *make*, *uucp*; el manual ya tenía 400 páginas (más dos apéndices de 400 cada uno). Se incluía también el compilador de C diseñado en los BTL por Kernighan y Ritchie, que se había creado para reescribir la mayor parte de UNIX, inicialmente en ensamblador y luego pasado a C con las partes de ensamblador que fuesen sólo dependientes de la arquitectura. Se incluyeron también una *shell* mejorada (*shell* de Bourne) y comandos como: *find*, *cpio* y *expr*.

La industria UNIX comenzó también a crecer, empezaron a aparecer versiones (implementaciones) de UNIX por parte de compañías como: Xenix, colaboración entre Microsoft (en los orígenes también trabajó con versiones de UNIX) y SCO para máquinas Intel 8086 (el primer PC de IBM); nuevas versiones BSD de Berkeley...

Pero apareció un nuevo problema, cuando AT&T se dio cuenta de que UNIX era un producto comercial valioso, en la licencia de la V7 se prohibió el estudio en centros académicos, para proteger el secreto comercial. Muchas universidades utilizaban hasta el momento el código fuente de UNIX para docencia de sistemas operativos, y dejaron de usarlo para dar sólo teoría.

Pero cada uno solucionó el problema a su modo. En Amsterdam, Andrew Tanenbaum (autor de prestigio de libros de teoría de sistema operativos) decidió escribir desde el principio un nuevo sistema operativo compatible con UNIX sin utilizar una sola línea de código de AT&T; llamó a este nuevo operativo Minix. Éste sería el que posteriormente le serviría en 1991 a un estudiante finlandés para crear su propia versión de UNIX, que llamó Linux.

ANOTACIONES

Bill Joy, que continuaba en Berkeley desarrollando BSD (ya estaba en la versión 4.1), decidió marcharse a una nueva empresa llamada SUN Microsystems, en la cual acabó los trabajos del 4.2BSD, que posteriormente acabaría modificando para crear el UNIX de SUN, el SunOS (hacia 1983). Cada empresa comenzó a desarrollar sus versiones: IBM con AIX, DEC con Ultrix, HP con HPUX, Microsoft/SCO con Xenix, etc. UNIX comenzó (desde el año 1980) su andadura comercial, AT&T sacó una ultima versión llamada UNIX SystemV (SV), de la cual derivan, junto con los 4.xBSD, los UNIX actuales, ya sea de la rama BSD o de la SystemV. La SV tuvo varias revisiones, por ejemplo, la SV Release 4 fue una de las más importantes. La consecuencia de estas últimas versiones es que más o menos todos los UNIX existentes se adaptaron uno al otro; en la práctica son versiones del SystemV R4 de AT&T o del BSD de Berkeley, adaptadas por cada fabricante. Algunos fabricantes lo especifican y dicen que su UNIX es de tipo BSD o SV, pero la realidad es que todos tienen un poco de las dos, ya que posteriormente se hicieron varios estándares de UNIX para intentar uniformizarlos; entre ellos encontramos los IEEE POSIX, UNIX97, FHS, etc.

Nota

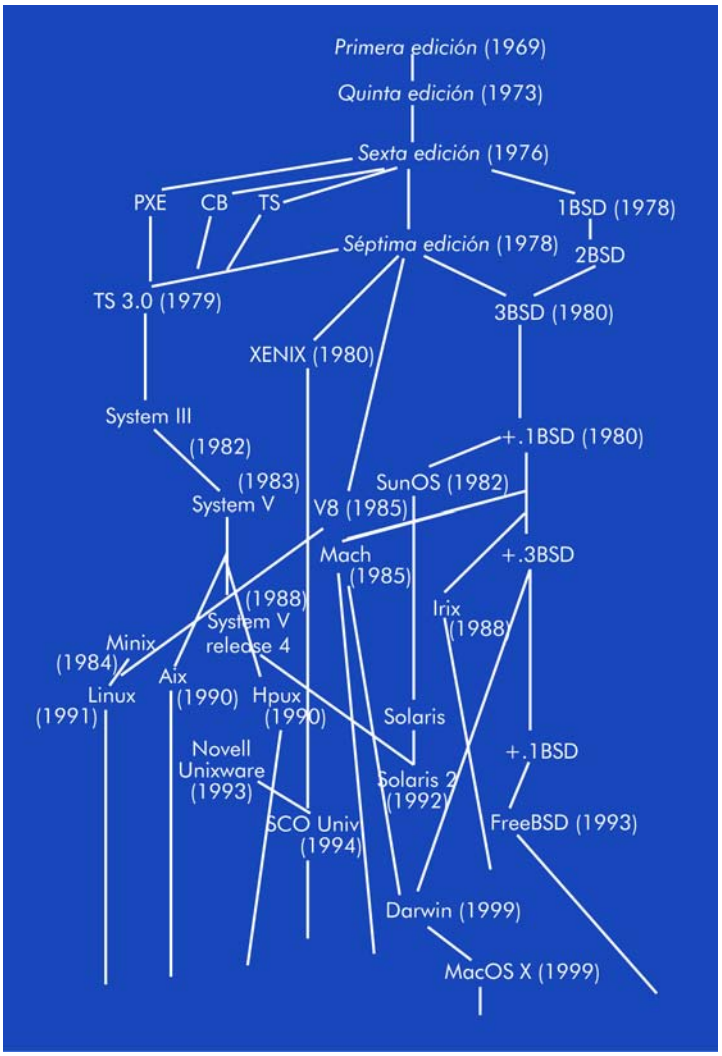
Con el tiempo, UNIX se dividió en varias ramas de sistema, siendo las dos principales, la que derivaba del AT&T UNIX o System V, y la de la universidad de California, el BSD. La mayoría de UNIX actuales derivan de uno u otro, o son una mezcla de los dos.

Pero AT&T en aquellos momentos (SVR4) pasó por un proceso judicial por monopolio telefónico (era la principal, si no la única, compañía telefónica en Estados Unidos), que hizo que se dividiera en múltiples empresas más pequeñas, y los derechos de UNIX originales comenzaron un baile de propietarios importante: en 1990 los tenían a medias el Open Software Foundation (OSF) y UNIX International (UI), después, UNIX Systems Laboratories (USL), que denunció a la Universidad de Berkeley por sus copias del BSD, pero perdió, ya que la licencia original no imponía derechos de propiedad al código de UNIX. Más tarde, los derechos UNIX se vendieron a la empresa Novell, ésta cedió parte a SCO, y hoy en día (2003) no está muy claro quién los tiene: los reclaman Novell, la OSF y SCO. Un ejemplo reciente de esta problemática puede ser el caso de SCO, que puso un pleito mul-

ANOTACIONES

timillonario a IBM porque ésta había cedido parte del código UNIX en versiones del *kernel* Linux, que supuestamente incluyen algún código UNIX original. El resultado hoy (a finales del 2003) es que el asunto continúa en los tribunales, con SCO convertida en un "paria" de la industria informática que amenaza a los usuarios Linux, IBM, y otros UNIX propietarios, con la afirmación de que tienen los derechos UNIX originales, y que los demás tienen que pagar por ellos. Habrá que ver cómo evoluciona todo esto.

Figura 1. Resumen histórico de varias versiones UNIX



El panorama actual de UNIX ha cambiado mucho desde la aparición de Linux (1991), que a partir de los años 1995-99 comenzó a convertirse en una alternativa seria a los UNIX propietarios, por la gran cantidad de plataformas hardware que soporta y el amplio apoyo de la comunidad internacional y empresas en el avance. Hay diferentes

ANOTACIONES

versiones UNIX propietarias que siguen sobreviviendo en el mercado, tanto por su adaptación a entornos industriales o por ser el mejor operativo existente en el mercado, como porque hay necesidades que sólo pueden cubrirse con UNIX y el hardware adecuado. Además, algunos de los UNIX propietarios todavía son mejores que GNU/Linux en cuanto a fiabilidad y rendimiento aunque cada vez acortando distancias, ya que las mismas empresas que tienen sus UNIX propietarios se interesan cada vez más en GNU/Linux, y aportan parte de sus desarrollos para incorporarlos a Linux. Es de esperar una muerte más o menos lenta de las versiones propietarias de UNIX hacia distribuciones basadas en Linux de los fabricantes adaptadas a sus equipos.

Un panorama general de estas empresas (en el verano del 2003):

Nota

Muchas de las empresas que disponen de UNIX propietarios participan en GNU/Linux y ofrecen algunos de sus desarrollos a la comunidad.

- SUN: dispone de su implementación de UNIX llamada Solaris (evolución del SunOS). Comenzó como un sistema BSD, pero ahora es mayoritariamente SV y partes de BSD; es muy utilizado en las máquinas Sun con arquitectura Sparc, y en máquinas multiprocesador (hasta unos 64 procesadores). Promocionan Linux como entorno de desarrollo para Java, y disponen de una distribución de Linux denominada Java Desktop System, que ha tenido una amplia aceptación. Además, ha comenzado a usar Gnome como escritorio, y ofrece apoyo financiero a varios proyectos como Mozilla, Gnome y OpenOffice.

- IBM: tiene su versión de UNIX llamada AIX, que está teniendo problemas con SCO por las licencias UNIX. Por otra parte, presta apoyo firme a la comunidad Open Source, proporciona entornos de desarrollo (eclipse.org) y tecnologías Java para Linux, incorpora Linux a sus grandes máquinas y diseña campañas publicitarias (marketing) para promocionar Linux.

- HP: tiene su UNIX HPUX, pero da amplio soporte a Linux, tanto en forma de código en Open Source, como instalando Linux en sus

ANOTACIONES

máquinas. Se dice que es la compañía que ha ganado más dinero con Linux.

- SGI: Silicon Graphics tiene un UNIX llamado IRIX para sus máquinas gráficas, pero está comenzando a vender máquinas con Windows, y puede que algunas con Linux (todavía no se define). A la comunidad Linux, le ofrece soporte de OpenGL (tecnología de gráficos 3D) y de diferentes sistemas de ficheros y control de dispositivos periféricos.

- Apple: se incorporó recientemente (a partir de mediados de los noventa) al mundo UNIX, cuando decidió sustituir su operativo por una variante UNIX. El núcleo llamado Darwin proviene de una versión 4.4BSD; este núcleo Open Source será el que, sumado a unas interfaces gráficas muy potentes, de a Apple su sistema operativo MacOS X. Considerado hoy en día como uno de los mejores UNIX y, como mínimo, uno de los más "bellos" en aspecto gráfico. También emplea software GNU como utilidades de sistema.

- Distribuidores Linux: tanto comerciales como organizaciones, mencionaremos a empresas como Red Hat, SuSe, Mandrake, y organizaciones no comerciales como Debian, etc. Entre éstas (las distribuciones con mayor despliegue) y las más pequeñas, se llevan el mayor desarrollo de Linux, y tienen el apoyo de la comunidad Linux y de la FSF con el software GNU, además de recibir contribuciones de las citadas empresas.

- BSD: aunque no sea una empresa como tal, mencionaremos cómo desde Berkeley y otros intermediarios se continúa el desarrollo de las versiones BSD, así como otros proyectos libres clones de BSD como los operativos FreeBSD, netBSD, OpenBSD (el UNIX considerado más seguro), TrustedBSD, etc., que también, más tarde o más temprano, suponen mejoras o incorporaciones de software a Linux. Además, una aportación importante es el *kernel* Darwin proveniente de 4.4BSD, y que desarrolló Apple como núcleo Open Source de su sistema operativo MacOS X.

- Microsoft: aparte de entorpecer el desarrollo de UNIX y GNU/Linux, poniendo trabas con incompatibilidades en diferentes tecnolo-

Nota

Ver la opinión de la FSF: http://www.gnu.org/philosophy/sco/sco.html

gías, no tiene participación directa en el mundo UNIX/Linux. Si bien recientemente compró una licencia UNIX a SCO, no están claros los motivos de Microsoft a la hora de realizar esta adquisición, aunque algunos sugieren que existe alguna relación con el hecho de proporcionar apoyo a SCO en su juicio contra IBM.

Nota

Ver:
http://www.wehavethewayout.com/us/index.asp
http://www.wehavethewayout.com/us/index.asp
http://www.soportelinux.com/articulo.php?articulo id = 6

Otra anécdota curiosa es que, junto a una empresa llamada UniSys, se dedican a hacer propaganda de cómo convertir sistemas UNIX a sistemas Windows; y aunque el objetivo podía ser más o menos loable, lo curioso era que el servidor original de la web empresarial estaba en una máquina FreeBSD con Apache. En ocasiones, también paga a algunas empresas "independientes" (algunos opinan que bastante poco) para que lleven a cabo estudios de rendimiento entre UNIX/Linux y Windows.

Como resumen general, algunos comentarios que suelen aparecer en la bibliografía UNIX apuntan a que UNIX es técnicamente un sistema sencillo y coherente diseñado con buenas ideas que se supieron llevar a la práctica, pero que no hay que olvidar que algunas de estas ideas se consiguieron gracias al apoyo entusiasta que brindó una gran comunidad de usuarios y desarrolladores que colaboraron entre sí, compartiendo una tecnología y gobernando su evolución.

Y como la historia se suele repetir, en este momento la evolución y el entusiasmo continúan con los sistemas GNU/Linux.

1.3. Sistemas GNU/Linux

Hace unos veinte años los usuarios de los primeros ordenadores personales no disponían de muchos sistemas operativos donde elegir. El mercado de los ordenadores personales lo dominaba un DOS de Microsoft. Otra posibilidad eran los MAC de Apple, pero a unos precios desorbitados en comparación con el resto. La otra opción importante, aunque reservada a grandes (y caras) máquinas, era UNIX.

ANOTACIONES

Una primera opción que apareció fue MINIX (1984), creado desde cero por Andrew Tanenbaum, que se pensó para la educación, para enseñar diseño e implementación de sistemas operativos.

MINIX fue pensado para ejecutarse sobre una plataforma Intel 8086, muy popular en la época porque era la base de los primeros IBM PC. La principal ventaja de este operativo radicaba en su código fuente, accesible a cualquiera (doce mil líneas de código entre ensamblador y C), ya que estaba incluido en el libro de operativos de Tanenbaum. Pero MINIX era más una herramienta de enseñanza que un sistema eficaz pensado para el rendimiento o para actividades profesionales.

En los noventa, la FSF (Free Software Foundation) y su proyecto GNU, motivó a muchos programadores para promover el software de calidad y de distribución libre. Y aparte de software de utilidades, se trabajaba en un núcleo (*kernel*) de operativo denominado HURD, que tendría varios años de desarrollo.

Mientras, en octubre de 1991, un estudiante finlandés llamado Linus Torvalds presentaría la versión 0.01 de su *kernel* de sistema operativo, que denominó Linux, orientado a máquinas Intel con 386, y lo ofreció bajo licencia GPL a foros de programadores y a la comunidad de Internet para que lo probaran y, si les gustaba, ayudaran a su desarrollo. El entusiasmo fue tal, que en poco tiempo había miles de programadores trabajando en el núcleo o en aplicaciones para él.

Algunas de las características que diferenciaron a Linux de los sistemas de su tiempo y que siguen siendo aplicables, y otras heredadas de UNIX podrían ser:

a) Sistema operativo de código abierto, cualquiera puede disponer de sus fuentes, modificarlas y crear nuevas versiones que poder compartir bajo la licencia GPL (que, de hecho, lo convierte en un software libre).

b) Portabilidad: tal como el UNIX original, Linux está pensado para depender muy poco de una arquitectura concreta de máquina; consecuentemente, Linux es, en su mayor parte, independiente de la máquina de destino y puede portarse a prácticamente cualquier arquitectura que disponga de un compilador C como el GNU *gcc*. Sólo restan algunas pequeñas partes de código ensamblador y de

ANOTACIONES

algunos dispositivos dependientes de la máquina, que tienen que ser reescritas en cada puerto a una nueva arquitectura. Gracias a esto, Linux es uno de los sistemas operativos que corre en mayor número de arquitecturas: Intel x86 y IA64, AMD x86 y x8664, Sparc de Sun, MIPS de Silicon, PowerPC (Apple), IBM S390, Alpha de Compaq, m68k Motorola, Vax, ARM, HPPArisc, ...

Nota

Proyecto original Mach: http://www2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

c) *Kernel* de tipo monolítico: el diseño del *kernel* está unido en un sola pieza, pero es conceptualmente modular en las diferentes tareas. Otra escuela de diseño de operativos propone los *microkernel* (un ejemplo es Mach), donde los servicios se implementan como procesos aparte, comunicados por un (micro) *kernel* más básico. Linux se decidió como monolítico, porque es difícil extraer buen rendimiento de los *microkernels* (es un trabajo bastante duro y complejo). Por otra parte, el problema de los monolíticos es el crecimiento, cuando se vuelven muy grandes se vuelven intratables en el desarrollo, esto se intentó solucionar con los módulos cargables.

d) Módulos dinámicamente cargables: permiten poner partes del sistema operativo, como *filesystems*, o *controladores de dispositivos*, como pedazos externos que se cargan (o enlazan) con el *kernel* en tiempo de ejecución bajo demanda. Esto permite simplificar el *kernel* y ofrecer estas funcionalidades como elementos que se pueden programar por separado. Con este uso de módulos, se podría considerar a Linux como un *kernel* mixto, ya que es monolítico, pero ofrece una serie de módulos que complementan el *kernel* (aproximación parecida al *microkernel*).

e) Desarrollo del sistema por una comunidad vinculada por Internet: los sistemas operativos nunca habían tenido un desarrollo tan amplio y disperso, no suelen salir de la compañía que los elabora (en el caso propietario) o de un pequeño conjunto de instituciones académicas y laboratorios que colaboran para crear uno. El fenómeno de la comunidad Linux permite que cada uno colabore en la medida que el tiempo y sus propios conocimientos se lo permitan. El resultado son: de cientos a miles de desarrolladores para Linux. Además, por su naturaleza de sistema de código fuente abierto, Linux es un laboratorio ideal para probar ideas de sistemas operativos al mínimo coste; se puede implementar, probar, tomar medidas y, si funciona, añadir la idea al *kernel*.

Los proyectos se sucedieron y –en el inicio de Linus con el *kernel*– a la gente de la FSF, con el software de utilidad GNU y, sobre todo,

ANOTACIONES

con su compilador de C (GCC), se les unieron otros proyectos importantes como las XFree (una versión PC de las X Window), los proyectos de escritorio como KDE y Gnome. Y el desarrollo de Internet con proyectos como el servidor web Apache, el navegador Mozilla, o las bases de datos MySQL y Postgres, acabaron por dar al *kernel* inicial Linux el recubrimiento de aplicaciones suficiente para construir los sistemas GNU/Linux y competir en igualdad de condiciones con los sistemas propietarios. Y convertir a los sistemas GNU/Linux en el paradigma del software de fuente abierta (Open Source).

Nota

Los sistemas GNU/Linux se han convertido en la punta de lanza de la comunidad Open Source, por la cantidad de proyectos que se han podido aglutinar y llevar a buen término.

El nacimiento de nuevas empresas, que crearon distribuciones Linux (empaquetamientos de *kernel* + aplicaciones) y le dieron apoyo, como Red Hat, Mandrake, SuSe, contribuyó a introducir Linux en las empresas, reacias, y a comenzar el imparable crecimiento que vivimos actualmente.

Comentaremos también el nombre de los sistemas como GNU/Linux. El término Linux como el sistema operativo con que se trabaja es ampliamente usado (para simplificar el nombre), aunque en opinión de algunos desmerece el trabajo de la FSF con el proyecto GNU, que ha sido el que ha proporcionado las principales herramientas del sistema. El término Linux para referirse al sistema operativo completo es ampliamente usado comercialmente.

Nota

GNU y Linux, por Richard Stallman: http://www.gnu.org/gnu/linux-and-gnu.html.

Cuando hablamos de Linux, nos estamos refiriendo sólo al núcleo (*kernel*) del sistema operativo. Esto crea cierta confusión, ya que hay gente que habla de "sistemas" o del "sistema operativo Linux" por abreviar. Cuando se trabaja con un sistema operativo GNU/Linux, se está trabajando sobre una serie de software de utilidades, en gran parte fruto del proyecto GNU, sobre el núcleo Linux. Por lo tanto, el sistema es básicamente GNU con un núcleo Linux.

El proyecto GNU de la FSF tenía por objetivo crear un sistema operativo de software libre al estilo UNIX denominado GNU [Sta02].

ANOTACIONES

Linus Torvalds consiguió en 1991 juntar su *kernel* Linux con las utilidades GNU cuando la FSF todavía no disponía de *kernel*. El *kernel* de GNU se denomina HURD, y hoy en día se trabaja bastante en él, y ya existen algunas versiones beta de distribuciones de GNU/HURD (ver más en el apartado dedicado a la administración del *kernel*).

Se calcula que en una distribución Linux hay un 28% de código GNU y un 3% que corresponde al código del *kernel* Linux; el porcentaje restante corresponde a código de terceros, ya sea de aplicaciones o de utilidades.

Para destacar la contribución de GNU [FSF03], podemos ver algunas de sus aportaciones incluidas en los sistemas GNU/Linux:

- El compilador de C y C++ (GCC)
- El *shell bash*
- El editor Emacs (GNU Emacs)
- El intérprete *postscript* (*ghostscript*)
- La biblioteca C estándar (*GNU C library*, o también *glibc*)
- El depurador (*GNU gdb*)
- *Makefile* (*GNU make*)
- El ensamblador (*GNU assembler* o *gas*)
- El *linker* (*GNU linker* o *gld*)

Los sistemas GNU/Linux no son los únicos en utilizar software GNU; por ejemplo, los sistemas BSD incorporan también utilidades GNU. Y algunos operativos propietarios como MacOS X (de Apple) también usan software GNU. El proyecto GNU ha producido software de alta calidad, que se ha ido incorporando a la mayor parte de las distribuciones de sistemas basadas en UNIX, tanto libres como propietarias.

Es justo para todo el mundo reconocer el trabajo de cada uno denominando GNU/Linux a los sistemas que trataremos.

ANOTACIONES

1.4. El perfil del administrador de sistemas

Las grandes empresas y organizaciones dependen cada vez más de sus recursos de computación y de cómo éstos son administrados para adecuarlos a las tareas. El gran incremento de las redes distribuidas, con sus equipos servidores y clientes, ha creado una gran demanda de un nuevo perfil laboral: el llamado *administrador de sistemas*.

El administrador de sistemas tiene una amplia variedad de tareas importantes. Los mejores administradores de sistema suelen ser bastante generalistas, tanto teóricamente como prácticamente. Pueden enfrentarse a tareas como: realizar cableados de instalaciones o reparar cables; instalar sistemas operativos o software de aplicaciones; corregir problemas y errores en los sistemas, tanto hardware como software; formar a los usuarios, ofrecer trucos o técnicas para mejorar la productividad en áreas que pueden ir desde aplicaciones de procesamiento de textos hasta áreas complejas de sistemas CAD o simuladores; evaluar económicamente compras de equipamiento de hardware y software; automatizar un gran número de tareas comunes, e incrementar el rendimiento general del trabajo en su organización.

Puede considerarse al administrador como un perfil de empleado que ayuda a los demás empleados de la organización a aprovechar mejor y más óptimamente los recursos disponibles, de forma que mejore toda la organización.

La relación con los usuarios finales de la organización puede establecerse de diferentes maneras: o bien mediante la formación de usuarios o bien por ayuda directa en el caso de presentarse problemas. El administrador es la persona encargada de que las tecnologías utilizadas por los usuarios funcione adecuadamente, o sea, que los sistemas cumplan las perspectivas de los usuarios, así como las tareas que éstos quieran realizar.

Hace años, y aún actualmente, en muchas empresas u organizaciones no hay una perspectiva clara del papel del administrador. En los

ANOTACIONES

inicios de la informática (años ochenta y noventa) en la empresa, el administrador era visto en un principio como la persona "entendida" en ordenadores (el "gurú") que se encargaba de poner máquinas y que vigilaba o las reparaba en caso de problemas. Normalmente, era una especie de informático polivalente que tenía que solucionar los problemas que fueran apareciendo. Su perfil de currículum no era claro, ya que no necesitaba tener amplios conocimientos, sino sólo tener conocimientos básicos de una decena (como mucho) de aplicaciones (el procesador de texto, la hoja de cálculo, la base de datos, etc.), y algunos conocimientos básicos de hardware eran suficientes para las tareas diarias. Así, cualquier simple "entendido" en el tema podía dedicarse a este trabajo, de manera que no solían ser informáticos tradicionales, y muchas veces incluso se llegaba a una transmisión oral de los conocimientos entre algún "administrador" más antiguo en la empresa y el nuevo aprendiz.

Con lo anterior, nos encontrábamos de alguna manera en la prehistoria de la administración de sistemas (aunque hay personas que siguen pensando que básicamente se trata del mismo trabajo). Hoy en día, en la época de Internet y de los servicios distribuidos, un administrador de sistemas es un profesional (con dedicación propia y exclusiva) que proporciona servicios en la "arena" del software y hardware de sistemas. El administrador tiene que llevar a cabo varias tareas que tendrán como destino múltiples sistemas informáticos, la mayoría heterogéneos, con objeto de hacerlos operativos para una serie de tareas.

Actualmente, los administradores necesitan tener unos conocimientos generales (teóricos y prácticos) de áreas muy diversas, desde tecnologías de redes, sistemas operativos, aplicaciones de ámbitos diversos, programación básica en una amplia variedad de lenguajes de programación, conocimientos amplios de hardware –tanto del ordenador como de los periféricos usados– tecnologías Internet, diseño de páginas web, bases de datos, etc. Y normalmente también es buscado con el perfil de conocimientos básicos sobre el área de trabajo de la empresa, ya sea química, física, matemáticas, etc. No es de extrañar, entonces, que en una empresa de tamaño medio a grande se haya pasado del "chapuzas" de turno a un pequeño grupo de profesionales con amplios conocimientos, la mayoría con nivel académico universitario, con diferentes tareas asignadas dentro de la organización.

ANOTACIONES

El administrador debe dominar un rango amplio de tecnologías para poder adaptarse a una multitud de tareas variadas, que pueden surgir dentro de la organización.

Debido a la gran cantidad de conocimientos, no es extraño que aparezcan a su vez diferentes subperfiles de la tarea del administrador. En una gran organización puede ser habitual encontrar a los administradores de sistemas operativos (UNIX, Mac, o Windows), que suelen ser diferentes: administrador de bases de datos, administrador de copias de seguridad, administradores de seguridad informática, administradores encargados de atención a los usuarios, etc.

En una organización más pequeña, varias o todas las tareas pueden estar asignadas a uno o pocos administradores. Los administradores de sistemas UNIX (o de GNU/Linux) serían una parte de estos administradores (cuando no el administrador que tendrá que hacer todas las tareas). Normalmente, su plataforma de trabajo es UNIX (o GNU/Linux en nuestro caso), y requiere de bastantes elementos específicos que hacen este trabajo único. UNIX (y variantes) es un sistema operativo abierto y muy potente, y, como cualquier sistema software, requiere de cierto nivel de adecuación, configuración y mantenimiento en las tareas para las que vaya a ser usado. Configurar y mantener un sistema operativo es una tarea seria, y en el caso de UNIX puede llegar a ser bastante frustrante.

Algunas áreas importantes por tratar son:

a) Que el sistema sea muy potente también indica que habrá bastantes posibilidades de adaptarlo (configurarlo) a las tareas que queremos hacer. Habrá que evaluar las posibilidades que se nos ofrecen y cuán adecuadas son para nuestro objetivo final.

b) Un sistema abierto y ejemplo claro de ello es nuestro GNU/Linux, que nos ofrecerá actualizaciones permanentes, ya sea en la corrección de errores del sistema, como en la incorporación de nuevas prestaciones. Y, evidentemente, todo esto tiene unos impactos directos importantes en costes de mantenimiento de las tareas de administración.

ANOTACIONES

c) Los sistemas se pueden utilizar para tareas de coste crítico, o en puntos críticos de la organización, donde no se pueden permitir fallos importantes, o que ralenticen o paren la marcha de la organización.

d) Las redes son actualmente un punto muy importante (si no el que más), pero también es un área de problemas potenciales muy crítica, tanto por su propia naturaleza distribuida como por la complejidad del sistema para encontrar, depurar y solucionar los problemas que se puedan presentar.

e) En el caso particular de los sistemas UNIX, y en nuestros GNU/Linux, la abundancia, tanto de versiones como de distribuciones diferentes del sistema, incorpora problemas adicionales a la administración, ya que es necesario conocer las problemáticas y diferencias de cada versión y distribución.

En particular, las tareas de administración del sistema y de la red suelen presentar particularidades diferentes, y a veces se tratan por separado (o por administradores diferentes). Aunque también pueden verse como dos caras del mismo trabajo, con el sistema propiamente dicho (máquina y software) por un lado, y el ambiente donde el sistema (el entorno de red) convive, por el otro.

Normalmente, por *administración de la red* se entiende la gestión del sistema como parte de la red, y hace referencia a los servicios o dispositivos cercanos necesarios para que la máquina funcione en un entorno de red; no cubre dispositivos de red como *switches*, *bridges* o *hubs* u otros dispositivos de red, pero unos conocimientos básicos son imprescindibles para facilitar las tareas de administración.

En este curso cubriremos primero aquellos aspectos locales del propio sistema, y en una segunda parte veremos las tareas de administración de red y sus servicios.

Ya hemos comentado el problema de determinar qué es exactamente un administrador de sistemas, ya que en el mercado laboral infor-

ANOTACIONES

mático no está demasiado claro. Era común pedir administradores de sistemas según categorías (establecidas en las empresas) de programador o ingenieros de software, las cuales no se adecuan correctamente.

Un programador es básicamente un productor de código; en este caso, un administrador obtendría poca producción, ya que en algunas tareas puede ser necesario, pero en otras no. Normalmente, será deseable que el administrador posea más o menos conocimientos dependiendo de la categoría laboral:

a) Alguna carrera o diplomatura universitaria. Preferentemente en informática, o en algún campo directamente relacionado con la empresa u organización.

> **Nota**
>
> El perfil del administrador suele incluir estudios informáticos o afines a la organización junto con experiencia demostrada en el campo y conocimientos amplios de sistemas heterogéneos y tecnologías de red.

b) Suele pedirse de 1 a 3 años de experiencia como administrador (a no ser que el puesto sea para ayudante de uno ya existente). La experiencia también puede ampliarse de 3 a 5 años.

c) Familiaridad o conocimientos amplios de entornos de red y servicios. Protocolos TCP/IP, servicios de ftp, telnet, ssh, http, nfs, nis, ldap, etc.

d) Conocimientos de lenguajes de *script* para prototipado de herramientas o automatización rápida de tareas (por ejemplo, shell *scripts*, Perl, tcl, Python, etc.) y experiencia en programación de un amplio rango de lenguajes (C, C++, Java, Asm, etc.).

e) Puede pedirse experiencia en desarrollo de aplicaciones grandes en cualquiera de estos lenguajes.

f) Conocimientos amplios de mercado informático, tanto de hardware como de software, en el caso que haya que evaluar compras de material o montar nuevos sistemas o instalaciones completas.

g) Experiencia en más de una versión de UNIX (o sistemas GNU/Linux), como Solaris, AIX, AT&T SystemV, BSD, etc.

ANOTACIONES

h) Experiencia en sistemas operativos no UNIX, sistemas complementarios que pueden encontrarse en la organización: msdos, Windows 9x/NT/2000/XP, Mac Os, VMS, sistemas IBM, etc.

i) Sólidos conocimientos del diseño e implementación de UNIX, mecanismos de páginas, intercambio, comunicación interproceso, controladores, etc,... por ejemplo, si las tareas de administración incluyen optimización de sistemas (*tuning*).

j) Conocimientos y experiencia en seguridad informática: construcción de cortafuegos (*firewalls*), sistemas de autentificación, aplicaciones de criptografía, seguridad del sistema de ficheros, herramientas de seguimiento de seguridad, etc.

k) Experiencia en bases de datos, conocimientos de SQL, etc.

l) Instalación y reparación de hardware y/o cableados de red y dispositivos.

1.5. Tareas del administrador

Según hemos descrito, podríamos separar las tareas de un administrador GNU/Linux (o UNIX en general) [Lev02] en dos partes principales: administración del sistema y administración de red. En los siguientes puntos mostramos de forma resumida en qué consisten en general estas tareas en los sistemas GNU/LINUX (o UNIX); la mayor parte del contenido se va a tratar con cierto detalle en este manual del curso; otra parte, por cuestiones de espacio o complejidad, se explicará superficialmente o no se tratará.

Las tareas de administración engloban una serie de conocimientos y técnicas de los cuales en este curso sólo podemos ver la "punta del *iceberg*"); en todo caso, en la bibliografía adjunta a cada unidad se aportarán referencias para ampliar dichos temas. Como se verá, hay una amplia bibliografía para casi cualquier punto que se trate.

Las tareas de administración del sistema se podrían resumir, por una parte, en la administración local del sistema, y por otra, en la administración de red.

ANOTACIONES

Tareas de administración local del sistema
(sin un orden concreto)

- **Arranque y apagado del sistema**: cualquier sistema basado en UNIX tiene unos sistemas de arranque y apagado valorables, de manera que podemos configurar qué servicios ofrecemos en el arranque de la máquina y cuándo hay que pararlos, o programar el apagado del sistema para su mantenimiento.

- **Gestión de usuarios y grupos**: dar cabida a los usuarios es una de las principales tareas de cualquier administrador. Habrá que decidir qué usuarios podrán acceder al sistema, de qué forma y bajo qué permisos; y establecer comunidades mediante los grupos. Un caso particular será el de los usuarios de sistema, pseudousuarios dedicados a tareas del sistema.

- **Gestión de recursos del sistema**: qué ofrecemos, cómo lo ofrecemos y a quién damos acceso.

- **Gestión de los sistemas de ficheros**: el ordenador puede disponer de diferentes recursos de almacenamiento de datos y dispositivos (disquetes, discos duros, ópticos, etc.) con diferentes sistemas de acceso a los ficheros. Pueden ser permanentes o extraíbles o temporales, con lo cual habrá que modelar y gestionar los procesos de montaje y desmontaje de los sistemas de ficheros que ofrezcan los discos o dispositivos afines.

- **Cuotas del sistema**: cualquier recurso que vaya a ser compartido tiene que ser administrado, y según la cantidad de usuarios, habrá que establecer un sistema de cuotas para evitar el abuso de los recursos por parte de los usuarios o establecer clases (o grupos) de usuarios diferenciados por mayor o menor uso de recursos. Suelen ser habituales sistemas de cuotas de espacio de disco, o de impresión, o de uso de CPU (tiempo de computación usado).

- **Seguridad del sistema**: seguridad local, sobre protecciones a los recursos frente a usos indebidos o accesos no permitidos a datos del sistema o de otros usuarios o grupos.

ANOTACIONES

- ***Backup* y restauración del sistema**: es necesario establecer políticas periódicas (según importancia de los datos), de copias de seguridad de los sistemas. Hay que establecer periodos de copia que permitan salvaguardar nuestros datos, de fallos del sistema (o factores externos) que puedan provocar pérdidas o corrupción de datos.

- **Automatización de tareas rutinarias**: muchas de las tareas rutinarias de la administración o del uso habitual de la máquina pueden ser fácilmente automatizables, ya debido a su simplicidad (y por lo tanto, a la facilidad de repetirlas), como a su temporización, que hace que tengan que ser repetidas en periodos concretos. Estas automatizaciones suelen hacerse, bien mediante programación por lenguajes interpretados de tipo *script* (*shells*, Perl, etc.), como por la inclusión en sistemas de temporización (*crontab*, *at*, ...).

- **Gestión de impresión y colas**: los sistemas UNIX pueden utilizarse como sistemas de impresión para controlar una o más impresoras conectadas al sistema, así como gestionar las colas de trabajo que los usuarios o aplicaciones puedan enviar a las mismas.

- **Gestión de módems y terminales**. Estos dispositivos suelen ser habituales en entornos no conectados a red local ni a banda ancha:

– Los **módems** permiten una conexión a la red por medio de un intermediario (el ISP o proveedor de acceso), o bien la posibilidad de conectar a nuestro sistema desde el exterior por acceso telefónico desde cualquier punto de la red telefónica.

– En el caso de los **terminales**, antes de la introducción de las redes solía ser habitual que la máquina UNIX fuese el elemento central de cómputo, con una serie de terminales "tontos", que únicamente se dedicaban a visualizar la información o a permitir la entrada de información por medio de teclados externos; solía tratarse de terminales de tipo serie o paralelo. Hoy en día, todavía suelen ser habituales en entornos industriales, y en nuestro sistema GNU/Linux de escritorio tenemos una cosa particular, que son los terminales de texto "virtuales", a los que se accede mediante las teclas Alt+Fxx.

ANOTACIONES

- **_Accounting_ (o _log_) de sistema**: para poder verificar el funcionamiento correcto de nuestro sistema, es necesario llevar políticas de *log* que nos puedan informar de los posibles fallos del sistema o del rendimiento que se obtiene de una aplicación, servicio o recurso hardware. O bien permitir resumir los recursos gastados, los usos realizados o la productividad del sistema en forma de informe.

- ***System performance tunning***: técnicas de optimización del sistema para un fin dado. Suele ser habitual que un sistema esté pensado para una tarea concreta y que podamos verificar su funcionamiento adecuado (por ejemplo, mediante *logs*), para examinar sus parámetros y adecuarlos a las prestaciones que se esperan.

- **Personalización del sistema**: reconfiguración del *kernel*. Los *kernels*, por ejemplo en GNU/Linux, son altamente personalizables, según las características que queramos incluir y el tipo de dispositivos que tengamos o esperemos tener en nuestra máquina, así como los parámetros que afecten al rendimiento del sistema o que consigan las aplicaciones.

Tareas de administración de red

- **Interfaz de red y conectividad**: el tipo de interfaz de red que utilizamos, ya sea el acceso a una red local, la conexión a una red mayor, o conexiones del tipo banda ancha con tecnologías DSL o RDSI. Además, el tipo de conectividades que vamos a tener, en forma de servicios o peticiones.

- ***Routing* de datos**: los datos que circularán, de dónde o hacia dónde se dirigirán, dependiendo de los dispositivos de red disponibles y de las funciones de la máquina en red; posiblemente, será necesario redirigir el trafico desde/hacia uno o más sitios.

- **Seguridad de red**: una red, sobre todo si es abierta (como Internet) a cualquier punto exterior, es una posible fuente de ataques y, por lo tanto, puede comprometer la seguridad de nuestros sistemas o los datos de nuestros usuarios. Hay que protegerse, detectar e impedir posibles ataques con una política de seguridad clara y eficaz.

ANOTACIONES

- **Servicios de nombres**: en una red hay infinidad de recursos disponibles. Los servicios de nombres nos permiten nombrar objetos (como máquinas y servicios) para poderlos localizar. Con servicios como el DNS, DHCP, LDAP, etc., se nos permitirá localizar servicios o equipos *a posteriori*...

- **NIS (*Network Information Service*)**: las grandes organizaciones han de tener mecanismos para poder organizar de forma efectiva los recursos y el acceso a ellos. Las formas UNIX estándar, como los *logins* de usuarios con control por *passwords* locales, son efectivos con pocas máquinas y usuarios, pero cuando tenemos grandes organizaciones, con estructuras jerárquicas, usuarios que pueden acceder a múltiples recursos de forma unificada o separada por diferentes permisos,... los métodos UNIX sencillos se muestran claramente insuficientes o imposibles. Entonces se necesitan sistemas más eficaces para controlar toda esta estructura. Servicios com NIS, NIS+, LDAP nos permiten organizar de forma efectiva toda esta complejidad.

- **NFS (*Network Fylesystems*)**: a menudo, en las estructuras de sistemas en red es necesario compartir informaciones (como los propios ficheros) por parte de todos o algunos de los usuarios. O sencillamente, debido a la distribución física de los usuarios, es necesario un acceso a los ficheros desde cualquier punto de la red. Los sistemas de ficheros por red (como NFS) permiten un acceso transparente a los ficheros, independientemente de nuestra situación en la red.

- ***UNIX remote commands***: UNIX dispone de comandos transparentes a la red, en el sentido de que, independientemente de la conexión física, es posible ejecutar comandos que muevan información por la red o permitan acceso a algunos servicios de las máquinas. Los comandos suelen tener una "r" delante, con el sentido de 'remoto', por ejemplo: *rcp*, *rlogin*, *rsh*, *rexec*, etc., que permiten las funcionalidades indicadas de forma remota en la red.

- **Aplicaciones de red**: aplicaciones de conexión a servicios de red, como telnet (acceso interactivo), ftp (transmisión de ficheros), en forma de aplicación cliente que se conecta a un servicio servido

ANOTACIONES

desde otra máquina. O bien que nosotros mismos podemos servir con el servidor adecuado: servidor de telnet, servidor ftp, servidor web, etc.

- **Impresión remota**: acceso a servidores de impresión remotos, ya sea directamente a impresoras remotas o bien a otras máquinas que ofrecen sus impresoras locales. Impresión en red de forma transparente al usuario o aplicación.

- **Correo electrónico**: uno de los primeros servicios proporcionados por las máquinas UNIX es el servidor de correo, que permite, ya sea el almacenamiento de correo o un punto de retransmisión de correo hacia otros servidores, si no iba dirigido a usuarios propios de su sistema. Para el caso web, también de forma parecida, un sistema UNIX con el servidor web adecuado ofrece una plataforma excelente para web. UNIX tiene la mayor cuota de mercado en cuanto a servidores de correo y web, y es uno de los principales mercados, donde tiene una posición dominante. Los sistemas GNU/Linux ofrecen soluciones de código abierto para correo y web y conforman uno de sus principales usos.

- **X Window**: un caso particular de interconexión es el sistema gráfico de los sistemas GNU/Linux (y la mayor parte de UNIX), X Window. Este sistema permite una transparencia total de red y funciona bajo modelos cliente servidor; permite que el procesamiento de una aplicación esté desligado de la visualización y de la interacción por medio de dispositivos de entrada, por lo que éstos se sitúan en cualquier parte de la red. Por ejemplo, podemos estar ejecutando una determinada aplicación en una máquina UNIX cuando desde otra visualizamos en pantalla los resultados gráficos, y entramos datos con el teclado y ratón locales de forma remota. Es más, el cliente, llamado cliente X, es tan sólo un componente software que puede ser portado a otros sistemas operativos, permitiendo ejecutar aplicaciones en una máquina UNIX y visualizarlas en cualquier otro sistema. Un caso particular son los llamados terminales X, que son básicamente una especie de terminales "tontos" gráficos que sólo permiten visualizar o interactuar (por teclado y ratón) con una aplicación en ejecución remota.

ANOTACIONES

1.6. Distribuciones de GNU/Linux

Al hablar de los orígenes de los sistemas GNU/Linux, hemos comprobado que no había un único sistema operativo claramente definido. Por una parte, hay tres elementos software principales que componen un sistema GNU/Linux:

1) El *kernel* Linux: como vimos, el *kernel* es tan sólo la pieza central del sistema. Pero sin las aplicaciones de utilidad, *shells*, compiladores, editores, etc. no podríamos tener un sistema entero.

2) Las aplicaciones GNU: en el desarrollo de Linux, éste se vio complementado con el software de la FSF existente del proyecto GNU, que le aportó editores (como *emacs*), compilador (gcc) y utilidades diversas.

3) Software de terceros: normalmente de tipo de código abierto en su mayor parte. Todo sistema GNU/Linux se integra además con software de terceros que permite añadir una serie de aplicaciones de amplio uso, ya sea el propio sistema gráfico de X Windows, servidores como el de Apache para web, navegadores, etc. Asimismo, puede ser habitual incluir algún software propietario, dependiendo del carácter libre que en mayor o menor grado quieran disponer los creadores de la distribución.

Al ser la mayoría del software de tipo de código abierto o libre, ya sea el *kernel*, software GNU o de terceros, normalmente hay una evolución más o menos rápida de versiones, ya sea por medio de corrección de errores o nuevas prestaciones. Esto obliga a que en el caso de querer crear un sistema GNU/Linux, tengamos que escoger qué software queremos instalar en el sistema, y qué versiones concretas de este software.

Nota

El mundo GNU/Linux no se limita a una empresa o comunidad particular, con lo que ofrece a cada uno la posibilidad de crear su propio sistema adaptado a sus necesidades.

ANOTACIONES

Normalmente, entre el conjunto de estas versiones siempre se encuentran algunas que son estables, y otras que están en desarrollo, en fases alfa o beta, que pueden tener errores o ser inestables, por lo que habrá que tener cuidado a la hora de crear un sistema GNU/Linux, con la elección de las versiones. Otro problema añadido es la selección de alternativas, el mundo de GNU/Linux es lo suficientemente rico para que haya más de una alternativa para un mismo producto de software. Hay que elegir entre las alternativas posibles, incorporar algunas o todas, si queremos ofrecer al usuario libertad para escoger su software.

Ejemplo

Un caso práctico lo forman los gestores de escritorio de X Window, en que, por ejemplo, se nos ofrecen dos entornos de escritorio diferentes como Gnome y KDE; los dos tienen características parecidas y aplicaciones semejantes o complementarias.

En el caso de un distribuidor de sistemas GNU/Linux, ya sea comercial o bien una organización sin beneficio propio, dicho distribuidor tiene como responsabilidad generar un sistema que funcione, seleccionando las mejores versiones y productos software que puedan conseguirse.

En este caso, una distribución GNU/Linux [Dis03] es una colección de software que forma un sistema operativo basado en el *kernel* Linux.

Un dato importante a tener en cuenta, y que causa más de una confusión, es que, como cada uno de los paquetes de software de la distribución tendrá su propia versión (independiente de la distribución en que esté ubicado), el número de distribución asignado no mantiene una relación con las versiones de los paquetes software.

El número de distribución sólo sirve para comparar las distribuciones que genera un mismo distribuidor, y no permite comparar entre otras distribuciones. Si queremos hacer comparaciones entre distribuciones, tendre-

ANOTACIONES

mos que examinar los paquetes software principales y sus versiones para poder determinar qué distribución aporta más novedades.

Ejemplo

Pongamos un ejemplo de algunas versiones actuales (las versiones que aparecen se refieren a finales del año 2003):

a) *Kernel* Linux: actualmente podemos encontrar distribuciones que ofrecen uno o más *kernels*, como los 2.2.x o 2.4.x o algún nuevo 2.6.x en versión beta.

b) XFree86: en el sistema gráfico X Window, en versión de código abierto, que podemos encontrar prácticamente en todos los sistemas GNU/Linux, se manejan versiones 4.1.x o 4.2.x o la beta de 4.3.x.

c) Gestor de ventanas o escritorio: podemos tener Gnome o KDE, o los dos; Gnome con versiones 1.4, 2.2, 2.3 o KDE 2, 3.0, 3.1.x.

Podríamos hacer una distribución que incluyese *kernel* 2.2, con XFree 4.2 y Gnome 2.2; o bien otra, por ejemplo, *kernel* 2.4, XFree 4.1, KDE 3. ¿Cuál es mejor?, es difícil compararlas, ya que suponen una mezcla de elementos, y, dependiendo de cómo se haga la mezcla, el producto saldrá mejor o peor. Normalmente, el distribuidor mantiene un compromiso entre la estabilidad del sistema y la novedad de las versiones incluidas.

En general, podría hacerse un mejor análisis de distribuciones a partir de los siguientes apartados, que habría que comprobar en cada una:

a) Versión del núcleo Linux: 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 normalmente implica mejoras en las prestaciones

ANOTACIONES

del núcleo: Y es par en los núcleos estables e impar en los desarrollos o pruebas. Y Z es la versión de construcción, que indica el número de la revisión de X.Y, en cuanto a parches o correcciones hechas. Los distribuidores no suelen incluir la ultima versión del núcleo, sino la que ellos hayan probado con más frecuencia y puedan verificar que es estable para el software que ellos incluyen.

b) Formato de empaquetado: es el mecanismo empleado para instalar y administrar el software de la distribución. Se suele conocer por el formato de los paquetes de software soportados. En este caso suelen estar los formatos RPM, DEB, tar.gz, mdk, aunque cada distribución suele utilizar varios formatos, suele tener uno por defecto. El software acostumbra a venir con sus archivos en un paquete que incluye información sobre su instalación y posibles dependencias con otros paquetes de software. El empaquetado es importante si se usa software de terceros que no venga con la distribución, ya que el software puede encontrarse sólo en algunos sistemas de paquetes, o incluso en uno sólo.

c) Estructura del sistema de archivos: la estructura del sistema de archivos principal (/) nos indica dónde podemos encontrar nuestros archivos (o los propios del sistema) en el *filesystem*. En GNU/Linux y UNIX hay algunos estándares de colocación de los archivos (como veremos en la unidad de herramientas), como por ejemplo el FHS (*Filesystem Hierarchy Standard*). Así, si tenemos una idea del estándar, sabremos dónde encontrar la mayor parte de los archivos; luego depende de que la distribución lo siga más o menos y de que nos avisen de los cambios que hayan hecho.

d) *Scripts* de arranque del sistema: los sistemas UNIX y Linux incorporan unos guiones de arranque (o *shell scripts*) que indican cómo debe arrancar la máquina y cual será el proceso (o fases) que se van a seguir, así como lo que deberá hacerse en cada paso. Para este arranque hay dos modelos, los de SysV o BSD (es una diferencia de las dos ramas de UNIX principales); y cada distribución podría escoger uno o otro. Aunque los dos sistemas tienen la misma funcionalidad, son diferentes en los detalles, y esto será importante en los temas de administración (lo veremos en la administración local). En nuestro caso, los sistemas analizados, tanto Red Hat como Debian, utilizan el sistema de SysV (será el que veremos en la unidad local), pero hay otras distribuciones como Slackware que utilizan el otro sistema BSD.

ANOTACIONES

e) Versiones de la biblioteca del sistema: todos los programas (o aplicaciones) que tenemos en el sistema dependen para su ejecución de un número (mayor o menor) de bibliotecas de sistema. Estas bibliotecas, normalmente de dos tipos, ya sean estáticas unidas al programa (archivos *libxxx.a*) o las dinámicas que se cargan en tiempo de ejecución (archivos *libxxx.so*), proporcionan gran cantidad de código de utilidad o de sistema que utilizarán las aplicaciones. La ejecución de una aplicación puede depender de la existencia de unas bibliotecas adecuadas y del número de versión concreto de estas bibliotecas (no es lo recomendable, pero puede pasar). Un caso bastante habitual es la biblioteca GNU C library, la biblioteca estándar de C, también conocida como *glibc*. Puede suceder que una aplicación nos pida que dispongamos de una versión concreta de la *glibc* para poder ejecutarse o compilarse. Es un caso bastante problemático, y por ello, uno de los parámetros que valoran la distribución es conocer qué versión de la *glibc* lleva. El problema aparece al intentar ejecutar o compilar un producto de software muy antiguo en una distribución moderna, o bien un producto de software muy nuevo en una distribución antigua.

El mayor cambio llegó al pasar a una *glibc* 2.0, en que había que recompilar todos los programas para poder ejecutarlos correctamente, y en las nuevas 2.1, 2.2 y 2.3 ha habido algunos cambios menores que podían afectar a alguna aplicación. En muchos casos, los paquetes de software comprueban si se tiene la versión correcta de la *glibc*, o en el mismo nombre mencionan la versión que hay que utilizar (ejemplo: *paquete-xxx-glibc2.rpm*).

f) Escritorio X Window: el sistema X Window es el estándar gráfico para Linux como visualización de escritorio. Fue desarrollado en el MIT en 1984 y prácticamente todos los UNIX tienen una versión del mismo. Linux trae una versión denominada XFree86. Normalmente, el X Window es una capa gráfica intermedia que confía a otra capa denominada **gestor de ventanas** la visualización de sus elementos. Además, podemos combinar el gestor de ventanas con utilidades y programas de aplicación variados para formar lo que se denomina un **entorno de escritorio**.

Linux tiene principalmente dos entornos de escritorio: Gnome y KDE. Cada uno tiene la particularidad de basarse en una biblioteca de componentes propios (o sea ventanas, botones, listas, etc.):

ANOTACIONES

gtk+ (en Gnome) y *Qt* (en KDE), que son las principales bibliotecas gráficas que se usan para programar aplicaciones en estos entornos. Pero además de estos entornos, hay muchos otros, gestores de ventanas o escritorios: XCFE, Motif, Enlightement, BlackIce, FVWM, etc., de modo que la posibilidad de elección es amplia. Además, cada uno de ellos permite cambiar la apariencia (*look & feel*) de ventanas y componentes al gusto del usuario, o incluso crearse el suyo propio.

g) Software de usuario: software añadido por el distribuidor, en su mayoría de tipo Open Source, para las tareas habituales (o no). Las distribuciones habituales son tan grandes, que pueden encontrarse de centenares a miles de estas aplicaciones (muchas de las distribuciones tienen de 1 a 4 CD de aplicaciones extra). Estas aplicaciones cubren casi todos los campos, desde el hogar hasta administrativos o científicos. Y en algunas distribuciones se añade software propietario de terceros (como, por ejemplo, alguna *suite* ofimática del tipo Office), software de servidor preparado por el distribuidor, como por ejemplo un servidor de correo, un servidor web seguro, etc.

Así es cómo cada distribuidor suele sacar diferentes versiones de su distribución, por ejemplo, a veces hay distinciones entre una versión personal, profesional o de tipo servidor.

El sistema GNU/Linux de fondo es el mismo, sólo hay diferencias (que se pagan) en el software añadido (en general, obra de la misma casa distribuidora). Por ejemplo, en servidores web o en servidores correo, ya sean propios, optimizados o mejorados. O bien la inclusión de mejores herramientas, desarrolladas por la distribuidora.

A menudo, este coste económico extra no tiene mucho sentido, ya que el software estándar es suficiente (con un poco de trabajo extra de administración); pero para las empresas puede ser interesante porque reduce tiempo de instalación y mantenimiento de los servidores, y además optimiza algunas aplicaciones y servidores críticos para la gestión informática de la empresa.

ANOTACIONES

1.6.1. Debian

El caso de Debian [Deb03b] es especial, en el sentido de que es una distribución guiada por una comunidad sin fines comerciales, aparte de mantener su distribución y promocionar el uso del software de código abierto y libre.

Debian es una distribución apoyada por una comunidad entusiasta de usuarios y desarrolladores propios, basada en el compromiso de la utilización de software libre.

Nota

Los documentos "Contrato social Debian" son consultables en: debian.org

El proyecto Debian se fundó en 1993 para crear la distribución Debian GNU/Linux. Desde entonces se ha vuelto bastante popular y rivaliza en uso con otras distribuciones comerciales como Red Hat o Mandrake. Por ser un proyecto comunitario, el desarrollo de esta distribución se rige por una serie de normas o políticas; existen unos documentos llamados "Contrato social Debian", que mencionan la filosofía del proyecto en su conjunto, y las políticas Debian, que especifican en detalle cómo se implementa su distribución.

La distribución Debian está bastante relacionada con los objetivos de la FSF y su proyecto de Software Libre GNU; por esta razón, incluyen siempre en su nombre: "Debian GNU/Linux"; además, su texto del contrato social ha servido como base de las definiciones de código abierto. En cuanto a las políticas, todo aquél que quiera participar en el proyecto de la distribución, tiene que seguirlas. Aunque no se sea un colaborador, estas políticas pueden ser interesantes porque explican cómo es la distribución Debian.

Figura 2.

Mencionamos también un aspecto práctico de cara a los usuarios finales: Debian ha sido siempre una distribución difícil. Suele ser la distribución que usan los *hackers* de Linux, en el buen sentido de los que destripan el *kernel*, aportan modificaciones, programadores de bajo nivel, los que desean estar a la última para probar software nuevo, los

ANOTACIONES

que quieren probar los desarrollos del *kernel* que todavía no han sido publicados, ... o sea, todo tipo de fauna de "locos" por GNU/Linux.

Las versiones anteriores de Debian se habían hecho famosas por su dificultad de instalación. La verdad es que no se hacía demasiado para que fuese fácil para los no expertos. Pero las cosas con el tiempo han mejorado. Ahora, la instalación, no sin ciertos conocimientos, puede hacerse guiada por menús (eso sí, textuales, a diferencia de otras comerciales que son absolutamente gráficos), y hay programas que facilitan la instalación de los paquetes. Pero aun así, los primeros intentos suelen ser un poco traumáticos.

Normalmente, suelen ser variantes (las llaman "sabores") de la distribución Debian. En este momento hay tres ramas de la distribución: la *stable*, la *testing* y la *unstable*. Y, como sus nombres indican, la ***stable*** es la que está destinada a entornos de producción (o usuarios que desean estabilidad), la ***testing*** ofrece software más nuevo que ha sido testado mínimamente (podríamos decir que es una especie de versión beta de Debian) y que pronto van a ser incluidos en la ***stable***. Y la ***unstable*** es la que presenta las últimas novedades de software, cuyos paquetes cambian en plazos muy cortos; en una semana, e incluso cada día pueden cambiar varios paquetes. Todas ellas son actualizables desde diversas fuentes (CD, ftp, web) por un sistema denominado APT que maneja los paquetes software DEB de Debian. Las tres distribuciones tienen nombres más comunes asignados:

- Woody (*stable*)
- Sarge (*testing*)
- Sid (*unstable*)

La versión previa *stable* se denominaba Potato (era la 2.2). La más actual (finales del 2003), es la Debian GNU/Linux Woody 3.0r1 (3.0 Release 1). Las versiones más extendidas son la Woody y la Sid, que son dos extremos. La Sid no está recomendada para entornos de trabajo diario, porque puede traer características a medias que aún se están probando y pueden fallar (aunque no es habitual); es la distribución que suelen usar los *hackers* Linux. Además, esta versión cambia a diario; suele ser normal, si se quiere actualizar, que haya de 10 a 20 paquetes de software nuevos por día.

ANOTACIONES

La Woody es quizás la mejor elección para el sistema de trabajo diario, se actualiza periódicamente para cubrir nuevo software o actualizaciones. Normalmente, no dispone del último software, y éste no se incluye hasta que la comunidad lo haya verificado.

Vamos a comentar brevemente algunas características de esta distribución (las versiones son las que se encuentran por defecto en la Woody y en Sid a día de hoy):

a) La distribución actual consta de 78 CD de la Woody 3.0r1. Normalmente hay dos CD de instalación iniciales (se instala uno o el otro) y unos 34 CD de aplicaciones, junto con documentación y paquetes de código fuente, así como un CD de actualizaciones. Esta distribución puede comprarse (a precios simbólicos de soporte físico, y de esta manera contribuimos a mantener la distribución) o puede bajarse desde debian.org o sus *mirrors*.

b) La Sarge y Sid no suelen tener CD, sino que puede convertirse una Woody a *testing* o *unstable* mediante cambios de configuración del sistema de paquetes APT.

c) Núcleo Linux: utiliza núcleos de la serie 2.2.x por defecto (pero incluye 2.4.x de forma opcional). Últimos *kernels*: en *Woody* 2.2.22, y en Sid, 2.4.21. El enfoque de Debian en Woody es potenciar la estabilidad y dejar a los usuarios la opción de otro producto de software, si lo necesitan.

d) Formato de empaquetado: Debian soporta uno de los más potentes, el APT. Los paquetes de software tienen un formato denominado DEB. El APT es una herramienta de más alto nivel para manejarlos y mantener una base de datos de los instalables y los disponibles en el momento. Además, el sistema APT puede obtener software de varias fuentes, ya sea desde CD, ftp, web.

e) El sistema con APT es actualizable en cualquier momento, mediante lista de sitios de fuentes de software Debian (fuentes APT), que pueden ser los sitios Debian por defecto (debian.org) o de terceros. No estamos así ligados a una empresa única ni a ningún sistema de pago por suscripción.

f) Algunas de las versiones utilizadas (a diciembre de 2003) son: XFree86(4.1.0), *glibc* (2.2.5), Shell bash (2.05a), ... Debian Sid tiene XFree86(4.3.0), *glibc* (2.3.2), Shell bash (2.05b).

ANOTACIONES

g) En el escritorio acepta tanto Gnome 1.4 (por defecto) como KDE 2.2 (K Desktop Environment). Sid con Gnome 2.2.3, y KDE 3.1.3.

h) En cuanto a aplicaciones destacables, incluye la mayoría de las que solemos encontrar en las distribuciones de GNU/Linux; en Sid: editores como *emacs* 21.3 (y *xemacs*), compilador *gcc* (3.3.1) y herramientas, servidor web Apache (2.0.47), navegador web Mozilla (1.4), software Samba (3.0a) para compartir archivos con Windows, etc.

i) Incluye también suites ofimáticas como OpenOffice (1.0.3) y KOffice.

j) Debian incluye muchos ficheros de configuración personalizados para su distribución en directorios de /etc.

k) Debian usa por defecto el gestor de arranque *lilo*.

l) La configuración de la escucha de los servicios de red TCP/IP, que se realiza como en la mayoría de UNIX, con el servidor inetd (/etc/inetd.conf).

m) Hay muchas distribuciones más basadas en Debian, ya que el sistema puede adaptarse fácilmente para hacer distribuciones más pequeñas o más grandes, o con más o menos software adaptado a un segmento. Una de las más famosas es Knoppix, una distribución de un único CD, tipo LiveCD (de ejecución en CD), que es muy usada para demos de Linux, o para probarlo en una máquina sin hacer una instalación previa, ya que arranca y se ejecuta desde CD, aunque también puede instalarse en disco duro y convertirse en una Debian estándar. Linex es otra distribución que ha conseguido bastante fama por su desarrollo apoyado por una administración, la de la comunidad autónoma de Extremadura.

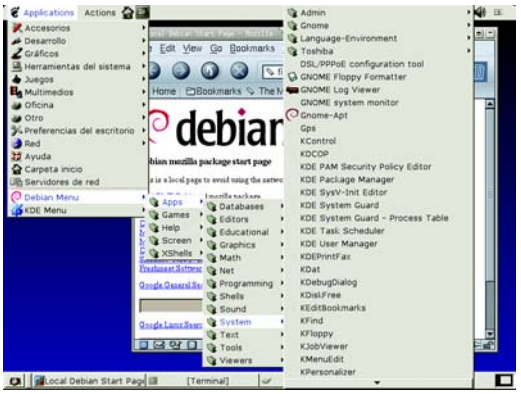
Nota

Debian puede usarse como base para otras distribuciones; por ejemplo, Knoppix es una distribución basada en Debian que puede ejecutarse desde el CD sin necesidad de instalarse en disco. Linex es una distribu-

ANOTACIONES

ción Debian adaptada por la administración de la Comunidad de Extremadura, en su proyecto de adoptar software de código abierto.

Figura 3. Entorno Debian Sid con Gnome 2.2



1.6.2. Red Hat

Red Hat inc. [Inc03a] es una de las principales firmas comerciales del mundo GNU/Linux, con una de las distribuciones con más éxito. Bob Young y Marc Ewing crearon Red Hat Inc en 1994. Estaban interesados en los modelos de software de código abierto y pensaron que sería una buena manera de hacer negocio. Su principal producto es su distribución Red Hat Linux (que abreviaremos como Red Hat), que está abierta a diferentes segmentos de mercado, tanto al usuario individual (versiones personal y profesional), como a las medianas o grandes empresas (con su versión Enterprise y sus diferentes subversiones).

Red Hat Linux es la principal distribución comercial de Linux, orientada tanto a mercado personal de escritorio como a servidores de gama alta. Además, Red Hat Inc es una de las empresas que más colaboran con el desarrollo de Linux, ya que varios miembros importantes de la comunidad trabajan para ella.

ANOTACIONES

Figura 4.

Aunque trabajan con un modelo de código abierto, se trata de una empresa, y por lo tanto sus fines son comerciales, por ello suelen añadir a su distribución básica valores por medio de contratos de soporte, suscripciones de actualización y otros métodos. En el caso empresarial, añaden software personalizado (o propio), para hacer que se adecue más el rendimiento a los fines de la empresa, ya sea por servidores optimizados o por software de utilidad propio de Red Hat.

Normalmente, Red Hat presenta una cierta visión conservadora de los elementos software que añade a su distribución, ya que su principal mercado de destino es el empresarial, e intenta hacer su distribución lo más estable posible, a pesar de que no cuenta con las últimas versiones. Lo que sí hace como valor añadido es depurar extensamente el *kernel* de Linux con su distribución, y genera correcciones y parches para mejorar su estabilidad. A veces, puede llegar a deshabilitar alguna funcionalidad (*drivers*) del *kernel*, si considera que éstos no son lo suficientemente estables. También ofrece muchas utilidades en el entorno gráfico y programas gráficos propios, incluidas unas cuantas herramientas de administración; en cuanto a los entornos gráficos, utiliza tanto Gnome (por defecto) como KDE, pero mediante un entorno modificado propio denominado BlueCurve, que hace que los dos escritorios sean prácticamente iguales (ventanas, menús, etc.).

La versión que utilizaremos en este manual será la Red Hat Linux 9 (también llamada *shrike*), la denominaremos simplemente como Red Hat. Esta versión en su mayor parte es bastante parecida a las versiones 7.x, 8.x, y las 9.x que salgan *a posteriori*, con lo cual, la mayoría de comentarios serían aplicables a las diferentes versiones, cuando comentemos versiones existentes de software se referirán al software presente en la versión 9.0 de la distribución, en otras puede haber ligeros cambios de versión.

ANOTACIONES

Nota

Ver:
http://fedora.redhat.com

En el momento de escritura de este manual, Red Hat ha anunciado que abandona la distribución del Red Hat Linux, en sus versiones de escritorio, centrando sus negocios en las versiones empresariales (Red Hat Linux Enterprise WS, ES, y AS). Por otra parte, cede su distribución Red Hat Linux 9 a la comunidad, para que a partir de ahora se desarrolle sin su intervención directa, ni soporte por su parte. A partir de ahora el proyecto Fedora será el encargado de continuar la versión *desktop* libre. Para la versión de Red Hat Linux 9 sólo se ofrece mantenimiento y soporte de erratas hasta abril del 2004, así, los clientes de Red Hat tendrán que decidir si continúan usando el proyecto libre (Fedora) o desean migrar a una versión empresarial de Red Hat.

A finales del 2003, nos encontramos ya con una versión de Fedora disponible, denominada Fedora Core 1, integrada por 3 CD, y que puede actualizar diferentes versiones Red Hat antiguas como las 7.x, 8.x y sobre la 9.

Vamos a comentar brevemente algunas características de esta distribución (las versiones son las que se encuentran por defecto en la Red Hat 9, actualizaciones posteriores pueden cambiar estos números):

a) La distribución actual consiste en 6 CD, de los cuales tres son básicos, junto con el primero, *bootable*, para su instalación. Existen también los tres CD extras que contienen documentación (manuales de Red Hat en diferentes idiomas) y código fuente de la mayoría del software instalado con la distribución.

b) Núcleo Linux: utiliza núcleos de la serie 2.4.x (la 9 viene con un 2.4.20), que puede irse actualizando con el servicio de suscripción Red Hat Network de la compañía (ver unidad del *kernel*), en principio de pago, pero puede obtenerse una cuenta "demo". Normalmente, Red Hat somete el *kernel* a muchas pruebas y crea parches para solucionar problemas, que normalmente también son integrados en la versión de la comunidad Linux, ya que bastantes de los colaboradores importantes de Linux trabajan para Red Hat.

c) Formato de empaquetado: Red Hat distribuye su software mediante el sistema de paquetes RPM (*Red hat Package Manager*), los cuales se gestionan mediante el comando *rpm* (lo comentaremos en la unidad de administración local). RPM es uno de los mejores sistemas de

ANOTACIONES

empaquetado existentes (bueno, el de Debian es quizás más potente), y algunos UNIX propietarios lo están incluyendo. Básicamente el sistema RPM mantiene una pequeña base de datos con los paquetes instalados, y verifica que el paquete que se va instalar con el comando *rpm*, no esté ya instalado, o entre en conflicto con algún otro paquete de software o por contra falte algún paquete software o versión de éste, necesaria para la instalación. El paquete RPM es básicamente un conjunto de ficheros comprimidos junto con información de sus dependencias o del software que necesita.

d) En cuanto al arranque, utiliza *scripts* de tipo *SystemV* (que veremos en la unidad de administración local).

e) Algunas de las versiones utilizadas son: XFree86 (4.3.0), glibc (2.3.2), Shell bash (2.05b), etc.

f) En el escritorio acepta tanto Gnome 2.2 (escritorio por defecto) o KDE 3.1 (K *Desktop* Environment) de forma opcional.

g) En cuanto a aplicaciones destacables, incluye la mayoría de las que solemos encontrar en la mayoría de distribuciones de GNU/Linux: editores como *emacs* 21.3 (y *xemacs*), compilador *gcc* (3.2) y herramientas, servidor web Apache (2.0.40), navegador web Mozilla (1.2.1), software Samba (2.2.7a) para compartir archivos con Windows, etc.

h) Incluye también suites ofimáticas como OpenOffice (1.0.2) y KOffice.

i) El software adicional puede obtenerse del servicio Red Hat Network con la herramienta *update* incluida, o bien por Internet mediante paquetes RPM pensados para la distribución.

j) Red Hat usa (por defecto) el cargador de arranque Grub para arrancar la máquina, a diferencia de la mayoría de distribuciones que usan *lilo*.

k) La configuración de escucha de los servicios de red TCP/IP, que se lleva a cabo en la mayoría de UNIX, con el servidor inetd (/etc/inetd.conf), en Red Hat ha sido substituido por xinetd, que tiene una configuración más modular (directorio /etc/xinetd.d).

l) Dispone en arranque de un programa denominado Kudzu que se encarga de verificar cambios de hardware y detectar el hardware nuevo instalado.

ANOTACIONES

m) Hay varias distribuciones más basadas en Red Hat que siguen muchas de sus características, a destacar Mandrake: una distribución francesa, que en su origen se basó en Red Hat y que sigue en los primeros puestos junto con Red Hat en las preferencias de los usuarios (sobre todo en trabajo de escritorio). Mandrake desarrolla software propio y multitud de asistentes para ayudar a la instalación y administración de las tareas más comunes; Caldera: de SCO, hoy desaparecida, ya que SCO se unió al UnitedLinux, un proyecto de crear un Linux para el mundo empresarial, antes de litigar contra IBM y atacar la comunidad Linux; y TurboLinux: distribución orientada al mercado asiático, china y Japón, y que suele trabajar con servidores en forma de *cluster* (agrupación de servidores trabajando en paralelo).

Figura 5. Escritorio Red Hat 9 con Gnome (entorno modificado BlueCurve)

Con respecto a la nueva distribución Fedora Core 1 presenta sólo unas pocas diferencias respecto a Red Hat 9, entre las que cabe destacar:

a) Es una distribución creada por la comunidad de programadores y usuarios basada en desarrollo que no cuenta con soporte ni de actualizaciones ni de mantenimiento por parte del fabricante. Este aspecto pasa a depender de la comunidad, de forma semejante al caso de la distribución Debian GNU/Linux.

b) Las versiones se van a producir con bastante rapidez, se esperan nuevas versiones de la distribución aproximadamente cada cuatro meses.

ANOTACIONES

c) Para la gestión de paquetes, también utiliza el sistema de paquetes RPM (heredado de Red Hat 9). Respecto al proceso de la actualización de los paquetes de la distribución o a la instalación de otros nuevos, pueden obtenerse por diferentes herramientas, ya sea vía *update* (igual que en Red Hat 9) con los canales de actualización de Fedora, o bien por los nuevos sistemas de actualización Yum y Apt (heredado de Debian, pero que trabaja con ficheros RPM).

d) Otras cuestiones más técnicas (algunas de las cuales veremos en los siguientes capítulos) pueden encontrarse en las notas de la versión de Fedora Core.

Nota

Ver: http://fedora.redhat.com/docs/releasenotes.

1.7. Qué veremos...

Una vez que hemos visto esta introducción "filosófica" al mundo del código abierto y la historia de los sistemas UNIX y GNU/Linux, así como definir cuáles serán las tareas de la figura del administrador del sistema, pasaremos a tratar las diferentes tareas típicas que nos encontraremos durante la administración de sistemas GNU/Linux.

A continuación, nos vamos a introducir por las diferentes áreas que implica la administración de sistemas GNU/Linux. En cada área, intentaremos examinar un mínimo de fundamentos teóricos que nos permitan explicar las tareas que hay que realizar y que nos permitan entender el funcionamiento de las herramientas que usaremos. Cada tema vendrá acompañado de algún "taller", donde veremos una pequeña sesión de trabajo de una tarea o el uso de algunas herramientas. Sólo recordaremos que, como dijimos en la presentación, el tema de la administración es "enorme" y cualquier intento de abarcarlo completamente (como éste) tiene que fallar por las dimensiones limitadas; por ello, en cada tema encontraréis abundante bibliografía (en forma libros, sitios web, howto's, etc.), donde ampliar la "pequeña" introducción que habremos hecho del tema.

Los temas que veremos son los siguientes:

- En el apartado que trata de **migración**, obtendremos una perspectiva del tipo de sistemas informáticos que se están utilizando y

ANOTACIONES

en qué ambientes de trabajo se usan; veremos, asimismo, cómo los sistemas GNU/Linux se adaptan mejor o peor a cada uno de ellos, y plantearemos una primera disyuntiva a la hora de introducir un sistema GNU/Linux: ¿cambiamos el sistema que teníamos o lo hacemos por etapas, coexistiendo ambos?

- En el apartado de **herramientas** estudiaremos (básicamente) aquel conjunto de útiles con el que el administrador tendrá que "vivir" (y sufrir) a diario, y que podrían formar la "caja de herramientas" del administrador. Hablaremos de los estándares GNU/Linux, que nos permitirán conocer aspectos comunes a todas las distribuciones Linux, o sea, lo que esperamos que aparezca en cualquier sistema. Otra herramienta básica serán los editores simples (o no tan simples): comandos básicos para conocer el estado del sistema u obtener información filtrada según nos interese; la programación de guiones de comandos (o *shell scripts*) que nos permitirán automatizar tareas; características de los lenguajes que podemos encontrarnos en las aplicaciones o en herramientas de administración; procesos básicos de compilación de programas a partir de los códigos fuente; herramientas de gestión del software instalado, al mismo tiempo que comentaremos la disyuntiva de uso de herramientas gráficas o las de línea de comandos.

- En el apartado dedicado al *kernel*, veremos el *kernel* Linux, y cómo, mediante el proceso de personalización, podemos adaptarlo mejor al hardware o a los servicios que queramos proporcionar desde nuestro sistema.

- En el apartado dedicado a **local**, trataremos aquellos aspectos de administración que podríamos considerar "locales" a nuestro sistema. Estos aspectos pueden conformar la mayor parte de las tareas típicas del administrador a la hora de manejar elementos tales como usuarios, impresoras, discos, software, procesos, etc.

- En el apartado dedicado a **red**, nos dedicaremos a todas aquellas tareas de administración que engloben nuestro sistema con su "vecindario" en la red, sea cual sea su tipo, así como a ver los diferentes tipos de conectividad que podemos tener con los sistemas vecinos, así como los servicios que les podemos ofrecer o recibir de ellos.

ANOTACIONES

- En el apartado dedicado a **servidores**, veremos algunas configuraciones típicas de los servidores habituales, que podemos encontrar en un sistema GNU/Linux.

- En el apartado dedicado a **datos**, nos dedicaremos a uno de los aspectos más importantes de hoy en día, a los mecanismos de almacenamiento y consulta de datos que nos pueden ofrecer los sistemas GNU/Linux, en concreto a los sistemas de base de datos, y los mecanismos de control de versiones.

- En el apartado dedicado a **seguridad**, atacaremos uno de los puntos más importantes de todo sistema GNU/Linux de hoy en día. La existencia de un "mundo" interconectado por Internet nos trae una serie de "peligros" importantes para el correcto funcionamiento de nuestros sistemas y plantea el tema de la fiabilidad, tanto de éstos, como de los datos que podamos recibir o proporcionar por la red. Con ello, tenemos que asegurar unos niveles mínimos de seguridad en nuestros sistemas, así como controlar y evitar los accesos indebidos o las manipulaciones de nuestros datos. Veremos qué tipos de ataques son los más frecuentes, qué políticas de seguridad cabría seguir y qué herramientas nos pueden ayudar para controlar nuestro nivel de seguridad.

- En el apartado dedicado a **optimización**, veremos cómo en los sistemas GNU/Linux debido a la gran cantidad de servidores y servicios ofrecidos, así como los diferentes ambientes para los que está pensado el sistema, suele haber muchos parámetros de funcionamiento que influyen en el rendimiento de las aplicaciones o los servicios ofrecidos. Podemos (o debemos) intentar extraer el máximo rendimiento, analizando las configuraciones del propio sistema y de los servidores, para adecuarlos a la calidad de servicio que queramos ofrecer a los clientes.

- En el apartado dedicado a ***clustering***, veremos algunas de las técnicas para proporcionar computación de altas prestaciones en los sistemas GNU/Linux, muy utilizados en áreas de computación científica, y cada vez más utilizadas por gran cantidad de industrias (farmacéutica, química, materiales, etc.), para el desarrollo y la investigación de nuevos productos. Así como la organización de varios sistema GNU/Linux en forma de *clusters*, para ampliar

ANOTACIONES

las prestaciones de los sistemas individuales, mediante la formación de grupos de sistemas que permitan escalar los servicios ofrecidos ante el aumento de demanda de los clientes.

1.8. Actividades para el lector

1) Leer el manifiesto Debian en http://www.debian.org/social/contract.

2) Documentarse sobre las diferentes distribuciones Debian: Knoppix, Linex. Aparte de los sitios de cada distribución, en la dirección http://www.distrowatch.com hay una buena guía de las distribuciones y su estado, así como el software que incluyen. En esta web, o bien en http://linuxiso.org, se pueden obtener las imágenes de los CD de la mayoría de las distribuciones.

1.9. Otras fuentes de referencia e información

- [LPD03a] The Linux Documentation Project (LDP), colección de Howtos, manuales y guías cubriendo cualquiera de los aspectos de GNU/Linux.

- [OSD03b] Comunidad de varios sitios web, noticias, desarrollos, proyectos, etc.

- [Sla03] Sitio de noticias comunidad Open Source y generales informática e Internet.

- [New03] [Bar03] Noticias Open Source.

- [Fre03] [Sou03] Listado de proyectos Open Source.

- [Dis03] Seguimiento de las distribuciones Linux y novedades de los paquetes software.

- [Lin03d] Imágenes de los CD de un gran número de distribuciones Linux.

ANOTACIONES

- [His03] [Bul03a] [Luc03] [LPD03c] Documentación general y comunidades de usuarios.

- [Lin03a] [Mag03] [Jou03] [Bul03b] Revistas Linux.

ANOTACIONES

2. Migración y coexistencia con sistemas no Linux

Una vez realizado un primer aprendizaje de los sistemas GNU/Linux, el siguiente paso es integrarlos en el entorno de trabajo como sistemas de producción. Según el sistema que esté en uso, podemos plantear, o bien una migración total a sistemas GNU/Linux, o bien una coexistencia mediante servicios compatibles.

Nota

La migración al entorno GNU/Linux puede hacerse de forma progresiva, sustituyendo servicios parcialmente, o bien sustituyendo todo por los equivalentes GNU/Linux del antiguo sistema.

En los entornos distribuidos actuales, el paradigma más presente es el de cliente/servidor. Cualquier tarea en el sistema global está gestionada por uno o más servidores dedicados, accediendo las aplicaciones o directamente los usuarios a los servicios prestados.

Respecto al ambiente de trabajo, ya sea desde el caso más simple como el usuario individual, o bien uno complejo como un entorno empresarial, cada entorno necesitará un conjunto de servicios que tendremos que seleccionar, adaptando luego las máquinas clientes y servidores, para que puedan acceder a éstos o proporcionar su uso.

Los servicios pueden englobar muchos aspectos diferentes, suelen estar presentes diversos tipos, ya sea para compartir recursos o información. Suelen ser habituales servidores de archivos, de impresión, de web, de nombres, correo, etc.

El administrador normalmente seleccionará un conjunto de servicios que deberán estar presentes en el ambiente de trabajo, dependiendo de las necesidades de los usuarios finales, y/o de la organización; y deberá configurar el soporte adecuado a la infraestructura, en forma de servidores que soporten la carga de trabajo esperada.

ANOTACIONES

2.1. Sistemas informáticos: ambientes

En el proceso de algunas instalaciones de distribuciones de GNU/Linux, podemos encontrarnos a menudo que nos preguntan por el tipo de ambiente, o tareas a las que va a estar dedicado nuestro sistema, esto permite muchas veces escoger un subconjunto de software que se nos instalará por defecto, por ser el más adecuado a la función prevista. Suele ser habitual que nos pregunten si el sistema será:

a) Estación de trabajo (*workstation*): este tipo de sistema normalmente incorpora algunas aplicaciones particulares que serán las más usadas. El sistema básicamente se dedica a la ejecución de estas aplicaciones y a un pequeño conjunto de servicios de red.

b) Servidor: básicamente se integran la mayoría de servicios de red o, en todo caso, algunos particulares.

c) Estación dedicada a cálculo: aplicaciones intensivas en cálculo, *renders*, aplicaciones científicas, gráficos CAD, etc.

d) Estación gráfica: escritorio con aplicaciones que necesitan de interactividad con el usuario en forma gráfica.

Normalmente podemos componer nuestro sistema GNU/Linux a partir de una o más de estas posibilidades.

Más en general, si tuviésemos que separar los ambientes de trabajo [Mor03] en que se puede utilizar un sistema GNU/Linux, podríamos identificar tres tipos principales de ambiente: **estación de trabajo** (*workstation*), **servidor** y **escritorio** (*desktop*).

También se podría incluir otro tipo de sistema, lo que llamaríamos un **dispositivo empotrado** (***embebbed***), o generalmente los que están más de moda hoy en día como un **sistema móvil** de pequeñas dimensiones, por ejemplo, un PDA. GNU/Linux ofrece asimismo soporte para estos dispositivos, con *kernels* reducidos y personalizados para ello.

Nota

Los sistemas GNU/Linux pueden dedicarse a funciones de servidor, estación de trabajo o escritorio.

ANOTACIONES

Ejemplo

Destacamos, por ejemplo, el trabajo realizado por la firma Sharp en sus modelos Zaurus, un PDA con Linux de altas prestaciones (existen cuatro o cinco modelos en el mercado). O también otras iniciativas Linux de tipo empotrado como los TPV (terminales punto de venta).

Respecto a los otros tres principales, veamos cómo se desarrolla cada uno de estos sistemas informáticos en un entorno GNU/Linux:

1) Un sistema de tipo ***workstation*** suele ser una máquina de alto rendimiento, utilizada para una tarea específica en lugar de para un conjunto general de tareas. La *workstation*, clásicamente, estaba compuesta de una máquina de altas prestaciones con hardware específico adecuado a la tarea que había que desarrollar; solía tratarse de una máquina Sun Sparc, IBM Risc o Silicon Graphics (entre otras) con sus variantes de UNIX comerciales. Estas máquinas de alto coste se orientaban a un segmento claro de aplicaciones, ya fuese el diseño gráfico 3D (caso Silicon o Sun) o bases de datos (IBM o Sun). Hoy en día, muchos de los actuales PC tienen un rendimiento comparable a estos sistemas (aunque no igual), y la frontera entre uno de estos sistemas y un PC no está ya tan clara, gracias a la existencia de GNU/Linux como alternativa a los sistemas UNIX comerciales.

2) Un sistema de tipo **servidor** tiene un objetivo concreto, que es ofrecer servicios a otras máquinas de la red: ofrece características o una funcionalidad clara al resto de máquinas. En sistemas informáticos pequeños (por ejemplo menor de 10 máquinas), no suele haber un sistema exclusivo de servidor, y suele estar compartido con otras funcionalidades, por ejemplo también como máquina de tipo escritorio. En sistemas medianos (unas pocas decenas de máquinas), suele haber una o más máquinas dedicadas a actuar de servidor, ya sea la máquina exclusiva que centra los servicios (correo, web, etc.) o un par de máquinas dedicadas a repartirse los servicios.

En sistemas grandes (un centenar o más de máquinas, incluso miles), por la capacidad de carga puede ser necesario que exista un buen grupo de servidores, dedicados normalmente cada uno de ellos a al-

ANOTACIONES

gún servicio en exclusiva, o incluso dedicar un conjunto de máquinas a exclusivamente a un servicio. Es más, si estos servicios se proporcionan –hacia dentro o hacia fuera de la organización–, mediante acceso por clientes directos o abierto a Internet, dependiendo de la capacidad de carga que tengamos que soportar, tendremos que recurrir a soluciones de tipo SMP (máquinas con varios procesadores) o de tipo *cluster* (agrupación de máquinas que se distribuyen la carga de un determinado servicio).

Los servicios que podemos necesitar de forma interna (o externa), podrían englobarse (entre otras) dentro de estas categorías de servicios:

a) **Aplicaciones**: el servidor dispone de ejecución de aplicaciones y como clientes sólo observamos la ejecución de éstas e interactuamos con ellas. Puede englobar servicios de terminales y ejecución de aplicaciones en web, por ejemplo.

b) **Ficheros**: se nos proporciona un espacio común y accesible desde cualquier punto de la red de donde almacenar/recuperar nuestros ficheros.

c) **Base de datos**: se centralizan datos que se van a consultar o producir por parte de las aplicaciones del sistema en red (o bien de otros servicios).

d) **Impresión**: se dispone de conjuntos de impresoras, donde se gestionan sus colas y los trabajos que se les envíen desde cualquier punto de la red.

e) **Correo electrónico**: se ofrecen servicios para recibir, enviar o reenviar correos procedentes o destinados tanto al interior como al exterior.

f) **Web**: servidor (o servidores) propios de la organización, de utilización interna o externa para los clientes.

g) **Información de red**: en organizaciones grandes es imprescindible poder localizar los servicios ofrecidos o recursos compartidos, o los mismos usuarios, se necesitan servicios que permitan esta localización y consulta de propiedades de cada tipo de objeto.

h) **Servicios de nombres**: se necesitan servicios que permitan nombrar y traducir los diversos nombres por los que se conoce a un mismo recurso.

ANOTACIONES

i) **Servicios de acceso remoto**: en caso de no disponer de acceso directo, debemos disponer de métodos alternativos que nos permitan interaccionar desde el exterior, que nos permitan acceder al sistema que queramos.

j) **Servicios de generación de nombres**: en el nombrado de máquinas, por ejemplo, puede darse una situación muy variable de número o que aquéllas no sean siempre las mismas. Debemos proporcionar métodos para identificarlas claramente.

k) **Servicios de acceso a Internet**: en muchas organizaciones no tiene por qué haber accesos directos, sino accesos por medio de pasarelas (*gateways*) o por intermediario (*proxys*).

l) **Servicios de filtrado**: medidas de seguridad para filtrar información incorrecta o que afecte a nuestra seguridad.

3) Un sistema de tipo ***desktop*** sería simplemente una máquina que se utiliza para las tareas informáticas rutinarias, de todos los días (por ejemplo el PC que tenemos en casa o en la oficina).

Ejemplo

Por ejemplo, podríamos poner las siguientes tareas como comunes (se incluyen algunos de los programas GNU/Linux más utilizados):

- Tareas ofimáticas: disponer de software clásico de una *suite* ofimática: procesador de texto, hoja de cálculo, presentaciones, alguna pequeña base de datos, etc. Podemos encontrar *suites* como OpenOffice (gratuita), StarOffice (de pago, producida por Sun), KOffice (de KDE), o varios programas como Gnumeric, AbiWord que formarían una *suite* para Gnome (denominada GnomeOffice).

- Navegación web: navegadores como Mozilla, Konqueror, Galeon, etc.

- Soporte hardware (dispositivos USB, de almacenamiento, ...). En GNU/Linux soportados por los controladores adecuados, normalmente proporcionados en el *kernel*, o bien por fabricantes. También hay herra-

ANOTACIONES

mientas de análisis de hardware nuevo, como *kudzu* (Red Hat) o *discover* (Debian). Media y entretenimiento (gráficos, procesamiento imágenes, fotografía digital, juegos y más). En GNU/Linux hay una cantidad enorme de estas aplicaciones, de calidad muy profesional: Gimp (retoque fotográfico), Sodipodi, Xine, Mplayer, gphoto, etc.

- Conectividad (acceso al escritorio de forma remota, acceso a otros sistemas). Este aspecto en GNU/Linux hay una cantidad enorme de herramientas, ya sea las propias TCP/IP como ftp, telnet, web, etc., como X Window, que tiene capacidades de escritorio remoto hacia cualquier máquina UNIX, rdesktop (para conectarse a escritorios Windows 2000/XP), o VNC (que permite conectarse a UNIX, Windows, Mac, etc.).

2.2. Servicios en GNU/Linux

Nota

GNU/Linux dispone de servidores adaptados para cualquier ambiente de trabajo.

Nota

http://www.gnome.org/gnomeoffice

Las categorías de los servicios que hemos comentado tienen equivalentes en servicios que podemos proporcionar desde nuestros sistemas GNU/Linux al resto de máquinas de la red (y de los que también podremos actuar como cliente):

a) Aplicaciones: GNU/Linux puede proporcionar servicios de terminales remotos, ya sea por conexión directa mediante interfases serie de terminales "tontos", que sirvan para visualizar o interactuar con las aplicaciones. Otra posibilidad es la conexión remota de modo textual, desde otra máquina, por medio de servicios TCP/IP como los *rlogin*, telnet, o de forma segura con *ssh*. GNU/Linux proporciona servidores para todos estos protocolos. En el caso de ejecutar aplicaciones gráficas, disponemos de soluciones mediante X Window de forma remota, cualquier cliente UNIX, Linux o Windows (u otros) que dispongan de un diente X Window puede visualizar la ejecución del entorno y sus aplicaciones. Asimismo, hay otras soluciones como VNC para el mismo problema. En cuanto al tema de aplicaciones vía web, GNU/Linux dispone del servidor Apache,

ANOTACIONES

y cualquiera de los múltiples sistemas de ejecución web están disponibles, ya sean Servlets (con Tomcat), JSP, Perl, PHP, xml, *webservices*, etc. Así como servidores de aplicaciones web como BEA Weblogic, IBM Websphere, JBoss (gratuito) que también se ejecutan sobre plataformas GNU/Linux.

b) Ficheros: pueden servirse ficheros de múltiples maneras, desde el acceso por ftp a los ficheros, como servirlos de forma transparente a otras máquinas UNIX y Linux con NFS, o bien actuar de cliente o servidor hacia máquinas Windows con Samba.

c) Base de datos: soporta una gran cantidad de bases de datos cliente/servidor de tipo relacional como MySQL, Postgres y varias comerciales como Oracle o IBM DB2 entre otras.

d) Impresión: puede servir impresoras locales o remotas, tanto a sistemas UNIX con protocolos TCP/IP, como a Windows mediante Samba.

e) Correo electrónico: ofrece tanto servicios para que los clientes obtengan correo en sus máquinas (servidores POP3 o IMAP), como agentes MTA (*Mail Transfer Agent*) para recuperar y retransmitir correo, como el servidor Sendmail (el estándar UNIX) u otro como Exim, y en el caso de envíos externos, el servicio de SMTP para el envío de mail externo.

f) Web: disponemos del servidor http Apache, ya sea en sus versiones 1.3.x o las nuevas 2.0.x. Además, podemos integrar servidores de aplicaciones web, como Tomcat para servir servlets, JSP...

g) Información de red: servicios como NIS, NIS+ o LDAP nos permiten centralizar la información de las máquinas, usuarios y recursos varios de nuestra red, facilitando la administración y los servicios a los usuarios, de manera que éstos no dependan de su situación en la red. O si nuestra organización tiene cierta estructura interna, estos servicios nos permiten modelarla dejando acceso a los recursos a quien los necesita.

h) Servicios de nombres: servicios como DNS para los nombres de las máquinas y su traducción desde IP o a IP, por medio de, por ejemplo, el servidor Bind (el DNS estándar UNIX).

i) Servicios de acceso remoto: ya sea para ejecutar aplicaciones o para obtener información remota de las máquinas. Los servidores

ANOTACIONES

podrían ser los que hemos comentado para aplicaciones: X Window, VNC, etc., y también los que permiten ejecutar algunos comandos remotos sin interactividad como *rexec*, *rsh*, *ssh*, etc.

j) Servicios de generación de nombres: servicios como DHCP permiten redes TCP/IP, una generación dinámica (o estática) de las direcciones IP que se disponen en función de las máquinas que las necesiten.

k) Servicios de acceso a Internet: en determinadas situaciones puede tenerse un único punto de salida a Internet (o varios). Estos puntos suelen actuar como *proxy*, ya que tienen el acceso y lo redirigen a los posibles accesos a Internet por parte de los clientes. También suelen actuar de caché de contenidos. En GNU/Linux podemos disponer, por ejemplo, del Squid. Dentro de esta categoría, también podría entrar la actuación de un sistema GNU/Linux de pasarela (*gateway*) o de *router*, ya sea para dirigir paquetes hacia otras redes o para buscar rutas de reenvío alternativas. También en el caso de pequeñas instalaciones como las de casa, podríamos incluir el acceso a Internet mediante módems por los servicios PPP.

l) Servicios de filtrado: una de las medidas de seguridad más utilizadas actualmente es la implantación de cortafuegos (o *firewalls*). Esto supone básicamente técnicas de filtrado de los paquetes entrantes o salientes, de los diferentes protocolos que estemos usando, para poner barreras a los no deseados. En GNU/Linux disponemos de mecanismos como ipchains e iptables (más moderno) para implementar los cortafuegos.

2.3. Tipologías de uso

Desde la perspectiva de los usuarios de los sistemas GNU/Linux, podríamos diferenciar:

Nota

Desde el punto de vista del usuario, GNU/Linux ofrece características válidas, desde al usuario personal hasta el usuario de una infraestructura de media o gran escala.

a) El **usuario individual** o **usuario doméstico**: normalmente, este tipo de usuario dispone de una o varias máquinas en su hogar, que serán compartidas o no. En general, en este ambiente, GNU/Linux se usaría para desarrollar un sistema de escritorio, con lo cual, será importante la parte gráfica: el escritorio de GNU/Linux.

Para este escritorio tenemos dos opciones principales, en forma de los entornos Gnome y KDE, los dos entornos constituyen opciones perfectamente válidas. Cualquiera de los dos entornos dispone de servicios de visualización y ejecución de las aplicaciones, así como de un amplio conjunto de aplicaciones propias básicas que nos permiten desarrollar todo tipo de tareas rutinarias. Los dos entornos ofrecen un escritorio visual con diferentes menús, barras de utilidad e iconos, así como navegadores de ficheros propios y aplicaciones de utilidad variadas. Cada entorno puede ejecutar sus aplicaciones y las del otro, aunque, del mismo modo que las aplicaciones, tienen mejor ejecución en su entorno propio por tener un aspecto visual más parecido al entorno para el que se diseñaron.

En cuanto a las aplicaciones para el usuario personal, incluiríamos las típicas del sistema de escritorio. En el caso de que el usuario dispusiera de una red en su casa, por ejemplo, un pequeño conjunto de ordenadores mediante una red Ethernet, podrían ser interesantes los servicios de compartir ficheros e impresoras entre las máquinas. Podrían ser necesarios servicios para compartir como NFS, si hay otras máquinas Linux; o bien Samba, si hay máquinas con Windows.

En el caso de tener una conexión a Internet por algún proveedor de acceso (ISP), según la forma de conexión utilizada, necesitaríamos controlar los dispositivos y los protocolos correspondientes:

- Conexión por módem: los módems telefónicos suelen utilizar el protocolo PPP de conexión con el proveedor. Tendríamos que habilitar este protocolo y configurar las cuentas que tengamos habilitadas en el proveedor. Un problema importante con Linux es el tema de los winModems, que ha traído de cabeza a más de uno. Estos módems (con excepciones) no están soportados, ya que no son reales, sino una simplificación, y la mayoría corren únicamente con Windows, por lo que hay que evitarlos (si no están soportados) y comprar módems reales.

- Conexión mediante un módem ADSL: el funcionamiento sería parecido, se podría utilizar el protocolo PPP u otro denominado EOPPP. Esto puede depender del fabricante del módem y del tipo de módem: Ethernet o USB.

ANOTACIONES

- Conexión por ADSL con *router*: la configuración es muy simple, debido a que en esta situación sólo hay que configurar la tarjeta de red Ethernet que acompaña al *router* ADSL.

Una vez la interfaz a Internet está conectada y configurada, el último punto es el tipo de servicios que necesitaremos. Si sólo queremos actuar como clientes en Internet, bastará con utilizar las herramientas cliente de los diferentes protocolos, ya sea ftp, telnet, el navegador web, el lector de correo o *news*, etc. Si además queremos ofrecer servicios hacia fuera –por ejemplo, publicar una web (servidor web) o permitir nuestro acceso externo a la máquina (servicios de *ssh*, telnet, ftp, X Window, VNC, etc.), en este caso, servidor– entonces, cabe recordar que esto será posible solamente si nuestro proveedor nos ofrece direcciones IP fijas para nuestra máquina. De otro modo, nuestra dirección IP cambiaría a cada conexión y la posibilidad de proporcionar un servicio se volvería muy difícil o imposible.

Otro servicio interesante seria la compartición de acceso a Internet entre las máquinas de que dispongamos.

b) **Usuario de media escala**: es un usuario de una organización de media escala, ya sea una pequeña empresa o un grupo de usuarios. Normalmente, este tipo de usuarios dispondrán de conectividad en red local (por ejemplo, una LAN) con algunas máquinas e impresoras conectadas. Y tendrá acceso directo a Internet, bien a través de algún *proxy* (punto o máquina destinada a la conexión externa), o bien habrá unas pocas máquinas conectadas físicamente a Internet. En general, en este ambiente, el trabajo suele ser en parte local y en parte compartido (ya sea recursos como las impresoras o aplicaciones comunes). Normalmente, necesitaremos sistemas de escritorio, por ejemplo, en una oficina podemos utilizar las aplicaciones ofimáticas junto con clientes Internet; y quizás también sistemas de tipo *workstation*; por ejemplo, en trabajos de ingeniería o científicos pueden utilizarse aplicaciones de CAD, de procesamiento de imágenes, aplicaciones de cálculo matemático intensivo, etc., y seguramente habrá algunas máquinas más potentes destinadas a estas tareas.

En este ambiente de uso, por lo común necesitaremos servicios de compartición de recursos como ficheros, impresoras, posiblemente aplica-

ANOTACIONES

ciones, etc. Por lo tanto, en un sistema GNU/Linux serán adecuados los servicios de NFS, servicios de impresión, Samba (si hay máquinas Windows con las que compartir ficheros o impresoras), así como también es posible que tengamos necesidad de entornos de bases de datos, algún servidor interno de web con aplicaciones compartidas, etc.

c) **Usuario de organización amplia**: este tipo de usuario es bastante parecido al anterior, y casi sólo se diferencia en el tamaño de la organización y en los recursos de los que puede disponer, que podrían llegar a ser muy altos, de modo que se necesitarían algunos recursos de sistemas de directorio de red de tipo NIS, NIS+ o LDAP para poder manejar la información de la organización y reflejar su estructura, así como, seguramente, disponer de grandes infraestructuras de servicios hacia los clientes externos, por lo general en forma de sitios web con aplicaciones diversas.

En este tipo de organizaciones se presentan niveles de heterogeneidad elevados, tanto en el hardware como en el software de los sistemas, y podríamos encontrar muchas arquitecturas y diferentes sistemas operativos, por lo que la tarea principal va a consistir en facilitar la compatibilidad de los datos vía bases de datos y formatos de documentos estándar y facilitar la interconectividad mediante protocolos, clientes y servidores estándar (normalmente con elementos TCP/IP).

2.4. Migrar o coexistir

A continuación vamos a plantear otro aspecto importante en el proceso de adopción de los sistemas GNU/Linux. Supongamos que somos nuevos en el manejo de este sistema; o, por el contrario, que somos experimentados y queremos adoptar uno o varios sistemas GNU/Linux como usuarios individuales, para el trabajo en nuestra pequeña organización, o nos estamos planteando sustituir la infraestructura completa (o parcial) de nuestra gran empresa.

Nota

La migración a un nuevo sistema no es trivial, hay que evaluarla mediante un estudio, donde se analicen tan-

ANOTACIONES

to los costes, como las prestaciones que esperamos obtener. Además, puede realizarse total o parcialmente, con cierto grado de coexistencia con los antiguos sistemas.

Estaremos delante de un proyecto de migración, total o parcial, de nuestros sistemas informáticos hacia GNU/Linux, y como administradores seremos responsables de este proceso.

Como en todo proyecto, habrá que estudiar el modo de responder a cuestiones como: ¿es rentable el cambio, en prestaciones, en costo?, ¿con qué objetivo lo hacemos?, ¿qué requerimientos queremos o debemos cumplir?, ¿podemos, hacer o es necesaria una migración completa?, ¿tiene que haber coexistencia con otros sistemas?, ¿habrá que formar de nuevo a los usuarios?, ¿podremos utilizar el mismo hardware, o necesitaremos uno nuevo?, ¿habrá costes añadidos importantes?, etc., o simplemente, ¿saldrá bien? Ésta y muchas preguntas más son las que tendremos que intentar responder. En el caso empresarial, esto normalmente pasaría por la definición de un proyecto de migración, con sus objetivos, análisis de requerimientos, proceso de implantación, estudios económicos, planes de formación de usuarios, etc. No entraremos en esto, pero nos plantearemos algunas de las cuestiones de forma sencilla. Y en el taller final examinaremos unos pequeños casos prácticos de cómo haríamos la migración.

Además, en el momento en que empecemos la migración a los sistemas GNU/Linux, es cuando comenzaremos a notar las ventajas que aportará a nuestra organización:

a) **Costes**: reducción de los costes, en licencias software del sistema y de las aplicaciones. GNU/Linux tiene un coste 0 en cuanto a las licencias, si se obtiene desde la red (por ejemplo, en forma de imágenes de los CD de la distribución), o un coste despreciable teniendo en cuenta que la mejor comparación de sistema sería un sistema Windows Advanced Server 2000 o el Windows Server 2003, con costes de que van de 1.500 a 3.000 euros, sin incluir mucho del software extra que proporciona GNU/Linux.

ANOTACIONES

Pero cuidado, no hay que desestimar los costes de mantenimiento y formación. Si nuestra organización sólo está formada por usuarios y administradores Windows, podemos tener un coste alto en nueva formación, personal, y quizás mantenimiento. Por eso, muchas grandes empresas desean depender de algún distribuidor comercial de GNU/Linux para que les implante el sistema, como por ejemplo las versiones empresariales que ofrecen Red Hat, SuSe y otros. Estas versiones GNU/Linux también tienen costes altos (comparables a Windows), pero por el contrario, están ya adaptadas a estructuras empresariales y traen software propio para gestionar la infraestructura informática de las empresas. Otro aspecto importante, que resumiría esta estimación de costes, es el concepto de TCO (*total cost of ownership*), como evaluación global de los costes asociados que nos encontraremos al emprender un desarrollo tecnológico; no sólo hay que evaluar las costes de licencias y maquinaria, sino también los costes de soporte y formación de las personas y productos implicados, los cuales pueden ser tan importantes o más que los de la solución implementada.

b) **Soporte**: Linux tiene el soporte de mantenimiento mayor que haya tenido un sistema operativo y gratis. Aparte, algunas empresas no adoptan GNU/Linux por ciertos temores, diciendo que no hay soporte del producto, y se dedican a comprar distribuciones adaptadas a distribuidores comerciales que les ofrecen contratos de soporte y mantenimiento. GNU/Linux tiene una comunidad de soporte mundial bien establecida, por medio de diferentes organizaciones que proporcionan documentación libre (los famosos HOWTO's), foros de usuarios especializados, comunidades de usuarios de prácticamente cualquier región o país del mundo, etc. Cualquier duda o problema con el que nos encontremos puede buscarse (por ejemplo, por alguno de los buscadores en Internet), y podemos tener respuestas en minutos. Cuando no, si hemos encontrado un *bug*, error, o situación no probada, podemos informar de ella, en varios lugares (foros, sitios de desarrollo, sitios de *bugs* de distribuciones, etc.), y obtener soluciones en horas o a lo sumo algunos días. Siempre que aparezca una duda o algún problema, intentar primero algunos procedimientos (así se aprende), y si no obtenemos solución en un tiempo prudencial, consultar a la comunidad GNU/Linux por si a algún otro usuario (o grupo de ellos) le ha ocurrido el mismo problema y ha obtenido solución, y

Nota

Linux Howto's: http://www.tldp.org/

ANOTACIONES

si no, siempre podemos informar del problema, y que nos planteen algunas soluciones.

2.4.1. Identificar requerimientos de servicios

Normalmente, si tenemos unos sistemas ya funcionando, tendremos que tener implantados algunos servicios de los cuales serán clientes los usuarios, o servicios que ayuden a la infraestructura del soporte informático. Los servicios entrarán dentro de alguna de las categorías vistas anteriormente, con las opciones GNU/Linux que comentamos.

Los sistemas GNU/Linux no son "nuevos" en absoluto, y derivan (como vimos en la introducción) de una historia de más de treinta años de uso y desarrollo de los sistemas UNIX. Por lo tanto, una de las primeras cosas que encontraremos es que no nos falta soporte para ningún tipo de servicio que queramos. Si acaso, habrá diferencias en la forma de hacer las cosas. Además, muchos de los servicios que se utilizan en los sistemas informáticos fueron pensados, investigados, desarrollados e implementados en su día para UNIX, y posteriormente adaptados a otros sistemas (como Windows, con más o menos acierto).

Cualquier servicio disponible en el momento podrá ser adaptado en los sistemas GNU/Linux con servicios equivalentes (cuando no iguales).

Ejemplo

Un caso famoso es el de los servidores Samba [Woo00]. Windows ofrece lo que él llama "compartir archivos e impresoras en red" mediante unos protocolos propios denominados genéricamente SMB (con apoyo de NetBios o NetBEUI). Estos protocolos permiten la compartición de carpetas de archivos (o discos) y de impresoras en una red de máquinas Windows. En UNIX esta idea ya era antigua, cuando apareció en Windows, y se disponía de servicios como NFS de compartición de archivos o la gestión remota de impresoras, bajo protocolos TCP/IP.

Nota

Un ejemplo de la evaluación de rendimiento en: http://www.vnunet.com/News/1144289

ANOTACIONES

Uno de los problemas de sustituir los servicios Windows de NetBios era cómo dar soporte a estos protocolos, ya que, si queríamos conservar las máquinas clientes con Windows, no podíamos utilizar los servicios UNIX. Para esto, Samba se desarrolló como un servidor para UNIX que soportaba los protocolos Windows, y podía sustituir a una máquina servidora Windows de forma transparente, los usuarios clientes con Windows no tenían por qué notar absolutamente nada. Es más, el resultado fue que en la mayor parte de los casos el rendimiento era mejor que en la máquina original con los servicios Windows.

2.4.2. Proceso de migración

En el proceso de migración, hay que tener en cuenta qué se quiere migrar, y si quiere hacerse de forma completa o parcial, coexistiendo con otros servicios o equipos con un sistema operativo diferente.

En ambientes como el de las grandes organizaciones, en donde encontramos un gran número de sistemas heterogéneos, habrá que tener en cuenta que seguramente no se migrarán todos, sobre todo en sistemas *workstation* dedicados a la ejecución de alguna aplicación básica para una tarea; puede que no exista la aplicación equivalente o simplemente podemos desear quedarnos con esos sistemas por razones de coste o de rentabilizar la inversión realizada.

Los elementos que se migren pueden ser:

a) Servicios o máquinas dedicadas a uno o más servicios. En la migración, pasaremos por la sustitución del servicio por otro equivalente, normalmente con el menor impacto posible si no queremos sustituir también a los clientes. En caso de clientes Windows, podemos usar el servidor Samba para sustituir los ser-

Nota

Podemos migrar varios elementos, ya sean los servicios que ofrecemos, las máquinas que los sirven o los clientes que acceden a ellos.

vicios de archivos que proporcionaban las máquinas Windows. Si se trata de otros servicios, podremos sustituirlos por los equivalentes GNU/Linux. En el caso de sustituir sólo algún servicio, normalmente se inhabilitará el servicio en la máquina que lo ofrecía y se habilitará en el sistema nuevo. Pueden ser necesarios cambios en los clientes (por ejemplo, direcciones de la nueva máquina o parámetros relacionados con el servicio).

Si la función la cumplía por entero una máquina servidora, hay que ver si la máquina estaba dedicada a uno o más servicios y si todos podrán ser sustituidos. En tal caso, sólo hay que reemplazar la máquina antigua por la nueva (o mantener la antigua) con los servicios bajo GNU/Linux, y en todo caso, modificar algún parámetro en los clientes si fuese necesario. Normalmente, antes de efectuar el cambio, es conveniente testar la máquina por separado con algunos clientes para asegurarse de que cumple su función correctamente y sustituir las máquinas en algún periodo de inactividad del sistema.

En cualquier caso, seguramente habrá que hacer *backups* de los datos anteriores al nuevo sistema, por ejemplo, el sistema de ficheros o las aplicaciones de que disponía el servidor original. Otro de los puntos previos a tener en cuenta es la portabilidad de los datos usados, un problema que a menudo presenta difícil solución si en la organización se utilizaban formatos de datos o aplicaciones dependientes de una plataforma.

Ejemplo

Por poner algunos casos prácticos de problemas con que se encuentran algunas empresas hoy en día:

- Aplicaciones en web con ASP: estas aplicaciones son sólo realizables en plataformas web con Windows y el servidor web IIS de Microsoft. Habría que evitarlas, si en algún momento pensamos hacer una migración de plataformas, y no queremos reescribirlas o pagar a una empresa para que lo haga. En plataformas GNU/Linux, está el servidor web Apache (el más utilizado en Internet), que también se puede usar con Windows, este servidor soporta ASP en Perl (en Windows se suele utilizar visual basic y Javascript), hay soluciones

ANOTACIONES

de terceros para migrar los ASP o más o menos convertirlos. Pero si nuestra empresa dependiese de esto, sería muy costoso en tiempo y dinero. Una solución práctica habría sido realizar los desarrollos web en Java (que sí que es portable entre plataformas) u otras soluciones como PHP.

- Bases de datos: usar un SQL *Server* de Microsoft, nos hace totalmente dependientes de su plataforma Windows, además, si utilizamos soluciones propietarias en un entorno concreto para aplicaciones de la base de datos, serán de difícil transferencia. Otras bases de datos como Oracle y DB2 (de IBM) son más portables por disponer de versión en diferentes plataformas, o por utilizar lenguajes de programación más portables. También se podría trabajar con sistemas de bases de datos Postgres o MySQL (también tiene versión para Windows) utilizadas en Linux, y que permiten una transición más fácil. Asimismo, si se combina con el desarrollo web tenemos muchas facilidades; en este sentido, hoy en día se utilizan sistemas como: aplicaciones web con Java ya sea servlets, applets, o Ejb; o bien soluciones como las famosas LAMP combinación de Linux, Apache, Mysql y Php.

b) Workstations: en éstas, el mayor problema parte de las aplicaciones, ya que son las que dan su razón de ser a la estación de trabajo, ya sean programas de CAD, de animación, programas de ingeniería o científicos. Aquí será importante que podamos sustituirlas por aplicaciones iguales o, como mínimo, compatibles con las mismas características. Normalmente, la mayor parte de estas aplicaciones ya provienen de un mundo UNIX, puesto que la mayoría de estas *worstations* estaban pensadas como máquinas UNIX. Con lo cual, quizás baste una recompilación o una adaptación mínima al nuevo sistema GNU/Linux, si disponemos del código fuente (como suele pasar en muchas aplicaciones científicas). Si se trata de aplicaciones comerciales, los fabricantes (de software de ingeniería y científico) comienzan a adaptarlas a Linux, aunque en estos casos las aplicaciones suelen ser muy caras (pueden ir perfectamente de miles a centenares de miles de euros).

ANOTACIONES

c) Máquinas clientes de escritorio. Las máquinas de escritorio continúan siendo un quebradero de cabeza en el mundo GNU/Linux, ya que ofrecen bastantes problemas adicionales. En los servidores, las máquinas se destinan a funcionalidades claras, en general no requieren interfaces gráficas complejas (muchas veces con comunicación textual basta), el hardware, normalmente *expreso* y de altas prestaciones, se compra para unas funcionalidades concretas y las aplicaciones suelen ser los propios servidores incluidos en el sistema operativo o en algunos terceros. Además, estas máquinas suelen estar gestionadas por personal de tipo administrador que tiene amplios conocimientos de lo que maneja. En el caso del escritorio, nos encontramos con un factor problemático (en sí mismo, y aún más para los administradores): los usuarios finales del sistema. Los usuarios de escritorio esperan disponer de potentes interfaces gráficas, más o menos intuitivas, y aplicaciones que permitan desarrollar sus tareas rutinarias, normalmente ofimáticas. Este tipo de usuario (con excepciones) no tiene por qué tener unos conocimientos informáticos elevados; en general, sus conocimientos son de tipo ofimático y suelen usar un reducido número de aplicaciones con mayor o menor dominio de las mismas. Aquí GNU/Linux tiene un problema claro, ya que UNIX como tal nunca fue pensado como un sistema puramente de escritorio, y sólo fue adaptado *a posteriori* por sistemas como X Window y los diferentes escritorios, como los actuales de GNU/Linux: Gnome y KDE. Además, el usuario final suele estar acostumbrado a sistemas Windows (que copan casi un 95% del mercado de escritorio).

En el caso del escritorio, GNU/Linux tiene que superar unos cuantos obstáculos. Uno de los más críticos es que no viene preinstalado en las máquinas, lo que obliga al usuario a tener conocimientos para poder instalarlo. Otros motivos podrían ser:

Nota

El ambiente de escritorio es una batalla todavía por librar para los sistemas GNU/Linux; tienen que vencer la desconfianza de los usuarios a cambiar de sistema, y saber dar a conocer que ofrecen alternativas, de sencillez y aplicaciones, que solucionan las tareas de los usuarios.

ANOTACIONES

- Desconfianza del usuario: una pregunta que se puede plantear un usuario es, ¿por qué debo cambiar de sistema? o ¿me ofrecerá lo mismo el nuevo entorno? Una de las razones básicas para hacer el cambio sería el software de calidad y su precio, del que una buena parte es libre. En este punto, afecta el tema de las copias de software ilegales. Parece ser que los usuarios consideran que su software es gratis, cuando en realidad están en una situación ilegal. El software GNU/Linux ofrece gran calidad a bajo coste (o gratis en muchos casos), y existen múltiples alternativas para una misma tarea.

- Sencillez: el usuario se muestra normalmente perdido si el sistema no le ofrece algunas referencias que lo hagan parecido a lo que ya conoce, como el comportamiento de la interfaz, o que las herramientas sean parecidas. Que, en general, no necesite mucho tiempo extra para aprender y manejar el nuevo sistema. GNU/Linux aún presenta algunos problemas en las instalaciones más o menos automáticas que, aunque mejoran día a día, todavía es necesario un cierto grado de conocimiento para hacer una instalación correcta del mismo. Otro problema radica en el soporte del hardware del PC, que cada día mejora, sin embargo, hasta que los fabricantes le presten la atención adecuada, no podremos tener el mismo soporte que en otros sistema propietarios (como en Windows).

- Transparencia: los entornos GNU/Linux tienen muchos mecanismos complejos, como los *daemons*, servicios, ficheros ASCII difíciles de configurar, etc. De cara a un usuario final, sería necesario poder ocultar todas estas complejidades, mediante programas gráficos, asistentes de configuración, etc. Es uno de los caminos que han tomado algunas distribuciones como Red Hat, Mandrake o SuSe.

- Soporte de aplicaciones conocidas: un usuario ofimático típico tendrá el problema de sus datos, o los formatos de éstos. ¿Qué hace con los datos que tenía hasta el momento? Este tema está mejorando bastante, gracias a las *suites* ofimáticas que comienzan a tener las funcionalidades necesarias para un usuario de escritorio. Por ejemplo, si nos planteamos una migración desde un uso de una *suite* Office de Windows, podemos encontrar *suites*

ANOTACIONES

como OpenOffice (software libre y gratuito) que puede leer (y crear) los formatos (con algunas restricciones) de ficheros Office, y leer o crear datos en estos formatos. La compatibilidad de formatos no es que sea difícil, cuando éstos están abiertos, pero Microsoft no es de esa opinión y continúa manteniendo una política de formatos cerrados; y hay que hacer un trabajo importante para poder utilizar estos formatos, mediante reingenieria inversa (proceso bastante costoso). Además, en la era de Internet, donde la información se supone que se mueve libremente, los formatos cerrados sin documentar son más un obstáculo que otra cosa. Lo mejor es utilizar formatos abiertos como RTF (aunque este caso también tiene algún problema, por las múltiples versiones que existen de él), o bien formatos basados en XML (OpenOffice genera sus documentos propios en XML), o PDF para la documentación de lectura.

Nota

Por ejemplo: http://linuxshop.ru/linuxbegin/win-lin-soft-en/table.shtml

- Proporcionar alternativas válidas: el software que se deja de usar tiene que tener alternativas que cumplan el trabajo anterior en el otro sistema. En la mayoría de aplicaciones existe una o varias alternativas con funcionalidades parecidas, cuando no superiores. Pueden encontrarse por Internet diferentes listas de equivalencias (más o menos completas) de aplicaciones Windows con sus correspondientes GNU/Linux.

- Soporte de ejecución de otras aplicaciones de diferentes sistemas: en algunas condiciones es posible ejecutar aplicaciones de otros sistemas UNIX (de la misma arquitectura, por ejemplo, Intel x86), o bien de msdos o Windows, mediante paquetes de compatibilidad o algún tipo de emuladores.

La mayor parte de estos problemas están superándose poco a poco y nos permitirán en el futuro disfrutar de una mayor cuota de usuarios GNU/Linux en el escritorio, y a medida que aumenten, disponer de mejores aplicaciones y que las empresas de software se dediquen a implementar versiones para Linux.

En el caso empresarial, esto puede superarse con una migración suave, primero de las etapas de servidores, y *workstations*, y después pasar por un proceso de formación amplia de los usuarios en los

ANOTACIONES

nuevos sistemas y aplicaciones, para, finalmente, integrarlos en su escritorio.

Un proceso que va a ayudar en gran medida a esto es también la introducción del software de código abierto, en las fases educativas y en administraciones publicas, como es el caso reciente de la Comunidad de Extremadura con su distribución GNU/Linux llamada Linex; o recientes medidas para llevar este software a la educación primaria, o las medidas de universidades de llevar a cabo cursos y materias con estos sistemas.

2.5. Taller de migración: análisis de casos de estudio

En este taller vamos a intentar aplicar lo estudiado en esta unidad para analizar unos procesos de migración sencillos, y algún detalle de las técnicas necesarias (en el caso de técnicas de red, las veremos en las unidades dedicadas a administración de redes).

Nos plantearemos los siguientes casos de estudio:

- Migración individual de un usuario de escritorio Windows a un sistema GNU/Linux.
- Migración de una pequeña organización que dispone de sistemas Windows y algunos UNIX.
- Migración de un servidor Windows a un servidor Samba en GNU/Linux.

a) Migración individual de un usuario de escritorio Windows a un sistema GNU/Linux

Un usuario se plantea la migración a GNU/Linux [Ray02b]. Normalmente, primero se pasará por un periodo de convivencia, de modo que el usuario dispondrá de los dos sistemas, y dedicará cada uno de ellos a una serie de tareas: continuará desarrollando tareas en Windows mientras aprende el nuevo sistema y encuentra software equivalente, o software nuevo que le permita hacer otras tareas para las que antes no disponía de software.

ANOTACIONES

Nota

La migración es para un usuario personal quizás uno de los procesos más complejos; hay que ofrecerle alternativas a lo que ya usa, de forma que no entrañe muchas complejidades extra y pueda adaptarse con facilidad al nuevo sistema.

Una primera posibilidad será hacer una instalación dual [Ban01] [Sko03b] del sistema original (un Windows 9x o un NT/XP por ejemplo), junto con el sistema GNU/Linux. Para ello, necesitaremos disponer o bien de espacio libre en disco no particionado, o bien, si estamos en particiones de tipo FAT/32, podemos liberar espacio con programas como FIPS, que permiten recortar la partición existente. En caso de no disponer de espacio, o de tener particiones NTFS, habrá que plantearse comprar un segundo disco duro, que dedicaremos a GNU/Linux.

Nota

Linux Hardware Howto: http://www.tldp.org/HOWTO/HardwareHOWTO/index.html

Una vez dispongamos del disco duro, tendríamos que verificar que nuestro hardware sea compatible con Linux [Pri02], ya sea por medio de alguna lista de compatibilidad de hardware o verificándolo en el fabricante, por si fuera necesario adquirir nuevos componentes o configurar de alguna forma el existente. Si desconocemos nuestro hardware, podemos verificarlo en Windows con el "administrador de dispositivos" (en el panel de control) o algún software de reconocimiento de hardware.

Una vez completada la revisión del hardware, tendremos que decidir la distribución del sistema GNU/Linux que usaremos. Si el usuario es poco experimentado en Linux, o tiene conocimientos básicos de informática, mejor decidirse por alguna de las distribuciones más "amigables" de cara al usuario, como Red Hat, Mandrake, SuSe, o similares. Si tenemos más conocimientos o estamos tentados a experimentar, podemos probar una Debian. En el caso de las comerciales, la distribución, en la mayoría de veces, con un hardware compatible se instala perfectamente sin problemas, y realiza configuraciones básicas que permiten ya utilizar el operativo. En el proceso tendremos que instalar el software, que normalmente vendrá definido por unos conjuntos de software orientados: a servidores, a aplicaciones concretas, o a aplicaciones de escritorio como las ofimáticas, aplicaciones de desarrollo (si nos interesa la programación), etc.

ANOTACIONES

Una vez instalado el sistema, se plantea el tema de la compartición de datos [Gon00] [Kat01], ¿cómo compartimos datos entre los dos sistemas? o ¿hay posibilidad de compartir algunas aplicaciones? Para esto hay varias soluciones:

a) Método por "intermediario": consiste en compartir los datos, por ejemplo, mediante disquete. Para ello, lo mejor son las utilidades denominadas *mtools*, son utilidades que permiten acceder a disquetes con formato msdos de forma transparente, y existen múltiples comandos de línea que funcionan de forma muy parecida a msdos o Windows. Estos comandos se llaman exactamente como los comandos msdos originales, sólo que con una "m" delante, por ejemplo: *mcd*, *mcopy*, *mdir*, *mdel*, *mformat*, *mtype*, etc.

b) Método directo: consiste en usar directamente los sistemas de ficheros de Windows. Como veremos en la unidad de administración local, GNU/Linux puede leer y escribir una gran cantidad de sistemas de ficheros, entre ellos el FAT, FAT32, y NTFS (sólo lectura). Se tiene que pasar por un proceso denominado "de montaje", que permite incorporar el sistema de ficheros de Windows a un punto como el de Linux; por ejemplo, podríamos montar nuestro disco Windows en /mnt/Windows y acceder desde este punto a sus carpetas y archivos, permitiendo escrituras y lecturas. Con los ficheros de texto ASCII, hay que andar con cuidado, ya que UNIX y Windows los tratan de modo diferente: en UNIX, el final de línea tiene un sólo carácter de retorno de carro (ASCII 13), mientras que en Windows hay dos, un retorno y un avance de línea (caracteres ASCII 13 y 10). Con lo cual, suele ser habitual que, al leer un fichero, éste contenga caracteres "raros" al final de línea. Hay editores como *emacs* que los tratan de forma transparente y, en todo caso, hay utilidades Linux que permiten convertirlos de uno a otro formato (con utilidades como *duconv*, *recode*, *dos2UNIX, UNIX2dos*).

c) Uso de aplicaciones: existen algunas alternativas para poder ejecutar las aplicaciones (no todas) de msdos y Windows. Para Linux hay un emulador de msdos llamado Dosemu [Sun02], y para Windows existe el Wine. Éste puede ejecutar algunas aplicaciones sencillas de Windows (por ejemplo, permite ejecutar un Office 97), pero se continúa mejorando. Si la ejecución de aplicaciones Windows es imprescindible, nos puede ayudar al-

ANOTACIONES

gún software comercial: parecidos a Wine existen Win4Lin y CrossOver. Otra posible solución son las máquinas virtuales; hay un software llamado VMware (una auténtica maravilla) que crea una máquina virtual: un PC completo virtual simulado por software, al cual se le puede instalar cualquier sistema operativo; se encuentran versiones para Windows y para GNU/Linux, lo que permite tener un GNU/Linux instalado con un Windows corriendo virtualmente sobre él, o un Windows con GNU/Linux en virtual.

Aparte de compartir la información, pueden buscarse aplicaciones GNU/Linux que sustituyan a las originales Windows a medida que el usuario vaya aprendiendo a utilizarlas.

Ejemplo

Por ejemplo, la *suite* ofimática puede pasarse a OpenOffice, que tiene un alto grado de compatibilidad con los ficheros de Office y un funcionamiento bastante parecido, o bien KOffice (para el escritorio KDE), o GNumeric y AbiWord (para Gnome). O para procesamiento de imágenes tomamos Gimp, tan potente como Photoshop. Y multitud de reproductores multimedia: Xine, Mplayer (o también una versión del RealPlayer). En Internet se pueden encontrar listas de equivalencias de programas entre Windows y GNU/Linux.

b) Migración de una pequeña organización que dispone de sistemas Windows y algunos UNIX

Nota

La migración en una organización (aunque sea pequeña) plantea muchas dificultades: tendremos diferentes ambientes de trabajo, hardware y software heterogéneo, y más de una vez, reticencias de los usuarios al cambio.

ANOTACIONES

Consideremos ahora una organización que tenga máquinas Windows y algunas máquinas UNIX dedicadas a servicios o *workstations* y unos usuarios un poco "anárquicos". Por ejemplo, estudiemos la siguiente situación: la organización tiene una pequeña red local de máquinas Windows repartidas por los usuarios, como máquinas de igual a igual en un grupo de trabajo Windows (no hay dominios NT). El grupo es variopinto: tenemos máquinas con Windows 98, ME, NT, XP, pero personalizadas por cada usuario con el software que necesita para su trabajo diario: ya sea Office, navegador, lector de correo, o entornos de desarrollo para los programadores de diferentes lenguajes (por ejemplo C, C++, Java).

Se dispone de algunos recursos hardware extras, como varias impresoras conectadas a la red local (aceptan trabajos TCP/IP), y permiten utilizarse desde cualquier punto de la organización. Y existe una máquina compartida, con algunos recursos especiales, como *scanner*, grabadora de CD y directorios compartidos por red, donde los usuarios pueden dejar sus directorios con sus ficheros para procesos de *backup* o para recuperar, por ejemplo, imágenes escaneadas.

También disponemos de varias *workstations*, en este caso Sun Microsystem Sparc, que ejecutan Solaris (UNIX, comercial de Sun). Estas estaciones están dedicadas a desarrollo y a algunas aplicaciones científicas y gráficas. Estas máquinas disponen de servicios de NFS para compartir archivos y NIS+ para manejar la información de los usuarios que se conectan a ellas y que puedan hacerlo desde cualquiera de ellas de forma transparente. Algunas de las máquinas incluyen servicios específicos; hay una destinada a servidor web de la organización y otra destinada a servidor de correo.

Se plantea la posibilidad de realizar una migración a GNU/Linux por intereses de desarrollo de software y por el interés particular de algunos usuarios de disponer de este sistema.

Además, se aprovechará la migración para intentar solucionar algunos problemas: de seguridad, algunos sistemas antiguos Windows no son la mejor forma de compartir archivos; se quiere restringir el uso de la impresora (el gasto en papel y el coste asociado es alto) a unas cuotas más razonables. Por otra parte, se quiere ofrecer cierta libertad a los usuarios, no se les obligará a cambiar de sistema, aunque se les hará la sugerencia. Y aprovecharemos para comprar hardware nuevo que complemente al existente, por ejemplo, si las estaciones de trabajo están faltas de espacio de disco, lo cual supone limitaciones de espacio para correo y cuentas de usuario.

Después de toda esta pequeña descripción de nuestra organización (en otros casos más complejos, podría llenar varias páginas o ser un documento entero), nos comenzamos a plantear posibilidades para solucionar todo esto:

1) ¿Qué hacemos con las *workstations* actuales? El coste en mantenimiento y licencias de software es elevado. Tenemos que cubrir

ANOTACIONES

el mantenimiento de fallos en las estaciones, hardware caro (en este caso, discos SCSI) y ampliaciones de memoria también caras. El coste del sistema operativo y sus actualizaciones también es caro. En este caso, se nos ofrecen dos posibilidades (dependiendo del presupuesto de que dispongamos para el cambio):

a) Podemos reducir costes convirtiendo las máquinas a sistemas GNU/Linux. Estos sistemas son de arquitectura Sparc y existen distribuciones que soportan esta arquitectura. Podríamos sustituir los servicios por sus equivalentes GNU/Linux; la sustitución sería prácticamente directa, puesto que ya usamos un sistema UNIX.

b) Otra posibilidad sería la eliminación del hardware propietario de Sun y convertir las estaciones en PC potentes con GNU/Linux; esto simplifica su mantenimiento posterior, aunque con un alto coste inicial.

2) ¿Y con el software de las *workstation*? Si las aplicaciones son de desarrollo propio, puede ser suficiente la recompilación o adaptación simple al nuevo entorno. Si son comerciales, tendremos que ver si la empresa puede proporcionarlos en entornos GNU/Linux, o si podemos encontrar reemplazos con funcionalidad parecida. En el caso de los desarrolladores, sus entornos de lenguajes C, C++ y Java pueden portarse fácilmente; en caso de C y C++, se puede utilizar el compilador GNU *gcc*, y existen multitud de IDE para el desarrollo (KDevelop, Anjuta, ...); en el caso de Java, se puede utilizar el *kit* de Sun en Linux y entornos varios de código abierto (Eclipse de IBM o Netbeans).

3) ¿Y con los usuarios? A aquellos que estén interesados en GNU/Linux les podemos instalar equipos duales con Windows y Linux para que comiencen a probar el sistema y, si les interesa, pasar finalmente a un único sistema GNU/Linux. Podemos encontrar dos tipos de usuarios: los puramente ofimáticos necesitaran básicamente la *suite*, navegador y correo; esto se les puede ofrecer con un escritorio GNU/Linux como Gnome o KDE y software como OpenOffice, navegador Mozilla, y correo Mozilla Mail (o cualquier otro Kmail,...). La equivalencia es más o menos directa, depende de las ganas que los usuarios tengan de probar y usar el nuevo software. Para los desarrolladores, el cambio puede ser más directo, ya que se les ofrecen muchos más entornos y herramientas flexibles; podrían pasarse completamente a sistemas GNU/Linux o trabajar directamente con las *workstations*.

ANOTACIONES

4) ¿Y las impresoras? Puede establecerse alguna estación de trabajo como servidor de impresión (ya sea por colas TCP/IP o por servidor Samba), y controlar las impresiones mediante cuotas.

5) ¿La máquina compartida? El hardware compartido puede dejarse en la misma máquina o se puede controlar desde un sistema GNU/Linux. En cuanto al espacio de disco compartido, puede moverse a un servidor Samba que sustituya al actual.

6) ¿Ampliamos el espacio del disco? Dependerá del presupuesto. Podemos mejorar el control mediante un sistema de cuotas que reparta el espacio de una forma equitativa y ponga límites a la saturación.

c) **Migración de un servidor Windows a un servidor Samba en GNU/Linux**

En este caso, el proceso básico necesario para efectuar una posible migración de un servidor Windows que comparte carpetas e impresora a un servidor Samba en un sistema GNU/Linux.

Nota

El proceso básico necesario suele ser bastante más extenso, consultar la bibliografía para ver los pasos completos del mismo.

Supongamos una máquina que pertenece a un grupo de trabajo GRUPO, que comparte una impresora llamada PRINTER y que tiene una carpeta compartida DATOS que no es más que el disco D de la máquina. Varios clientes Windows acceden a la carpeta para lectura/escritura, dentro de una red local con direcciones IP 192.168.1.x, donde x será 1 para nuestro servidor Windows, y los clientes tienen otros valores (las redes 192.168.x.x se utilizan a menudo como direcciones para montar redes privadas internas).

Nota

Gracias al software como Samba, la migración desde entornos Windows es muy flexible y rápida e incluso con mejoras de prestaciones.

En nuestro proceso vamos a construir un servidor Samba, que es la que permite ejecución en Linux del protocolo SMB (*Server Message Block*). Este protocolo permite la interacción del sistema de archivos y de la impresora por medio de redes de múltiples sistemas operativos. Podemos montar carpetas pertenecientes a Windows en las máquinas Linux, o bien parte de los archivos de Linux en Windows, y lo mismo con las impresoras de uno u otro. El servidor está compuesto de dos *daemons* (procesos de sistema) llamados *smbd* y *nmbd*.

El proceso *smbd* gestiona las peticiones de los clientes hacia los archivos o impresoras compartidos. El *nmbd* gestiona el sistema de

ANOTACIONES

nombres de las máquinas y los recursos bajo el protocolo NetBIOS (creado por IBM). Este protocolo es independiente de la red que se usa (actualmente, Microsoft utiliza en NT/2000/XP Netbios sobre TCP/IP). El *nmbd* también proporciona servicios WINS, que es el servicio de asignación de nombres, que normalmente se ejecuta sobre Windows NT si tenemos una colección de máquinas; es una especie de combinación de DNS y DHCP para entornos Windows. El proceso es un poco complejo, pero en resumen: cuando una máquina Windows arranca, o bien tiene una dirección IP estática, o bien dinámica a través de un servidor DHCP, y además puede que tenga un nombre NetBIOS (el nombre que el usuario asigna a la máquina: en identificación de red), entonces, el cliente WINS contacta con el servidor para informar de cuál es su IP; si una máquina de red pregunta posteriormente por el nombre NetBios, se contacta con el servidor WINS para obtener su dirección IP y se establecen las comunicaciones. El *nmbd* ejecuta este proceso sobre GNU/Linux.

Como cualquier otro servicio de red, no se debería ejecutar sin considerar qué riesgos puede suponer su activación y cómo podemos minimizarlos. Respecto a Samba, hay que tener presentes los temas de seguridad, puesto que estamos abriendo parte de nuestros archivos y las impresoras locales o de la red. Tendremos que verificar bien las restricciones de comunicación que ponemos para no dar acceso a usuarios o máquinas no deseadas. En este ejemplo básico, no vamos a comentar estos temas; en un caso real, tendríamos que examinar las opciones de seguridad y restringir el acceso sólo a quien realmente deseemos.

En el proceso de migración, primero tendremos que configurar el sistema GNU/Linux para el soporte de Samba [Woo00], se necesita el soporte en el *kernel* de los *filesystems* Samba (*smbfs*), que normalmente ya viene activado. Necesitaremos instalar los paquetes software de Samba: habrá que examinar qué paquetes relacionados con Samba hay en la distribución e instalar los que tengan que ver con el funcionamiento de servidor. Y también, si se quiere, los relacionados como cliente, en el caso de que deseemos ser clientes de máquinas Windows o testar desde nuestro Linux los recursos compartidos de las máquinas Windows. En una distribución Debian, estos paquetes son: Samba, Sambacommon, Smbclient, Smbfs. También puede ser interesante instalar *swat*, que es una herramienta gráfica basada en web para la administración de los servicios Samba. Para nuestro

ANOTACIONES

servidor GNU/Linux de Samba [Woo00] [War03], tendremos que copiar el anterior disco D de la máquina original a la nueva máquina y colocar su contenido en algún *path*, por ejemplo, /home/DATOS, así como el contenido de los archivos Windows que teníamos, ya sea por copia de *backup*, transferencia ftp, o usando Samba como cliente para transferir los archivos.

En cuanto a la utilización de GNU/Linux como cliente Samba, es bastante sencilla. Mediante el uso de comandos cliente:

a) Montamos un directorio compartido Windows (sea *host* el nombre del servidor):

```
smbmount //host/carpeta /mnt/Windows
```

b) Colocamos el acceso a la carpeta Windows de la máquina *host* en nuestro directorio local:

```
/mnt/Windows
```

c) A continuación, podemos desmontar el recurso con:

```
smbumount /mnt/Windows
```

Si no conocemos los recursos compartidos, podemos obtener una lista con:

```
smbclient -L host
```

Y también podemos utilizar smbclient //host/carpeta, que es un programa parecido a un cliente ftp.

Nota

Ver el archivo:
man smbclient.

Para montar el servidor Samba, una vez tengamos instalado todo el software Samba, tendremos que configurar el servidor a través de su fichero de configuración. Según la versión (o la distribución), este fichero puede estar en /etc/smb.conf o bien en /etc/samba/smb.conf. Las opciones aquí mostradas pertenecen a un Samba 2.2.3 instalado sobre una Debian Woody. Otras versiones pueden tener algunas modificaciones menores.

Nota

Consultar siempre las páginas ***man***, o 'manuales', que acompañen al software.

Durante la instalación de los paquetes de software es habitual que se nos pregunten algunos datos sobre su configuración. En el caso de

Samba, se nos pregunta por el grupo de trabajo al que va a servir; habrá que colocar el mismo nombre del grupo que en Windows. También se nos pregunta si deseamos contraseñas encriptadas (recomendable por seguridad, antes en los Windows 9x se enviaban en texto tal cual, en lo que constituye un claro ejemplo de escasa seguridad y de alta vulnerabilidad del sistema).

A continuación pasamos a ver el proceso de configuración del fichero smb.conf. Este fichero tiene tres secciones principales:

1) *Global* (características básicas de funcionamiento).

2) *Browser* (controla lo que otras máquinas ven de nuestros recursos).

3) *Share* (controla qué compartimos).

En el manual (extenso) de este fichero pueden verse las opciones disponibles (*man smb.conf*). Editaremos el fichero con algún editor e iremos viendo algunas de las líneas del fichero (los caracteres '#' o ';' a principio de línea son comentarios, si la línea contiene ';' es un comentario; para habilitar la línea, quitar el ';'):

```
workgroup = GRUPO
```

Aquí esta indicado el grupo de trabajo Windows del cual las máquinas Windows clientes serán miembros.

```
server string = %h server (Samba %v
```

Podemos colocar una descripción textual de nuestro servidor, la *h* y la *v* que aparecen son variables de Samba, que hacen referencia al nombre del *host* y a la versión de Samba. Por seguridad, es mejor quitar la *v*, ya que con ello se informa al exterior de qué versión de Samba tenemos; si hay *bugs* de seguridad conocidos, esto puede aprovecharse.

```
hosts allow = 192.168.1
```

Esta línea puede estar presente o no, y la podemos incluir para habilitar qué *hosts* serán servidos; en este caso, todos los del rango 192.168.1.x.

ANOTACIONES

```
printcap name = /etc/printcap
```

El fichero *printcap* es donde GNU/Linux guarda la definición de las impresoras, y es aquí donde Samba buscará la información acerca de éstas.

```
guest account = nobody
```

Ésta es la cuenta de "invitado". Podemos crear una cuenta diferente, o solamente habilitar el acceso a Samba a los usuarios dados de alta en el sistema GNU/Linux.

```
log file = /var/log/samba/log.%n
```

Esta línea nos dice dónde se van a guardar los ficheros del registro de sesión (*logs*) de Samba. Se guarda uno por cada cliente (variable *n* es el nombre del cliente conectado).

```
encrypt passwords = true
```

Es conveniente, por seguridad, usar encriptación de contraseñas (*passwords*) si tenemos máquinas clientes con Windows 98, NT o superiores. Estas contraseñas se guardan en un fichero /etc/samba/smbpasswd, que normalmente se genera para los usuarios de la instalación de Samba. Las contraseñas se pueden cambiar con el comando *smbpasswd*. También hay una opción llamada *UNIX passwdord sync*, que permite que el cambio sea simultáneo a las dos contraseñas.

A continuación, saltaremos ahora a la sección "Share Definitions":

```
[homes]
```

Estas líneas permiten dar acceso a las cuentas de los usuarios desde las máquinas Windows. Si no lo queremos, añadimos unos ';' al inicio de estas líneas, y las máquinas al conectarse verán el nombre *comment*. En principio, la escritura está deshabilitada, pero para habilitarla, sólo hay que poner "yes" en la opción *writable*.

Cualquier compartición de un directorio concreto (en Samba se suele denominar *partición* a un grupo de datos compartidos), se hará

ANOTACIONES

como los ejemplos que aparecen (ver, por ejemplo, la definición de compartir el CD-ROM en las líneas que empiezan por [cdrom]). En *path* se coloca la ruta de acceso.

Ejemplo

En nuestro caso, por ejemplo, pondríamos un nombre DATOS a la partición en la ruta /home/DATOS, donde habíamos copiado el disco D de la máquina original Windows y el *path* donde se puede encontrar, además de un alto grupo de opciones que pueden modificar el usuario que podrá acceder y la manera de hacerlo.

Nota

Ver: *man smb.conf*

También hay una definición [*profiles*], que permite controlar los perfiles (*profiles*) de los usuarios Windows, o sea, el directorio donde se guarda su configuración de escritorio Windows, el menú de inicio, etc.

El método es parecido para las impresoras: se hace una partición con el nombre de la impresora (el mismo que se haya dado en GNU/Linux), y en el *path* se coloca la dirección de la cola de la impresora (en GNU/Linux: /var/spool/samba/PRINTER). Y la opción *printable = yes*, si queremos que se envíen trabajos con Samba. Y también se puede restringir qué usuarios acceden (*valid users*).

Una vez hechos estos cambios, sólo tenemos que guardarlos y reiniciar Samba para que lea la nueva configuración. En Debian:

```
/etc/init.d/samba restart
```

Ahora, nuestro directorio compartido y la impresora por Samba estarán disponibles, de manera que sirvan a los usuarios sin que éstos noten diferencia alguna respecto a las conexiones anteriores con el servidor Windows.

2.6. Actividades para el lector

1) En la descripción de servicios GNU/Linux, ¿se encuentra a faltar alguna funcionalidad?, ¿qué otro tipo de servicios añadiríais?

ANOTACIONES

2) En el segundo caso de estudio del taller (el de la organización), ¿cómo cambiaríais la infraestructura informática si dispusierais de un presupuesto de coste cero, un presupuesto medio, o un presupuesto alto? Presentad algunas soluciones diferentes a las expuestas.

3) El software VMware Workstation es una máquina virtual por software, que permite instalar operativos sobre un PC virtual. Se puede conseguir una demo en http://www.vmware.com. Probar (en el caso de disponer de una licencia Windows) a instalarla sobre Windows, y entonces un GNU/Linux sobre el PC virtual (o al revés). ¿Qué ventajas nos aporta este sistema de compartir los operativos? ¿Qué problemas ocasiona?

4) Si se dispone de dos máquinas para instalar un servidor Samba, podemos probar la instalación o configuración del servidor en configuraciones de cliente Samba UNIX-servidor Windows, o cliente Windows-servidor Samba en GNU/Linux. En una sola máquina puede probarse, utilizando la misma máquina como servidor y cliente Samba.

2.7. Otras fuentes de referencia e información

- [LPD03a] Linux Documentation Project proporciona los Howto's de los diferentes aspectos de un sistema GNU/Linux y un conjunto de manuales más elaborados.

- [Mor03] Buena referencia de configuración de sistemas Linux, con algunos casos de estudio en diferentes entornos; comenta diferentes distribucions Debian y Red Hat.

ANOTACIONES

3. Herramientas básicas para el administrador

El administrador de sistemas GNU/Lsinux tiene que enfrentarse diariamente a una gran cantidad de tareas. En general, en la filosofía UNIX no suele haber una única herramienta para cada tarea o una sola manera de hacer las cosas. Lo común es que los sistemas UNIX proporcionen una gran cantidad de herramientas más o menos simples para afrontar las diferentes tareas.

Será la combinación de las herramientas básicas, cada una con una tarea muy definida, la que nos dará la posibilidad de solucionar un problema o tarea de administración.

Nota

GNU/Linux posee un conjunto muy amplio de herramientas con funcionalidades básicas, cuya potencia está en su combinación.

En esta unidad veremos diferentes grupos de herramientas y algunos ejemplos básicos de sus usos. Identificaremos algunas funciones básicas de estas herramientas y su uso. Comenzaremos por examinar algunos estándares del mundo Linux, que nos permitirán hallar algunas de las características básicas que esperamos de cualquier distribución de GNU/Linux. Estos estándares, como el LSB (o Linux Standard Base) [Lin03c] y el FHS (*Filesystem Hierarchy Standard*) [Lin03b], nos hablan de herramientas que esperamos encontrar disponibles, de una estructura común para el sistema de ficheros, así como de diversas normas que tienen que cumplirse para que una distribución sea considerada un sistema GNU/Linux y mantenga reglas comunes para la compatibilidad entre ellos.

En la automatización de tareas de administración suelen utilizarse comandos agrupados en *shell scripts* (también llamados *guiones de comandos*), lenguajes interpretados por el *shell* (intérprete de comandos) del sistema, y nos permiten unir los comandos del sistema con estructuras de control de flujo y disponer de un entorno de prototipo rápido de herramientas para la automatización de tareas.

ANOTACIONES

Otro esquema habitual es la utilización de herramientas de compilación y depuración de lenguajes de alto nivel (como por ejemplo C). En general, serán utilizadas por el administrador para generar nuevos desarrollos de aplicaciones o herramientas, o para incorporar al sistema aplicaciones que vengan como código fuente y tenga que adaptarse y compilarse.

También analizaremos el uso de algunas herramientas gráficas con respecto a las de la línea de comandos. Estas herramientas suelen facilitar las tareas al administrador, pero su uso es limitado, ya que dependen fuertemente de la distribución de GNU/Linux, o incluso de cada versión. Aun así, hay algunas herramientas útiles que son exportables entre distribuciones.

Por último, analizaremos un grupo de herramientas imprescindibles para mantener el sistema actualizado, las herramientas de gestión de paquetes. El software servido en la distribución GNU/Linux, o incorporado posteriormente, se suele ofrecer en forma de paquetes y cada distribución suele aportar software de gestión para mantener listas de paquetes instalados o por instalar, así como control de versiones, o posibilidades de actualización de diferentes fuentes.

3.1. Herramientas gráficas y líneas de comandos

Existen muchas más herramientas de las que examinamos en esta unidad, y de las que veremos en las unidades siguientes que son más concretamente de administración, ya sean proporcionadas por terceros independientes a la distribución o por el mismo distribuidor del sistema GNU/Linux.

Estas herramientas pueden cubrir más o menos aspectos de la administración de una tarea concreta y presentarse con múltiples interfaces diferentes: ya sean herramientas de línea de comandos con múltiples opciones y/o ficheros de configuración asociados o herramientas textuales con algún tipo de menús elaborados; o bien herramientas gráficas con interfaces más adecuadas para el manejo de información, o asistentes que automaticen las tareas, o bien interfaces web de administración.

Nota

Las herramientas gráficas de administración no suelen ofrecer una funcionalidad completa, y es interesante conocer cuáles son los efectos de sus acciones.

APLICACIONES

Todo esto nos ofrece un gran número de posibilidades de cara a la administración, pero siempre tenemos que valorar su facilidad de uso con las prestaciones y los conocimientos que posea el administrador que se dedica a estas tareas.

Las tareas habituales del administrador GNU/Linux pueden pasar por trabajar con diferentes distribuciones (por ejemplo, las que comentaremos Red Hat [Inc03a] o Debian [Deb03b], o cualquier otra), o incluso trabajar con variantes comerciales de otros UNIX. Esto conlleva que tengamos que establecer una cierta manera de trabajar que nos permita hacer de forma uniforme las tareas en los diferentes sistemas.

Por eso, a lo largo de las unidades intentaremos destacar todos aquellos aspectos más comunes, y las técnicas de administración serán realizadas en su mayor parte a bajo nivel, mediante una línea de comandos o con edición de ficheros de configuración asociados.

Cualquiera de las distribuciones de GNU/Linux suele aportar herramientas del tipo línea de comandos, textual o gráfico, que complementan las anteriores y simplifican en mayor o menor medida la administración de las tareas [S+02]. Pero hay que tener en cuenta varias cosas:

a) Estas herramientas son una interfaz más o menos elaborada de las herramientas básicas de línea de comandos y los correspondientes ficheros de configuración.

b) Normalmente no ofrecen todas las prestaciones o configuraciones que pueden realizarse a bajo nivel.

c) Los errores pueden no gestionarse bien, o simplemente proporcionar mensajes tipo "la tarea no se ha podido realizar".

d) El uso de estas herramientas oculta, a veces completamente, el funcionamiento interno del servicio o tarea. Comprender bien el funcionamiento interno es un conocimiento básico para el administrador, y más si tiene que desarrollar tareas de corrección de errores u optimización de servicios.

ANOTACIONES

e) Estas herramientas son útiles en la mejora de la producción, una vez que el administrador tiene los conocimientos adecuados, puede manejar con ellas de forma más eficaz las tareas rutinarias y automatizarlas.

f) O también el caso contrario, la tarea puede ser tan compleja, o necesitar tantos parámetros, o generar tantos datos, que se vuelve imposible controlarla de forma manual. En estos casos, las herramientas de alto nivel pueden ser muy útiles y volver practicables algunas tareas que de otra manera son difíciles de controlar. Por ejemplo, dentro de esta categoría entrarían las herramientas de visualización, monitorización o resumen de actividades o servicios complejos.

g) En la automatización de tareas, estas herramientas (de más alto nivel) pueden no ser las adecuadas, pueden no haber estado pensadas para los pasos que hay que realizar, o bien hacerlo de una forma no eficaz. Por ejemplo, un caso concreto puede ser la creación de usuarios, una herramienta visual puede ser muy atrayente, por la forma de introducir los datos, pero ¿qué pasa cuando en lugar de introducir uno o pocos usuarios queremos introducir una lista de decenas o centenares de éstos?, la herramienta, si no está preparada, se vuelve totalmente ineficiente.

h) Por ultimo, los administradores suelen querer personalizar sus tareas utilizando las herramientas que consideran más cómodas y fáciles de adaptar. En este aspecto, suele ser habitual la utilización de las herramientas básicas de bajo nivel y la utilización de *shell scripts* (veremos los fundamentos en esta unidad) para combinarlas de modo que formen una tarea.

Tenemos que saber valorar estas herramientas extra según la valía que tengan para nuestras tareas.

Podemos dar a estas herramientas un uso casual, si tenemos los conocimientos suficientes para tratar los errores que puedan producirse, o bien facilitar algún proceso para el que haya sido pensada la herramienta, pero siempre controlando lo que hacemos.

ANOTACIONES

3.2. Documentos de estándares

Los estándares, ya sean genéricos del mundo UNIX o particulares de GNU/Linux, nos permiten seguir unos criterios básicos, por los que nos guiamos en el momento de aprender o realizar una tarea, y que nos proporcionan información básica para comenzar nuestro trabajo.

En GNU/Linux podemos encontrarnos con estándares como el FHS (*Filesystem Hierarchy Standard*) [Lin03b], que nos explica qué podemos encontrarnos (o dónde buscarlo) en la estructura del sistema de ficheros de nuestro sistema. O el LSB (*Linux Standard Base*), que nos comenta diferentes componentes que solemos encontrar en los sistemas [Lin03c].

Nota

Ver FHS en:
http://www.pathname.com/fhs

En el **estándar FHS** (***Filesystem Hierchachy Standard***) se describen la estructura de árbol del sistema de ficheros principal (/), donde se especifica la estructura de los directorios y los principales ficheros que contendrán. Este estándar es usado en mayor o menor medida también para los UNIX comerciales, en los cuales al principio hubo muchas diferencias que hicieron que cada fabricante cambiara la estructura a su gusto. El estándar pensado en origen para GNU/Linux se hizo para normalizar esta situación y evitar cambios drásticos. Aun así, el estándar es seguido con diferentes grados, la mayoría de distribuciones siguen en un alto porcentaje el FHS, realizando cambios menores o aportando ficheros o directorios que no existían en el estándar.

Un esquema básico de directorios podría ser:

Nota

El estándar FHS es una herramienta básica para el conocimiento de una distribución, que nos permite conocer la estructura y funcionalidad del sistema de archivos principal del sistema.

- /bin: utilidades de base del sistema, normalmente programas empleados por los usuarios, ya sean desde los comandos básicos del sistema (como /bin/ls, listar directorio), pasando por los *shells* (/bin/bash), etc.

- /boot: archivos necesarios durante el arranque del sistema, por ejemplo la imagen del *kernel* Linux, en /boot/vmlinuz.

- /dev: aquí encontramos ficheros especiales que representan los dispositivos posibles en el sistema, el acceso a los periféricos en sistemas UNIX se hace como si fueran periféricos. Podemos encontrar ficheros como /dev/console, /dev/modem, /dev/mouse, /dev/cdrom, /dev/floppy, ... que suelen ser enlaces a dispositivos más específicos del tipo de controlador o interfaz que utilizan los dispositivos: /dev/mouse ⇒ /dev/psaux, un ratón de tipo PS2; o /dev/cdrom ⇒ /dev/hdc, un CD-ROM que es un dispositivo del segundo conector IDE y máster. Aquí encontramos los dispositivos IDE como /dev/hdx, los *scsi* /dev/sdx, ... con *x* variando según el dispositivo.

- /etc: ficheros de configuración. La mayoría de tareas de administración necesitarán examinar o modificar los ficheros contenidos en este directorio. Por ejemplo: /etc/passwd contiene la información de las cuentas de los usuarios del sistema.

- /home: contiene las cuentas de los usuarios, es decir, los directorios personales de cada usuario.

- /lib: las bibliotecas del sistema, compartidas por los programas de usuario, ya sean estáticas (extensión .a) o dinámicas (extensión .so). Por ejemplo, la biblioteca C estándar, en ficheros *libc.so* o *libc.a*.

- /mnt: punto para montar (comando *mount*) sistemas de ficheros extraíbles; por ejemplo: /mnt/cdrom, para montar el lector de CD-ROM.

- /opt: suele colocarse el software añadido al sistema posterior a la instalación; otra instalación válida es en /usr/local.

- /sbin: utilidades de base del sistema. Suelen ser comandos reservados al administrador (*root*). Por ejemplo: /sbin/fsck para verificar el estado de los sistemas de ficheros.

- /tmp: ficheros temporales de las aplicaciones o del propio sistema.

- /*usr*: diferentes elementos instalados en el sistema. Algún software de sistema más completo se instala aquí, además de complemen-

ANOTACIONES

tos multimedia (iconos, imágenes, sonidos, por ejemplo en: /usr/share) y la documentación del sistema (/usr/doc). También en /usr/local se suele utilizar para instalar software.

- /var: ficheros de registro de sesión o de estado (ficheros de tipo *log*) y/o errores del propio sistema y de diversos servicios, tanto locales como de red. Por ejemplo, ficheros de sesión en /var/log, contenido de los mails en /var/spool/mail, o trabajos de impresión en /var/spool/lpd.

Respecto a las distribuciones, Red Hat sigue el estándar FHS muy de cerca. Sólo presenta algunos cambios en los archivos presentes en /usr, /var. En /etc suele existir un directorio por componente configurable, y en /opt, /usr/local no suele existir software instalado, a no ser que el usuario lo instale. Debian, por su parte, sigue el estándar, aunque añade algunos directorios de configuración en /etc.

Nota

http://www.linuxbase.org/spec/refspecs/LSB_1.3.0/gLSB/gLSB/rstandards.html

Otro estándar en proceso es el **LSB** (*Linux Standard Base*) [Lin03c]. La idea de éste es definir unos niveles de compatibilidad entre las aplicaciones, bibliotecas y utilidades, de manera que sea posible la portabilidad de las aplicaciones entre distribuciones sin demasiados problemas. Además del estándar, proporcionan conjuntos de prueba (tests) para verificar el nivel de compatibilidad. LSB en sí mismo es un recopilatorio de varios estándares aplicados a GNU/Linux.

Nota

http://www.linuxbase.org/spec/

3.3. Documentación del sistema en línea

Uno de los aspectos más importantes para nuestras tareas de administración será disponer de la documentación correcta para nuestro sistema y el software instalado. Hay muchas fuentes de información, pero destacaremos las siguientes:

a) **man** es la ayuda por excelencia. Nos permite consultar el manual de GNU/Linux, que está agrupado en varias secciones, correspondientes a comandos administración, formatos de fi-

ANOTACIONES

cheros, comandos de usuario, llamadas de lenguaje C, etc. Normalmente, para obtener la ayuda asociada, tendremos suficiente con:

`man comando`

Cada página describiría el comando junto con sus opciones y, normalmente, algunos ejemplos de utilización. A veces, puede haber más de una entrada en el manual. Por ejemplo, puede que haya una llamada C con igual nombre que un comando; en este caso, hay que especificar qué sección quiere mirarse:

`man n comando` siendo *n* el número de sección.

Existen también unas cuantas herramientas de exploración de los manuales, por ejemplo *xman y tkman*, que mediante interfaz gráfica facilitan el examen de las diferentes secciones, así como índices de los comandos. Otro comando interesante es *apropos palabra*, que nos puede servir para localizar páginas *man* que hablen de un tema determinado (asociado con la palabra buscada).

b) **info** es otro sistema de ayuda habitual. Es un programa desarrollado por GNU para la documentación de muchas de sus herramientas. Es básicamente una herramienta textual en la que los capítulos y páginas se pueden recorrer por medio de un sistema de navegación simple (basado en teclado).

c) Documentación de las aplicaciones: además de ciertas páginas *man*, es habitual incluir en las aplicaciones documentación extra, ya sea en forma de manuales o tutoriales, o simples guías de usuario. Normalmente, estos componentes de documentación se instalan en el directorio /usr/doc, en donde se crea un directorio por paquete de aplicación.

d) Sistemas propios de las distribuciones. Red Hat suele venir con unos CD de manuales de consulta que son instalables en el sistema y tienen formatos HTML o PDF. Debian trae los manuales como un paquete de software más y suelen instalarse en /usr/doc. Por otra parte, dispone de herramientas que clasifican la documentación presente en el sistema, y la organizan por menús para su visualización, como

ANOTACIONES

dwww o *dhelp*, las cuales presentan interfaces web para examinar la documentación del sistema.

e) Por último, los escritorios X, como Gnome y KDE, normalmente también traen sistemas de documentación propios con su documentación y manuales, así como información para desarrolladores, ya sea en forma de ayudas gráficas en sus aplicaciones, o en aplicaciones propias que recopilan las ayudas (por ejemplo *devhelp* en Gnome).

3.4. *Shells* y *scripts*

El término genérico *shell* se utiliza para denominar un programa que sirve de interfaz entre el usuario y el núcleo (*kernel*) del sistema GNU/Linux. En este apartado nos centraremos en los *shells* interactivos de texto, que serán los que nos encontraremos como usuarios una vez estemos validados y dentro del sistema.

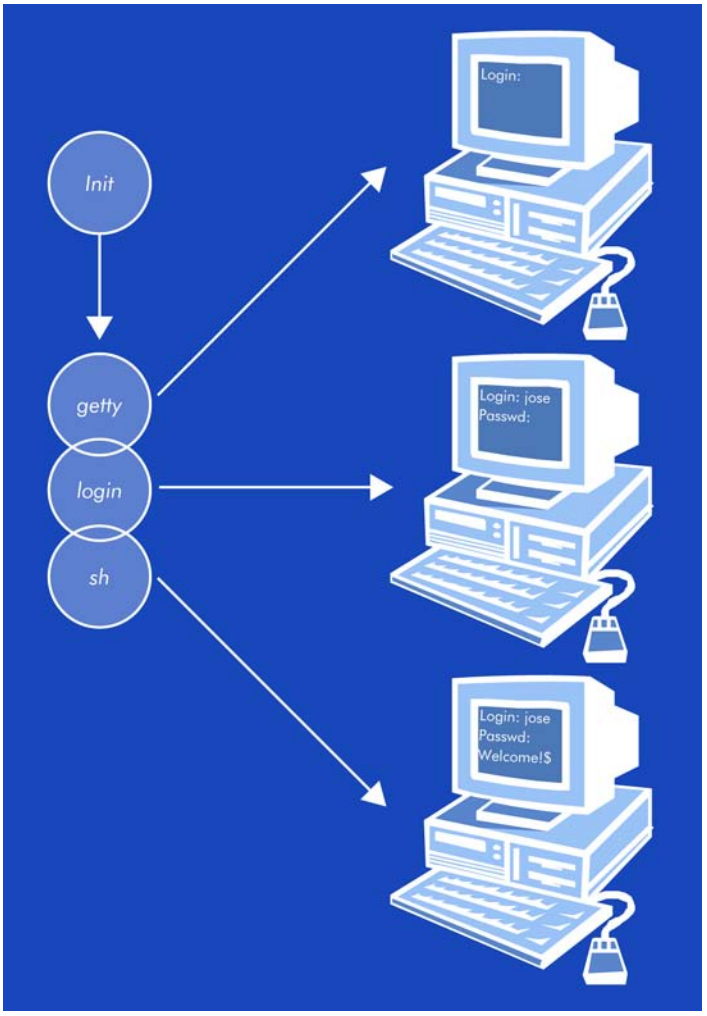
El *shell*, como programa, es una utilidad que permite a los usuarios interactuar con el *kernel* por interpretación de comandos que el mismo usuario introduce en la línea de comandos o en los ficheros de tipo *shell script*.

El *shell* es lo que los usuarios ven del sistema. El resto del sistema operativo permanece esencialmente oculto a sus ojos. El *shell* está escrito de la misma forma que un proceso (programa) de usuario; no está integrado en el *kernel*, sino que se ejecuta como un programa más del usuario.

Cuando nuestro sistema GNU/Linux arranca, suele presentar a los usuarios una interfaz de cara determinada; esta interfaz puede ser tanto de texto, como gráfica. Dependiendo de los modos (o niveles) de arranque del sistema, ya sea con los diferentes modos de consola de texto o con modos donde directamente tengamos arranque gráfico en X Window.

ANOTACIONES

Figura 6. Ejemplo de arranque *shell* textual y procesos de sistema involucrados [O'K00]



En los modos de arranque gráfico, la interfaz está compuesta por algún administrador de acceso que gestiona el proceso de *login* del usuario desde una "carátula" gráfica, en la que se le pide la información de entrada correspondiente: su identificador como usuario y su palabra de paso (o *passsword*). En GNU/Linux suelen ser habituales los gestores de acceso: *xdm* (propio de X Window), *gdm* (Gnome) y *kdm* (KDE), así como algún otro asociado a diferentes gestores de ventanas (*window managers*). Una vez validado nuestro acceso, nos encontraremos dentro de la interfaz gráfica de X Window con algún gestor de ventanas, como Gnome o KDE. Para interactuar desde un *shell* interactivo, sólo tendremos que abrir alguno de los programas de emulación de terminal disponibles.

Si nuestro acceso es por modo consola (en texto), una vez validados obtendremos el acceso directo al *shell* interactivo.

ANOTACIONES

Otro caso de obtención de un *shell* interactivo es el acceso remoto a la máquina, ya sea vía cualquiera de las posibilidades de texto como telnet, rlogin, ssh, o gráficas como los emuladores X Window.

3.4.1. *Shells* interactivos

Una vez iniciado el *shell* interactivo [Qui01], se muestra un *prompt* de cara al usuario, indicándole que puede introducir una línea de comando. Tras la introducción, el *shell* asume la responsabilidad de validarla y poner los procesos necesarios en ejecución, mediante una serie de fases:

- Leer e interpretar la línea de comandos.
- Evaluar los caracteres "comodín" como `$ * ?` u otros.
- Gestionar las redirecciones de E/S necesarias, los *pipes* y los procesos en segundo plano (*background*) necesarios (&).
- Manejar señales.
- Preparar la ejecución de los programas.

Normalmente, las líneas de comandos podrán ser ejecuciones de comandos del sistema, comandos propios del *shell* interactivo, puesta en marcha de aplicaciones o *shell scripts*.

Los *shell scripts* son ficheros de texto que contienen secuencias de comandos de sistema, más una serie de comandos propios del *shell* interactivo, más las estructuras de control necesarias para procesar el flujo del programa (tipo *while*, *for*, etc.).

Los ficheros *script* son directamente ejecutables por el sistema bajo el nombre que se haya dado al fichero. Para ejecutarlos, se invoca el *shell* junto con el nombre del fichero, o bien se dan permisos de ejecución al *shell script*.

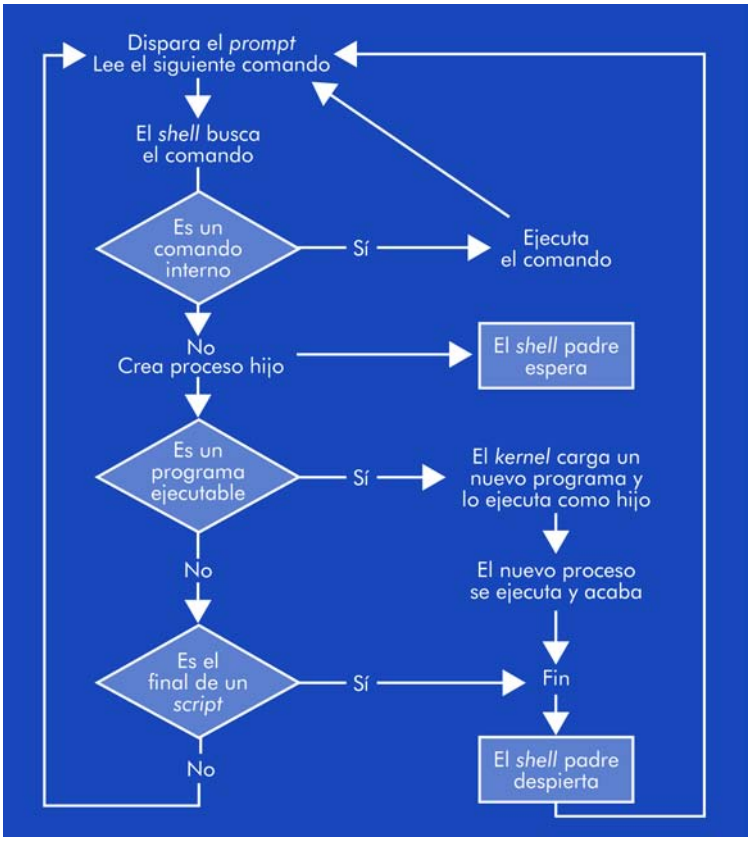
ANOTACIONES

En cierta manera, podemos ver el *shell script* como código de un lenguaje interpretado que se ejecuta sobre el *shell* interactivo correspondiente. Para el administrador, los *shell scripts* son muy importantes básicamente por dos razones:

1) La configuración del sistema y de la mayoría de los servicios proporcionados se hacen mediante herramientas proporcionadas en forma de *shell scripts*.

2) La principal forma de automatizar procesos de administración es mediante la creación de *shell scripts* por parte del administrador.

Figura 7. Flujo de control básico de un *shell*



Todos los programas invocados mediante un *shell* poseen tres ficheros predefinidos, especificados por los correspondientes descriptores de ficheros (*file handles*). Por defecto, estos ficheros son:

1) *standard input* (entrada estándar): normalmente asignada al teclado del terminal (consola); usa el descriptor número 0 (en UNIX los ficheros utilizan descriptores enteros).

ANOTACIONES

2) *standard output* (salida estándar): normalmente asignada a la pantalla del terminal; usa el descriptor 1.

3) *standard error* (salida estándar de errores): normalmente asignada a la pantalla del terminal; utiliza el descriptor 2.

Esto nos indica que cualquier programa ejecutado desde el *shell* tendrá por defecto la entrada asociada al teclado del terminal, su salida hacia la pantalla y, en el caso de producirse errores, también los envía a la pantalla.

Además, los *shells* suelen proporcionar los tres mecanismos siguientes:

1) **Redirección**: dado que los dispositivos de E/S y los ficheros se tratan de la misma manera en UNIX, el *shell* los trata a todos simplemente como ficheros. Desde el punto de vista del usuario, se pueden redefinir los descriptores de los ficheros para que los flujos de datos de un descriptor vayan a cualquier otro descriptor; a esto se le llama *redirección*. Por ejemplo, nos referiremos a la redirección de los descriptores 0 o 1 como a la redirección de la E/S estándar.

2) **Tuberías** (*pipes*): la salida estándar de un programa puede usarse como entrada estándar de otro por medio de *pipes*. Varios programas pueden ser conectados entre sí mediante *pipes* para formar lo que se denomina un *pipeline*.

3) **Concurrencia** de programas de usuario: los usuarios pueden ejecutar varios programas simultáneamente, indicando que su ejecución se va a producir en segundo plano (*background*), en términos opuestos a primer plano (o *foreground*), donde se tiene un control exclusivo de pantalla. Otra utilización consiste en permitir trabajos largos en segundo plano cuando interactuamos con el *shell* con otros programas en primer plano.

En los *shells* UNIX/Linux estos tres aspectos suponen en la práctica:

- **Redirección**: un comando va a poder recibir su entrada o salida desde otros ficheros o dispositivos.

ANOTACIONES

Ejemplo

Por ejemplo:

comando op *fichero* donde op puede ser:

- < : recibir entrada del fichero.
- > : enviar salida al fichero.
- >>: indica que se añada la salida (por defecto, con > se crea de nuevo el fichero).

- *Pipes*: encadenamiento de varios comandos, con transmisión de sus datos:

```
comando1 | comando2 | comando3
```

Esta instrucción nos indica que *comando1* recibirá entrada posiblemente de teclado, enviará su salida a *comando2*, que la recibirá como entrada, y éste producirá salida hacia *comando3* , que la recibe y produce su salida hacia salida estándar (la pantalla, por defecto).

- **Concurrencia** en segundo plano: cualquier comando ejecutado con el '&' al final de línea se ejecuta en segundo plano y el*prompt* del *shell* se devuelve inmediatamente mientras continúa su ejecución. Podemos seguir la ejecución de los comandos con el comando *ps* y sus opciones, que nos permite ver el estado de los procesos en el sistema. Y también disponemos de la orden *kill*, que nos permite eliminar procesos que todavía se estén ejecutando o que hayan entrado en alguna condición de error: *kill 9 pid* permite "matar" el proceso número *pid*. *pid* es el identificador asociado al proceso, un número entero que el sistema le asigna y que puede obtenerse con el comando *ps*.

3.4.2. *Shells* disponibles

La independencia del *shell* respecto al *kernel* del operativo (el *shell* es sólo una capa de interfaz), nos permite disponer de varios de ellos en el sistema [Qui01]. Algunos de los más comunes son:

a) El *shell* Bash (*bash*). El *shell* Linux por defecto.

ANOTACIONES

b) El *shell* Bourne (*sh*). Éste ha sido desde siempre el *shell* estándar UNIX, y el que todos los UNIX poseen en alguna versión. Normalmente, es el *shell* por defecto del administrador (*root*). En GNU/Linux suele ser el Bash, una versión mejorada del Bourne. El *sh* fue creado por Stephen Bourne en AT&T a finales de los setenta. El indicador (o *prompt*) por defecto suele ser un '$' (en *root* un '#').

c) El *shell* Korn (*ksh*). Es un superconjunto del Bourne (se mantiene cierta compatibilidad), escrito en AT&T por David Korn (a mediados de los ochenta), en el cual se hizo cierta mezcla de funcionalidades del Bourne y del C, más algún añadido. El *prompt* por defecto es el $.

d) El *shell* C (*csh*). Fue desarrollado en la Universidad de Berkeley por Bill Joy a finales de los setenta y tiene unos cuantos añadidos interesantes al Bourne, como un histórico de comandos, alias, aritmética desde la línea de comandos, completa nombres de ficheros y control de trabajos en segundo plano. El *prompt* por defecto para los usuarios es '%'. Los usuarios UNIX suelen preferir este *shell* como interactivo, pero los administradores UNIX prefieren utilizar el Bourne, ya que los *scripts* suelen quedar más compactos, y la ejecución suele ser más rápida. Por otro lado, una ventaja de los *scripts* en C shell es que, como su nombre indica, su sintaxis está basada en el lenguaje C.

e) Otros, como versiones restringidas o especializadas de los anteriores.

Nota

El *shell* es un programa más de usuario, existiendo diferentes posibilidades de elección, con funcionalidades y prestaciones diferentes.

El *shell* Bash (*Bourne Again Shell*) [G00] [Coo03] ha adquirido importancia desde su inclusión en los sistemas GNU/Linux como *shell* por defecto. Este *shell* forma parte del software GNU. Es un intento de combinar los tres *shell* anteriores (Bourne, C y Korn), manteniendo la sintaxis del *shell* Bourne original. Es en el que nos vamos a fijar para desarrollar ejemplos posteriores.

Una forma rápida de conocer bajo qué *shell* nos encontramos como usuarios es mediante la variable *$SHELL*, desde una línea de comandos con la instrucción:

```
echo $SHELL
```

Algunas cuestiones que encontraremos comunes a todos los *shells*:

- Todos permiten la escritura de *shell scripts* que son luego interpretados ejecutándolos bien por el nombre (si el fichero tiene permiso de ejecución) o bien pasándolo como parámetro al comando del *shell*.

ANOTACIONES

- Los usuarios del sistema tienen un *shell* por defecto asociado a ellos. Esta información se proporciona al crear las cuentas de los usuarios. El administrador asigna un *shell* a cada usuario, o bien si no se asigna el *shell* por defecto (*bash* en GNU/Linux). Esta información se guarda en el fichero de *passwords* en /etc/passwd.

- Cada *shell* es en realidad un comando ejecutable, normalmente presente en los directorios /bin en GNU/Linux (o /usr/bin).

- Se pueden escribir *shell scripts* en cualquiera de ellos, pero ajustándose a la sintaxis de cada uno, que es normalmente diferente (a veces hay sólo pequeñas diferencias). La sintaxis de las construcciones, así como los comandos internos, están documentados en la página man de cada *shell*.

- Cada *shell* tiene algunos ficheros de arranque asociados (ficheros de inicialización), cada usuario puede adaptarlos a sus necesidades, incluyendo código, variables, caminos (*path*), ...

- La potencia en la programación está en combinar la sintaxis de cada *shell* (de sus construcciones), con los comandos internos de cada *shell*, y una serie de comandos UNIX muy utilizados en los *scripts*, como por ejemplo los *grep, sed, awk*.

- Si como usuarios estamos utilizando un *shell* determinado, nada impide arrancar una copia nueva de *shell* (lo llamamos *subshell*), ya sea el mismo u otro diferente. Sencillamente, lo invocamos por el nombre del ejecutable, ya sea el *sh*, `bash`, *csh* o *ksh*. También cuando ejecutamos un *shell script* se lanza un *subshell* con el *shell* que corresponda para ejecutar el *script* pedido.

Algunas diferencias básicas entre ellos [Qui01]:

a) Bash es el *shell* por defecto en GNU/Linux (si no se especifica lo contrario al crear la cuenta del usuario). En otros sistemas UNIX suele ser el *shell* Bourne (*sh*). Bash es compatible con *sh*, y además incorpora algunas características de los otros *shells*, *csh* y *ksh*.

b) Ficheros de arranque: *sh*, *ksh* tienen *.profile* (en la cuenta del usuario, y se ejecuta en el *login* del usuario) y también *ksh* suele tener un *.kshrc* que se ejecuta a continuación, *csh* utiliza *.login* (se ejecuta al iniciarse el *login* del usuario una sola vez), *.logout* (an-

ANOTACIONES

tes de la salida de la sesión del usuario) y *.cshrc* (parecido al *.profile*, en cada *subshell* C que se inicia). Y Bash utiliza el *.bashrc* y el *.bash_profile*. Además, el administrador puede colocar variables y caminos comunes en el fichero /etc/profile que se ejecutará antes que los ficheros que tenga cada usuario. Los ficheros de inicialización de los *shell* se ponen en la cuenta del usuario al crearla (normalmente se copian del directorio /etc/skel), donde el administrador puede dejar unos esqueletos de los ficheros preparados.

c) Los *scripts* de configuración del sistema o de servicios suelen estar escritos en *shell* Bourne (*sh*), ya que la mayoría de los UNIX así los tenían. En GNU/Linux también nos podemos encontrar con algunos en Bash.

d) Podemos identificar en qué *shell* se ejecuta el *script* mediante el comando *file*, por ejemplo *file <nombrescript>*. O bien examinando la primera línea del *script*, que suele ser: #!/bin/nombre, donde nombre es `bash`, *sh*, *csh*, *ksh* ... Esta línea le dice, en el momento de ejecutar el *script,* qué *shell* hay que utilizar a la hora de interpretarlo (o sea, qué *subshell* hay que lanzar para ejecutarlo). Es importante que todos los *scripts* la contengan, ya que si no, se intentarán ejecutar en el *shell* por defecto (Bash en nuestro caso), y la sintaxis puede no corresponder, causando muchos errores sintácticos en la ejecución.

3.4.3. Variables de sistema

Algunas variables de sistema útiles que pueden consultarse (con el comando *echo*), son:

Variable	Valor ejemplo	Descripción
HOME	/home/juan	Directorio raíz del usuario
LOGNAME	juan	Id del usuario en el *login*
PATH	/bin:/*usr*/local/bin:/*usr*/X11/bin	Caminos
SHELL	/bin/*bash*	*Shell* del usuario
PS1	$	*Prompt* del *shell*, el usuario puede cambiarlo
MAIL	/var/mail/juan	Directorio del buzón de correo
TERM	xterm	Tipo de terminal que el usuario utiliza
PWD	/home/juan	Directorio actual del usuario

Las diferentes variables del entorno pueden verse con el comando *env*. Por ejemplo:

```
$ env
SSH_AGENT_PID = 598
MM_CHARSET = ISO-8859-15
TERM = xterm
DESKTOP_STARTUP_ID =
SHELL = /bin/bash
WINDOWID = 20975847
LC_ALL = es_ES@euro
USER = juan
LS_COLORS = no = 00:fi = 00:di = 01;34:ln = 01;
SSH_AUTH_SOCK = /tmp/ssh-wJzVY570/agent.570
SESSION_MANAGER = local/aopcjj:/tmp/.ICE-unix/570
USERNAME = juan
PATH = /soft/jdk/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
MAIL = /var/mail/juan
PWD = /etc/skel
JAVA_HOME = /soft/jdk
LANG = es_ES@euro
GDMSESSION = Gnome
JDK_HOME = /soft/jdk
SHLVL = 1
HOME = /home/juan
GNOME_DESKTOP_SESSION_ID = Default
LOGNAME = juan
DISPLAY = :0.0
COLORTERM = gnome-terminal
XAUTHORITY = /home/juan/.Xauthority
_ = /usr/bin/env
OLDPWD = /etc
```

ACTIVIDADES

3.4.4. Programación *scripts* en Bash

Aquí veremos algunos conceptos básicos de los *shell script* en Bash, se recomienda ampliar en [G00] [Coo03].

Todos los *scripts* Bash tienen que comenzar con la línea:

```
#!/bin/bash
```

Esta línea le indica al *shell* usado por el usuario, el que tengamos activo en el momento, con qué *shell* hay que ejecutar el *script* que aparezca a continuación.

El *script* puede ejecutarse de dos modos diferentes:

1) Por ejecución directa desde la línea de comandos, siempre que tenga permiso de ejecución. Si no se da el caso, ponemos el permiso con: `chmod +x script`.

2) Por ejecución mediante el *shell*, llamamos al *shell* explícitamente: `bash script`.

Hay que tener en cuenta que, sea cuál sea el método de ejecución, siempre estamos creando un *subshell* donde se va a ejecutar nuestro *script*.

Variables en Bash

La asignación de variables se realiza por:

```
variable = valor
```

El valor de la variable se puede ver con:

```
echo $variable
```

donde '$' nos hace referencia al valor de la variable.

La variable por defecto sólo es visible en el *script* (o en el *shell*). Si la variable tiene que ser visible fuera del *script*, a nivel de *shell* o de

ANOTACIONES

cualquier *shell* hijo (o *subshell*) que se genere *a posteriori*, será necesario "exportarla" además de asignarla. Podemos hacer dos cosas:

- Asignar y exportar después:

```
var = valor
export var
```

- Exportar en la asignación:

```
export var = valor
```

En los *scripts* Bash tenemos algunas variables predeterminadas accesibles:

- `$1-$N`: Guarda los argumentos pasados como parámetros al *script* desde la línea de comandos.
- `$0` : Guarda el nombre del *script*, sería el parámetro 0 de la línea de comandos.
- `$*` : Guarda todos los parámetros del 1 al *N* en esta variable.
- `$` : Guarda todos los parámetros, pero con comillas dobles (" ") en cada uno de ellos.
- `$?` , "Status": guarda el valor devuelto por el último comando ejecutado. Útil para verificar condiciones de error, ya que UNIX suele devolver 0 si la ejecución ha sido correcta, y un valor diferente como código de error.

Otra cuestión importante en las asignaciones es el uso de las comillas:

- Las "dobles" permiten que sea considerado todo como una unidad.
- Las 'simples' son parecidas, pero se ignoran los caracteres especiales que se encuentren dentro.

ANOTACIONES

- Las inclinadas hacia la izquierda `comando`, son utilizadas para evaluar el interior, si hay alguna ejecución o sustitución que hacer. Primero se ejecuta el contenido, y se sustituye lo que había por el resultado de la ejecución. Por ejemplo: var = `ls` guarda el listado del directorio en $var.

Comparaciones

Para las condiciones se suele utilizar la orden *test expresión* o directamente [*expresión*]. Podemos agrupar las condiciones disponibles en:

- Comparación numérica: *-eq*, *-ge*, *-gt*, *-le*, *-lt*, *-ne*, correspondiendo a: igual que, más grande o igual que (ge), más grande que, menor o igual que (le), menor que, distinto que.

- Comparación de cadenas: = , ! = , *-n*, *-z*, correspondiendo a cadenas de caracteres: iguales, diferentes, con longitud mayor que 0, longitud igual a cero o vacío.

- Comparación de ficheros: *-d*, *-f -r*, *-s*, *-w*, *-x*. El fichero es: un directorio, un fichero ordinario, es leíble, es no vacío, es escribible, es ejecutable.

- Booleanos entre expresiones: !, *-a*, *-o*, condiciones de *not*, *and* y *or*.

Estructuras de control

Respecto a la programación interna del *script*, hay que pensar que nos vamos a encontrar básicamente con:

- Comandos propios del operativo.

- Comandos propios internos del *shell* Bash (ver: *man bash*).

- Las estructuras de control propias de programación (*for*, *while*, ...), con la sintaxis propia de *Bash*.

ANOTACIONES

La sintaxis básica de las estructuras de control es la siguiente:

a) Estructura *if...then*, se evalúa la expresión y si se obtiene un valor cierto, entonces se ejecutan los *commands*.

```
if [ expresion ]
   then
      commands
fi
```

b) Estructura *if..then...else*, se evalúa la expresión, y si se obtiene un valor de cierto, entonces se ejecutan los*commands1* , en caso contrario se ejecutan los *commands2*:

```
if [ expresion ]
   then
      commands1
   else
      commands2
fi
```

c) Estructura *if..then...else if...else*, misma utilización que la anterior, con anidamientos de estructuras *if*.

```
if [ expresion ]
   then
      commands
   elif [ expresion2 ]
      then
         commands
   else
      commands
fi
```

d) Estructura *case select*, estructura de selección múltiple según valor de selección (en *case*)

```
case string1 in
   str1)
      commands;;
   str2)
      commands;;
   *)
      commands;;
esac
```

Nota

Los *shells* como Bash, ofrecen un conjunto amplio de estructuras de control que los hace comparables a cualquier otro lenguaje.

e) Bucle *for*, sustitución de variable por cada elemento de la lista:

```
for var1 in list
do
   commands
done
```

f) Bucle *while*, mientras se cumpla la expresión:

```
while [ expresion ]
do
   commands
done
```

g) Bucle *until*, hasta que se cumpla la expresión:

```
until [ expresion ]
do
   commands
done
```

h) Declaración de funciones:

```
fname() {
   commands
}
```

o bien con llamada acompañada de parámetros:

```
fname2(arg1,arg2...argN) {
   commands
}
```

y la llamadas de la función con *fname* o *fname2 p1 p2 p3 ... pN*.

3.5. Herramientas de gestión de paquetes

En cualquier distribución, los paquetes son el elemento básico para tratar las tareas de instalación de nuevo software, actualización del existente o eliminación del no utilizado.

ANOTACIONES

Básicamente, un **paquete** es un conjunto de ficheros que forman una aplicación o una unión de varias aplicaciones relacionadas, normalmente formando un único fichero (denominado *paquete*), con un formato propio y normalmente comprimido, que es el que se distribuye, ya sea vía CD, disquete o mediante acceso a servicios de ftp o web.

El uso de paquetes facilita añadir o quitar software al considerarlo una unidad y no tener que trabajar con los ficheros individuales.

En el contenido de la distribución (sus CD) los paquetes suelen estar agrupados por categorías como: a) base: paquetes indispensables para el funcionamiento del sistema (útiles, programas de inicio, bibliotecas de sistema); b) sistema: útiles de administración, comandos de utilidad; c) desarrollo (*development*): útiles de programación: editores, compiladores, depuradores,... d) gráficos: controladores e interfaces gráficas, escritorios, gestores de ventanas, ... e) otras categorías.

Normalmente, para la instalación de un paquete será necesario efectuar una serie de pasos:

1) Previo (preinstalación): comprobar que existe el software necesario (y con las versiones correctas) para su funcionamiento (dependencias), ya sean bibliotecas de sistema u otras aplicaciones que sean usadas por el software.

2) Descomprimir el contenido del paquete, copiando los ficheros a sus localizaciones definitivas, ya sean absolutas (tendrán una posición fija) o si se permite reubicarlas a otros directorios.

3) Postinstalación: retocar los ficheros necesarios, configurar posibles parámetros del software, adecuarlo al sistema, ...

Dependiendo de los tipos de paquetes, estos pasos pueden ser automáticos en su mayoría (así es en el caso de RPM [Bai03] y DEB

ANOTACIONES

[Deb02]), o pueden necesitar hacerlos todos a mano (caso *.tgz*) dependiendo de las herramientas que proporcione la distribución.

Veremos a continuación quizás los tres paquetes más clásicos de la mayoría de distribuciones. Cada distribución tiene uno por estándar y soporta alguno de los demás.

3.5.1. Paquete TGZ

Nota

Los paquetes TGZ son una herramienta básica a la hora de instalar software no organizado. Además, son una herramienta útil para realizar procesos de *backup* y restauración de archivos.

Los paquetes TGZ son quizás los de utilización más antigua. Las primeras distribuciones de GNU/Linux los utilizaban para instalar el software, y aún varias distribuciones los usan (por ejemplo, Slackware) y algunos UNIX comerciales. Son una combinación de ficheros unidos por el comando *tar* en un único fichero .tar, que luego ha sido comprimido por la utilidad *gzip*, suele aparecer con la extensión .tgz o bien .tar.gz. Asimismo, hoy en día es común encontrar los tar.bz2 que utilizan en lugar de *gzip* otra utilidad llamada *bzip2*, que en algunos casos consigue mayor compresión del archivo.

Este tipo de paquete no contiene ningún tipo de información de dependencias, y puede presentar tanto contenido de aplicaciones en formato binario como en código fuente. Podemos considerarlo como una especie de colección de ficheros comprimida.

En contra de lo que pudiera parecer, es un formato muy utilizado, y sobre todo por creadores o distribuidores de software externo a la distribución. Muchos creadores de software que trabajan para plataformas varias, como varios UNIX comerciales, y diferentes distribuciones de GNU/Linux lo prefieren como sistema más sencillo y portable.

Ejemplo

Un ejemplo es el proyecto GNU, que distribuye su software en este formato (en forma de código fuente), ya que puede utilizarse en cualquier UNIX, ya sea un sistema propietario, una variante BSD o una distribución GNU/Linux.

ANOTACIONES

Si se trata de formato binario, tendremos que tener en cuenta que sea adecuada para nuestro sistema, por ejemplo, suele ser común alguna denominación como la que sigue (en este caso, la versión 1.4 del navegador web Mozilla):

```
mozilla-i686-pc-linux-gnu-1.4-installer.tar.gz
```

donde tenemos el nombre del paquete, como Mozilla, arquitectura a la que ha destinado i686 (Pentium II o superiores o compatibles), podría ser i386, i586, i686, k6 (*amd k6*), k7 (*amd athlon*) otras para máquinas sparc, powerpc, hppa,... después nos indica qué es para Linux, en una máquina PC, la versión del software 1.4.

Si fuese en formato fuente, suele aparecer como:

```
mozilla-source-1.4.tar.gz
```

donde se nos indica la palabra *source*; en este caso no menciona versión de arquitectura de máquina, esto nos indica que está preparado para compilarse en diferentes arquitecturas.

De otro modo, habría diferentes códigos para cada sistema operativo o fuente: Linux, Solaris, Irix, bsd, ...

El proceso básico con estos paquetes consiste en:

1) Descomprimir el paquete (no suelen utilizar *path* absoluto, con lo que se pueden descomprimir en cualquier parte):

   ```
   tar -zxvf fichero.tar.gz (o fichero.tgz)
   ```

   Con el comando *tar* ponemos opciones de *z*: descomprimir, *x*: extraer ficheros, *v*: ver proceso, *f*: fichero por tratar.

   También se puede hacer por separado (sin la *z* del *tar*):

   `gunzip fichero.tar.gz` (nos deja un fichero *tar*)
   `tar -xvf fichero.tar`

ANOTACIONES

2) Una vez tenemos descomprimido el tgz, tendremos los ficheros que contenía, normalmente el software debe incluir algún fichero de tipo *readme* o *install*, donde nos especificarán las opciones de instalación paso a paso, y también posibles dependencias del software.

En primer lugar habrá que verificar las dependencias por si disponemos del software adecuado, y si no, buscarlo e instalarlo.

Si se trata de un paquete binario, la instalación suele ser bastante fácil, ya que o bien directamente ya será ejecutable donde lo hayamos dejado, o traerá algún instalador propio. Otra posibilidad será que tengamos que hacerlo manualmente, con lo que bastará con copiar (*cp -r*, copia recursiva) o mover (comando *mv*) el directorio a la posición deseada.

Otro caso es el formato de código fuente. Entonces, antes de instalar el software tendremos que pasar por un paso de compilación. Para eso habrá que leerse con cierto detalle las instrucciones que lleve el programa. Pero la mayoría de desarrolladores usan un sistema de GNU llamado *autoconf* (de *autoconfiguración*), en el que habitualmente se usan los siguientes pasos (si no aparecen errores):

- ./configure: se trata de un *script* que configura el código para poder ser compilado en nuestra máquina, verifica que existan las herramientas adecuadas. La opción *--prefix = directorio* permite especificar dónde se instalará el software.

- make: compilación propiamente dicha.

- make install: instalación del software a un lugar adecuado, normalmente especificado previamente como opción al *configure* o asumida por defecto.

Éste es un proceso general, pero depende del software que lo siga o no, hay casos bastante peores donde todo el proceso se tiene que realizar a mano, compilando uno a uno los ficheros, pero esto, por suerte, es cada vez menos habitual.

ANOTACIONES

En caso de querer borrar el software instalado, habrá que utilizar el desinstalador si nos lo proporcionan, o si no, borrar directamente el directorio o ficheros que se instalaron, teniendo cuidado de posibles dependencias.

Los paquetes tgz son bastante habituales como mecanismo de *backup* en tareas de administración, por ejemplo, para guardar copias de datos importantes, hacer *backups* de cuentas de usuario, o guardar copias antiguas de datos que no sabemos si volveremos a necesitar. Suele utilizarse el siguiente proceso: supongamos que queremos guardar copia del directorio "dir" *tar -cvf dir.tar dir* (c: compactar dir en el fichero *dir.tar*) *gzip dir.tar* (comprimir) o bien en una sola instrucción como:

```
tar -zcvf dir.tgz dir
```

El resultado será un fichero *dir.tar.gz*.

3.5.2. Red Hat: paquetes RPM

El sistema de paquetes RPM [Bai03] creado por Red Hat supone un paso adelante, ya que incluye la gestión de dependencias y tareas de configuración del software. Además, el sistema guarda una pequeña base de datos con los paquetes ya instalados, que puede consultarse y se actualiza con las nuevas instalaciones.

Los paquetes RPM, por convención, suelen usar un nombre como:

paquete-*version*-rev.arq.rpm

donde *paquete* es el nombre del software, *version* es la versión del software, *rev* suele ser la revisión del paquete RPM, que indica las veces que se ha construido, y *arq*, la arquitectura a la que va destinado el paquete, ya sea Intel (i386, i586, i686) u otras como Alpha, Sparc, PPC. La arquitectura Noarch suele usarse cuando es independiente, por ejemplo, un conjunto de *scripts*, y *src* en el caso de que se trate de paquetes de código fuente. La ejecución típica incluye la ejecución de *rpm*, las opciones de la operación a realizar, junto con uno o más nombres de paquetes por procesar juntos.

Ejemplo

El paquete:

apache-1.3.19-23.i686.rpm

indicaría que se trata del software Apache (el servidor web), en su versión 1.3.19, revisión del paquete RPM 23, para arquitecturas Pentium II o superiores.

APLICACIONES

Las operaciones típicas con los paquetes RPM incluyen:

Nota

Los paquetes RPM incorporan la idea de gestión de dependencias y de base de datos de los paquetes existentes.

- **Información del paquete**: se consulta sobre el paquete una información determinada, se usa la opción *-qp* acompañada del nombre del paquete instalado. Si el paquete no ha sido instalado todavía, la opción sería *-q* acompañada de la información que se quiera, y si se quiere preguntar a todos los paquetes a la vez, la opción sería *-qa*. Por ejemplo, preguntas a un paquete instalado:

Consulta	Opciones RPM	Resultados
Archivos	rpm -qpl	Lista de los archivos que contiene
Información	rpm -qpi	Descripción del paquete
Requisitos	rpm -qpR	Requisitos previos, bibliotecas o software

- **Instalación**: simplemente *rpm -i paquete.rpm*. Esto podrá realizarse siempre que se estén cumpliendo las dependencias del paquete, ya sea software previo o bibliotecas que deberían estar instaladas. En caso de no cumplirlo, se nos listará qué software falta, y el nombre del paquete que lo proporciona. Puede forzarse la instalación (a riesgo de que no funcione) con las opciones *--force* o *--nodeps*, o simplemente se ignora la información de las dependencias.

- **Actualización**: equivalente a la instalación pero comprobando primero que el software ya existe *rpm -u paquete.rpm*. Se encargará de borrar la instalación previa.

- **Verificación**: durante el funcionamiento normal del sistema, muchos de los archivos instalados cambian. En este sentido, RPM permite verificar los archivos para detectar las modificaciones, bien por proceso normal, bien por algún error que podría indicar datos corrompidos. Mediante *rpm -V paquete* verificamos un paquete concreto, y mediante *rpm -Va* los verificará todos.

- **Eliminación**: borrar el paquete del sistema RPM de paquete; si hay dependencias, puede ser necesario eliminar otros primero. El sistema RPM también ofrece la posibilidad de trabajar con paquetes remotos, que pueden descargarse desde servidores FTP o web, sólo hay que utilizar la sintaxis ftp:// o http:// para dar la localización del paquete.

ANOTACIONES

Ejemplo

Por ejemplo:

```
rpm -i ftp://sitio/directorio/paquete.rpm
```

nos permitiría descargar el paquete desde el sitio ftp o web proporcionado, con su localización de directorios, y proceder en este caso a la instalación del paquete.

Hay que vigilar con la procedencia de los paquetes, y sólo utilizar fuentes de paquetes conocidas y fiables, ya sea del propio fabricante de la distribución, o sitios en los que confiemos. Normalmente se nos ofrecen junto con los paquetes, alguna "firma" digital de estos para que podamos comprobar la autenticidad, suelen utilizarse las firmas md5, u otros sistemas como GPG. Hay también en Internet diferentes almacenes de paquetes RPM, donde están disponibles para diferentes distribuciones que usen o permitan el formato RPM.

Nota

Ver el sitio: www.rpmfind.net.

En cuanto a las distribuciones, en Red Hat (y también en Fedora), RPM es el formato por defecto de paquetes y el que usa ampliamente la distribución para las actualizaciones, y la instalación de software. En Debian se utiliza el formato DEB, hay soporte para RPM (existe el comando *rpm*), pero sólo para consulta o información de paquetes. En el caso de que sea imprescindible instalar un paquete, se recomienda utilizar la utilidad *alien*, que permite convertir formatos de paquetes, en este caso de RPM a DEB, y proceder a la instalación con el paquete convertido.

3.5.3. Debian: paquetes DEB

Debian tiene herramientas interactivas como *tasksel*, que permiten escoger unos subconjuntos de paquetes agrupados por tipo de tareas: paquetes para X, para desarrollo, para documentación, etc., o como *dselect* que nos permite navegar por toda la lista de paquetes disponible (hay miles), y escoger aquellos que queramos instalar o desinstalar.

En el nivel de línea de comandos dispone de *dpkg*, que es el comando de más bajo nivel, para gestionar directamente los paquetes DEB

ANOTACIONES

de software [Deb02], típicamente *dpkg -i paquete.deb* para realizar la instalación. Pueden realizarse todo tipo de tareas, de información, instalación, borrado o cambios internos a los paquetes de software.

Otro nivel intermedio lo presentan las herramientas APT (la mayoría son comandos *apt-xxx*), donde se gestionan los paquetes a través de una lista de paquetes actuales y disponibles a partir de varias fuentes de software, ya sea desde los propios CD de la instalación, sitios ftp o web (HTTP). Esta gestión se hace de forma transparente, de manera que el sistema es independiente de las fuentes de software.

La configuración del sistema APT se efectúa desde los archivos disponibles en /etc/apt, donde /etc/apt/sources.list es la lista de fuentes disponibles; un ejemplo podría ser:

```
deb http://http.us.debian.org/debian stable main contrib non-free
deb http://non-us.debian.org/debian-non-US stable/non-US main contrib non-free
deb http://security.debian.org stable/updates main contrib non-free
debsrc http://http.us.debian.org/debian stable main contrib non-free
debsrc http://nonus.debian.org/debian-non-US stable non-US

#Sources Oficiales de Debian STABLE "Woody"
deb http://ftp.debian.org/debian/ woody main non-free contrib
debsrc http://ftp.debian.org/debian/ woody main non-free contrib
deb http://non-us.debian.org/debian-non-US woody/non-US main contrib non-free
debsrc http://non-us.debian.org/debian-non-US woody/non-US main contrib non-free
```

En que hay recopiladas algunas de las fuentes "oficiales" para una Debian *Woody*, desde donde se pueden obtener los paquetes de software, así como las actualizaciones que estén disponibles. Básicamente, se especifica el tipo de fuente (web en este caso), el sitio, la versión de la distribución (*stable* o *woody*), y categorías del software que se buscará (libre, o contribuciones de terceros o comercial).

Los paquetes de software están disponibles para las diferentes distribuciones Debian, existen paquetes para las versiones *stable*, *testing*, y *unstable*. El uso de unos u otros determina el tipo de distribución. Pueden tenerse fuentes de paquetes mezcladas, pero no es muy re-

Nota

Los paquetes DEB de Debian es quizás el sistema de instalación más potente existente en GNU/Linux. Una prestación destacable es la independencia del sistema de las fuentes de los paquetes (mediante APT).

comendable, ya que se podrían dar conflictos entre las versiones de las diferentes distribuciones.

Una vez tenemos las fuentes de software configuradas, la principal herramienta para manejarlas en nuestro sistema es *apt-get*, que nos permite instalar, actualizar o borrar desde el paquete individual, hasta actualizar la distribución entera. Existe también un reemplazo a *apt-get*, llamado *aptitude*, cuya interfaz de opciones es prácticamente igual; como ventaja aporta una mejor gestión de dependencias de los paquetes y permite una interfaz interactiva.

Algunas funciones básicas de *apt-get*:

- Instalación de un paquete particular:

```
apt-get install paquete
```

- Borrado de un paquete:

```
apt-get remove paquete
```

- Actualización de la lista de paquetes disponibles:

```
apt-get update
```

- Actualización de la distribución, podríamos efectuar los pasos:

```
apt-get update
apt-get upgrade
apt-get dist-upgrade
```

Mediante el último proceso, podemos mantener nuestra distribución actualizada permanentemente, actualizando los paquetes instalados y verificando las dependencias con los nuevos. Una herramienta útil para construir esta lista es *apt-spy*, que intenta buscar los sitios oficiales más rápidos. Por otro lado, podemos buscar las fuentes oficiales (podemos configurarlas con apt-setup), o bien copiar algún fichero de fuentes disponible. Software adicional (de terceros) puede necesitar añadir otras fuentes más, se pueden obtener listas de sitios de fuentes disponibles (por ejemplo: http://www.apt-get.org)

ANOTACIONES

Otra herramienta útil es el comando *apt-cache*, que nos permite interactuar con las listas de paquetes de software Debian.

Ejemplo

La herramienta *apt-cache* dispone de comandos que nos permiten buscar información sobre los paquetes, como por ejemplo:

- Buscar paquetes sobre la base de un nombre incompleto:

```
apt-cache search nombre
```

- Mostrar la descripción del paquete:

```
apt-cache show paquete
```

Otras tareas más especificas necesitarán realizarse con la herramienta de más bajo nivel, como *dpkg*. Por ejemplo, obtener la lista de archivos de un paquete determinado:

```
dpkg -L paquete
```

Cabe destacar que el software APT es muy flexible y potente a la hora de gestionar las actualizaciones, y poco a poco se está imponiendo en las distribuciones. Por ejemplo, en la actual Fedora se puede utilizar para gestionar la instalación de paquetes RPM.

3.6. Herramientas genéricas de administración

En el campo de la administración, también prodríamos considerar algunas herramientas, como las pensadas de forma genérica para la administración:

a) **Linuxconf**: es una herramienta genérica de administración que agrupa diferentes aspectos de administración en una interfaz de menús textual; puede utilizarse en prácticamente cualquier distribución GNU/Linux, y soporta diversos detalles propios de cada una.

ANOTACIONES

b) **Webmin**: es otra herramienta de administración pensada desde interfaz web; funciona con una serie de *plugins* que se pueden añadir para cada servicio a administrar; normalmente cuenta con formularios donde se especifican los parámetros de configuración de los servicios; además, ofrece la posibilidad (si se activa) de permitir administración remota desde cualquier máquina con navegador.

Los entornos de escritorio de Gnome y KDE suelen disponer del concepto de "Panel de control", que permite gestionar tanto el aspecto visual de las interfaces gráficas, como tratar algunos parámetros de los dispositivos del sistema.

En cuanto a las herramientas gráficas individuales de administración, la propia distribución de GNU/Linux ofrece algunas directamente (herramientas que acompañan tanto a Gnome como KDE), herramientas dedicadas a gestionar un dispositivo (impresoras, sonido, tarjeta de red, etc.), y otras para la ejecución de tareas concretas (conexión a Internet, configurar arranque de servicios del sistema, configurar X Window, visualizar *logs*, ...). Muchas de ellas son simples *frontends* (o carátulas) a las herramientas básicas de sistema, o bien están adaptadas a particularidades de la distribución.

3.7. Otras herramientas

En el espacio limitado de esta unidad no pueden llegar a comentarse todas aquellas herramientas que nos pueden aportar beneficios para la administración. Citaremos algunas de las herramientas que podríamos considerar básicas:

- Los múltiples comandos UNIX básicos: *grep*, *awk*, *sed*, *find*, *diff*, *gzip*, *bzip2*, *cut*, *sort*, *df*, *du*, *cat*, *more*, *file*, *which*...

- Los editores, imprescindibles para cualquier tarea de edición, cuentan con editores como: Vi, muy utilizado en tareas de administración por la rapidez de efectuar pequeños cambios en los ficheros. Vim es el editor compatible Vi, que suele traer GNU/Linux; permite sintaxis coloreada en varios lenguajes. Emacs, editor muy

Nota

Ver material asociado a curso de introducción a GNU/Linux, o las páginas man de los comandos, o una referencia de herramientas como [Stu01].

completo, adaptado a diferentes lenguajes de programación (sintaxis y modos de edición); dispone de un entorno muy completo y de una versión X denominada Xemacs. Joe, editor compatible con Wordstar. Y muchos otros...

- Lenguajes de tipo *script*, útiles para administración, como: Perl, muy útil para tratamiento de expresiones regulares, y análisis de ficheros (filtrado, ordenación, etc.). PHP, lenguaje muy utilizado en entornos web. Python, otro lenguaje que permite hacer prototipos rápidos de aplicaciones, ...

- Herramientas de compilación y depuración de lenguajes de alto nivel: GNU gcc (compilador de C y C++), gdb (depurador), xxgdb (interfaz X para gdb), ddd (depurador para varios lenguajes).

3.8. Actividades para el lector

1) Hacer una lectura rápida del estándar FHS, que nos servirá para tener una buena guía a la hora de buscar archivos por nuestra distribución.

2) Para repasar y ampliar conceptos y programación de *shell scripts* en *bash* ver: [G00] [Coo03].

3) Para los paquetes RPM, ¿cómo haríamos algunas de las siguientes tareas?:

- Conocer qué paquete instaló un determinado comando.

- Obtener la descripción del paquete que instaló un comando.

- Borrar un paquete cuyo nombre completo no conocemos.

- Mostrar todos los archivos que estaban en el mismo paquete que un determinado archivo.

4) Efectuar las mismas tareas que en la actividad anterior, pero para paquetes Debian, usando herramientas APT.

Nota

FHS:
http://www.pathname.com/fhs

ANOTACIONES

5) Actualizar una distribución Debian (o Red Hat).

6) Instalar en nuestra distribución alguna herramienta genérica de administración, ya sea por ejemplo Linuxconf o Webadmin. ¿Qué nos ofrecen? ¿Entendemos las tareas ejecutadas y los efectos que provocan?

3.9. Otras fuentes de referencia e información

- [G00][Coo03] ofrecen una amplia introducción (y conceptos avanzados) a la programación de *shell scripts* en `bash`, así como números ejemplos. [Qui01] comenta los diferentes *shells* de programación en GNU/Linux, así como sus semejanzas y diferencias.

- [Deb02][Bai03] ofrecen una amplia visión de los sistemas de paquetes de software de las distribuciones Debian y Red Hat.

- [Stu01] es una amplia introducción a las herramientas disponibles en GNU/Linux.

ANOTACIONES

4. El *kernel*

El *kernel* del sistema GNU/Linux (al que habitualmente se le denomina Linux) [Vas03b] es el corazón del sistema: Se encarga de arrancar el sistema y, una vez éste es ya utilizable por las aplicaciones y los usuarios, gestiona los recursos de la máquina en forma de gestión de la memoria, sistema de ficheros, entrada/salida, procesos e intercomunicación de procesos.

Su origen se remonta al año 1991, cuando en agosto, un estudiante finlandés llamado Linus Torvalds anunció en una lista de *news* que había creado su propio núcleo de sistema operativo que funcionaba conjuntamente con software GNU, y lo ofrecía a la comunidad de desarrolladores para que ésta lo probara y sugiriera mejoras para una mejor utilización. Así, se constituyó en el origen del *kernel* del operativo, que más tarde se llamaría Linux.

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

Nota

El *kernel* Linux se remonta al año 1991, cuando Linus Torvalds lo puso a disposición de la comunidad. Es de los pocos operativos que, siendo ampliamente usado, se puede disponer de su código fuente.

La otra ventaja principal es que, al disponer de las fuentes, podemos recompilarlas para adaptarlas mejor a nuestro sistema y, como veremos posteriormente (en la unidad de optimización), podemos asimismo configurar para dar un mejor rendimiento al sistema.

En esta unidad veremos cómo manejar este proceso: cómo, partiendo de las fuentes, podemos obtener una nueva versión del *kernel* adaptada a nuestro sistema. Del mismo modo, trataremos cómo se desarrolla la configuración y la posterior compilación y cómo realizar pruebas con el nuevo *kernel* obtenido.

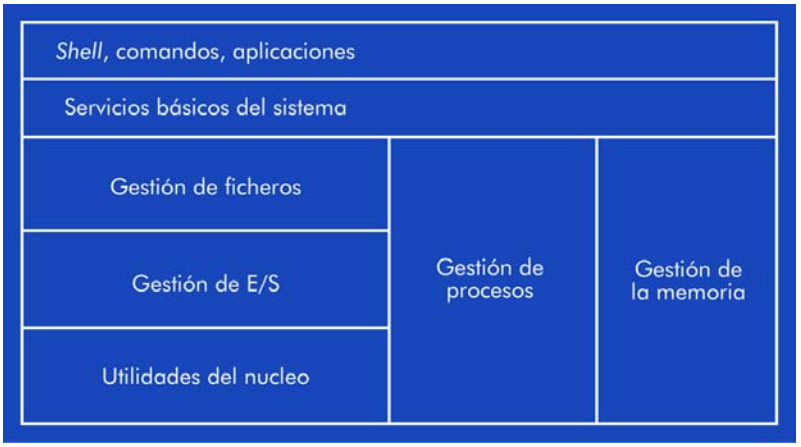
ANOTACIONES

4.1. El *kernel* del sistema GNU/Linux

El núcleo o *kernel* es la parte básica de cualquier sistema operativo [Tan87], y en él descansa el código de los servicios fundamentales para controlar el sistema entero. Básicamente, su estructura se puede separar en:

- Gestión de procesos: qué tareas se van a ejecutar y en qué orden y prioridad. Un aspecto importante es la planificación de 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 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 el acceso a los mismos.

Figura 8. Funciones básicas de un *kernel* respecto de las aplicaciones y comandos ejecutados



En los sistemas propietarios, el *kernel* está perfectamente "oculto" bajo las capas del software del sistema operativo, y el usuario fi-

ANOTACIONES

nal no tiene una perspectiva clara de qué es el *kernel*, y no tiene ninguna posibilidad de cambiarlo u optimizarlo, si no es por el uso de exotéricos editores de "registros" internos, o programas especializados de terceros (normalmente de alto coste). Además, el *kernel* suele ser único, es el que el fabricante proporciona, y éste 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 ofrecen como "parches" de errores.

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 son de seguridad, todavía más, ya que hasta que no estén corregidos no podemos garantizar la seguridad del sistema. 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 *kernel* 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 hacerlo. Esto permite a los usuarios críticos controlar mejor sus aplicaciones y el propio sistema, y poder montar sistemas con el propio operativo "a la carta", personalizado al gusto de cada uno. Y disponer a su vez de un operativo con código abierto, desarrollado por una comunidad de programadores coordinados desde Internet, 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 las necesidades de cada uno.

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

ANOTACIONES

Siguiendo un poco de historia (rápida) del *kernel* [Ker03a] [Ker03b]: lo comenzó a desarrollar un estudiante finlandés llamado Linus Torvalds, en 1991, con la intención de realizar una versión parecida a Minix [Tan87] (versión para PC de UNIX [Bac86]) para el procesador 386 de Intel. La primera versión publicada oficialmente fue la de Linux 1.0 en marzo de 1994, en la cual se incluía sólo la ejecución para la arquitectura i386 y soportaba máquinas de un sólo 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) [Tum02]. En Linux 2.2, enero de 1999, se incrementaron las prestaciones de SMP de manera significativa, y se añadieron controladores para gran cantidad de hardware. Y finalmente, en la 2.4, en enero del 2001, la más utilizada hoy en día, se mejoró el soporte SMP, y se integraron controladores para dispositivos USB, PC card (PCMCIA de los portátiles), parte de PnP (*plug and play*), etc. En la 2.6 (en versión beta a finales del 2003), se espera un mayor soporte para los sistemas grandes (otra vez SMP) y en los más pequeños (PDA, dispositivos móviles).

Nota

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

Respecto al desarrollo, el *kernel*, desde su creación por Linus Torvalds en 1991 (versión 0.01), lo ha seguido manteniendo él mismo, pero a medida que su trabajo se lo permitía, y a medida que el *kernel* maduraba (y crecía), se ha visto ayudado a mantener las diferentes versiones estables del *kernel* por diferentes colaboradores, mientras Linus continuaba (en la medida de lo posible) desarrollando y recopilando aportaciones para la última versión del *kernel*. Los colaboradores principales en estas versiones han sido [lkm03]:

- 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.

ANOTACIONES

Para ver un poco la complejidad del *kernel* de Linux, veamos una tabla con un poco de su historia resumida en las diferentes versiones y su tamaño respectivo del código fuente. En la tabla se indican las versiones de producción únicamente; el tamaño está especificado en miles de líneas (K) de código fuente:

Versión	Fecha de publicación	Líneas de código (en miles)
0,01	09-1991	7,5
1,00	03-1994	158
1,20	03-1995	277
2,00	07-1996	649
2,20	01-1999	1.536
2,40	01-2001	2.888
2,60	08/09-2003	4.200

Como podemos comprobar, hemos pasado de unas siete mil líneas a cuatro millones.

Nota

El *kernel* hoy en día ha alcanzado un grado de madurez y complejidad significativo.

A finales del 2003, la última versión estable es la 2.4.x, que incluyen la mayoría de distribuciones como versión por defecto (algunas tienen un 2.2.x, pero el 2.4.x es una opción en la instalación). Ya está publicada una versión beta del 2.6.x [Pra03] (mantenida por Andrew Morton), y se espera que la corrección de errores y parches pueda tener una primera versión en producción a finales de año o principios del 2004.

Actualmente, cabe suponer que se acelerarán los trabajos del *kernel*, ya que tanto Linus Torvalds (que antes trabajaba en una empresa llamada Transmeta), como Andrew Morton (mantiene la 2.6) se han incorporado a OSDL (Open Source Developement Laboratory) [OSD03a], un consorcio de empresas con el fin de promocionar el uso Open Source y Linux en la empresa (en el consorcio se encuentran muchas empresas con intereses en Linux: HP, IBM, Intel, Fujitsu, Hitachi, Toshiba, Red Hat, Suse, Transmeta, ... y parece que se unirá también Sun). En estos momentos se da una situación interesante, ya que el consorcio OSDL ha conseguido tener al mantenedor de la versión estable del *kernel* (2.6 por Andrew)

Nota

OSDL: http://www.osdl.org

y la de desarrollo (2.5 por Linus), trabajando a tiempo completo en las versiones.

Si bien hay que tener en cuenta que con las versiones actuales del *kernel* 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. Además, otro factor a considerar es el tamaño y el número de personas que están trabajando en el desarrollo. En un principio había unas pocas personas que tenían un conocimiento global del *kernel* entero, mientras hoy en día tenemos muchas personas desarrollándolo (se cuenta que cerca de dos mil), aunque con unos conocimientos parciales del *kernel*; además, ni todos trabajan simultáneamente, ni su aportación es igual de relevante (algunas de ellas sólo corrigen errores sencillos), mientras son unas pocas (como los mantenedores) las que disponen de un conocimiento total del *kernel*. Esto supone que se puedan alargar los desarrollos, y que las aportaciones se tengan que depurar, para ver que no entren en conflicto entre ellas, o decidir entre posibles alternativas de prestaciones para escoger.

Respecto a la numeración de las versiones del *kernel* de Linux ([lkm03][DB03]), cabe tener en cuenta los aspectos siguientes:

a) Las versiones del *kernel* Linux están divididas en dos series: una es la denominada "experimental" (la numerada impar en la segunda cifra, como 1.3.xx, 2.1.x o 2.5.x) y la otra es la de producción (serie par, como 1.2.xx, 2.0.xx, 2.2.x, 2.4.x y más). La serie experimental son versiones que se mueven rápidamente y son utilizadas para probar nuevas prestaciones, algoritmos o controladores de dispositivo, etc. Por la propia naturaleza de los *kernels* experimentales, pueden tener comportamientos impredecibles, como pérdidas de datos, bloqueos aleatorios de la máquina, etc. Por lo tanto, no deberían utilizarse en máquinas destinadas a producción, a no ser que se quiera probar una característica determinada (con los consecuentes peligros).

b) Los *kernels* de producción (serie par) o estables son *kernels* con un conjunto de prestaciones bien definido, con un número bajo de errores conocidos y controladores de dispositivos probados.

ANOTACIONES

Se publican con menos frecuencia que los experimentales, y existe variedad de versiones, unas más buenas que otras. Los sistemas GNU/Linux se suelen basar en un determinado *kernel* estable elegido, no necesariamente el último *kernel* de producción publicado.

c) Para obtener el último *kernel* publicado, hay que acudir al archivo de *kernels* Linux (en http://www.kernel.org) o al *mirror* local (en España http://www.es.kernel.org). También podrán encontrarse algunos parches al *kernel* original, que corrijan errores detectados *a posteriori* de la publicación del *kernel*.

Algunas de las características técnicas ([DB03][Arc03]) del *kernel* Linux que podríamos destacar son:

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

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

- *Threads* de *kernel*: para el funcionamiento interno se usan varios hilos (*threads*) de ejecución internos al *kernel*, que pueden estar asociados a un programa de usuario o bien a una funcionalidad interna del *kernel*. En Linux no se hace un uso intensivo de este concepto, sólo unas pocas funcionalidades, pero en las próximas versiones del *kernel* puede ir a más.

- Soporte de aplicaciones *multithread*: soporte de aplicaciones de usuario de tipo *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 *threads* que puede usarse para las aplicaciones *multithread*, pero también se esperan mejoras im-

ANOTACIONES

portantes en este aspecto en las nuevas versiones (ver la unidad 4.6, que trata del futuro del *kernel*).

- El *kernel* es de tipo no apropiativo (*nonpreemptive*): esto supone que dentro del *kernel* no pueden pararse llamadas a sistema (en modo supervisor) mientras se está resolviendo la tarea de sistema, y cuando ésta acaba, se reasume la ejecución de la tarea anterior. Por lo tanto, el *kernel* dentro de una llamada no puede ser interrumpido para atender otra tarea. Normalmente, los *kernels* apropiativos están asociados a sistemas que trabajan en tiempo real, en que debe permitirse lo anterior para tratar eventos críticos. Hay algunas versiones especiales del *kernel* de Linux para tiempo real, que permiten esto por medio de la introducción de unos puntos fijos donde pueden intercambiarse. También se está probando este concepto en la versión 2.5/2.6 de desarrollo del *kernel*. Este concepto de *kernel* 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, lo que llama *multiprocesamiento simétrico* (SMP). Este concepto suele englobar máquinas que van desde el caso simple de 2 hasta 64 CPU. Linux puede usar múltiples procesadores, donde cada procesador puede manejar una o más tareas. Pero hay algunas partes del *kernel* que disminuyen el rendimiento, ya que están pensadas para una única CPU y obligan a parar el sistema entero. SMP es una de las técnicas más estudiadas en la comunidad del *kernel* de Linux, y se esperan mejoras importantes para las nuevas versiones, ya que se depende en gran medida del rendimiento SMP para la adopción de Linux en los sistemas empresariales.

- Sistemas de ficheros: el *kernel* tiene una buena arquitectura de los sistemas de ficheros, 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 operativo que más sistemas de ficheros soporta, desde los propios ext2 y ext3, hasta msdos, vfat, sistemas con *journal* como ReiserFS, JFS(IBM), XFS(Silicon), NTFS(sólo lectura), iso9660 (CD), udf,

ANOTACIONES

etc. y se van añadiendo más; podéis ver como ejemplo la lista de particiones soportadas actualmente:

0	Empty	1c	Hidden Win95 FA	70	DiskSecure Mult	bb	Boot Wizard hid
1	FAT12	1e	Hidden Win95 FA	75	PC/IX	be	Solaris boot
2	XENIX root	24	NEC DOS	80	Old Minix	c1	DRDOS/sec (FAT-
3	XENIX usr	39	Plan 9	81	Minix / old Lin	c4	DRDOS/sec (FAT-
4	FAT16 <32M	3c	PartitionMagic	82	Linux swap	c6	DRDOS/sec (FAT-
5	Extended	40	Venix 80286	83	Linux	c7	Syrinx
6	FAT16	41	PPC PReP Boot	84	OS/2 hidden C:	da	Non-FS data
7	HPFS/NTFS	42	SFS	85	Linux extended	db	CP/M / CTOS / .
8	AIX	4d	QNX4.x	86	NTFS volume set	de	Dell Utility
9	AIX bootable	4e	QNX4.x 2nd part	87	NTFS volume set	df	BootIt
a	OS/2 Boot Manag	4f	QNX4.x 3rd part	8e	Linux LVM	e1	DOS access
b	Win95 FAT32	50	OnTrack DM	93	Amoeba	e3	DOS R/O
c	Win95 FAT32 (LB	51	OnTrack DM6 Aux	94	Amoeba BBT	e4	SpeedStor
e	Win95 FAT16 (LB	52	CP/M	9f	BSD/OS	eb	BeOS fs
f	Win95 Ext'd (LB	53	OnTrack DM6 Aux	a0	IBM Thinkpad hi	ee	EFI GPT
10	OPUS	54	OnTrackDM6	a5	FreeBSD	ef	EFI (FAT-12/16/
11	Hidden FAT12	55	EZDrive	a6	OpenBSD	f0	Linux/PA-RISC b
12	Compaq diagnost	56	Golden Bow	a7	NeXTSTEP	f1	SpeedStor
14	Hidden FAT16 <3	5c	Priam Edisk	a8	Darwin UFS	f4	SpeedStor
16	Hidden FAT16	61	SpeedStor	a9	NetBSD	f2	DOS secondary
17	Hidden HPFS/NTF	63	GNU HURD or Sys	ab	Darwin boot	fd	Linux raid auto
18	AST SmartSleep	64	Novell Netware	b7	BSDI fs	fe	LANstep
1b	Hidden Win95 FA	65	Novell Netware	b8	BSDI swap	ff	BBT

Otras características menos técnicas (un poco de marketing):

a) Linux es gratis: junto con el software GNU, y el incluido en cualquier distribución, podemos tener un sistema UNIX completo por el coste del hardware y los costes de distribución Linux; podemos obtenerla gratis, pero no está de más pagar por una distribución completa, con los manuales y apoyo técnico, a un coste ridículo comparado con lo que se paga por algunos sistemas propietarios. O contribuir con la compra al desarrollo de las distribuciones que más nos gusten.

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

ANOTACIONES

c) 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.

d) Linux es un sistema potente: el objetivo principal en Linux es la eficiencia, se aprovecha al máximo el hardware disponible.

e) Alta calidad: los sistemas Linux son muy estables, bajo ratio de fallos y reducen el tiempo dedicado a mantener los sistemas.

f) El *kernel* es bastante reducido y compacto: es posible colocarlo, junto con algunos programas fundamentales en un sólo disco de 1.44 MB (existen varias distribuciones de un sólo disquete con programas básicos).

g) 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/recibir servicios de cualquiera de estos sistemas. Además, también con ciertas bibliotecas puede ejecutar programas de otros sistemas (como msdos, Windows, BSD, Xenix, etc.) en la arquitectura x86.

h) Linux está ampliamente soportado: no hay ningún otro sistema que tenga la rapidez y cantidad de parches y actualizaciones que Linux, ni en los sistemas propietarios. Para un problema determinado, hay infinidad de listas de correo y foros, que en pocas horas pueden permitir solucionar cualquier problema. El único problema está en los controladores de hardware reciente, que muchos fabricantes todavía se resisten a proporcionar si no es para sistemas propietarios. Pero esto está cambiando poco a poco, y varios de los fabricantes más importantes de sectores como tarjetas de vídeo (NVIDIA, ATI) e impresoras (Epson, HP,..) comienzan ya a proporcionar los controladores para sus dispositivos.

ANOTACIONES

4.2. Personalizar o actualizar el *kernel*

Como usuarios o administradores de sistemas GNU/Linux, tenemos que tener en cuenta las posibilidades que nos ofrece el *kernel* 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, Mandrake, Suse o "comunitarias" como Debian.

Estas distribuciones aportan, en el momento de la instalación, una serie de *kernels* Linux binarios ya preconfigurados y compilados, y normalmente tenemos que elegir qué *kernel* del conjunto de los disponibles se adapta mejor a nuestro hardware. Hay *kernels* orientados a dispositivos IDE, otros a SCSI, otros que ofrecen una mezcla de controladores de dispositivos [AR01], etc.

Otra opción de instalación suele ser la versión del *kernel*. Normalmente las distribuciones usan una 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 versiones 2.2.x del *kernel* por defecto, ya que se considera la versión más estable del momento. Aunque se incluyen como opción en el momento de la instalación la posibilidad de usar como alternativa una 2.4.x con mejor soporte para dispositivos más modernos (de última generación).

Los distribuidores suelen, además, modificar el *kernel* para mejorar el comportamiento de su distribución o corregir errores que han detectado en el *kernel* en el momento de las pruebas. Otra técnica bastante común en las comerciales es deshabilitar prestaciones problemáticas, que pueden causar fallos a los usuarios. Por ejemplo, en algunas versiones de Red Hat estaba desactivado el uso de algunos modos DMA en los discos duros, ya que en algunos discos esto provocaba que el sistema se colgase o se perdieran datos, el efecto colateral es que para otros usuarios esto supone una pérdida de prestaciones de entre un 40-70% del rendimiento de los discos.

Nota

La posibilidad de actualizar y personalizar el *kernel* a medida ofrece buena adaptación a cualquier sistema, permitiendo una optimización y sintonización al mismo.

Esto nos lleva a considerar que, por muy bien que un distribuidor haga el trabajo de adaptar el *kernel* a su distribución, siempre nos podemos encontrar con una serie de problemas:

- El *kernel* no está actualizado en la última versión disponible; algunos dispositivos modernos no lo soportan.

ANOTACIONES

- El *kernel* estándar no dispone de soporte de los dispositivos que tenemos, porque no han estado habilitados.

- Los controladores que nos ofrece un fabricante necesitan una nueva versión del *kernel* o modificaciones.

- A la inversa, el *kernel* es demasiado moderno, tenemos hardware antiguo que ya no tiene soporte en los *kernels* modernos.

- El *kernel*, tal como está, no ofrece las máximas prestaciones de nuestros dispositivos.

- Algunas aplicaciones que queremos usar requieren soporte de un *kernel* nuevo o de algunas de sus prestaciones.

- Somos muy "modernos" o arriesgados, y queremos estar a la última en el *kernel* Linux.

- Nos gusta investigar o probar los nuevos avances del *kernel* o bien queremos tocar o modificar el *kernel*.

- Queremos programar un controlador para un dispositivo no soportado.

- Otros.

Por éstos y otros motivos podemos no estar contentos con el *kernel* que tenemos; se nos plantean entonces dos posibilidades: actualizar el *kernel* binario de la distribución o bien personalizarlo a partir de las fuentes.

Vamos a ver someramente las opciones y qué suponen:

1) Actualización del *kernel* de la distribución: el distribuidor normalmente publica también en las actualizaciones del *kernel*. Cuando la comunidad Linux crea una nueva versión del *kernel*, cada distribuidor la une a su distribución y hace las pruebas pertinentes. Posterior-

Nota

La actualización automática es un proceso sencillo (si no surgen errores), y en el taller al final de esta unidad veremos cómo se realiza en las distribuciones Red Hat y Debian.

ANOTACIONES

mente al periodo de prueba, se identifican posibles errores, los corrige y produce la actualización del *kernel* pertinente a la que ofrecía en los CD de la distribución. Los usuarios pueden bajarse la distribución del sitio web o bien actualizarla mediante algún sistema automático vía Internet. Normalmente, se verifica qué versión tiene el sistema, se baja el *kernel* nuevo y se hacen los cambios necesarios para que la siguiente vez arranque con el nuevo *kernel*, y se mantenga la versión antigua por si hay problemas.

Este tipo de actualización nos simplifica mucho el proceso, pero no tiene por qué solucionar nuestros problemas, ya que puede ser que nuestro hardware no esté todavía soportado o la característica por probar del *kernel* no esté todavía en la versión que tenemos de la distribución; cabe recordar que no tiene por qué usar la última disponible.

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

2) Personalizar el *kernel* (proceso descrito con detalle en el apartado siguiente). En este caso, iremos a las fuentes del *kernel* y adaptaremos "a mano" el hardware o las características deseadas. Pasaremos por un proceso de configuración y compilación de las fuentes del *kernel* para, finalmente, crear un *kernel* binario que instalaremos en el sistema, y tenerlo, así, disponible en el siguiente arranque del sistema.

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.

ANOTACIONES

4.3. Proceso de configuración y compilación

La personalización del *kernel* [Vas03b] es un proceso costoso y necesita amplios conocimientos de lo que se está haciendo, y además, es una de las tareas críticas, de la cual depende la estabilidad del sistema, por la propia naturaleza del *kernel,* puesto que es el elemento central del sistema.

Nota

El proceso de obtención de un nuevo *kernel* personalizado pasa por obtener las fuentes, adaptar la configuración, compilar e instalar el kernel obtenido en el sistema.

Cualquier error de procedimiento puede comportar la inestabilidad o la pérdida del sistema. Por lo tanto, no está de más llevar a cabo cualquier tarea de *backup* de los datos de usuarios, datos de configuraciones que hayamos personalizado o, si disponemos de dispositivos adecuados, el *backup* completo del sistema. También es recomendable disponer de algún disquete de arranque por si surgen problemas, o bien un disquete de rescate (*rescue disk*) que la mayoría de distribuciones permiten crear desde los CD de la distribución.

Sin ánimo de exagerar, casi nunca aparecen problemas si se siguen los pasos adecuadamente, se tiene conciencia de lo que se está haciendo y se toman algunas precauciones.

Vamos a ver el proceso necesario para instalar y configurar un *kernel* Linux; en este caso supondremos que se trata de un *kernel* de la serie 2.4, un 2.4.0., por ejemplo. Las instrucciones son específicas para la arquitectura x86 Intel, mediante usuario *root* (aunque parte del proceso puede hacerse como usuario normal):

1) Obtener el *kernel*: por ejemplo, podemos acudir a www.kernel.org (o su servidor ftp) y bajarnos la versión que necesitemos. Hay *mirrors* para diferentes países, podemos por ejemplo acudir a www.es.kernel.org. En la mayoría de las distribuciones de GNU/Linux, como Red Hat o Debian, también se ofrece el código fuente del *kernel* (normalmente con algunas modificaciones incluidas), si se trata de la versión del *kernel* que necesitamos, quizás sea preferible usar éstas. Si queremos los últimos *kernels*, quizás no estén disponibles en la distribución y tendremos que ir a kernel.org.

ANOTACIONES

2) Desempaquetar el *kernel*: las fuentes del *kernel* suelen colocarse y desempaquetarse sobre el directorio /usr/src, por ejemplo, si las fuentes venían en un fichero comprimido de tipo *bzip2*:

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

Si las fuentes venían en un fichero gz, reemplazamos *bzip2* por *gzip*. Al descomprimir las fuentes, se habrá generado un directorio *linux-version-kernel*, donde entraremos para establecer la configuración del *kernel*.

Para la configuración del *kernel* [Vas03b], tenemos varios métodos:

- **make config**: desde 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 bien la configuración larga, en la que se nos piden muchas respuestas, y dependiendo de la versión, igual tenemos que responder a casi un centenar de preguntas.

- **make oldconfig**: sirve por si queremos reutilizar una configuración ya usada (normalmente almacenada en un fichero *.config*), hay que tener en cuenta que sólo es válida si estamos compilando la misma versión del *kernel*, ya que diferentes versiones del *kernel* pueden variar en sus opciones.

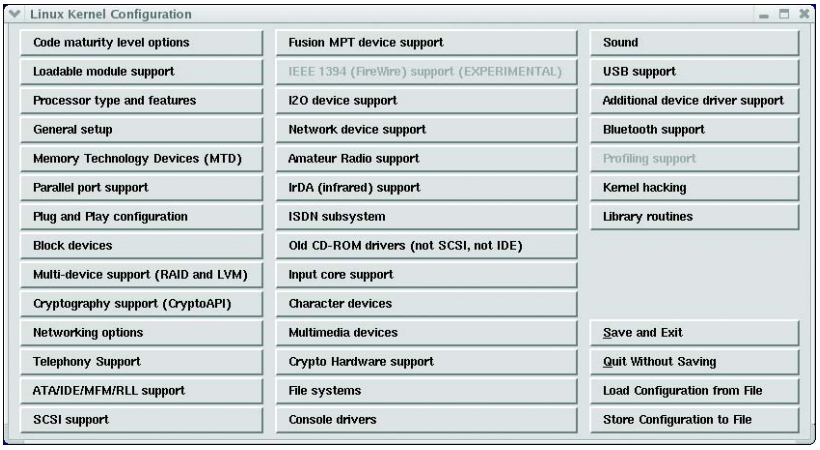
- **make menuconfig**: configuración basada en menús textuales, bastante cómoda; podemos habilitar o inhabilitar lo que queramos, más rápida que el *make config*.

- **make xconfig**: la más cómoda, basada en diálogos gráficos en X Window. Es necesario tener instaladas las bibliotecas de tcl/tk, ya que esta configuración está programada en este lenguaje. La configuración se basa en cuadros de diálogos y botones / casillas de activación, se hace bastante rápida y dispone de ayuda con comentarios de muchas de las opciones. Pero hay un defecto, puede ser que algunas de las opciones no aparezcan (depende de que el programa de configuración esté actualizado, y a veces no lo está). En este último caso, el *make config* es el único que asegura disponer de todas las opciones elegibles; en los otros tipos de configuración depende de

ANOTACIONES

que se haya podido adaptar a tiempo los programas a las nuevas opciones cuando se libera el *kernel*.

Figura 9. Configuración del *kernel* desde interfaz gráfica en X Window



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

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

El *kernel* de Linux ha comenzado a ser muy grande, tanto por complejidad como por los controladores (*drivers*) de dispositivo [AR01] que incluye. Si lo integrásemos todo, se podría crear un fichero del *kernel* bastante grande y ocupar mucha memoria, ralentizando, de ese modo, algunos aspectos de funcionamiento. Los módulos del *kernel* [Hen03] son un método que permite separar parte del *kernel* en pequeños trozos, que serán cargados bajo demanda dinámicamente, cuando o bien por carga explícita o por uso de la característica sean necesarios.

ANOTACIONES

La elección más normal es integrar lo que se considere básico para el funcionamiento o crítico en rendimiento dentro del *kernel*. Y aquellas partes o controladores de los que se vaya hacer un uso esporádico o que se necesiten para futuras ampliaciones del equipo dejarlos como módulos.

Ejemplo

Por ejemplo:

- Un caso claro son los controladores de dispositivo, por ejemplo, si estamos actualizando la máquina, puede ser que a la hora de crear el *kernel* no sepamos seguro qué hardware va a tener: por ejemplo, qué tarjeta de red; pero sí que sabemos que estará conectada a red, entonces, el soporte de red estará integrado en el *kernel*, pero en 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 tenemos que cambiar una tarjeta por otra después, sólo tendremos que cambiar el módulo que se va a cargar. Si hubiese sólo un controlador integrado en el *kernel* y cambiamos la tarjeta, esto obligaría a reconfigurar y recompilar el *kernel* 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 puede pasar por ejemplo con la impresora con cable paralelo y hardware que se conecta al puerto paralelo). Por lo tanto, en este caso tenemos que colocar como módulos los controladores y cargar/descargar el que sea necesario.

- Otro ejemplo podrían conformarlo los sistemas de ficheros (*filesystems*) normalmente esperaremos que nuestro sistema tenga acceso a algunos de ellos, por ejemplo ext2 o ext3 (propios de Linux) o vfat (de los Windows 95/98/ME), y les daremos de

ANOTACIONES

alta en la configuración del *kernel*. Si en otro momento tuviéramos que leer otro tipo no esperado, por ejemplo, datos guardados en un disco o partición de sistema NTFS de Windows NT/XP, no podríamos: el *kernel* 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 *filesystems* como módulos.

1) Compilación del *kernel*

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

```
make dep
make bzImage
```

Cuando este proceso acabe, tenemos la parte integrada del *kernel*; nos faltan las partes que hayamos puesto como módulos:

```
make modules
```

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

2) Instalación

Comenzamos instalando los módulos:

```
make modules_install
```

y la instalación del nuevo *kernel* (desde el directorio /usr/src/linux-version):

```
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.0
cp System.map /boot/System.map-2.4.0
```

ANOTACIONES

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

3) El siguiente paso es decirle al sistema con qué *kernel* tiene que arrancar, aunque esto depende del sistema de arranque de Linux:

a) Desde arranque con *lilo* [Zan01][Sko03a], ya sea en el MBR (*Master Boot Record*) o desde partición propia, hay que añadir al fichero de configuración (en: /etc/lilo.conf), por ejemplo las líneas:

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

donde *image* es el *kernel* 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 *kernel* antiguo. Se recomienda añadirlas y dejar el *kernel* antiguo por, si aparecen problemas, poder recuperar el viejo. 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 que se arranca por defecto. También podemos añadirle a las líneas anteriores un *root = /dev/...* para indicar la partición de disco donde está el sistema de archivos principal (el '/'), recordar que los discos tienen dispositivos como /dev/hda (1.r disco *ide*) /dev/hdb (2.º disco *ide*), y la partición se indicaría como *root* = /dev/hda2 si el '/' de nuestro Linux estuviese en la segunda partición del primer disco *ide*. Además con "*append* = " podemos añadir parámetros al arranque del *kernel* [Gor03]. Después de cambiar la configuración del *lilo*, hay que escribirla para que arranque:

```
/sbin/lilo
```

Reiniciamos (*reset*) y arrancamos con el nuevo *kernel*.

ANOTACIONES

Si tuviésemos problemas, podemos recuperar el antiguo *kernel*, escogiendo la opción del viejo *kernel*, y luego mediante la acción de retocar el *lilo.conf*, podemos devolver la antigua configuración o estudiar el problema y reconfigurar y recompilar el *kernel* de nuevo.

b) Arranque con *grub* [Kan01][Pro02]. El manejo en este caso es bastante simple, cabe añadir una nueva configuración formada por el *kernel* nuevo y sumarla como una opción más al fichero del *grub*. A continuación, reiniciar procediendo de forma parecida a la del *lilo*, pero recordando que en *grub* es suficiente con editar el fichero y reiniciar. También es mejor dejar la antigua configuración para poder recuperarse de posibles errores.

4.4. Parchear el *kernel*

En algunos casos también puede ser habitual la aplicación de parches (*patch*) al *kernel* [lkm03].

Un fichero de parche (*patch file*) respecto al *kernel* 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 kernel (normalmente en /usr/src).

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

ANOTACIONES

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

```
patchxxxx-2.4.21-pversion.bz2
```

donde *xxxx* suele ser algún mensaje sobre el tipo o finalidad del parche, 2.4.21 sería la versión del*kernel* 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 las fuentes del *kernel* (normalmente instaladas, como vimos, en /usr/src/linux).

Una vez dispongamos del parche, tendremos que aplicarlo, veremos el proceso a seguir en algún fichero *readme* que acompañe al parche, pero generalmente el proceso consta (una vez comprobados los requerimientos previos) de descomprimir el parche en el directorio de los ficheros fuentes y aplicarlo sobre las fuentes del *kernel*, como por ejemplo:

```
cd/usr/src/linux (o /usr/src/linux-2.4.21 o la versión
                  que sea).
bunzip2 patch-xxxxx-2.4.21-version.bz2
patch -p1 < patch-xxxxx-2.4.21-version
```

y posteriormente tendremos que recompilar el *kernel*, para volverlo a generar.

Los parches pueden obtenerse de diferentes lugares. Lo más normal es encontrarlos en el sitio de almacén de los *kernels* (www.kernel.org) o bien en www.linuxhq.com, que tiene un buen archivo de ellos. En 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 *kernel* (o de controladores) para que funcionen mejor sus dispositivos (un ejemplo conocido es NVIDIA y sus *drivers* Linux para sus tarjetas gráficas).

ANOTACIONES

Nota

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

Por último, señalaremos que en las distribuciones de GNU/Linux, muchas de ellas (Red Hat, Mandrake, ...) ya ofrecen *kernels* parcheados por ellos mismos, y sistemas para actualizarlos (algunos incluso de forma automática, como en el caso de Red Hat y Debian). Normalmente, en sistemas de producción es más recomendable seguir las actualizaciones del fabricante, aunque éste no ofrecerá necesariamente el último *kernel* publicado, sino el que crea más estable para su distribución, bajo pena de perder prestaciones de última generación.

4.5. Los módulos del *kernel*

El *kernel* es capaz de cargar dinámicamente porciones de código (módulos) bajo demanda [Hen03], para complementar su funcionalidad (se dispone de esta posibilidad desde la versión 1.2 del *kernel*). Por ejemplo, los módulos pueden añadir soporte para un sistema de ficheros o para dispositivos hardware específicos. Cuando la funcionalidad proporcionada por el módulo no es necesaria, el módulo puede ser descargado, liberando memoria.

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

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/ (el directorio estándar para los módulos), la version-kernel es la versión del *kernel* actual, se utiliza el comando *uname -r* para determinarla. Por lo tanto, los módulos en forma binaria están relacionados con una versión concreta del *kernel*, y suelen colocarse en /lib/modules/version-kernel.

Si hay que compilarlos, se tiene que disponer de las fuentes y/o *headers* de la versión del núcleo al cual está destinado.

Nota

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

Hay unas cuantas utilidades que nos permiten trabajar con módulos (suelen aparecer en un paquete software llamado *modutils*):

- **lsmod**: podemos ver los módulos cargados en el *kernel* (la información se obtiene del pseudofichero /proc/modules). Se listan los nombres, las dependencias con otros (en [ ]), 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          37.344     3        (autoclean)
apm              10.024     1        (autoclean)
parport_pc       23.304     1        (autoclean)
lp                6.816     0        (autoclean)
parport          25.992     1        [parport_pc lp]
snd              30.884     0
af_packet        13.448     1        (autoclean)
NVIDIA        1.539.872    10
es1371           27.116     1
soundcore         3.972     4        [snd es1371]
ac97_codec       10.9640    0        [es1371]
gameport          1.676     0        [es1371]
3c59x            26.960     1
```

b) **modprobe**: intenta la carga de un módulo y de sus dependencias.

c) **insmod**: carga un módulo determinado.

d) **depmod**: analiza dependencias entre módulos y crea fichero de dependencias.

e) **rmmod**: saca un módulo del *kernel*.

f) Otros comandos pueden ser utilizados para depuración o análisis de los módulos, como *modinfo*: lista algunas informaciones asociadas al módulo, o *ksyms*, permite examinar los símbolos exportados por los módulos (también en /proc/ksyms).

Ya sea por el mismo *kernel*, 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 que

ANOTACIONES

sean positivos, 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
```

4.6. Futuro del *kernel* y alternativas

Los avances en el *kernel* de Linux en determinados momentos fueron muy rápidos, pero actualmente, ya con una situación bastante estable con los *kernels* de la serie 2.x.x, cada vez pasa más tiempo entre las versiones que van apareciendo, y 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.

Nota

El *kernel* continúa evolucionando, incorporando las últimas novedades en soporte hardware y mejoras en las prestaciones.

Comentaremos en este apartado algunas de las ideas de los últimos *kernels*, y algunas que están por venir, para dar indicaciones de lo que será el futuro próximo del *kernel*.

En la última versión estable, serie 2.4.x [DB03], incluida en la mayoría de distribuciones actuales, se realizaron algunas aportaciones en:

- Cumplimiento de los estándares IEEE POSIX, esto permite que muchos de los programas existentes de UNIX pueden recompilarse y ejecutarse en Linux.
- Mejor soporte de dispositivos: PnP, USB, Puerto Paralelo, SCSI,...
- Soporte para nuevos *filesystems*, como UDF (CD-ROM reescribibles como un disco). Otros sistemas con *journal*, como los Reiser de IBM o el ext3, éstos permiten tener un *log* (*journal*) de las modificaciones del *filesystems*, 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 *kernels* 1.2.x) que no soportaban más de 128 MB de memoria (en aquel tiempo era mucha memoria).

ANOTACIONES

- Se mejoró la interfaz /proc. Ésta es un pseudosistema de ficheros (el directorio /proc) que no existe realmente en disco, sino que es simplemente una forma de acceder a datos del *kernel* y del hardware de una manera organizada.

- Soporte del sonido en el *kernel*, se añadieron parcialmente los controladores Alsa que antes se configuraban por separado.

El nuevo *kernel* 2.6.x [Pra03], ahora (finales del 2003) en versión beta se espera que tenga entre otras características:

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

- Mejoras en el planificador de CPU (*scheduler*).

- Mejoras en el soporte multithread para las aplicaciones de usuario. Se incorporan nuevos modelos de *threads* NGPT (IBM) y NPTL (Red Hat) (estos últimos ya están implementados en las versiones del *kernel* 2.4.x que trae Fedora Core).

- Soporte para USB 2.0.
- Controladores Alsa de sonido incorporados en el *kernel*.
- Nuevas arquitecturas de CPU de 64 bits, se soportan AMD x8664 y PowerPC 64.
- Sistemas de ficheros con *journal*: JFS (IBM), y XFS (Silicon Graphics).
- Mejoras de prestaciones en E/S, y nuevos modelos de controladores unificados.
- Mejoras en implementación TCP/IP, y sistema NFSv4 (compartición sistema de ficheros por red con otros sistemas).
- *Kernel* apropiativo: permite que internamente el *kernel* gestione varias tareas que se pueden interrumpir entre ellas, imprescindible para implementar eficazmente sistemas de tiempo real.

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

ANOTACIONES

- 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 testar una versión de Linux sobre otro sistema, esto es útil tanto para el propio desarrollo del *kernel*, como para un análisis de seguridad del *kernel*.

En el futuro, se tiene pensado mejorar los aspectos siguientes:

- El soporte de dispositivos PnP, ya que ahora se están realizando con unas utilidades llamada *isapnptools*, o bien asignando IRQ y DMA manualmente.

- El soporte de SMP (máquinas multiprocesador), de CPU de 64 bits (Itanium de Intel, y Opteron de AMD), el cumplimiento de los estándar POSIX, etc.

Nota

POSIX:
http://www.UNIX- systems.org/

Nota

El proyecto GNU:
http://www.gnu.org/gnu/thegnuproject.html

Nota

GNU y Linux, por Richard Stallman:
http://www.gnu.org/gnu/linux-and-gnu.html

También, aunque se aparta de los sistemas Linux, la FSF (Free Software Foundation) y su proyecto GNU siguen trabajando en su proyecto de acabar un sistema operativo completo. Cabe recordar que el proyecto GNU tenía como principal objetivo conseguir un clon UNIX de software libre, y las utilidades GNU sólo son el software de sistema necesario. A partir de 1991, cuando Linus consigue conjuntar su *kernel* 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.

Hurd fue pensado como el núcleo para el sistema GNU sobre 1990 cuando comenzó su desarrollo, ya que entonces la mayor parte del software GNU estaba desarrollado, y sólo faltaba el *kernel*. Fue en 1991 cuando Linus combinó GNU con su *kernel* 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.

ANOTACIONES

En concreto, Hurd estaba pensada como una colección de servidores implementados sobre un *microkernel* Mach [Vah96], el cual es un diseño de núcleo tipo *microkernel* (a diferencia de Linux, que es de tipo monolítico) desarrollado por la Universidad de Carnegie Mellon y posteriormente por la de Utah. La idea base era modelar las funcionalidades del *kernel* 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 éste se publicó finalmente como software libre, que permitiría usarlo para desarrollar Hurd. Comentar en este punto la importancia de Mach, ya que muchos operativos se han basado en ideas extraídas de él; el más destacado 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 (verano 2003) ya se disponen de las primeras versiones de producción así como de versiones de prueba de distribución GNU/Hurd.

Puede que en un futuro no tan lejano coexistan sistemas GNU/Linux con GNU/Hurd, o incluso sea sustituido el núcleo Linux por el Hurd, si prosperasen algunas demandas judiciales contra Linux (léase caso SCO frente a IBM), ya que sería una posibilidad para evitar problemas posteriores. En todo caso, tanto los unos como los otros sistemas, tienen un prometedor futuro por delante. El tiempo dirá hacia dónde se inclina la balanza.

4.7. Taller: Configuración del *kernel* a las necesidades del usuario

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

Una primera cosa imprescindible, antes de comenzar, es conocer la versión actual que tenemos del *kernel* con *uname -r*, para poder de-

ANOTACIONES

terminar 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 CD de la instalación, disquete de arranque creado durante la instalación (veremos cómo crear uno *a posteriori*), o disquete de rescate. Además de hacer *backup* de nuestros datos o configuraciones importantes.

Veremos las siguientes posibilidades:

1) Actualización del *kernel* de la distribución. Caso automático de Debian.

2) Actualización automática en Red Hat.

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

4.7.1. Actualizar *kernel* en Debian

En el caso de la distribución Debian *Woody* (o Sid), 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 (gnome-apt, synaptic, ...).

Vamos a efectuar la instalación por línea de comandos con *apt-get*, suponiendo que el acceso a las fuentes *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 *kernel*:

```
apt-cache search kernelimage
```

ANOTACIONES

3) Elegir una versión adecuada a nuestra arquitectura (genérica, 386/586/686 para Intel, k6 o k7 para *amd*). La versión va acompañada de versión *kernel*, revisión de Debian del *kernel* y arquitectura. Por ejemplo: 2.4.21-4-k7, *kernel* para AMD Athlon, revisión Debian 4 del *kernel* 2.4.21.

4) Comprobar, para la versión elegida, que existan los módulos accesorios extras (con el mismo número de versión). Con *apt-cache* buscamos los *pcmcia-modules* (solamente si tenemos un portátil), y los *Alsa-modules* (para tarjetas de sonido). Si queremos estos módulos, los números de versión del *kernel* tienen que coincidir.

5) Buscar, si queremos, también el código fuente del *kernel*, los *kernel- source-version* (sólo el 2.4.21, o sea, los números principales), y los *kernel-headers* correspondientes, por si luego queremos hacer un *kernel* personalizado: en este caso, el *kernel* genérico correspondiente parcheado por Debian.

6) Instalar lo que hayamos decidido; si queremos compilar desde las fuentes o simplemente disponer del código:

```
apt-get install kernel-image-version
```

`apt-get install Alsa-modules-version` (si fuera necesario)

y

```
apt-get install kernel-sources-version-generica
apt-get install kernel-headers-version
```

7) Instalar el nuevo *kernel*, por ejemplo en el *bootloader lilo*. Normalmente esto se hace automáticamente. Si se nos pregunta por si tenemos el *initrd* activado, habrá que verificar el fichero de *lilo* (/etc/lilo.conf) e incluir en la configuración de la imagen de arranque una línea:

```
initrd = /initrd.img
```

ANOTACIONES

una vez hecha esta configuración, tendríamos que tener un *lilo* del modo (fragmento):

```
default = Linux

image = /vmlinuz
     label = Linux
     read-only
     initrd = /initrd.img
#    append = "idebus = 66 ide0 = ata66 ide1 = ata66"
#    restricted
#    alias = 1
image = /vmlinuz.old
     label = LinuxOLD
     read-only
     optional
#    restricted
#    alias = 2
```

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

4.7.2. Actualizar *kernel* en Red Hat

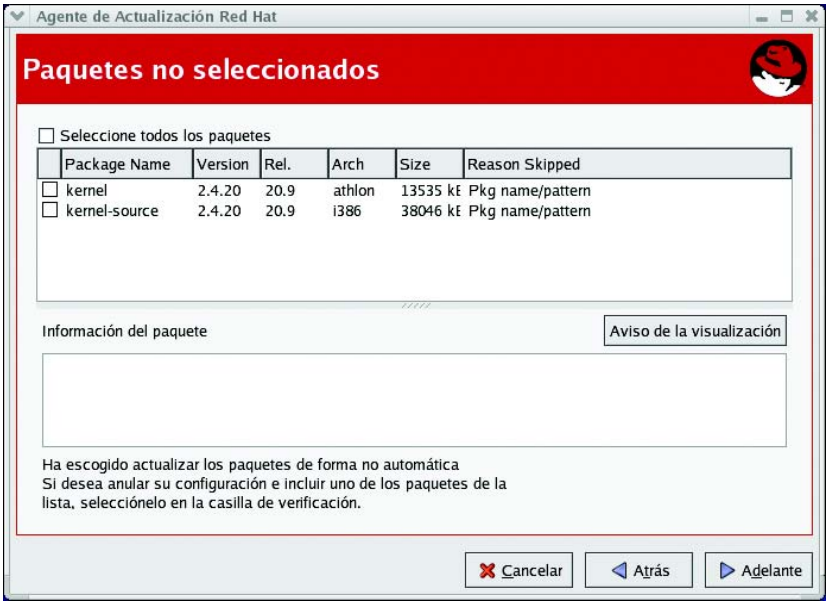
La actualización del *kernel* en la distribución Red Hat 9 es totalmente automática por medio de su servició Red Hat Network y del programa gráfico que incluye la distribución llamado *up2date*. Normalmente, lo encontraremos en la barra de tareas o en el menú de Herramientas de sistema de Red Hat.

Este programa básicamente verifica los paquetes de la distribución actual frente a una base de datos de Red Hat, y da la posibilidad de bajarse los paquetes actualizados, entre ellos los del *kernel*. Este servicio de Red Hat funciona por una cuenta de servicio, y Red Hat lo ofrece por pago, pero se puede dar de alta una cuenta de demo, a la que no nos garantizan que, en momentos de máxima demanda, tengamos acceso. Aparte de lo que se ha comentado, suele funcionar bastante bien y la actualización del *kernel* es automática.

ANOTACIONES

Por ejemplo, en la figura 10, observamos que una vez puesto en ejecución nos ha detectado una nueva versión del *kernel* disponible y podemos seleccionarla para que nos la descargue:

Figura 10. El servicio de actualización de Red Hat (Red Hat Network up2date) muestra la actualización del *kernel* disponible y sus fuentes.



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

```
#fichero grub.conf
default = 3
timeout = 10
splashimage = (hd0,1)/boot/grub/splash.xpm.gz
title Red Hat Linux (2.4.20-18.9)
root (hd0,1)
kernel /boot/vmlinuz-2.4.20-18.9 ro root = LABEL = /
initrd /boot/initrd-2.4.20-18.9.img
title Red Hat Linux (2.4.20-13.9)
root (hd0,1)
kernel /boot/vmlinuz-2.4.20-13.9 ro root = LABEL = /
initrd /boot/initrd-2.4.20-13.9.img
```

ANOTACIONES

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 *kernel*; y el fichero *initrd*, este último es una especie de imagen de disco RAM que se utiliza en el arranque. El *kernel* Linux normalmente se carga en dos fases [O'K00]: primero se crea un disco RAM en memoria, donde se copia una estructura básica del sistema, se carga el *kernel*, se descomprime (normalmente es una imagen comprimida con *gzip*) y se inicia el proceso de arranque desde la raíz / en la segunda fase.

En el caso de que dispongamos de *lilo* en la Red Hat como gestor, el sistema también lo actualiza (fichero /etc/lilo.conf), pero luego habrá que reescribir el arranque con el comando *lilo* manualmente. Normalmente, *up2date* nos informará de los pasos que hay que seguir.

Notar también que con la instalación anterior teníamos posibilidades de bajarnos las fuentes del kernel, éstas, una vez instaladas, están en /usr/src/linux-version, y pueden configurarse y compilarse por el procedimiento habitual como si fuese un *kernel* genérico. Hay que mencionar que la empresa Red Hat lleva a cabo un gran trabajo de parches y correcciones para el *kernel*, y que sus *kernel* son modificaciones al estándar genérico con bastantes añadidos, por lo cual puede ser mejor utilizar las fuentes propias de Red Hat, a no ser que queramos un *kernel* más nuevo o experimental que el que nos proporcionan.

4.7.3. Personalizar e instalar un *kernel* genérico

Vamos a ver el caso general de instalación de un *kernel* a partir de sus fuentes. Supongamos que tenemos unas fuentes ya instaladas en /usr/src. Normalmente, tendremos un directorio *linux*, o *linux-version* o sencillamente *version*; éste será el árbol de las fuentes del *kernel*.

Estas fuentes pueden provenir de la misma distribución (o que las hayamos bajado en una actualización previa), primero será interesante comprobar si son las últimas disponibles, como ya hemos hecho antes con Red Hat o Debian. O si queremos tener las últimas y genéricas versiones podemos ir a kernel.org y bajar la última versión disponible, mejor la estable que las experimentales, a no ser que estemos interesados en el desarrollo del *kernel*. Descargamos el archi-

ANOTACIONES

vo y descomprimimos en /usr/src las fuentes del *kernel*. También podríamos buscar si existen parches para el *kernel* y aplicarlos (según hemos visto en el apartado 3.4.4).

A continuación comentaremos los pasos que habrá que realizar; lo haremos brevemente, ya que muchos de ellos los tratamos anteriormente cuando trabajamos la configuración y personalización.

Nota

Sería conveniente releer el apartado 3.4.3.

1) Crear disco de arranque (por si las cosas fallan), nos permitirá arrancar la configuración actual desde disquete:

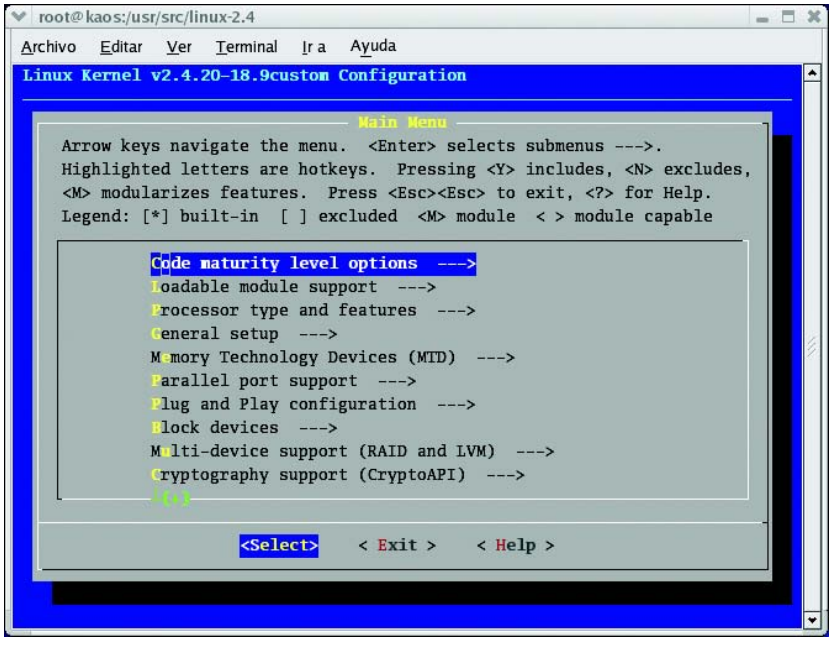
```
fdformat /dev/fd0H1440
uname -r                    (muestra la versión actual del kernel)
mkbootdisk --device /dev/fd0H1440 versión-anterior
```

Estamos formateando un disquete a bajo nivel y verificándolo. Habrá que asegurarse de que no dé errores y, si los da, volveremos a formatearlo o, para mayor seguridad, utilizaremos otro disquete. Con *mkbootdisk* creamos un disco.

2) Limpiar el directorio de pruebas anteriores (si es el caso): *make mrproper*.

3) Configurar el *kernel* con, por ejemplo: *make menuconfig* (o *xconfig*, *config* o *oldconfig*). Lo vimos en el apartado 3.4.3.

Figura 11. Configuración del *kernel* por menús textuales



ANOTACIONES

4) Dependencias y limpieza de compilaciones anteriores:

```
make dep
make clean
```

5) Compilación y creación de la imagen del *kernel*: *make bzImage*. También sería posible *zImage* si la imagen fuera más pequeña, pero es más normal *bzImage*, que optimiza el proceso de carga y la compresión para *kernels* más grandes. En algún hardware antiguo puede no funcionar y ser necesario *zImage*. El proceso puede durar de uno a decenas de minutos en hardware moderno (CPU de 1-2 GHz) y horas en hardware antiguo (hasta 100 MHz). Cuando acaba, la imagen se halla en: /usr/src/directorio-fuentes/arch/i386/boot.

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

7) En el caso de los módulos, hay que tener cuidado, ya que en alguna versión del *kernel* se sobreescribían los antiguos (en las últimas ya no es así). Si probamos alguna versión antigua (rama 2.2 o similar) o si estamos compilando una versión que es la misma que tenemos (los módulos se sobreescribirán), mejor hacer un *backup* de los módulos:

```
cd /lib/modules

tar cvfz 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
```

8) Ahora podemos pasar a la instalación del *kernel*, por ejemplo con:

```
cp /usr/src/directorio-Fuentes/System.map /boot/System.map-versionkernel
ln -s /boot/System.map-versionkernel /boot/System.map

cp bzImage /boot/bzImage-versionkernel
```

Así colocamos el fichero de símbolos del *kernel* (*System.map*) y la imagen del *kernel*.

9) Ya sólo nos queda poner la configuración necesaria en el fichero de configuración del gestor de arranque, ya sea *lilo* (/etc/lilo.conf) o *grub* (/boot/grub/grub.conf) según las configuraciones que ya vimos con Red Hat o Debian. Y recordar, en el caso de *lilo*, que habrá que volver a actualizar la configuración con /sbin/lilo o /sbin/lilo -*v*.

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

4.8. Actividades para el lector

1) Determinar la versión actual del *kernel* incorporado en nuestra distribución. Comprobar las actualizaciones disponibles de forma automática, ya sea en Debian (*apt*) o en Red Hat (via *up2date*).

2) Realizar una actualización automática de nuestra distribución. Comprobar posibles dependencias con otros módulos utilizados (ya sean PCMCIA, Alsa u otros), y con el *bootloader* (*lilo* o *grub*) utilizado. Se recomienda *backup* de los datos importantes del sistema (cuentas de usuarios y ficheros de configuración modificados), o bien realizarlo en otro sistema del que se disponga para pruebas.

3) Para nuestra serie del *kernel*, determinar la última versión disponible (consultar www.kernel.org), y realizar una instalación manual con los pasos examinados en la unidad. La instalación final puede dejarse opcional, o bien poner una entrada dentro del *bootloader* para las pruebas del nuevo *kernel*.

4) En el caso de la distribución Debian, además de los pasos manuales explicados en la unidad, existe una forma especial (recomendada) de instalar el *kernel* a partir de sus fuentes mediante el paquete de software kernel-package, ver en:

- http://myrddin.org/howto/debian-kernel-recompile.html, o en
- http://www.debian.org/doc/manuals/reference/chkernel.en.html.

ANOTACIONES

4.9. Otras fuentes de referencia e información

- [Ker03b] Sitio que proporciona almacén de las diversas versiones del *kernel* Linux y sus parches.

- [Ker03a] [lkm03] Sitios web que recogen a una parte de la comunidad del *kernel* de Linux, dispone de varios recursos de documentación y listas de correo de la evolución del *kernel*, su estabilidad y las nuevas prestaciones que se desarrollan.

- [DB03] Libro sobre el *kernel* de Linux 2.4, que detalla los diferentes componentes y su implementación y diseño. Existe una primera edición sobre el *kernel* 2.2.

- [Pra03] Artículo que describe algunas de las principales novedades de la nueva serie 2.6 del *kernel* Linux.

- [Ker02] [Mur02] Proyectos de documentación del *kernel*, incompletos pero con material útil.

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

- [Sko03a][Zan01][Kan01][Pro02] Para tener más información sobre los cargadores *lilo* y *grub*.

ANOTACIONES

5. Administración local

Una de las primeras tareas con la que tendrá que enfrentarse el administrador será la gestión de los recursos locales presentes en la máquina. En el curso de introducción a GNU/Linux, se cubrieron algunos de estos aspectos de forma básica. En el presente, veremos algunas de estas tareas de administración de forma más profunda, y algunos de los aspectos de personalización y rendimiento de los recursos.

Comenzaremos por analizar el proceso de arranque de un sistema GNU/Linux, que nos hará comprender la estructura inicial del sistema y su relación con los diversos servicios que éste proporciona.

A continuación aprenderemos cómo obtener una visión general del estado actual del sistema por medio de los diferentes procedimientos y comandos de que se dispone para evaluar las diversas partes del sistema; de este modo podremos tomar decisiones de administración si detectamos algún fallo o deficiencia de rendimiento, o la falta de algún recurso.

Nota

La administración local engloba muchas tareas variadas, que quizás sean las más utilizadas por el administrador en su trabajo diario.

Uno de los principales puntos de la administración es la gestión de usuarios, ya que cualquier configuración de la máquina estará destinada a que pueda ser utilizada por éstos; veremos cómo definir nuevos usuarios al sistema y controlar su nivel de acceso a los recursos.

En cuanto a los periféricos del sistema, como discos e impresoras, disponemos de diferentes posibilidades de gestión, ya sea vía diferentes servidores (caso impresión) o diferentes sistemas de archivos que podemos tratar, así como algunas técnicas de optimización del rendimiento de los discos.

También examinaremos el problema de la actualización del sistema y cómo mantenerlo actualizado; así como la nueva incorporación de software de aplicación y cómo hacerlo disponible a los usuarios. Asi-

ANOTACIONES

mismo, analizaremos la problemática de ejecutar trabajos temporizados en el sistema.

En el taller final examinaremos la evaluación de estado de una máquina, siguiendo los puntos vistos en esta unidad, y llevaremos a cabo algunas de las tareas de administración básicas descritas. En el desarrollo de la unidad comentaremos algunos comandos, y posteriormente, en el taller, veremos algunos de ellos con más detalle en lo que respecta a su funcionamiento y opciones.

5.1. Distribuciones: particularidades

Intentamos destacar ahora algunas diferencias menores en las distribuciones (Red Hat y Debian) utilizadas [Mor03], que iremos viendo con más detalle a lo largo de las unidades, a medida que vayan apareciendo.

Nota

Es importante conocer los detalles de una distribución, ya que pueden ser básicos para resolver una tarea o acelerar su solución (por ejemplo, si dispone de herramientas propias).

Cambios o particularidades de Red Hat:

- Uso del gestor de arranque *grub* (un software GNU), a diferencia de la mayoría de distribuciones que suelen usar *lilo*, Red Hat utiliza *grub*. GRUB (*Grand Unified Bootloader*) tiene una configuración en modo texto (normalmente en /boot/grub/grub.conf) bastante sencilla, y que puede modificarse en el arranque. Es quizás más flexible que *lilo*.

- Gestión de alternativas. En el caso de que haya más de un software presente, mediante un directorio (/etc/alternatives), se indica cuál es la alternativa que se usa. Este sistema se tomó prestado de Debian, que hace un uso amplio de él en su distribución.

- Programa de escucha de puertos TCP/IP basado en xinetd; en /etc/xinetd.d podemos encontrar de forma modular los ficheros de configuración para algunos de los servicios TCP/IP, junto con el fichero de configuración /etc/xinetd.conf. En los sistemas UNIX clásicos, el programa utilizado es el *inetd*, que poseía un único fichero de configuración en /etc/inetd.conf, caso, por ejemplo, de la distribución Debian que utiliza *inetd*.

ANOTACIONES

- Algunos directorios de configuración especiales: /etc/profile.d, archivos que se ejecutan cuando un usuario abre un *shell*; /etc/xinetd.d, configuración de algunos servicios de red; /etc/sysconfig, datos de configuración de varios aspectos del sistema; /etc/cron., varios directorios donde se especifican trabajos para hacer periódicamente (mediante *crontab*); /etc/pam.d, donde PAM son los denominados *módulos de autentificación*: en cada uno de cuyos archivos se configuran permisos para el programa o servicio particular; /etc/logrotate.d, configuración de rotación (cuando hay que limpiar, comprimir, etc.) de algunos de los ficheros de *log* para diferentes servicios.

- Dispone de un software llamado *kudzu*, que examina el hardware en arranque para detectar posibles cambios de configuración y generar los dispositivos o configuraciones adecuadas.

En el caso de Debian:

- Sistema de empaquetado propio basado en los paquetes DEB, con herramientas de varios niveles para trabajar con los paquetes como: *dpkg* , *apt-get*, *dselect*, *tasksel*.

- Debian sigue el FHS, sobre la estructura de directorios, añadiendo algunos particulares en /etc, como por ejemplo: /etc/default, archivos de configuración, y valores por defecto para algunos programas; /etc/network, datos y guiones de configuración de las interfaces de red; /etc/dpkg y /etc/apt, información de la configuración de las herramientas de gestión de paquetes; /etc/alternatives, enlaces a los programas por defecto, en aquellos que hay (o puede haber) varias alternativas disponibles.

- Sistema de configuración de muchos paquetes de software por medio de la herramienta dpkg-reconfigure. Por ejemplo:

  ```
  dpkg-reconfigure gdm
  ```

  permite escoger el gestor de entrada para X, o:

  ```
  dpkg-reconfigure X-Window-system
  ```

  nos permite configurar los diferentes elementos de X.

- Utiliza configuración de servicios TCP/IP por *inetd*, configuración en fichero /etc/inetd.conf; dispone de una herramienta *update-inetd* para inhabilitar o crear entradas de servicios.

ANOTACIONES

- Algunos directorios de configuración especiales: /etc/cron., varios directorios donde se especifican trabajos para hacer periódicamente (mediante *crontab*); /etc/pam.d, donde PAM son módulos de autentificación.

5.2. Niveles de arranque y servicios

Un primer punto importante en el análisis del comportamiento local del sistema es su posible funcionamiento en los llamados niveles de ejecución (o *runlevels*), y en los servicios que se proporcionan en cada nivel [W+02].

Un **servicio** es una funcionalidad proporcionada por la máquina. La activación o parada de servicios se realiza mediante la utilización de *scripts*. La mayoría de los servicios estándar, los cuales suelen tener su configuración en el directorio /etc, suelen controlarse mediante los *scripts* presentes en /etc/init.d/. En este directorio suelen aparecer *scripts* con nombres similares al servicio donde van destinados, y se suelen aceptar parámetros de activación o parada. Se realiza:

`/etc/init.d/servicio start`	arranque del servicio.
`/etc/init.d/servicio stop`	parada del servicio.
`/etc/init.d/servicio restart`	parada y posterior arranque del servicio.

Cuando un sistema GNU/Linux arranca, primero se carga el *kernel* del sistema, después se inicia el primer proceso, denominado *init*, que es el responsable de ejecutar y activar el resto del sistema, mediante la gestión de los niveles de ejecución (o *runlevels*).

Un **nivel de ejecución** es sencillamente una configuración de programas y servicios que se ejecutarán orientados a un determinado funcionamiento.

ANOTACIONES

Los niveles típicos suelen ser (puede haber algunas diferencias):

Runlevel	Función	Descripción
0	Parada	Finaliza servicios y programas activos, así como desmonta *filesystems* activos y para la CPU.
1	Monousuario	Finaliza la mayoría de servicios, permitiendo sólo la entrada del administrador (*root*). Se usa para tareas de mantenimiento y corrección de errores críticos.
2	Multiusuario sin red	No se inician servicios de red, permitiendo sólo entradas locales en el sistema.
3	Multiusuario	Inicia todos los servicios excepto los gráficos asociados a X Window.
4	Multiusuario	No suele usarse, típicamente es igual que el 3.
5	Multiusuario	Igual que 3, pero con soporte X para la entrada de usuarios (*login* gráfico).
6	Reinicio	Para todos los programas y servicios. Reinicia el sistema.

Estos niveles suelen estar configurados en los sistemas GNU/Linux (y UNIX) por dos sistemas diferentes, el BSD, o el System V. En el caso de Red Hat y Debian, se utiliza el sistema System V, que es el que explicaremos, pero otros UNIX y alguna distribución GNU/Linux (como Slackware) utilizan el modelo BSD.

En el caso del modelo *runlevel* de System V, cuando el proceso *init* arranca, utiliza un fichero de configuración llamado /etc/inittab para decidir el modo de ejecución en el que va a entrar. En este fichero se define el *runlevel* por defecto en arranque, y una serie de servicios de terminal por activar para atender la entrada del usuario.

Después, el sistema, según el *runlevel* escogido, consulta los ficheros contenidos en /etc/rcn.d, donde *n* es el número asociado al *runlevel*, en el que se encuentra una lista de servicios por activar o parar en caso de que arranquemos en el *runlevel*, o lo abandonemos. Dentro del directorio encontraremos una serie de *scripts* o enlaces a los *scripts* que controlan el servicio.

Cada *script* posee un nombre relacionado con el servicio, una S o K inicial que indica si es el *script* para iniciar (S) o matar (K) el servicio, y un número que refleja el orden en que se ejecutarán los servicios.

Una serie de comandos de sistema sirven de ayuda para manejar los niveles de ejecución, cabe mencionar:

- Los *scripts*, que ya hemos visto, en /etc/init.d/ nos permiten arrancar, parar o reiniciar servicios individuales.

ANOTACIONES

- *telinit*, nos permite cambiar de nivel de ejecución, sólo tenemos que indicar el número. Por ejemplo, necesitamos hacer una tarea crítica en *root*; sin usuarios trabajando, podemos hacer un *telinit* 1 (también puede usarse S) para pasar a *runlevel* monousuario, y después de la tarea un *telinit 3* para volver a multiusuario. También puede utilizarse el comando *init* para la misma tarea, aunque *telinit* aporta algún parámetro extra. Por ejemplo, el reinicio típico de un sistema UNIX se hacía con *sync; sync; sync; init 6*, el comando *sync* fuerza el vaciado de los *buffers* del sistema de archivos, y luego reiniciamos en *runlevel* 6.

- *shutdown*, permite parar ('h' de *halt*) o reiniciar el sistema ('r' de *reboot*). Puede darse también un intervalo de tiempo para hacerse, o bien inmediatamente. Para estas tareas también existen los comandos *halt* y *reboot*.

- *wall*, permite enviar mensajes de advertencia a los usuarios del sistema. Concretamente, el administrador puede anunciar que se va a parar la máquina en un determinado momento. Comandos como *shutdown* suele utilizarlo de forma automática.

- *pidof*, permite averiguar el PID (*process ID*) asociado a un proceso. Con *ps* obtenemos los listados de procesos, y si queremos eliminar un servicio o proceso mediante *kill*, necesitaremos su PID.

Respecto a todo el modelo de arranque, las distribuciones presentan algún pequeño cambio:

- Red Hat: el *runlevel* 4 no tiene un uso declarado. Los directorios /etc/rcn.d existen como enlaces hacia subdirectorios de /etc/rc.d, donde están centralizados los *scripts* de arranque. Los directorios son, así: /etc/rc.d/rcn.d; pero como existen los enlaces, es transparente al usuario. El *runlevel* por defecto es el 5 con arranque con X un programa llamado *prefdm* gestiona el escritorio preferido.

  Los comandos y ficheros relacionados con el arranque del sistema están en los paquetes de software *sysvinit* y *initscripts*.

  Respecto a los cambios de ficheros y guiones en Red Hat, cabe destacar: en /etc/sysconfig podemos encontrar archivos que es-

ANOTACIONES

pecifican valores por defecto de la configuración de dispositivos o servicios. El guión /etc/rc.d/rc.sysinit es invocado una vez cuando el sistema arranca; el guión /etc/rc.d/rc.local se invoca al final del proceso de carga y sirve para indicar configuraciones específicas de la máquina.

El arranque real de los servicios se hace por medio de los guiones almacenados en /etc/rc.d/init.d. Hay también un enlace desde /etc/init.d. Además, Red Hat proporciona unos *scripts* de utilidad para manejar servicios: /sbin/service para parar o iniciar un servicio por el nombre; y /sbin/chkconfig, para añadir enlaces a los ficheros S y K necesarios para un servicio.

- Debian dispone de comandos de gestión de los *runlevels* como *updaterc.d*, que permite instalar o borrar servicios arrancándolos o parándolos en uno o más *runlevels*; *invoke-rc.d*, permite las clásicas acciones de arrancar, parar o reiniciar el servicio.

El *runlevel* por defecto en Debian es el 2, el X Window System no se gestiona desde /etc/inittab, sino que existe el gestor (por ejemplo, *gdm* o *kdm*) como si fuera un servicio más del *runlevel* 2.

5.3. Observar el estado del sistema

Una de las principales tareas del administrador (*root*) en su día a día, será verificar el correcto funcionamiento del sistema y vigilar la existencia de posibles errores o de saturación de los recursos de la máquina (memoria, discos, etc.). Pasaremos a detallar en los siguientes subapartados los métodos básicos para examinar el estado del sistema en un determinado momento y llevar a cabo las acciones necesarias para evitar problemas posteriores.

En el taller final de esta unidad, realizaremos un examen completo del sistema para que se puedan ver algunas de estas técnicas.

5.3.1. Arranque del sistema

En el arranque de un sistema GNU/Linux, se produce todo un volcado de información interesante; cuando el sistema arranca, suelen

ANOTACIONES

aparecen los datos de detección de las características de la máquina, detección de dispositivos, arranque de servicios de sistema, etc., y se mencionan los problemas aparecidos.

En la mayoría de las distribuciones esto puede verse en la consola del sistema directamente durante el proceso de arranque. Sin embargo, o la velocidad de los mensajes o algunas modernas distribuciones que los ocultan tras carátulas gráficas pueden impedir seguir los mensajes correctamente, con lo que necesitaremos una serie de herramientas para este proceso.

Básicamente, podemos utilizar:

- Comando *dmesg*: da los mensajes del último arranque del *kernel*.
- Fichero /var/*log*/messages: *log* general del sistema, que contiene los mensajes generados por el *kernel* y otros *daemons*.
- Comando *uptime*: indica cuánto tiempo hace que el sistema está activo.
- Ficheros /proc: ficheros que utiliza el *kernel* para almacenar la información que gestiona.

5.3.2. *Kernel*: Directorio/proc

El *kernel* durante su arranque pone en funcionamiento un *pseudo-filesystem* llamado /proc, donde vuelca la información que recopila de la máquina, así como muchos de sus datos internos. El directorio /proc está implementado sobre memoria y no se guarda en disco. Los datos contenidos son tanto de naturaleza estática como dinámica (varían durante la ejecución).

Hay que tener en cuenta que al ser /proc fuertemente dependiente del *kernel*, propicia que su estructura dependa del *kernel* de que dispone el

ANOTACIONES

sistema y la estructura y los ficheros incluidos pueden cambiar (los comentados pertenecen a una serie 2.4 del *kernel*).

Una de las características interesantes es que en el directorio /proc podremos encontrar las imágenes de los procesos en ejecución, junto con la información que el *kernel* maneja acerca de ellos. Cada proceso del sistema se puede encontrar en el directorio /proc/pid-proceso, donde hay un directorio con ficheros que representan su estado. Esta información es útil para programas de depuración, o bien para los propios comandos del sistema como *ps* o *top*, que pueden utilizarla para ver el estado de los procesos.

Nota

El directorio /proc es un recurso extraordinario para obtener información de bajo nivel sobre el funcionamiento del sistema, muchos comandos de sistema se apoyan en él para sus tareas.

Por otra parte, en /proc podemos encontrar otros ficheros de estado global del sistema. Comentamos brevemente algunos ficheros que podremos examinar:

Fichero	Descripción
/proc/bus	Directorio con información de los buses PCI y USB
/proc/cmdline	Línea de arranque del *kernel*
/proc/cpuinfo	Información de la CPU
/proc/devices	Dispositivos del sistema de caracteres o bloques
/proc/drive	Información de algunos módulos de hardware
/proc/*filesystems*	Sistemas de ficheros habilitados en el *kernel*
/proc/ide	Directorio de información del bus IDE, características de discos
/proc/interrups	Mapa de interrupciones hardware (IRQ) utilizadas
/proc/ioports	Puertos de E/S utilizados
/proc/meminfo	Datos del uso de la memoria
/proc/modules	Módulos del *kernel*
/proc/net	Directorio con toda la información de red
/proc/pci	Dispositivos pci del sistema
/proc/scsi	Directorio de dispositivos SCSI, o IDE emulados por SCSI
/proc/version	Versión y fecha del *kernel*

ANOTACIONES

5.3.3. Procesos

Los procesos que se encuentren en ejecución en un determinado momento serán, en general, de diferente naturaleza. Podemos encontrar:

- **Procesos de sistema**, o bien procesos asociados al funcionamiento local de la máquina y del *kernel*, o bien procesos (denominados *daemons*) asociados al control de diferentes servicios, ya sean locales, o de red, porque estamos ofreciendo el servicio (actuamos de servidor) o estamos recibiendo el servicio (actuamos de clientes). La mayoría de estos procesos aparecerán asociados al usuario *root*, aunque no estemos presentes en ese momento como usuarios. Puede haber algunos servicios asociados a otros usuarios de sistema (lp, bin, www, mail, etc.), estos son usuarios "virtuales" que utiliza el sistema para ejecutar ciertos procesos.

- **Procesos del usuario administrador**: en caso de actuar como *root*, nuestros procesos interactivos o aplicaciones lanzadas también aparecerán como procesos asociados al usuario *root*.

- **Procesos de usuarios del sistema**: asociados a la ejecución de sus aplicaciones, ya sea tareas interactivas en modo texto o en modo gráfico.

Como comandos rápidos y más útiles podemos utilizar:

- *ps*: el comando estándar, lista los procesos con sus datos de usuario, tiempo, identificador de proceso y línea de comandos usada. Una de las opciones utilizada es *ps ef*, pero hay muchas opciones disponibles (ver *man*).

- *top*: una versión que nos da una lista actualizada a intervalos.

- *kill*: nos permite eliminar procesos del sistema mediante el envío de señales al proceso como, por ejemplo, la de terminación *kill -9 PID*, donde indicamos el identificador del proceso. Útil para procesos con comportamiento inestable o programas interactivos que han dejado de responder.

ANOTACIONES

5.3.4. *Logs* del sistema

Tanto el *kernel* como muchos de los *daemons* de servicios, así como diferentes aplicaciones o subsistemas de GNU/Linux, pueden generar mensajes que vayan a parar a ficheros *log*, ya sea para tener una traza de su funcionamiento, o bien para detectar errores o advertencias de malfuncionamiento o situaciones críticas. Este tipo de *logs* son imprescindibles en muchos casos para las tareas de administración y se suele emplear bastante tiempo de administración en el procesamiento y análisis de sus contenidos.

La mayor parte de los *logs* se generan en el directorio /var/log, aunque algunas aplicaciones pueden modificar este comportamiento; la mayoría de *logs* del propio sistema sí que se encuentran en este directorio.

Un *daemon* particular del sistema (importante) es el *daemon* Syslogd. Este *daemon* se encarga de recibir los mensajes que se envían por parte del *kernel* y otros *daemons* de servicios y los envía a un fichero *log* que se encuentra en /var/*log*/messages. Éste es el fichero por defecto, pero Syslogd es también configurable (fichero /etc/syslog.conf), de manera que se puede generar otro fichero o bien, según el *daemon* que envía el fichero, dirigirlo a un *log* u a otro (clasificar por fuente) o clasificar los mensajes por importancia: *alarm* , *warning* , *error*, *critical* , etc.

Nota

El *daemon* Syslogd es el servicio más importante de obtención de información dinámica de la máquina. El proceso de análisis de los *logs* nos ayuda a entender el funcionamiento, los posibles errores y el rendimiento del sistema.

Dependiendo de la distribución, puede estar configurado de diferentes modos por defecto, en /var/log suele generar (por ejemplo) en Debian ficheros como: *kern.log*, *mail.err*, *mail.info*, ... que son los *logs* de diferentes servicios. Podemos examinar la configuración para determinar de dónde provienen los mensajes y en qué ficheros los guarda. Una opción que suele ser útil es la posibilidad de enviar los mensajes a una consola virtual de texto, de manera que podremos ir viéndolos a medida que se produzcan. Esto suele ser útil para monitorizar la ejecución del sistema sin tener que estar mirando el fichero a cada momento. Una modificación simple de este método podría ser introducir, desde un terminal, la instrucción siguiente:

```
tail -f /var/log/messages
```

ANOTACIONES

Esta sentencia nos permite dejar el terminal o ventana de terminal, de manera que irán apareciendo los cambios que se produzcan en el fichero.

Otros comandos relacionados:

- *uptime*: tiempo que hace que el sistema está activo.
- *last*: analiza *log* de entradas/salidas del sistema (/var/*log*/wtmp).

5.3.5. Memoria

Respecto a la memoria del sistema, tendremos que tener en cuenta que disponemos: a) de la memoria física de la propia máquina, b) memoria virtual, que puede ser direccionada por los procesos. Normalmente, no dispondremos de cantidades demasiado grandes, de modo que la memoria física será menor que el tamaño de memoria virtual necesario. Esto obligará a utilizar una zona de intercambio (*swap*) sobre disco.

Esta zona de intercambio (*swap*) puede implementarse como un fichero en el sistema de archivos, pero es más habitual encontrarla como una partición de intercambio (llamada de *swap*), creada durante la instalación del sistema. En el momento de particionar el disco, se declara como de tipo Linux Swap.

Para examinar la información sobre memoria tenemos varios métodos y comandos útiles:

- Fichero /etc/fstab: aparece la partición *swap* (si existiese). Con un comando *fdisk* podemos averiguar su tamaño.
- Comando *ps*: permite conocer qué procesos tenemos, y con las opciones de porcentaje y memoria usada.
- Comando *top*: es una versión *ps* dinámica actualizable por periodos de tiempo. Puede clasificar los procesos según la memoria que usan o el tiempo de CPU.

ANOTACIONES

- Comando *free*: informa sobre el estado global de la memoria. Da también el tamaño de la memoria virtual.

- Comando *vmstat*: informa sobre el estado de la memoria virtual, y el uso que se le da.

5.3.6. Discos y *filesystems*

Examinaremos qué discos tenemos disponibles, cómo están organizados y de qué particiones y archivos de sistemas (*filesystems*) disponemos.

Para crear una partición y asociarla a un determinado *filesystem* accesible, tendremos que realizar un proceso de montaje, ya sea explícitamente o bien programada en arranque. En el proceso de montaje se conecta el sistema de archivos asociado a la partición a un punto del árbol de directorios.

Para conocer los discos (o dispositivos de almacenamiento) que tenemos en el sistema, podemos basarnos en la información de arranque del sistema (*dmesg*), donde se detectan los presentes, como los /dev/hdx para los dispositivos IDE o los SCSI con dispositivos /dev/sdx. Otros dispositivos, como discos duros conectados por USB, discos flash (los de tipo *pen drive*), unidades zips, CD-ROM externos, suelen ser dispositivos con algún tipo de emulación SCSI, por lo que se verán como dispositivo de este tipo.

Cualquier dispositivo de almacenamiento presentará una serie de particiones de su espacio. Típicamente, un disco IDE soporta un máximo de 4 particiones físicas, o más si éstas son lógicas (que permiten colocar varias particiones de este tipo sobre una física). Cada partición puede contener tipos de *filesystems* diferentes, ya sean de un mismo operativo o de operativos diferentes.

Para examinar la estructura de un dispositivo conocido, o cambiar su estructura particionando el disco, podemos utilizar el comando *fdisk*,

ANOTACIONES

o cualquiera de sus variantes más o menos interactivas (*fdisk, sfdisk*). Por ejemplo, al examinar un disco ide /dev/hda, nos da la siguiente información:

```
# fdisk /dev/hda (opción p)
Disk /dev/hda: 20.5 GB, 20520493056 bytes
255 heads, 63 sectors/track, 2494 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1	*	1	1305	10482381	7	HPFS/NTFS
/dev/hda2	*	1306	2429	9028530	83	Linux
/dev/hda3		2430	2494	522112+	82	Linux swap

Disco de 20 GB con tres particiones (se identifican con el número añadido al nombre del dispositivo), donde observamos dos particiones con arranque (columna *Boot* con) de tipo NTFS y Linux, lo que supone la existencia de un Windows NT/2000/XP junto con una distribución Linux, y la ultima partición que es usada de *swap* para Linux. Además tenemos información de la estructura del disco y de los tamaños de cada partición.

De los discos y particiones de que dispongamos, algunos de ellos se encontrarán montados en nuestro sistema de ficheros, o estarán preparados para montarse bajo demanda, o bien montarse en el momento en que dispongan de medio (en el caso de dispositivos extraíbles).

Esta información la podemos obtener de diferentes maneras (veremos más detalle en el taller final):

- Fichero /etc/fstab. Indica dispositivos que están preparados para montarse en el arranque o los extraíbles que podrán ser montados. No tienen por qué estar todos los del sistema, sino sólo aquellos que queramos tener en arranque. Los demás podemos montarlos bajo demanda con el comando *mount*, o desmontarlos con *umount*.

- Comando *mount*. Nos informa de los *filesystems* montados en ese momento (ya sean dispositivos reales o *filesystem* virtuales como

ANOTACIONES

/proc). Podemos obtener esta información también desde el fichero /etc/mtab.

- Comando *df -k*. Nos informa de los *filesystems* de almacenamiento, y nos permite verificar el espacio usado y disponible. Comando básico para controlar espacio de disco disponible.

Respecto a este ultimo comando *df –k*, una de nuestras tareas básicas de administración de la máquina es controlar los recursos de la máquina, y en este caso el espacio disponible en los *filesystems* utilizados. Estos tamaños hay que monitorizarlos con cierta frecuencia para evitar la caída del sistema; nunca tendría que dejarse un *filesystem* (y más si es el /) por debajo de un 10 o 15%, ya que hay muchos procesos *daemons* que están escribiendo normalmente información temporal o *logs*, que pueden generar gran información; un caso particular lo forman los ficheros *core,* ya comentados, que pueden suponer (dependiendo del proceso) tamaños muy grandes de archivo. Normalmente, habrá que tener algunas precauciones de "limpieza del sistema" si se detectan situaciones de saturación del *filesystem*:

- Eliminar temporales antiguos. Los directorios /tmp y /var/tmp suelen acumular muchos archivos generados por diferentes usuarios o aplicaciones. Algunos sistemas o distribuciones ya toman medidas de limpieza, como limpiar /tmp en cada arranque del sistema.

- *Logs*: evitar su crecimiento excesivo, según la configuración del sistema (por ejemplo de Syslogd) la información generada de mensajes puede ser muy grande. Normalmente, habrá que limpiar periódicamente al llegar a determinados tamaños, y en todo caso, si necesitamos la información para posteriores análisis, podemos realizar *backups* en medios extraíbles.

- Hay otros puntos del sistema que suelen crecer mucho, como pueden ser: a) ficheros core de los usuarios: podemos eliminarlos periódicamente o eliminar su generación; b) el sistema de correo electrónico: almacena todos los correos enviados y recibidos, podemos pedir a los usuarios que hagan limpieza periódica, o bien poner sistemas de cuotas; c) las cachés de los navegadores u otras aplicaciones: también suelen tener tamaños grandes, otra limpieza que habrá que hacer periódicamente; d) las cuentas de los propios usuarios: pueden tener cuotas para no exceder los tamaños prefijados; ...

ANOTACIONES

5.4. Sistema de ficheros

En cada máquina con un sistema GNU/Linux podemos encontrarnos con diferentes sistemas de ficheros de diferentes tipos [Hin00].

Para empezar, es habitual encontrarse con los propios sistemas de ficheros Linux creados en diversas particiones de los discos [Koe01]. La configuración habitual suele ser de dos particiones: la correspondiente a "/" (*root filesystem*) y la correspondiente al fichero de intercambio o de *swap*.

La de *swap* es de tipo Linux *swap*, y la correspondiente a / suele ser de alguno de los sistemas de ficheros estándar, ya sea ext2 (el tipo por defecto hasta los *kernels* 2.4), o el nuevo ext3, que es una mejora del ext2 compatible pero con *journaling*, lo que permite tener un *log* de lo que va pasando al sistema de ficheros, para recuperaciones más rápidas en caso de error.

Otra configuración habitual puede ser de 3 particiones: /, *swap*, /home, donde la /home se dedicará a las cuentas de los usuarios. Esto permite separar las cuentas de los usuario del sistema, aislando en dos particiones separadas, y podemos dar el espacio necesario para las cuentas en otra partición.

Otro esquema muy utilizado es el de separar en particiones las partes estáticas del sistema de las dinámicas, por ejemplo, una partición donde se coloca / con la parte estática (/bin /sbin y /usr en algunos casos) que se espera que no va a crecer o lo va a hacer muy poco, y otra o varias con la parte dinámica (/var /tmp /opt), suponiendo que /opt, por ejemplo, es el punto de instalación del software nuevo. Esto permite ajustar mejor el espacio de disco y dejar más espacio para las partes del sistema que lo necesiten.

5.4.1. Puntos de montaje

Aparte del *filesystem* principal / y de sus posibles divisiones en particiones extras (/usr /var /tmp /home), cabe tener en cuenta la posibilidad de dejar puntos de montaje preparados para el montaje de otros *filesystems*, ya sea particiones de disco u otros dispositivos de almacenamiento.

ANOTACIONES

En las máquinas en que GNU/Linux comparte la partición con otros sistemas operativos, mediante algún sistema de arranque (*lilo* o *grub*), pueden existir varias particiones asignadas a los diferentes operativos. Muchas veces es interesante compartir datos con estos sistemas, ya sea para leer sus ficheros o modificarlos. A diferencia de otros sistemas (que sólo tienen en cuenta sus propios datos, léase Windows, en el cual en algunas versiones no se soportan algunos de sus propios sistemas de ficheros), GNU/Linux es capaz de tratar con una cantidad enorme de sistemas de ficheros de diferentes operativos y poder compartir la información.

Ejemplo

Si en los PC personales hemos instalado GNU/Linux, seguramente encontraremos más de un operativo, por ejemplo, otra versión de GNU/Linux con ext2 o 3 de sistema de ficheros, podríamos encontrar un antiguo msdos con su sistema de ficheros FAT, un Windows98/ME/XP Home con FAT32 (o vfat para Linux), o un Windows NT/2000/XP profesional con sistemas NTFS (ntfs para Linux) y FAT32 (vfat) a la vez.

Nuestro sistema GNU/Linux puede leer datos (o sea ficheros y directorios) de todos estos sistemas de ficheros y escribir en la mayoría de ellos; en el caso de NTFS, este soporte existe pero en versión experimental (ya que existen dos versiones llamadas NTFS y NTFS2, y los llamados volúmenes dinámicos; hay ciertas incompatibilidades entre ellos y podrían causar corrupciones de datos o errores en el sistema de ficheros). En la versión 2.6 del *kernel* Linux se espera tener un soporte más o menos completo de NTFS (mientras Microsoft no haga más cambios en la especificación).

Para que se puedan leer o escribir los datos, la partición tiene que estar disponible dentro de nuestro sistema de ficheros raíz (/). Por lo tanto, hay que llevar a cabo un proceso de "montaje" del sistema de ficheros en algún punto de nuestro árbol de directorios. Se seguirá el mismo proceso si se trata de un dispositivo de almacenamiento, ya sea disquete o *floppy*.

ANOTACIONES

Dependiendo de la distribución, se usan unos u otros, o también los podemos crear nosotros. Normalmente suelen existir o bien como subdirectorios de la raíz, por ejemplo /cdrom /win /floppy, o bien son subdirectorios dentro de /mnt, el punto estándar de montaje (aparecen como /mnt/cdrom /mnt/floppy ...).

El proceso de montaje se realiza mediante la orden *mount* con el siguiente formato:

```
mount -t filesystem-type device mount-point
```

El tipo de *filesystem* puede ser msdos (fat), vfat (fat32), ntfs (ntfs lectura), iso9660 (para cdrom), ...

El dispositivo es la entrada correspondiente en el directorio /dev a la localización del dispositivo, los IDE tenían /dev/hdxy donde *x* es a,b,c, o d (1 master, 1 slave, 2 master, 2 slave) e *y*, el número de partición, los SCSI (/dev/sdx) donde *x* es a,b,c,d ... (según el ID SCSI asociado 0,1,2,3,4 ...).

Vamos a ver algunos ejemplos:

`mount -t iso9660 /dev/hdc /mnt/cdrom`
montaría el CD-ROM (si es el IDE que está en el segundo IDE de forma master) en el punto /mnt/cdrom.

`mount -t iso9660 /dev/cdrom /mnt/cdrom`
montaría el CD-ROM; /dev/cdrom se usa como sinónimo (es un *link*) del dispositivo donde está conectado.

`mount -t vfat /dev/fd0H1440 /mnt/floppy`
montaría el disquete, /dev/fd0H1440. Sería la disquetera A en alta densidad (1.44 MB), también puede usarse /dev/fd0.

`mount -t ntfs /dev/hda2 /mnt/winXP`
montaría la segunda partición del primer dispositivo IDE (la C:), de tipo NTFS (por ejemplo un Windows XP).

Si estas particiones son más o menos estables en el sistema (o sea, no cambian frecuentemente) y las queremos utilizar, lo mejor será in-

ANOTACIONES

cluir los montajes para que se hagan en tiempo de ejecución, mediante la configuración del fichero /etc/fstab:

```
# /etc/fstab: Información estática del sistema de ficheros
#
# <Sis. ficheros>   <Punto montaje>   <Tipo>    <Opciones>          <volcado>   <pasada>
/dev/hda2           /                 ext3      errors = remountro  0           1
/dev/hdb3           none              swap      sw                  0           0
proc                /proc             proc      defaults            0           0
/dev/fd0            /floppy           auto      user,noauto         0           0
/dev/cdrom          /cdrom            iso9660   ro,user,noauto      0           0
/dev/sdb1           /mnt/usb          vfat      user,noauto         0           0
```

Por ejemplo, esta configuración incluye algunos de los sistemas estándar, como la raíz en /dev/hda2, la partición de *swap* que está en hdb3, el sistema *proc* (que utiliza el *kernel* para guardar su información). Y el disquete, el CD-ROM, y en este caso un disco USB de tipo Flash (que se ve como un dispositivo SCSI). En algunos casos, se especifica *auto* como tipo de *filesystem*. Esto permite que lo autodetecte. Si se conoce, es mejor darlo en la configuración, y por otra parte, el *noauto* en las opciones permite que no sea montado de forma automática siempre, sino bajo petición.

Si tenemos esta información en el fichero, el proceso de montaje se simplifica mucho, ya que se hará o bien en ejecución o bien bajo demanda (para los *noauto*). Y puede hacerse ahora simplemente pidiendo que se monte el dispositivo o el punto de montaje:

```
mount /mnt/cdrom
mount /dev/fd0
```

dado que el sistema ya tiene el resto de la información.

El proceso contrario, el desmontaje, es bastante sencillo, el comando *umount* con punto o dispositivo:

```
umount /mnt/cdrom
umount /dev/fd0
```

ANOTACIONES

En el caso de medios extraíbles, tipo CD-ROM (u otros), puede usarse *eject* para la extracción del medio:

`eject /dev/cdrom` o, en este caso, sólo: `eject`

Los comandos *mount* y *umount* montan o desmontan todos los sistemas disponibles. En el fichero /etc/mtab se mantiene una lista de los sistemas montados en un momento concreto se puede consultar o ejecutar *mount* sin parámetros para obtener esta información.

5.4.2. Permisos

Otro tema que habrá que controlar en el caso de los ficheros y directorios es el de los permisos que queremos establecer en cada uno de ellos, recordando que cada fichero puede disponer de la serie de permisos: *rwxrwxrwx* donde se corresponden con *rwx* del propietario, *rwx* del grupo al que el usuario pertenece, y *rwx* para otros usuarios. En cada uno se puede establecer el permiso de lectura (*r*), escritura (*w*) o ejecución (*x*). En el caso de un directorio, *x* denota el permiso para poder entrar en ese directorio (con el comando *cd*, por ejemplo).

Para modificar los derechos sobre un directorio o fichero, existen los comandos:

- *chown*: cambiar propietario de los ficheros.
- *chgrp*: cambiar grupo de los ficheros.
- *chmod*: cambiar permisos específicos (*rwx*) de los archivos.

Estos comandos también permiten la opción *–R*, que es recursiva si se trata de un directorio.

5.5. Usuarios y grupos

Los usuarios de un sistema GNU/Linux disponen normalmente de una cuenta asociada con sus datos, junto con el espacio en disco

ANOTACIONES

para que puedan desarrollar sus archivos y directorios. Este espacio está asignado al usuario, y sólo puede ser usado por éste (a no ser que los permisos especifiquen cosas diferentes.

Dentro de las cuentas asociadas a usuarios podemos encontrar diferentes tipos:

- La del administrador, con identificador *root*, que sólo es (o debería ser) utilizada para las operaciones de administración. El usuario *root* es el que dispone de más permisos y acceso completo a la máquina y a los archivos de configuración. Por lo tanto, también es el que más daño puede causar por errores u omisiones. Es mejor evitar usar la cuenta de *root* como si fuese un usuario más, por lo que se recomienda dejarla sólo para operaciones de administración.

- Cuentas de usuarios: las cuentas normales para cualquier usuario de la máquina tienen los permisos restringidos al uso de ficheros de su cuenta, y a algunas otras zonas particulares (por ejemplo, los temporales en /tmp), así como a utilizar algunos dispositivos para los que se le haya habilitado.

- Cuentas especiales de los servicios: *lp*, *news*, *wheel*, *www-data*, ... cuentas que no son usadas por personas, sino por servicios internos del sistema, que los usa bajo estos nombres de usuario. Algunos de los servicios también son usados bajo el nombre de *root*.

Un usuario normalmente se crea mediante la especificación de un nombre (o identificador de usuario), una palabra de paso (*password*) y un directorio personal asociado (la cuenta).

La información de los usuarios del sistema está incluida en los siguientes archivos:

```
/etc/passwd
/etc/shadow
/etc/group
/etc/gshadow
```

ANOTACIONES

Ejemplo de unas líneas del /etc/passwd:

```
juan:x:1000:1000:Juan Garcia,,,:/home/juan:/bin/bash

root:x:0:0:root:/root:/bin/bash
```

donde se indica (si aparecen :: seguidos, el campo está vacío):

- juan: identificador de usuario en el sistema.
- x: palabra de paso del usuario codificada, si hay una "x" es que se encuentra en el fichero /etc/shadow.
- 1000: código del usuario, lo usa el sistema como código de identidad del usuario.
- 1000: código del grupo principal al que pertenece, la información del grupo en /etc/group.
- Juan García: comentario, suele ponerse el nombre completo del usuario.
- /home/juan: directorio personal asociado a su cuenta.
- /bin/bash: *shell* interactivo que utilizará el usuario al interactuar con el sistema, en modo texto, o por *shell* gráfico. En este caso, el *shell* Bash de GNU, que es el utilizado por defecto. El fichero /etc/passwd solía contener las palabras de paso de los usuarios de forma encriptada, pero el problema estaba en que cualquier usuario podía ver el fichero, y en su momento se diseñaron *cracks* que intentaban encontrar en forma bruta la palabra de paso mediante la palabra de paso encriptada como punto de partida (codificado con el sistema *crypt*).

Para evitar esto, hoy en día ya no se colocan las palabras de paso en este archivo, sólo una "x" que indica que se encuentran en otro fichero, que es sólo de lectura para el usuario *root*, /etc/shadow, el contenido del cual podría ser algo parecido a lo siguiente:

```
juan:a1gNcs82ICst8CjVJS7ZFCVnu0N2pBcn/:12208:0:99999:7:::
```

donde se encuentra el identificador del usuario junto con la palabra de paso encriptada. Además, aparecen (como campos separados por ":":

- Días desde 1 de enero de 1970 en que la palabra de paso se cambió por ultima vez.
- Días que faltan para que se cambie (0 no hay que cambiarla).
- Días después en que hay que cambiarla (o sea, plazo de cambio).
- Días en que el usuario será avisado antes de que le expire.
- Días una vez expirado cuando se le deshabilitará la cuenta.
- Días desde 1 enero 1970 en que la cuenta está deshabilitada.
- Y un campo reservado.

Además, las claves de encriptación pueden ser más difíciles, ya que ahora puede utilizarse un sistema denominado md5 (suele aparecer como opción a la hora de instalar el sistema) para proteger las palabras de paso de los usuarios. Veremos más detalles al respecto en la unidad dedicada a la seguridad.

En /etc/group está la información de los grupos de usuarios:

```
jose:x:1000:
```

donde tenemos:

```
nombre-grupo:contraseña grupo:identificador-del grupo:lista-usuarios
```

La lista de usuarios del grupo puede estar presente o no, ya que la información ya está en /etc/passwd, no suele ponerse en /etc/group. Si se pone, suele aparecer como una lista de usuarios separada por comas. Los grupos también pueden poseer una contraseña asociada (aunque no suele ser tan común), como en el caso de los de usuario, también existe un fichero de tipo shadow: /etc/gshadow.

ANOTACIONES

Otros ficheros interesantes son los del directorio /etc/skel, donde se hallan los ficheros que se incluyen en cada cuenta de usuario al crearla. Recordar que, como vimos con los *shell* interactivos, podíamos tener unos *scripts* de configuración que se ejecutaban al entrar o salir de la cuenta. En el directorio *skel* se guardan los "esqueletos" que se copian en cada usuario al crearlo. Suele ser responsabilidad del administrador crear unos ficheros adecuados para los usuarios, poniendo los *path* necesarios de ejecución, inicialización de variables de sistema, variables que se necesiten para el software, etc.

A continuación vamos a ver una serie de comandos útiles para esta administración de usuarios (mencionamos su funcionalidad y en el taller haremos algunas pruebas):

- *useradd*: añadir un usuario al sistema.
- *userdel*: borrar un usuario del sistema.
- *usermod*: modificar un usuario del sistema.
- *groupadd, groupdel, groupmod* lo mismo para grupos.
- *newusers, chpasswd*: pueden ser de utilidad en grandes instalaciones con muchos usuarios, ya que permiten crear varias cuentas desde la información introducida en un fichero (*newusers*) o bien cambiar las contraseñas a un gran número de usuarios (*chpasswd*).
- *chsh*: cambiar el *shell* de *login* del usuario.
- *chfn*: cambiar la información del usuario, la presente en el comentario del fichero /etc/passwd.
- *passwd*: cambia la contraseña de un usuario. Puede ejecutarse como usuario, y entonces pide la contraseña nueva y la vieja. En el caso de hacerlo, el *root* tiene que especificar el usuario al que va a cambiar la contraseña (si no, estaría cambiando la suya) y no necesita la contraseña antigua. Es quizás el comando más usado por el *root*, de cara a los usuarios cuando se les olvida la contraseña antigua.

ANOTACIONES

- *su*: una especie de cambio de identidad. Lo utilizan tanto usuarios, como el *root* para cambiar el usuario actual. En el caso del administrador, es bastante utilizado para testar que la cuenta del usuario funcione correctamente; hay diferentes variantes: su (sin parámetros, sirve para pasar a usuario *root*, siempre que se tenga el *passwd* de *root*, nos permite, cuando estamos en una cuenta de usuario, pasar a *root* para hacer alguna tarea). La sentencia *su iduser* (cambia el usuario a *iduser*, pero dejando el entorno como está, o sea, en el mismo directorio, ...). El mandato *su iduser* (hace una sustitución total, como si el otro usuario hubiese entrado en el sistema haciendo un *login*).

Respecto a la administración de usuarios y grupos, lo que hemos comentado aquí hace referencia a la administración local de una sola máquina. En sistemas con múltiples máquinas que comparten los usuarios suele utilizarse otro sistema de gestión de la información de los usuarios. Estos sistemas, denominados genéricamente **sistemas de información de red**, como NIS, NIS+ o LDAP, utilizan bases de datos para almacenar la información de los usuarios y grupos, de manera que se utilizan máquinas servidoras, donde se almacena la base de datos, y otras máquinas clientes, donde se consulta esta información. Esto permite tener una sola copia de los datos de los usuarios, y que éstos puedan entrar en cualquier máquina disponible.

Podemos comprobar si estamos en un entorno de tipo NIS si en las líneas *passwd* y *group* del archivo de configuración /etc/nsswitch.conf aparece *compat*, si estamos trabajando con los ficheros locales, o bien *nis* o *nisplus* según el sistema con que estemos trabajando. En general, para el usuario simple no supone ninguna modificación, ya que la gestión de las máquinas le es transparente, y más si se combina con ficheros compartidos por NFS que permite disponer de su cuenta sin importar con qué máquina. La mayor parte de los comandos anteriores pueden seguir usándose sin problema bajo NIS o NIS+, son equivalentes a excepción del cambio de contraseña, que en lugar de *passwd*, se suele hacer con *yppasswd* (NIS) o *nispasswd* (NIS+); pero el administrador los puede renombrar (por un enlace) a *passwd*, y los usuarios no notarán la diferencia.

Veremos este modo de configuración en las unidades de administración de red.

ANOTACIONES

5.6. Servidores de impresión

El sistema de impresión de GNU/Linux [GT03] [Smi02] está heredado de la variante BSD de UNIX; este sistema se denominaba LPD (*Line Printer Daemon*). Éste un sistema de impresión muy potente, ya que integra capacidades para gestionar tanto impresoras locales como de red. Y ofrece dentro del mismo, tanto el cliente como el servidor de impresión.

LPD es un sistema bastante antiguo, ya que se remonta a las orígenes de la rama BSD de UNIX (mediados de los ochenta). Por lo tanto, a LPD le suele faltar soporte para los dispositivos modernos, ya que en origen el sistema no estuvo pensado para el tipo de impresoras actuales. El sistema LPD no estuvo pensado como un sistema basado en controladores de dispositivo, ya que normalmente se producían sólo impresoras serie o paralelas de escritura de caracteres texto.

Nota

Los sistemas UNIX disponen, quizás, de los sistemas de impresión más potentes y complejos, que aportan una gran flexibilidad a los entornos de impresión.

Para la situación actual, el sistema LPD, se combina con otro software común, como el sistema Ghostscript, que ofrece salida de tipo *postscript* para un rango muy amplio de impresoras para las que posee controladores. Además, se suele combinar con algún software de filtrado, que según el tipo de documento a imprimir, selecciona filtros adecuados. Así, normalmente el proceso que se sigue es (básicamente):

1) El trabajo es iniciado por un comando del sistema LPD.

2) El sistema de filtro identifica qué tipo de trabajo (o fichero) es utilizado y convierte el trabajo a un fichero *postscript* de salida, que es el que se envía a la impresora. En GNU/Linux y UNIX, la mayoría de aplicaciones suponen que la salida será hacia una impresora *postscript*, y muchas de ellas generan salida *postscript* directamente, y por esta razón se necesita el siguiente paso.

3) Ghostscript se encarga de interpretar el fichero *postscript* recibido, y según el controlador de la impresora a la que ha sido enviado el trabajo, realiza la conversión al formato propio de la impresora; si es de tipo *postscript*, la impresión es directa, si no, habrá que realizar la traducción. El trabajo se manda a la cola de impresión.

Nota

Ghostscript: http://www.cs.wisc.edu/ghost/

ANOTACIONES

Además del sistema de impresión LPD (con origen en los BSD UNIX), también existe el denominado sistema System V (de origen en la otra rama UNIX de System V). Normalmente, por compatibilidad, actualmente la mayor parte de UNIX integran ambos, de manera que o bien uno u otro es el principal, y el otro se simula sobre el principal. En el caso de GNU/Linux, pasa algo parecido, según la instalación que hagamos podemos tener sólo los comandos LPD de sistema de impresión, pero también será habitual disponer de los comandos System V. Una forma sencilla de identificar los dos sistemas (BSD o System V), es con el comando principal de impresión (el que envía los trabajos al sistema), en BSD es **lpr**, y en System V es **lp**.

Éste era el panorama inicial de los sistemas de impresión de GNU/Linux, pero en los últimos años han surgido más sistemas, que permiten una mayor flexibilidad y una mayor disposición de controladores para las impresoras. Los dos principales sistemas son **LPRng** y **CUPS**.

Nota

LPRng: http://www.lprng.org

CUPS: http://www.cups.org

Los dos son una especie de sistema de más alto nivel, pero que no se diferencia en mucho de cara al usuario respecto a los BSD y System V estándar, por ejemplo, se utilizan los mismos comandos clientes para imprimir. Para el administrador sí que supondrá diferencias, ya que los sistemas de configuración son diferentes. En cierto modo podemos considerar a LPRng y CUPS como nuevas arquitecturas de sistemas de impresión, que son compatibles de cara al usuario con los comandos antiguos.

En las distribuciones GNU/Linux actuales podemos encontrarnos con los diferentes sistemas de impresión. Si la distribución es antigua, puede que lleve incorporado tan sólo el sistema BSD LPD; en las actuales: Red Hat utiliza LPRng (por defecto), pero puede cambiarse a CUPS (hay una herramienta, Print switch, que permite cambiar el sistema).

En el caso de Fedora Core, el sistema de impresión por defecto es CUPS (desapareciendo LPRng), y la herramienta Print switch ya no existe por no ser necesaria. Debian por defecto utiliza BSD LPD, pero es común instalar CUPS, y también puede utilizar LPRng. Además, cabe recordar que también teníamos la posibilidad (vista en la unidad de migración) de interaccionar con sistemas Windows mediante

ANOTACIONES

protocolos Samba, que permitían compartir las impresoras y el acceso a éstas.

Respecto a cada uno de los sistemas [GT03]:

- **BSD LPD**: es uno de los estándares de UNIX, y algunas aplicaciones asumen que tendrán los comandos y el sistema de impresión disponibles, por lo cual, tanto LPRng como CUPS emulan el funcionamiento y los comandos de BDS LPD. El sistema LPD es utilizable, pero no muy configurable, sobre todo en el control de acceso, por eso las distribuciones se han movido a los otros dos sistemas más modernos.

- **LPRng**: se diseñó básicamente para ser un reemplazo del BSD, por lo tanto, la mayor parte de la configuración es parecida y únicamente difiere en algunos ficheros de configuración.

- **CUPS**: se trata de una desviación mayor del BSD original, y la configuración es propia. Se proporciona información a las aplicaciones sobre las impresoras disponibles (también en LPRng). En CUPS tanto el cliente como el servidor tienen que disponer de software CUPS.

Los dos sistemas tienen emulación de los comandos System V.

Para la impresión en GNU/Linux, hay que tener en cuenta varios aspectos:

- Sistema de impresión que se utiliza: BSD, LPRng o CUPS.

- Dispositivo de impresión (impresora): puede disponer de conexión local a una maquina o estar colocada en red. Las impresoras actuales pueden estar colocadas por conexiones locales a una máquina mediante interfaces serie, paralelo, USB, etc. O puestas simplemente en red, como una máquina más. Las conectadas a red, pueden normalmente actuar ellas mismas de servidor de impresión (por ejemplo, muchas láser HP son servidores BSD LPD), o bien pueden colgarse de una máquina que actúe de servidor de impresión para ellas.

ANOTACIONES

- Protocolos de comunicación utilizados con la impresora, o el sistema de impresión: ya sea TCP/IP directo (por ejemplo, una HP con LPD), o bien otros de más alto nivel sobre TCP/IP, como IPP (CUPS), JetDirect (algunas impresoras HP), etc. Este parámetro es importante, ya que lo debemos conocer para instalar la impresora en un sistema.

- Sistema de filtros usado, cada sistema de impresión soporta uno o varios.

- Y los controladores de las impresoras, en GNU/Linux hay muchos, podemos mencionar por ejemplo, controladores de: CUPS, propios o de los fabricantes (por ejemplo HP y Epson los proporcionan); Gimp, el programa de retoque de imágenes también posee *drivers* optimizados para la impresión de imágenes; Foomatic es un sistema de gestión de controladores que funciona con la mayoría de sistemas (CUPS, LPD, LPRng, y otros); los controladores de Ghostscript, etc. En casi todas las impresoras tienen uno o más controladores de estos conjuntos.

Nota

Se puede encontrar información de las impresoras más adecuadas y de los controladores en:

http://www.linuxprinting.org/foomatic.html

Respecto a la parte cliente del sistema, los comandos básicos son iguales para los diferentes sistemas, y éstos son los comandos del sistema BSD (cada sistema soporta emulación de estos comandos:

- *lpr*: envía un trabajo a la cola de la impresora por defecto (o a la que se selecciona), el *daemon* de impresión (lpd) se encarga de enviarlo a la cola correspondiente, y asigna un número de trabajo, que será usado con los otros comandos. Normalmente, la impresora por defecto estaría indicada por una variable de sistema PRINTER, o se utilizará la primera que exista definida, o en algunos sistemas se utiliza la cola *lp*.

Ejemplo

Ejemplo de *lpr*:

```
lpr -Pepson datos.txt
```

Esta instrucción mandaría el fichero datos.txt a la cola de impresión asociada a una impresora que hemos definido como "epson".

ANOTACIONES

- *lpq*: nos permite examinar los trabajos existentes en la cola.

Ejemplo

```
# lpq -P epson

Rank   Owner  Job  Files     Total Size
1st    juan   15   datos.txt 74578 bytes
2nd    marta  16   fpppp.F   12394 bytes
```

Este comando nos muestra los trabajos en cola, con el orden y tamaños de éstos; los ficheros pueden aparecer con nombres diferentes, ya que depende de si los hemos enviado con *lpr*, o con otra aplicación que puede cambiarlos de nombre al enviarlos, o si han tenido que pasar por algún filtro al convertirlos.

- *lprm*: elimina trabajos de la cola, podemos especificar un número de trabajo, o un usuario para cancelar los trabajos.

Ejemplo

```
lprm -Pepson 15
```

Eliminar el trabajo con *id* 15 de la cola.

Respecto a la parte administrativa (en BSD), el comando principal sería *lpc*; este comando permite activar, desactivar colas, mover trabajos en el orden de las colas y activar o desactivar las impresoras (se pueden recibir trabajos en las colas pero no se envían a las impresoras).

Cabe mencionar asimismo que, para el caso de System V, los comandos de impresión suelen también estar disponibles, normalmente simulados sobre los de BSD. En el caso cliente, los comandos son: *lp, lpstat, cancel* y para temas de administración: *lpadmin, accept, reject, lpmove, enable, disable, lpshut.*

En las siguientes secciones veremos cómo hay que configurar un servidor de impresión para los tres sistemas principales. Estos servidores

ANOTACIONES

sirven tanto para la impresión local, como para atender las impresiones de clientes de red (si están habilitados).

5.6.1. BSD LPD

En el caso del servidor BSD LPD, hay dos ficheros principales para examinar, por una parte, la definición de las impresoras en /etc/printcap y por otra, los permisos de acceso por red en /etc/hosts.lpd.

Respecto al tema de los permisos, por defecto BSD LPD sólo deja acceso local a la impresora, y por lo tanto, hay que habilitarlo expresamente en /etc/hosts.lpd.

Ejemplo

El fichero podría ser:

```
#fichero hosts.lpd
second
first.the.com
192.168.1.7
+@groupnis
-three.the.com
```

que indicaría que está permitida la impresión a una serie de máquinas, listadas bien por su nombre DNS o por la dirección IP. Se pueden añadir grupos de máquinas que pertenezcan a un servidor NIS (como en el ejemplo *groupnis*) o bien no permitir acceso a determinadas máquinas indicándolo con un guión "-".

Respecto a la configuración del servidor en /etc/printcap, se definen entradas, donde cada una representa una cola del sistema de impresión a la que pueden ir a parar los trabajos. La cola puede estar tanto asociada a un dispositivo local, como a un servidor remoto, ya sea éste una impresora u otro servidor.

En cada entrada pueden existir las opciones:

- lp = , nos indica a qué dispositivo está conectada la impresora, por ejemplo *lp* = /dev/lp0 indicaría el primer puerto paralelo. Si

ANOTACIONES

la impresora es de tipo LPD, por ejemplo una impresora de red que acepta el protocolo LPD (como una HP), entonces podemos dejar el campo vacío y rellenar los siguientes.

- **rm** = , dirección con nombre o IP de la máquina remota que dispone de la cola de impresión. Si se trata de una impresora de red, será la dirección de ésta.

- **rp** = , nombre de la cola remota, en la máquina indica antes con *rm*.

Veamos un ejemplo:

```
# Entrada de una impresora local
lp|epson|Epson C62:\
  :lp = /dev/lp1:sd = /var/spool/lpd/epson:\
  :sh:pw#80:pl#72:px#1440:mx#0:\
  :if = /etc/magicfilter/StylusColor@720dpi-filter:\filtro
  :af = /var/log/lp-acct:lf = /var/log/lp-errs:
# Entrada de impresora remota
hpremota|hpr|hp remota del departamento|:\
  :lp = :\
  :rm = servidor:rp = colahp:\
  :lf = /var/adm/lpd_rem_errs:\fichero de log.
  :sd = /var/spool/lpd/hpremota:spool local asociado
```

5.6.2. LPRng

En el caso del sistema LPRng, ya que éste se hizo para mantener la compatibilidad con BSD, y entre otros mejorar aspectos de acceso, el sistema es compatible a nivel de configuración de colas, y se lleva a cabo a través del mismo formato de fichero de /etc/printcap, con algunos añadidos propios.

Donde la configuración resulta diferente es en el tema del acceso, en este caso se realiza a través de un fichero /etc/lpd.perms en general para todo el sistema, y pueden existir también configuraciones individuales de cada cola, con el fichero lpd.perms, colocado en el directorio correspondiente a la cola, normalmente /var/spool/lpd/nombre-cola.

ANOTACIONES

Estos ficheros *lpd.perms* tienen una capacidad superior de configurar el acceso, permitiendo los siguientes comandos básicos:

```
DEFAULT ACCEPT
DEFAULT REJECT
ACCEPT [ key = value[,value]* ]*
REJECT [ key = value[,value]* ]*
```

donde los dos primeros nos permiten establecer el valor por defecto, de aceptar todo, o rechazar todo, y los dos siguientes, aceptar o rechazar una configuración concreta especificada en la línea. Se pueden aceptar (o rechazar) peticiones de un *host*, usuario, o puertos IP específicos. Asimismo, se puede configurar qué tipo de servicio se proporcionará al elemento: X (puede conectarse), P (impresión de trabajos), Q (examinar cola con *lpq*), M (borrar trabajos de la cola, *lprm*), C (control de impresoras, comando *lpc*), entre otros, así en el fichero:

```
ACCEPT SERVICE = M HOST = first USER = jose
ACCEPT SERVICE = M SERVER REMOTEUSER = root
REJECT SERVICE = M
```

Se permite borrar trabajos de la cola, al usuario (*jose*) de la máquina (*first*), y al usuario *root*, del servidor donde esté alojado el servicio de impresión (*localhost*), además, se rechazan cualesquiera otras peticiones de borrar de la cola trabajos, que no sean las peticiones ya establecidas.

Con esta configuración hay que tener especial cuidado, porque en algunas distribuciones, los servicios LPRng están por defecto abiertos. Puede limitarse la conexión por ejemplo con:

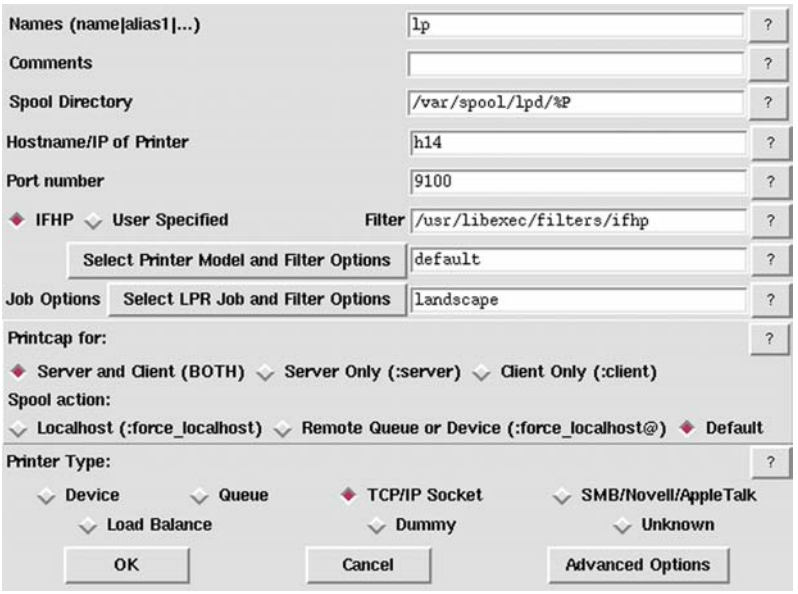
```
ACCEPT SERVICE = X SERVER
REJECT SERVICE = X NOT REMOTEIP = 100.200.0.0/255
```

Servicio de conexión sólo accesible a la máquina local del servidor, y rechazado si no se pertenece a nuestra subred (en este caso, suponemos que sea 100.200.0.x).

ANOTACIONES

Para la administración de línea de comandos, se usan las mismas herramientas que el BSD, estándar. En el terreno de la administración gráfica del sistema, cabe destacar la herramienta *lprngtool*.

Figura 12. lprngtool, configuración de una impresora



Hay varios paquetes de software relacionados con LPRng, por ejemplo en una Debian Sid encontramos:

```
lprng - lpr/lpd printer spooling system
lprngdoc - lpr/lpd printer spooling system (documentation)
lprngtool - GUI frontend to LPRng based /etc/printcap
ifhp - Printer filter for HP LaserJet printers
lprfax - Utility to allow printing to a fax modem
pdq - Simple printing system for workstations
printbill - Sophisticated print billing / accounting system for lprng
printop - Graphical interface to the LPRng print system.
```

ANOTACIONES

5.6.3. CUPS

CUPS es una nueva arquitectura para el sistema de impresión bastante diferente, tiene una capa de compatibilidad hacia BSD LPD, que le permite interaccionar con servidores de este tipo. Soporta también un nuevo protocolo de impresión llamado IPP (basado en http), pero sólo disponible cuando cliente y servidor son de tipo CUPS. Además, utiliza

un tipo de *drivers* denominados PPD que identifican las capacidades de la impresora, CUPS ya trae algunos de estos controladores, y algunos fabricantes también los ofrecen (caso HP y Epson).

CUPS tiene un sistema de administración completamente diferente, basado en diferentes ficheros: /etc/cups/cupsd.conf centraliza la configuración del sistema, y /etc/cups/printers.conf controla la definición de impresoras, y /etc/cups/classes.conf los grupos de éstas.

En /etc/cups/cupsd.conf, configuramos el sistema según una serie de secciones del archivo y las directivas de las diferentes acciones. El archivo es bastante grande, destacaremos algunas directivas importantes:

- Allow: nos permite especificar qué máquinas podrán acceder al servidor, ya sean grupos o máquinas individuales, o segmentos IP de red.

- AuthClass: permite indicar si se pedirá que se autentifiquen los usuarios clientes o no.

- BrowseXXX: hay una serie de directivas relacionadas con la posibilidad de examinar la red para encontrar impresoras servidas, esta posibilidad está activada por defecto (*browsing* en *on*), por lo tanto, normalmente encontraremos disponibles todas las impresoras disponibles en la red. Podemos desactivarla, para solamente observar las impresoras que hayamos definido. Otra opción importante es BrowseAllow, que dice a quién le damos la posibilidad de preguntar por nuestras impresoras; por defecto está habilitada, por lo que cualquiera puede ver nuestra impresora desde nuestra red.

Destacar que CUPS en principio está pensado para que tanto clientes, como el servidor funcionen bajo el mismo sistema, si los clientes utilizan LPD o LPRng, hay que instalar un *daemon* de compatibilidad llamado *cups-lpd*. En este caso, CUPS acepta trabajos que provengan de un sistema LPD o LPRng, pero no controla los accesos (*cupsd.conf* sólo sirve para el propio sistema CUPS), por lo tanto, habrá que implementar alguna estrategia de control de acceso, tipo *firewall* por ejemplo (ver unidad seguridad).

Para la administración desde línea de comandos, CUPS es un tanto peculiar, ya que acepta tanto comandos LPD como System V en los clientes, y la administración suele hacerse con el comando lpadmin

ANOTACIONES

de System V. En cuanto a herramientas gráficas, disponemos de KUPS, ESP Print Pro (propietario) o la interfaz por web que trae el mismo sistema CUPS, accesible en http://localhost:631.

Figura 13. Interfaz para la administración del sistema CUPS

Respecto a los paquetes software relacionados con CUPS, en una Debian Sid encontramos (entre otros):

```
cupsys - Common UNIX Printing System(tm) - server
cupsys-bsd - Common UNIX Printing System(tm) - BSD commands
cupsys-client - Common UNIX Printing System(tm) - client programs (SysV)
cupsys-driver-gimpprint - Gimp-Print printer drivers for CUPS
cupsys-pt - Tool for viewing/managing print jobs under CUPS
cupsomatic-ppd - linuxprinting.org printer support - transition package
foomatic-db - linuxprinting.org printer support - database
foomatic-db-engine - linuxprinting.org printer support - programs
foomatic-db-gimp-print - linuxprinting - db Gimp-Print printer drivers
foomatic-db-hpijs - linuxprinting - db HPIJS printers
foomatic-filters - linuxprinting.org printer support - filters
foomatic-filters-ppds - linuxprinting - prebuilt PPD files
foomatic-gui - GNOME interface for Foomatic printer filter system
gimpprint-doc - Users' Guide for GIMP-Print and CUPS
gimpprint-locales - Locale data files for gimp-print
gnome-cups-manager - CUPS printer admin tool for GNOME
gtklp - Frontend for cups written in gtk
```

5.7. Discos y gestión *filesystems*

Respecto a las unidades de almacenamiento, como hemos ido viendo poseen una serie de dispositivos asociados, dependiendo del tipo de interfaz:

- IDE: dispositivos /dev/hda disco *master*, primer conector IDE; /dev/hdb disco *slave* del primer conector, /dev/hdc *master* segundo conector, /dev/hdd *slave* segundo conector.
- SCSI: dispositivos /dev/sda, /dev/sdb, ... siguiendo la numeración que tengan los periféricos en el Bus SCSI.
- Disquetes: dispositivos /dev/fdx, con x número de disquetera (comenzando en 0). Hay diferentes dispositivos dependiendo de la capacidad del disquete, por ejemplo, el disquete de 1.44 MB en la disquetera A sería /dev/fd0H1440.

Respecto a las particiones presentes, el número que sigue al dispositivo representa el índice de la partición dentro del disco, y es tratado como un dispositivo independiente: /dev/hda1 primera partición del primer disco IDE, o /dev/sdc2, segunda partición del tercer dispositivo SCSI. En el caso de los discos IDE, éstos permiten cuatro particiones denominadas primarias, y más lógicas, así, si /dev/hdan, *n* será inferior o igual a 4, se tratará de una partición primaria, si no, se tratará de una partición lógica con *n* superior o igual a 5.

Con los discos y los *filesystems* asociados, los procesos básicos que podemos realizar los englobamos en:

- Creación de particiones, o modificación de éstas. Mediante comandos como *fdisk* o parecidos (*cfdisk, sfdisk*).
- Formateo de disquetes: en caso de disquetes, pueden utilizarse diferentes herramientas: *fdformat* (formateo de bajo nivel), *superformat* (formateo a diferentes capacidades en formato msdos), *mformat* (formateo específico creando *filesystem* msdos estándar).
- Creación de *filesystems* linux, en particiones, mediante el comando *mkfs*. Hay versiones específicas para crear *filesystems* diversos *mkfs.ext2, mkfs.ext3*, y también *filesystems* no linux: *mkfs.vfat*, *mkfs.msdos*, *mkfs.minix*. U otros para CD-ROM como *mkisofs* para crear los iso9660 (con extensiones *joliet* o *rock ridge*), sobre CD-ROM, y junto con œmandos como *cdrecord* permite

ANOTACIONES

crear/grabar CD-ROM. Otro caso particular es la orden *mkswap*, que permite crear áreas de *swap* en particiones, que después se pueden activar o desactivar con *swapon* y *swapoff*.

- Montaje de los *filesystems*: comandos *mount*, *umount*.
- Verificación de estado: la principal herramienta de verificación de *filesystems* Linux es el comando *fsck*. Este comando comprueba las diferentes áreas del sistema de ficheros para verificar la consistencia y comprobar posibles errores y, en los casos que sea posible, corregirlos. El propio sistema activa automáticamente el comando en el arranque cuando detecta situaciones donde se ha producido una parada incorrecta (un apagón eléctrico o accidental de la máquina), o bien pasado un cierto número de veces en que el sistema se ha arrancado; esta comprobación suele comportar cierto tiempo, normalmente algunos minutos. También existen versiones particulares para otros sistemas de ficheros: *fsck.ext2*, *fsck.ext3*, *fsck.vfat*, *fsck.msdos*, etc. El proceso del *fsck* normalmente se realiza con el dispositivo en modo de "sólo lectura" con particiones montadas; se recomienda desmontar las particiones para realizar el proceso si se detectan errores y hay que aplicar correcciones. En determinados casos, por ejemplo si el sistema por testar es la raíz / y se detecta algún error crítico, se nos pedirá que cambiemos de modo de ejecución del sistema (*runlevel*) hacia modo sólo *root*, y hagamos allí la verificación. En general, si hay que hacerla verificación, se recomienda hacer éstas en modo superusuario (podemos conmutar de modo de *runlevel* con los comandos *init* o *telinit*).
- Procesos de *backup*: ya sean del disco, bloques de disco, particiones, *filesystems*, ficheros,... Hay varias herramientas útiles para ello: *tar* nos permite copiar ficheros hacia un fichero o a unidades de cinta; *cpio*, de forma parecida, puede realizar *backups* de ficheros hacia un fichero; tanto *cpio* como *tar* mantienen información de permisos y propietarios de los ficheros; *dd* permite copias, ya sea de ficheros, dispositivos, particiones o discos a fichero; es un poco complejo y hay que conocer información de bajo nivel, tipo, tamaño, bloque o sector, y puede enviarse también a cintas.
- Utilidades diversas: algunos comandos individuales, algunos de ellos utilizados por los procesos anteriores para hacer tratamientos diversos: *badblocks* para encontrar bloques defectuosos en el dispositivo; *dumpe2fs* para obtener información sobre *filesystems*

ANOTACIONES

Linux; *tune2fs* permite hacer *tunning* del *filesystem* Linux de tipo ext2 o ext3 y ajustar diferentes parámetros de comportamiento.

5.8. Software: actualización

Para la administración de instalación o actualización de software en nuestro sistema, vamos a depender en primera instancia del tipo de paquetes software que utilice nuestro sistema:

- RPM: paquetes que utiliza la distribución Red Hat. Se suelen manejar a través del comando *rpm*. Contienen información de dependencias del software con otros.
- DEB: paquetes de Debian, se suelen manejar con un conjunto de herramientas que trabajan a diferentes niveles con paquetes individuales o grupos. Entre éstas, cabe mencionar: *dselect*, *tasksel*, *dpkg*, y *apt-get*.
- Tar, o bien los tgz (también tar.gz): son puramente paquetes de ficheros unidos y comprimidos mediante comandos estándar como *tar* y *gzip* (se usan los mismos para la descompresión). Estos paquetes no contienen información de dependencias y normalmente pueden instalarse en diferentes lugares, si no es que llevan información de ruta (*path*) absoluta.

Para manejar estos paquetes existen varias herramientas gráficas, como RPM: Kpackage; DEB: Synaptic, *Gnome-apt*; Tgz: Kpackage o el propio gestor de ficheros gráficos. También suelen existir utilidades de conversiones de paquetes. Por ejemplo, en Debian existe alien, que permite convertir paquetes RPM a DEB.

Dependiendo del uso de los tipos de paquetes o herramientas: la actualización o instalación de software de nuestro sistema se podrá producir de diferentes maneras:

1) Desde los propios CD de instalación del sistema, normalmente, todas las distribuciones buscan el software en sus CD. Pero hay que tener en cuenta que este software no sea antiguo e incluya, por esta razón, algunos parches como actualizaciones, o nuevas versiones con más prestaciones; con lo cual, si se instala a partir de CD, es bastante común verificar después que no exista alguna versión más nueva.
2) Mediante servicios de actualización o búsqueda de software, ya sea de forma gratuita, como el caso de la herramienta *apt-get* de

ANOTACIONES

Debian, o servicios de suscripción (de pago o con facilidades básicas) como el Red Hat Network de Red Hat.

3) Por repositorios de software que ofrecen paquetes de software preconstruidos para una distribución determinada.

4) Por el propio creador o distribuidor del software, que ofrece una serie de paquetes de instalación de su software. Podemos no encontrar el tipo de paquetes necesario para nuestra distribución.

5) Software sin empaquetamiento o con un empaquetamiento de sólo compresión sin ningún tipo de dependencias.

6) Sólo código fuente, en forma de paquete o bien fichero comprimido.

5.9. Trabajos no interactivos

En las tareas de administración, suele ser necesaria la ejecución a intervalos de ciertas tareas, ya sea por programar las tareas para realizarlas en horarios menos usados, o bien por la propia naturaleza periódica de las tareas.

Para realizar este tipo de trabajos "fuera de horas", como servicios periódicos o programados, hay diversos sistemas que nos permiten construir un tipo de agenda de tareas:

- **nohup** es quizás el caso más simple utilizado por los usuarios, les permite la ejecución de una tarea no interactiva una vez hayan salido de su cuenta. Normalmente, al salir de la cuenta, el usuario pierde sus procesos; *nohup* permite dejarlos corriendo, a pesar de que el usuario se desconecte.

- **at** nos permite lanzar una acción para más tarde, programando un determinado instante en el que va a iniciarse, especificándose la hora (hh:mm) y fecha, o bien si se hará hoy (*today*) o mañana (*tomorrow*). Ejemplos:

`at 10pm tarea`	realizar la tarea a las diez de la noche.
`at 2am tomorrow tarea`	realizar la tarea a las dos de la madrugada.

ANOTACIONES

- **cron** permite establecer una lista de trabajos por realizar con su programación; esta configuración se guarda en /etc/crontab; concretamente, en cada entrada de este fichero tenemos: minutos y hora en que se va a efectuar la tarea, qué día del mes, qué mes, qué día de la semana, junto con qué (ya sea una tarea, o bien un directorio donde estarán las tareas a ejecutar). Por ejemplo, de forma estándar el contenido es:

```
25 6 * * * root test -e /usr/sbin/anacron || run-parts --report /etc/cron.daily
47 6 * * 7 root test -e /usr/sbin/anacron || run-parts --report /etc/cron.weekly
52 6 1 * * root test -e /usr/sbin/anacron || run-parts --report /etc/cron.monthl
```

donde se está programando que una serie de tareas van a hacerse: cada día ("" indica 'cualquiera'), semanalmente (el 7.º día de la semana), o mensualmente (el 1.º de cada mes). Normalmente, los trabajos serían ejecutados por el comando *crontab*, pero el sistema *cron* supone que la máquina está siempre encendida, si no es así, es mejor utilizar *anacron*, que verifica si la acción no se realizó cuando habría debido hacerlo, y la ejecuta. En cada línea del anterior fichero se verifica que exista el comando *anacron* y se ejecutan los *scripts* asociados a cada acción, que en este caso están guardados en unos directorios asignados para ello, los *cron*.

También pueden existir unos ficheros *cron.allow*, *cron.deny*, para limitar quién puede colocar (o no) trabajos en *cron*. Mediante el comando *crontab*, un usuario puede definir trabajos en el mismo formato que hemos visto antes, que habitualmente se guardarán en /var/spool/cron/crontabs. En el caso de la distribución Debian, existe también un directorio /etc/cron.d donde se pueden colocar trabajos y que es tratado como si fueran una extensión del fichero /etc/crontab.

5.10. Taller: prácticas combinadas de los diferentes apartados

Comenzaremos por examinar el estado general de nuestro sistema. Vamos a hacer los diferentes pasos en un sistema Debian. Se trata de un sistema Debian Sid (la versión inestable, pero más actualizada); sin embargo, los procedimientos son en su mayor parte trasladables a otras distribuciones como Red Hat (mencionaremos algunos de los cambios más importantes). El hardware consiste en un AMD

ANOTACIONES

Athlon a 900 Mhz con 512 MB y varios discos, DVD y grabador de CD, además de otros periféricos, pero ya iremos obteniendo esta información paso a paso.

Veamos primero cómo ha arrancado nuestro sistema la última vez:

```
# uptime
19:03:52 up 13 min, 3 users, load average: 0.08, 0.19, 0.16
```

Este comando nos da el tiempo que lleva el sistema "levantado" desde que se arrancó la última vez, 13 minutos, en nuestro caso tenemos 3 usuarios. Éstos no tienen por qué ser tres usuarios diferentes, sino que normalmente serán las sesiones de usuario abiertas (por ejemplo, mediante un terminal). El comando *who* permite listar estos usuarios. El *load average* es la carga media del sistema en tanto por uno de CPU, nos indica el nivel de trabajo medio en los últimos 1, 5 y 15 minutos.

Veamos el *log* del arranque del sistema, las líneas que se iban generando en la carga del sistema (se han suprimido diferentes líneas por claridad):

```
# dmesg | more
Linux version 2.4.20-3-k7 (herbert@gondolin) (gcc version 3.3 (Debian))
#1 Sun Jun 8 01:35:14 EST 2003
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
 BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 000000000f00000 (usable)
 BIOS-e820: 000000000f00000 - 0000000001000000 (reserved)
 BIOS-e820: 0000000001000000 - 000000001fff0000 (usable)
 BIOS-e820: 000000001fff0000 - 000000001fff3000 (ACPI NVS)
 BIOS-e820: 000000001fff3000 - 0000000020000000 (ACPI data)
 BIOS-e820: 00000000ffff0000 - 0000000100000000 (reserved)
0MB HIGHMEM available.
511MB LOWMEM available.
```

Estas primeras líneas ya nos indican varios datos interesantes: la versión del *kernel* Linux es la 2.4.20-3-k7, una versión 2.4 revisión 20

a revisión 3 de Debian, y para máquinas k7 (Amd Athlon). Indica también que estamos arrancando un sistema Debian, con este *kernel* que fue compilado con un compilador GNU *gcc* versión 3.3 y la fecha. A continuación, existe un mapa de zonas de memoria usadas (reservadas) por la BIOS, y a continuación el total de memoria detectada en la máquina: 511 MB, a las que habría que sumar el primer 1 MB, total de 512 MB.

```
kernel command line: auto BOOT_IMAGE = Linux ro root = 302
Initializing CPU#0
Detected 900.065 MHz processor.
Console: colour VGA+ 80x25
Calibrating delay loop... 1795.68 BogoMIPS
Memory: 511848k/524224k available (986k kernel code,
10964k reserved, 391k data, 100k init, 0k highmem)
```

Aquí nos refiere cómo ha sido el arranque de la máquina, qué línea de comandos se le ha pasado al *kernel* (pueden pasarse diferentes opciones, por ejemplo desde el *lilo* o *grub*). Hay una sola CPU (puede haber varias) de 900 MHz, y estamos arrancando en modo consola de 80 x 25 caracteres (esto se puede cambiar). Los BogoMIPS son una medida interna del *kernel* de la velocidad de la CPU, hay arquitecturas donde es difícil detectar con cuántos MHz funciona la CPU, y por eso se utiliza esta medida de velocidad. Después nos da más datos de la memoria principal, y para qué se está usando en este momento del arranque.

```
CPU: L1 I Cache: 64K (64 bytes/line), D cache 64K (64 bytes/line)
CPU: L2 Cache: 256K (64 bytes/line)
...
CPU: AMD Athlon(tm) processor stepping 02
..... CPU clock speed is 900.0460 MHz.
..... host bus clock speed is 200.0102 MHz.
```

Además, proporciona datos varios de la CPU, el tamaño de las caché de primer nivel (caché interna a la CPU, L1) y caché externa (L2), tipo de CPU, velocidad de ésta y del bus del sistema.

ANOTA

```
PCI: PCI BIOS revision 2.10 entry at 0xfb4e0, last bus = 1
...
Linux NET4.0 for Linux 2.4
Starting kswapd
VFS: Diskquotas version dquot_6.4.0 initialized
pty: 256 UNIX98 ptys configured
Serial driver version 5.05c (20010708) with HUB6 MANY_PORTS ...
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 4096 buckets, 32Kbytes
TCP: Hash tables configured (established 32768 bind 65536)
Freeing initrd memory: 3040k freed
```

Continúan las inicializaciones del *kernel* y dispositivos, menciona que tenemos la versión 4 de la interfaz de red para Linux, que activa *kswapd* (*daemon* interno del *kernel* que gestiona el intercambio en disco). Los terminales (*pty*), los puertos serie ttyS00 (sería el *com1*), ttyS01 (el *com2*), información IP de protocolos disponible. Da información de los discos RAM que usamos, y nos dice que ha liberado el espacio de initrd (recordar, como hemos visto en la unidad de *kernel*, que se trataba de un archivo para realizar la primera fase del arranque del sistema).

```
Uniform Multi-Platform E-IDE driver Revision: 6.31
VP_IDE: IDE controller on PCI bus 00 dev 39
VP_IDE: VIA vt82c686a (rev 22) IDE UDMA66 controller on pci00:07.1
ide0: BM-DMA at 0xd000-0xd007, BIOS settings: hda:DMA, hdb:DMA
ide1: BM-DMA at 0xd008-0xd00f, BIOS settings: hdc:DMA, hdd:DMA
hda: ST320430A, ATA DISK drive
hdb: ST340016A, ATA DISK drive
hdc: _NEC DV5700A, ATAPI CD/DVD-ROM drive
hdd: HLDTST GCE8160B, ATAPI CD/DVD-ROM drive
...
Journalled Block Device driver loaded
kjournald starting. Commit interval 5 seconds
EXT3fs: mounted filesystem with ordered data mode.
Adding Swap: 763076k swapspace (priority -1)
EXT3 FS 2.4-0.9.19, 19 August 2002 on ide0(3,2), internal journal
```

Detección de dispositivos IDE, detecta el chip IDE en el bus PCI, e informa de que está controlando cuatro dispositivos: *hda*, *hdb*, *hdc*,

hdd, que son respectivamente: un disco duro (*seagate*, acrónimo ST), un segundo disco duro, un DVD Nec, y una grabadora de CD (en este caso es una LG). Más adelante, detecta el *filesystem* principal de Linux, un ext3 con *journal*, que activa y añade el espacio de *swap* disponible en una partición.

```
Real Time Clock Driver v1.10e
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
usb-uhci.c: High bandwidth mode enabled
PCI: Found IRQ 11 for device 00:07.2
PCI: Sharing IRQ 11 with 00:07.3
usb-uhci.c: USB UHCI at I/O 0xd400, IRQ 11
usb-uhci.c: Detected 2 ports
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
hub.c: 2 ports detected
...
usb-uhci.c: v1.275:USB Universal Host Controller Interface driver
```

Más detección de dispositivos, USB en este caso (el *kernel* 2.4 tiene un soporte bastante amplio de dispositivos USB), ha detectado dos dispositivos (*2 USB ports*).

```
3c59x: Donald Becker and others. www.scyld.com/network/vortex.html
00:0f.0: 3Com PCI 3c900 Cyclone 10Mbps TPO at 0xe000. Vers LK1.1.16
es1371: version v0.30 time 02:11:59 Jun 8 2003
PCI: Found IRQ 9 for device 00:09.0
es1371: found chip, vendor id 0x1274 device id 0x1371 revision 0x06
ac97_codec: AC97 Audio codec, id: CRY19(Cirrus Logic CS4297A rev A)
0: NVIDIA: loading NVIDIA Linux x86 NVIDIA.o kernel Module
   1.04363 Sat Apr 19 17:46:46 PDT 2003
Ensoniq AudioPCI soundcard not found or device busy
parport0: PC-style at 0x378 (0x778) [PCSPP,TRISTATE]
parport_pc: Via 686A parallel port: io = 0x378
lp0: using parport0 (polling).
apm: BIOS version 1.2 Flags 0x07 (Driver version 1.16)
Linux agpgart interface v0.99 (c) Jeff Hartmann
```

Y la detección final del resto de dispositivos: tarjeta de red (3com), tarjeta sonido es1371 (SoundBlaster PCI128), se ha cargado el controlador de tarjetas gráfican NVIDIA (este controlador no es estándar, lo proporciona NVIDIA y hay que instalarlo por separado), la máquina tiene una Riva TNT2. Detección del puerto paralelo. Bios de tipos APM, que permite la suspensión e hibernación de la máquina, y módulo final de control del modo AGP de la tarjeta de vídeo.

Toda esta información que hemos visto mediante el comando *dmesg*, también la podemos encontrar volcada en el *log* principal del sistema /var/*log*/messages, aquí encontraremos los mensajes del *kernel* y de los *daemons*, y errores de red o dispositivos que comunican sus mensajes a un *daemon* especial llamado *syslog*, que es el encargado de escribir los mensajes en este fichero. Si hemos arrancado la máquina recientemente, observaremos que las últimas líneas contienen exactamente la misma información que el comando *dmesg*, por ejemplo, si nos quedamos con la parte final del fichero (suele ser muy grande):

```
tail 200 /var/log/messages
```

Observamos las líneas de antes, y también algunas informaciones más, como por ejemplo:

```
kernel: kernel logging (proc) stopped.
kernel: kernel log daemon terminating.
exiting on signal 1
...
syslogd 1.4.1#11: restart.
...
kernel: klogd 1.4.1#11, log source = /proc/kmsg started.
kernel: Inspecting /boot/System.map-2.4.20-3-k7
kernel: Loaded 15461 symbols from /boot/System.map-2.4.20-3-k7.
kernel: Symbols match kernel version 2.4.20.
kernel: Loaded 4714 symbols from 19 modules.
```

La primera parte corresponde a la parada anterior del sistema, nos informa de que el *kernel* ha dejado de colocar información en /proc, se está parando el sistema,... Al principio del arranque nuevo, se ac-

tiva el *daemon* Syslogd que genera el *log* y comienza la carga del sistema, que nos dice que el *kernel* comenzará a escribir información en su sistema, /proc; que utiliza los archivos de /boot para obtener los símbolos del *kernel* y módulos que va a cargar, y que coinciden con la versión que se está cargando. Y luego viene lo que hemos visto en *dmesg*.

En este punto, otro comando útil para saber cómo se ha producido la carga es *ismod*, que nos permitirá saber qué modulos se han cargado en el *kernel*:

```
# lsmod
```

```
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
keybdev                 2084     0   (unused)
usbkbd                  3640     0   (unused)
input                   3520     0   [keybdev usbkbd]
usbuhci                23248     0   (unused)
usbcore                62220     0   [usbkbd usbuhci]
rtc                     6792     0   (autoclean)
ext3                   63808     1   (autoclean)
jbd                    41764     1   (autoclean) [ext3]
ide-disk               12448     2   (autoclean)
ide-probe-mod           9744     0   (autoclean)
ide-mod               167736     2   (autoclean) [ide-disk ide-probe-mod]
```

Vemos que básicamente tenemos los controladores para el hardware que hemos detectado.

Ya tenemos, pues, una idea de cómo se ha cargado el *kernel* y sus módulos. En este proceso puede que ya hubiésemos observado algún error, si hay hardware mal configurado o módulos del *kernel* mal compilados (no eran para la versión del *kernel* adecuada), inexistentes, etc.

El paso siguiente será la observación de los procesos en el sistema, con el comando *ps* (*Process Status*), por ejemplo (sólo se han sacado los procesos de sistema, no los de los usuarios):

```
ps ef
UID PID PPID C STIME TTY TIME CMD
```

Información de los procesos, UID usuario que ha lanzado el proceso (o con qué identificador se ha lanzado), PID, código del proceso asignado por el sistema, son consecutivos a medida que se lanzan los procesos, el primero siempre es el 0, que corresponde al proceso de *init*. PPID es el id del proceso padre del actual. STIME, tiempo en que fue arrancado el proceso, TTY, terminal asignado al proceso (si tiene alguno), CMD, línea de comando con que fue lanzado.

```
root   1  0  0  18:50  ?  00:00:00  init   [2]
root   2  1  0  18:50  ?  00:00:00  [keventd]
root   3  1  0  18:50  ?  00:00:00  [ksoftirqd_CPU0]
root   4  1  0  18:50  ?  00:00:00  [kswapd]
root   5  1  0  18:50  ?  00:00:00  [bdflush]
root   6  1  0  18:50  ?  00:00:00  [kupdated]
root  32  1  0  18:50  ?  00:00:00  [kjournald]
root  82  1  0  18:50  ?  00:00:00  [khubd]
```

Varios *daemons* de sistema, como kswapd *daemon*, que controla el intercambio de páginas con memoria virtual. Manejo de *buffers* del sistema (*kupdated*, *bdflush*). Manejo de *journal* de *filesystem* (*kjournald*), manejo de USB (*khubd*). En general, no siempre, los *daemons* suelen identificarse por una *d* final, y si llevan un *k* inicial, normalmente son hilos (*threads*) internos del *kernel*.

ANOTACIONES

```
root    181  1   0   18:50  ?  00:00:00  dhclient pf /var/run/dhclient.eth0
daemon  185  1   0   18:50  ?  00:00:00  /sbin/portmap
root    403  1   0   18:50  ?  00:00:00  /sbin/syslogd
root    406  1   0   18:50  ?  00:00:00  /sbin/klogd
root    423  1   0   18:50  ?  00:00:00  /usr/sbin/cupsd
root    444  1   0   18:50  ?  00:00:00  /usr/sbin/inetd
root    448  1   0   18:50  ?  00:00:00  /usr/sbin/lpd
```

Tenemos *dhclient*, esto indica que esta máquina es cliente de un servidor DHCP, para obtener su IP. Syslogd, *daemon* que envía mensajes al *log*. Los *daemons lpd* y *chupad*, como vimos, relacionados con el sistema de impresión. E*inetd*, que, como veremos en la parte de redes, es una especie de "superservidor" o intermediario de otros *daemons* relacionados con servicios de red.

```
root      460    1    0   18:50  ?  00:00:00  /usr/sbin/sshd
root      469    1    0   18:50  ?  00:00:00  /usr/bin/X11/xfs -daemon
root      548    1    0   18:50  ?  00:00:00  /sbin/rpc.statd
daemon    551    1    0   18:50  ?  00:00:00  /usr/sbin/atd
root      554    1    0   18:50  ?  00:00:00  /usr/sbin/cron
root      558    1    0   18:50  ?  00:00:00  /usr/sbin/apache
wwwdata   562  558    0   18:50  ?  00:00:00  /usr/sbin/apache
wwwdata   563  558    0   18:50  ?  00:00:00  /usr/sbin/apache
```

También está *sshd*, servidor de acceso remoto seguro (una versión mejorada que permite servicios compatibles con telnet y ftp). *xfs* es el servidor de fuentes (tipos de letra) de X Window. Los comandos *atd* y *cron* sirven para manejar tareas programadas en un momento determinado. Apache es el servidor web, que puede tener varios hilos activos (*threads*) para atender diferentes peticiones.

```
root      567    1    0   18:50     ?   00:00:00  /usr/bin/gdm
root      574    1    0   18:50   tty1  00:00:00  /sbin/getty 38400 tty1
root      575    1    0   18:50   tty2  00:00:00  /sbin/getty 38400 tty2
root      576    1    0   18:50   tty3  00:00:00  /sbin/getty 38400 tty3
root      580  572    7   18:50     ?   00:05:53  /usr/bin/X11/X :0 -defer
root      582    1    0   18:50     ?   00:00:00  [kapmd]
root      1605 949    0   20:07  pts/0  00:00:00  ps -ef
```

Gdm es el *login* gráfico del sistema de escritorio Gnome (la ventana que nos pide el *login* y contraseña), los procesos *getty* son los que gestionan los terminales virtuales de texto (los que podemos ver con las teclas Alt+Fx (o Ctrl+Alt+Fx si estamos en modo gráfico). X es el proceso de servidor gráfico de X Window System, imprescindible para que corra cualquier entorno de escritorio encima de él. Tenemos *kapmd*, que gestiona la suspensión o hibernación de la máquina, y finalmente el proceso que hemos generado al hacer este *ps* desde la línea de comandos.

El comando *ps* tiene muchas opciones de línea de comandos para ajustar la información que queremos de cada proceso, ya sea tiempo que hace que se ejecuta, porcentaje de CPU usado, memoria utilizada, etc... (ver *man* de *ps*). Otro comando muy interesante es *top*, que hace lo mismo que *ps* pero de forma dinámica, o sea, se actualiza con cierto intervalo, podemos clasificar los procesos por uso de CPU, de memoria, y también nos da información del estado de la memoria global.

Otros comandos útiles para uso de recursos son *free* y *vmstat*, que nos dan información sobre la memoria utilizada y el sistema de memoria virtual:

```
# free
```

```
                 total       used       free     shared    buffers     cached
Mem:            514988     268708     246280          0      24960     144584
-/+ buffers/cache:          99164     415824
Swap:           763076          0     763076
```

```
# vmstat
```

```
 procs -----------memory---------- ---swap-- ---io---- --system-- ----cpu--------
 r  b swpd   free  buff   cache   si   so   bi   bo   in   cs  us  sy  id  wa
 0  0    0 246264 24964  144596    0    0   21   16  179  202  10   1  89   0
```

Nota

Ver *man* de los comandos para interpretar las salidas.

En el comando *free* también se puede observar el tamaño del *swap* presente, aproximadamente de unas 700 MB, que de momento no se está usando por tener suficiente espacio de memoria física, todavía quedan unas 200 MB libres. El espacio de memoria y el *swap* (a

partir de los *kernels* 2.4) son aditivos para componer el total de memoria del sistema, haciendo en este caso un total de 1,2 GB de memoria disponible. Esto puede parecer mucho, pero dependerá de las aplicaciones que se estén ejecutando.

5.11. Actividades para el lector

1) El espacio de *swap* permite complementar la memoria física para disponer de mayor memoria virtual. Dependiendo de los tamaños de memoria física y *swap*, ¿puede llegar a agotarse la memoria? ¿Podemos solucionarlo de otro modo, que no sea añadiendo más memoria física?

2) Supongamos que tenemos un sistema con dos particiones Linux, una / y la otra de *swap*. ¿Cómo solucionaríamos el caso de que las cuentas de los usuarios agotasen el espacio de disco? Y en el caso de tener una partición /home aislada que se estuviera agotando, ¿cómo lo solucionaríamos?

3) Instalar el sistema de impresión CUPS, definir nuestra impresora para que funcione con CUPS y probar la administración vía interfaz web. ¿Tal como está el sistema, ¿sería recomendable modificar de algún modo la configuración que trae por defecto CUPS? ¿Por qué?

4) Examinar la configuración por defecto que traiga el sistema GNU/Linux de trabajos no interactivos por cron. ¿Qué trabajos y cuándo se están realizando? ¿Alguna idea para nuevos trabajos que haya que añadir?

5) Reproducir el analisis del taller (más los otros apartados de la unidad) sobre la máquina que se disponga, ¿se observan en el sistema algunos errores o situaciones anómalas? En tal caso, ¿cómo las corregimos?

5.12. Otras fuentes de referencia e información

- [W+02] [Fri02] [Smi02] Libros de administración de GNU/Linux y UNIX, donde se comentan ampliamente aspectos de administración local y gestión de sistemas de impresión.

ANOTACIONES

- [GT03] Se puede encontrar información actualizada de los sistemas de impresión y su configuración, así como detalles de algunas impresoras. Para detalles concretos de modelos de impresora, y controladores, podemos dirigirnos a http://www.linuxprinting.org/.

- [Hin00][Koe01] Encontramos información sobre los diferentes sistemas de ficheros disponibles y los esquemas de creación de particiones para la instalación del sistema.

ANOTACIONES

6. Administración de red

El sistema operativo UNIX (GNU/Linux) es tomado como ejemplo de una arquitectura de comunicaciones estándar. Desde el mítico UUCP (servicio de copia entre sistemas operativos UNIX) hasta las redes actuales, UNIX siempre ha mostrado su versatilidad en aspectos relacionados a la comunicación y el intercambio de información. Con la introducción de redes de ordenadores (área local LAN, área amplia WAN o las más actuales área metropolitana MAN) ofreciendo enlaces multipunto a diferentes velocidades (56kbits/seg hasta 1Gbit/seg), han ido surgiendo nuevos servicios basados en protocolos más rápidos, portables entre diferentes ordenadores y mejor adaptados, como el TCP/IP (Transport Control Program / Internet Protocol). [Com01, Mal96, Cis00, Gar98, KD00]

6.1. Introducción a TCP/IP (TCP/IP suite)

El protocolo TCP/IP sintetiza un ejemplo de estandarización y una voluntad de comunicación a nivel global.

El protocolo TCP/IP es en realidad un conjunto de protocolos básicos que se han ido agregando a principal para satisfacer las diferentes necesidades en la comunicación ordenador-ordenador como son TCP, UDP, IP, ICMP, ARP. [Mal96]

La utilización más frecuente de TCP/IP para el usuario en la actualidad son la conexión remota a otros ordenadores (telnet, SSH *Secure Shell*), la utilización de ficheros remotos (*Network File System* NFS) o su transferencia (*File Transfer Protocol* FTP, *Hipertext Markup Protocol,* HTTP).

Nota

Utilización típica de TCP/IP remote *login*:

```
telnet localhost
Debian GNU/Linux 3.0
login.
```

6.1.1. Servicios sobre TCP/IP

Los servicios TCP/IP tradicionales más importantes son [Gar98]:

- **Transferencia de archivos**: el *File Transfer Protocol* (FTP) permite a un usuario de un ordenador obtener/enviar archivos de un ordenador hacia otro ordenador. Para ello, el usuario deberá tener una cuenta en el ordenador remoto e identificarse a través de su nombre (*login*) y una palabra clave (*password*) o a ordenadores donde existe un repositorio de información (software, documentación, ...), y el usuario se conectará como anónimo (*anonymous*) para transferir (leer) estos archivos a su ordenador. Esto no es lo mismo que los más recientes sistemas de archivos de red, NFS, *Network File System*, (o protocolos **netbios** sobre tcp/ip, "invento" totalmente inseguro sobre Windows y que es mejor reemplazar por una versión más antigua pero más segura del mismo concepto llamado **netbeui**) que permiten virtualizar el sistema de archivos de una máquina para que pueda ser accedido en forma interactiva sobre otro ordenador.

- **Conexión (*login*) remota**: el protocolo de terminal de red (telnet) permite a un usuario conectarse a un ordenador remotamente. El ordenador local se utiliza como terminal del ordenador remoto y todo es ejecutado sobre éste permaneciendo el ordenador local invisible desde el punto de vista de la sesión. Este servicio en la actualidad se ha reemplazado por el SHH (*Secure Shell*) por razones de seguridad. En una conexión remota mediante telnet, los mensajes circulan tal cual (texto plano), o sea, si alguien "observa" los mensajes en la red, equivaldrá a mirar la pantalla del usuario. SSH codifica la información (que significa un coste añadido a la comunicación) que hace que los paquetes en la red sean ilegibles a un nodo extraño.

- **eMail**: este servicio permite enviar mensajes a los usuarios de otros ordenadores. Este modo de comunicación se ha transformado en un elemento vital en la vida de los usuarios y permiten que los *e-mails* (correos electrónicos) sean enviados a un servidor central para que después puedan ser recuperados por medio de programas específicos (clientes) o leídos a través de una conexión web.

El avance de la tecnología y el bajo coste de los ordenadores ha permitido que determinados servicios se hayan especializado en ello y

ANOTACIONES

se ofrecen configurados sobre determinados ordenadores trabajando en un modelo cliente-servidor. Un servidor es un sistema que ofrece un servicio específico para el resto de la red. Un cliente es otro ordenador que utiliza este servicio. Todos estos servicios generalmente son ofrecidos dentro de TCP/IP:

- **Sistemas de archivos en red** (*Network File Systems*): permite a un sistema acceder a los archivos sobre un sistema remoto en una forma más integrada que FTP. Los dispositivos de almacenamiento (o parte de ellos) son exportados hacia el sistema que desea acceder y éste los puede "ver" como si fueran dispositivos locales. Este protocolo permite a quien exporta poner las reglas y la formas de acceso, lo que (bien configurado) hace independiente el lugar donde se encuentra la información físicamente del sitio donde se "ve" la información.

- **Impresión remota**: permite acceder a impresoras conectadas a otros ordenadores.

- **Ejecución remota**: permite que un usuario ejecute un programa sobre otro ordenador. Existen diferentes maneras de realizar esta ejecución: o bien a través de un comando (*rsh*, ssh, *rexec*) o a través de sistemas con RPC (*Remote Procedure Call*) que permiten a un programa en un ordenador local ejecutar una función de un programa sobre otro ordenador. Los mecanismos RPC han sido objeto de estudio y existen diversas implementaciones, pero las más comunes son Xerox's Courier y Sun's RPC (esta última adoptada por la mayoría de los UNIX).

- **Servidores de nombre** (*name servers*): en grandes instalaciones existen un conjunto de datos que necesitan ser centralizados para mejorar su utilización, por ejemplo, nombre de usuarios, palabras claves, direcciones de red, etc. Tod ello facilita que un usuario disponga de una cuenta para todas las máquinas de una organización. Por ejemplo, Sun's Yellow Pages (NIS en las versiones actuales de *sun's*) está diseñado para manejar todo este tipo de datos y se encuentra disponible para la mayoría de UNIX. El DNS (*Domain Name System*) es otro servicio de nombres pero que guarda una relación entre el nombre de la máquina y la identificación lógica de esta máquina (dirección IP).

ANOTACIONES

- **Servidores de terminal** (*terminal servers*): conecta terminales a un servidor que ejecuta telnet para conectarse al ordenador central. Este tipo de instalaciones permite básicamente reducir costes y mejorar las conexiones al ordenador central (en determinados casos).

- **Servidores de terminales gráficas** (*network-oriented window systems*): permiten que un ordenador pueda visualizar información gráfica sobre un *display* que está conectado a otro ordenador. El más común de estos sistemas es X Window.

6.1.2. ¿Qué es TCP/IP?

TCP/IP son en realidad dos protocolos de comunicación entre ordenadores independientes uno del otro.

Por un lado, TCP (transmission control protocol), define las reglas de comunicación para que un ordenador (host) pueda 'hablar' con otro (si se toma como referencia el modelo de comunicaciones OSI/ISO se describe la capa 4, ver tabla siguiente).

TCP es orientado a conexión, es decir, equivalente a un teléfono, y la comunicación se trata como un flujo de datos (*stream*).

Por otro lado, IP (*Internet Protocol*), define el protocolo que permite identificar las redes y establecer los caminos entre los diferentes ordenadores.

Es decir, encamina los datos entre dos ordenadores a través de las redes. Corresponde a la capa 3 del modelo OSI/ISO y es un protocolo sin conexión (ver tabla siguiente) [Com01, Rid00, Dra99].

Una alternativa al TCP la conforma el protocolo UDP (*User Datagram Protocol*), el cual trata los datos como un mensaje (datagrama) y en-

ANOTACIONES

vía paquetes. Es un protocolo sin conexión (el ordenador destino no debe necesariamente estar escuchando cuando un ordenador establece comunicación con él) y tiene la ventaja de que ejerce una menor sobrecarga a la red que las conexiones de TCP, pero la comunicación no es fiable (los paquetes pueden no llegar o llegar duplicados).

Existe otro protocolo alternativo llamado ICMP (*Internet Control Message Protocol*). ICMP se utiliza para mensajes de error o control. Por ejemplo, si uno intenta conectarse a un equipo (*host*), el ordenador local puede recibir un mensaje ICMP indicando *"host unreachable"*. ICMP también puede ser utilizado para extraer información sobre una red. ICMP es similar a UDP, ya que maneja mensajes (datagramas), pero es más simple que UDP, ya que no posee identificación de puertos (los puertos son buzones donde se depositan los paquetes de datos y desde donde las aplicaciones servidoras leen dichos paquetes) en el encabezamiento del mensaje.

En el modelo de comunicaciones de la OSI/ISO (*OSI, Open Systems Interconnection Reference Model, ISO, International Standards Organization*), es un modelo teórico adoptado por muchas redes. Existen siete capas de comunicación donde cada una tiene una interfaz para comunicarse con la anterior y la posterior:

Nivel	Nombre	Utilización
7	Aplicación	SMTP, *simple mail transfer protocol*, el servicio propiamente dicho
6	Presentación	Telnet, FTP implementa el protocolo del servicio
5	Sesión	Generalmente no se utiliza
4	Transporte	TCP, UDP transformación de acuerdo a protocolo de comunicación
3	Red	IP permite encaminar el paquete (*routing*)
2	Link	Controladores (*drivers*) transformación de acuerdo al protocolo físico
1	Físico	Ethernet, ADSL, ... envío del paquete físicamente

En resumen, TCP/IP es una familia de protocolos (que incluyen IP, TCP, UDP) que proveen un conjunto de funciones a bajo nivel utilizadas por la mayoría de las aplicaciones. [KD00, Dra99].

Algunos de los protocolos que utilizan los servicios mencionados han sido diseñados por Berkeley, Sun u otras organizaciones. Ellos no for-

ANOTACIONES

man oficialmente parte de *Internet Protocol Suite* (IPS). Sin embargo, son implementados utilizando TCP/IP y por lo tanto considerados como parte formal de IPS. Una descripción de los protocolos disponibles en Internet puede consultarse la RFC 1011 (ver referencias sobre RFC [IET03]) que lista todos los protocolos disponibles. Existe actualmente una nueva versión del protocolo IPv6, también llamado IPng (IP *next generation*) que reemplaza al IPv4. Este protocolo mejora notablemente el anterior en temas tales como mayor número de nodos, control de tráfico, seguridad o mejoras en aspectos de *routing*.

6.1.3. Dispositivos físicos (hardware) de red

Desde el punto de vista físico (capa 1 del modelo OSI), el hardware más utilizado para LAN es conocido como Ethernet (o FastEthernet o GigaEthernet). Sus ventajas son su bajo coste, velocidades aceptables (10, 100, o 1,000 megabits por segundo) y facilidad en su instalación.

Existen tres modos de conexión en función del tipo de cable de interconexión: grueso (*thick*), fino (*thin*), y par trenzado (*twisted par*).

Las dos primeras están obsoletas (utilizan cable coaxial), mientras que la última se realiza a través de cables (pares) trenzados y conectores similares a los telefónicos (se conocen como RJ45). La conexión par trenzado es conocida como 10baseT o 100baseT (según la velocidad) y utiliza repetidores llamados *hubs* como puntos de interconexión. La tecnología Ethernet utiliza elementos intermedios de comunicación (*hubs*, *switchs*, *routers*) para configurar múltiples segmentos de red y dividir el tráfico para mejorar las prestaciones de transferencia de información. Normalmente, en las grandes instituciones estas LAN Ethernet están interconectadas a través de fibra óptica utilizando tecnología FDDI (*Fiber Distributed Data Interface*) que es mucho más cara y compleja de instalar, pero se pueden obtener velocidades de transmisión equivalentes a Ethernet y no tienen la limitación de la distancia de ésta (FDDI admite distancias de hasta 200 km). Su coste se justifica para enlaces entre edificios o entre segmentos de red muy congestionados.[Rid00, KD00].

ANOTACIONES

Existe además otro tipo de hardware menos común, pero no menos interesante como es ATM (*Asynchronous Transfer Mode*). Este hardware permite montar LAN con una calidad de servicio elevada y es una buena opción cuando deben montarse redes de alta velocidad y baja latencia, como por ejemplo aquellas que involucren distribución de vídeo en tiempo real.

Existe otro hardware soportado por GNU/Linux para la interconexión de ordenadores, entre los cuales podemos mencionar: Frame Relay o X.25 (utilizada en ordenadores que acceden o interconectan WAN y para servidores con grandes necesidades de transferencias de datos), Packet Radio (interconexión vía radio utilizando protocolos como AX.25, NetRom o Rose) o dispositivos *dialing up*, que utilizan líneas series, lentas pero muy baratas, a través de módems analógicos o digitales (RDSI, DSL, ADSL, etc.). Estas últimas son las que comúnmente se utilizan en pymes o uso doméstico y requieren otro protocolo para la transmisión de paquetes, tal como SLIP o PPP. Para virtualizar la diversidad de hardware sobre una red, TCP/IP define una interfaz abstracta mediante la cual se concentrarán todos los paquetes que serán enviados por un dispositivo físico (lo cual también significa una red o un segmento de esta red). Por ello, por cada dispositivo de comunicación en la máquina tenderemos una interfaz correspondiente en el *kernel* del sistema operativo.

Ejemplo

Ethernet en GNU/Linux se llaman con **ethx** (donde en todas, *x* indica un número de orden comenzando por 0), la interfaz a líneas series (módems) se llaman por **pppx** (para PPP) o **slx** (para SLIP), para FDDI son **fddix**. Estos nombres son utilizados por los comandos para configurarlas y asignarles el número de identificación que posteriormente permitirá comunicarse con otros dispositivos en la red.

En GNU/Linux puede significar tener que incluir los módulos adecuados para el dispositivo (*network interface card* NIC) adecuado (en el *kernel* o como módulos), esto significa compilar el *kernel* después de haber escogido con, porejemplo, *make menuconfig* el NIC adecuado, indicándole como interno o como módulo (en este último caso se deberá compilar el módulo adecuado también).

ANOTACIONES

Nota

¿Cómo ver las interfaces de red disponibles?

```
ifconfig -a
```

Este comando muestra todas las interfaces/parámetros por defecto de cada una.

Los dispositivos de red se pueden mirar en el directorio /dev que es donde existe un archivo (especial, ya sea de bloque o de caracteres según su transferencia), que representa a cada dispositivo hardware.[KD00, Dra99].

6.2. Conceptos en TCP/IP

Como se ha observado, la comunicación significa una serie de conceptos que ampliaremos a continuación [Mal96, Com01]:

- **Internet/intranet**: el término *intranet* se refiere a la aplicación de tecnologías de Internet (red de redes) dentro de una organización básicamente para distribuir y tener disponible información dentro de la compañía. Por ejemplo, los servicios ofrecidos por GNU/Linux como servicios Internet e intranet incluyen correo electrónico, WWW, *news*, etc.

- **Nodo**: se denomina nodo (*host*) a una máquina que se conecta a la red (en un sentido amplio un nodo puede ser un ordenador, una impresora, una torre (*rack*) de CD, etc.), es decir, un elemento activo y diferenciable en la red que reclama o presta algún servicio y/o comparte información.

- **Dirección de red Ethernet (*Ethernet address* o *MAC address*)**: un número de 48 bits (por ejemplo 00:88:40:73:AB:FF –en octal– 0000 0000 1000 1000 0100 0000 0111 0011 1010 1011 1111 1111 en binario) que se encuentra en el dispositivo físico (hardware) del controlador (NIC) de red Ethernet y es grabado por el fabricante del mismo (este número debe ser único en el mundo, por lo que cada fabricante de NIC tiene un rango preasignado).

- *Host name*: cada nodo debe tener además un único nombre en la red. Ellos pueden ser sólo nombres o bien utilizar un esquema de nombres jerárquico basado en dominios (*hierarchical domain naming scheme*). Los nombres de los nodos deben ser únicos, lo cual resulta fácil en pequeñas redes, más dificultoso en redes extensas e imposible en Internet si no se realiza algún control. Los nombres

ANOTACIONES

deben ser de un máximo de 32 caracteres entre a-zA-Z0-9.-, y que no contengan espacios o # comenzando por un carácter alfabético.

- **Dirección de Internet (*IP address*)**: está compuesto por cuatro números en el rango 0-255 separados por puntos (por ejemplo 192.168.0.1) y es utilizado universalmente para identificar los ordenadores sobre una red o Internet. La traslación de nombres en direcciones IP es realizada por un servidor DNS (*Domain Name System*) que transforma los nombres de nodo (legibles por humanos) en direcciones IP (este servicio es realizado por una aplicación denominada *named*).

Nota

Nombre de la máquina:
`more /etc/hostname`

- **Puerto (*port*)**: identificador numérico del buzón en un nodo que permite que un mensaje (TCP, UDP) pueda ser leído por una aplicación concreta dentro de este nodo (por ejemplo, dos máquinas que se comuniquen por telnet lo harán por el puerto 23, pero las dos mismas máquinas pueden tener una comunicación ftp por el puerto 21). Se pueden tener diferentes aplicaciones comunicándose entre dos nodos a través de diferentes puertos simultáneamente.

Nota

Dirección IP de la máquina:
`more /etc/hosts`

- **Nodo *router* (*gateway*)**: es un nodo que realiza encaminamientos (transferencia de datos *routing*). Un *router*, según sus características, podrá transferir información entre dos redes de protocolos similares o diferentes y puede ser además selectivo.

Nota

Puertos preasignados en UNIX:
`more /etc/services`
Este comando muestra los puertos predefinidos por orden y según soporten TCP o UDP.

- ***Domain Name System* (DNS)**: permite asegurar un único nombre y facilitar la administración de las bases de datos que realizan la traslación entre nombre y dirección de Internet y se estructuran en forma de árbol. Para ello, se especifican dominios separados por puntos, de los que el más alto (de derecha a izquierda) describe una categoría, institución o país (COM, comercial, EDU, educación, GOV, gubernamental, MIL, militar (gobierno), ORG, sin fin de lucro, XX dos letras por país, ...). El segundo nivel representa la organización, el tercero y restantes departamentos, secciones o divisiones dentro de una organización (por ejemplo, www.uoc.edu o nteum@pirulo.remix.es). Los dos primeros nombres (de derecha a izquierda, *uoc.edu* en el primer caso, *remix.es* (en el segundo) deben ser asignados (aprobados) por el SRI-NIC (órgano mundial gestor de Internet) y los restantes pueden ser configurados/asignados por la institución.

Nota

Configuración del *routing*:
`netstat -r`

ANOTACIONES

Nota

Dominio y quién es nuestro servidor de DNS:

```
more /etc/default domain
more /etc/resolv.conf
```

Nota

Tablas de arp:

```
arp a NombreNodo
```

- **DHCP, bootp**: DHCP y bootp son protocolos que permiten a un nodo cliente obtener información de la red (tal como la dirección IP del nodo). Muchas organizaciones con gran cantidad de máquinas utilizan este mecanismo para facilitar la administración en grandes redes o donde existe una gran cantidad de usuarios móviles.

- **ARP, RARP**: en algunas redes (como por ejemplo IEEE 802 LAN que es el estándar para Ethernet), las direcciones IP son descubiertas automáticamente a través de dos protocolos miembros de *Internet protocol suite: Address Resolution Protocol* (ARP) y *Reverse Address Resolution Protocol* (RARP). ARP utiliza mensajes (*broadcast messages*) para determinar la dirección Ethernet (especificación MAC de la capa 3 del modelo OSI) correspondiente a una dirección de red particular (IP). RARP utiliza mensajes de tipo *broadcast* (mensaje que llega a todos los nodos) para determinar la dirección de red asociada con una dirección hardware en particular. RARP es especialmente importante en máquinas sin disco, en las cuales la dirección de red generalmente no se conoce en el momento del inicio (*boot*).

- **Biblioteca de *sockets***: en UNIX toda la implementación de TCP/IP forma parte del *kernel* del sistema operativo (o bien dentro del mismo o como un módulo que se carga en el momento del inicio como el caso de GNU/Linux con los controladores de dispositivos).

  La forma de utilizarlas por un programador es a través de la API (*Application Programming Interface*) que implementa ese operativo. Para TCP/IP la API más común es la Berkeley Socket Library (Windows utiliza un sustituto que "compró" a otra compañía y se llama Winsocks). Esta biblioteca permite crear un punto de comunicación (*socket*), asociar éste a una dirección de un nodo remoto/puerto (*bind*) y ofrecer el servicio de comunicación (a través de *connect, listen, accept, send, sendto, recv, recvfrom,* por ejemplo). La biblioteca provee además de la forma más general de comunicación (familia AF INET) comunicaciones más optimizadas para casos que los procesos que se comunican en la misma máquina (familia AF UNIX). En GNU/Linux, la biblioteca de *socket* es parte de la biblioteca estándar de C, Libc, (Libc6 en las versiones actuales), y soporta AF_INET, AF_UNIX, AF_IPX (para protocolos de redes Novell), AF_X25 (para el pro-

ANOTACIONES

tocolo X.25), AF_ATMPVC-AF_ATMSVC (para el protocolo ATM) y AF_AX25, F_NETROM, AF_ROSE (para el *amateur radio protocol*).

6.3. ¿Cómo se asigna una dirección Internet?

Esta dirección es asignada por el NIC y tiene dos campos. El izquierdo representa la identificación de la red y el derecho la identificación del nodo. Considerando lo mencionado anteriormente (4 números entre 0-255, o sea 32 bits o cuatro bytes), cada byte representa o bien la red o bien el nodo. La parte de red es asignada por el NIC y la parte del nodo es asignada por la institución o el proveedor).

Existen algunas restricciones: 0 (por ejemplo, 0.0.0.0) en el campo de red está reservado para el *routing* por defecto y 127 (por ejemplo, 127.0.0.1) es reservado para la autorreferencia (*local loopback* o *local host*), 0 en la parte de nodo se refiere a esta red (por ejemplo, 192.168.0.0) y 255 es reservado para paquetes de envío a todas las máquinas (*broadcast*) (por ejemplo, 198.162.255.255). En las diferentes asignaciones se puede tener diferentes tipos de redes o direcciones:

- **Clase A** (*red.host.host.host*): 1.0.0.1 a 126.254.254.254 (126 redes, 16 millones de nodos) definen las grandes redes. El patrón binario es 0 + 7 bits red + 24 bits de nodos.

- **Clase B** (***red.red***.*host.host*): 128.1.0.1 a 191.255.254.254 (16K redes, 65K nodos) generalmente se utiliza el primer byte de nodo para identificar subredes dentro de una institución). El patrón binario es **10** + 14 bits de red + 16 bits de nodos.

- **Clase C** (***red.red.red***.*host*): 192.1.1.1 a 223.255.255.254 (2 millones de bits de redes, 254 de nodos). El patrón binario es **110** + 21 bits red + 8 bits de nodos.

ANOTACIONES

- **Clase D y E** (*red.red.red.host*): 224.1.1.1 a 255.255.255.254 reservado para *multicast* (desde un nodo a un conjunto de nodos que forman parte de un grupo) y propósitos experimentales.

Algunos rangos de direcciones han sido reservados para que no correspondan a redes públicas, sino a redes privadas (máquinas que se conectan entre ellas sin tener conexión con el exterior) y los mensajes no serán encaminados a través de **Internet**, lo cual es comúnmente conocido como **Intranet**). Éstas son para la clase A 10.0.0.0 hasta 10.255.255.255, clase B 172.16.0.0 hasta 172.31.0.0 y clase C 192.168.0.0 hasta 192.168.255.0.

La dirección de *broadcast* es especial, ya que cada nodo en una red escucha todos los mensajes (además de su propia dirección). Esta dirección permite que datagramas (generalmente información de *routing* y mensajes de aviso) puedan ser enviados a una red y todos los nodos del mismo segmento de red los puedan leer. Por ejemplo, cuando ARP busca encontrar la dirección Ethernet correspondiente a una IP, éste utiliza un mensaje de *broadcast*, el cual es enviado a todas las máquinas de la red simultáneamente. Cada nodo en la red lee este mensaje y compara la IP que se busca con la propia y le retorna un mensaje al nodo que hizo la pregunta si hay coincidencia.

Dos conceptos complementarios a lo descrito anteriormente es el de **subredes** y *routing* entre ellas. Subredes significa subdividir la parte del nodo en pequeñas redes dentro de la misma red para, por ejemplo, mejorar el tráfico. Una subred toma la responsabilidad de enviar el tráfico a ciertos rangos de direcciones IP extendiendo el mismo concepto de redes A, B, C, pero sólo aplicando esta redirección en la parte nodo de la IP. El número de bits que son interpretados como identificador de la subred es dado por una máscara de red (*netmask*) que es un número de 32 bits (igual que la IP). Para obtener el identificador de la subred, se deberá hacer una operación lógica Y (*AND*) entre la máscara y la IP, lo cual dará la IP de la subred. Por ejemplo, sea una institución que tienen una red clase B con número 172.17.0.0, y su *netmask* es, por lo tanto, 255.255.0.0. Internamente, esta red está formada por pequeñas redes (una planta del edificio por ejemplo). Así, el rango de direcciones es reasignado en 20 subnets (plantas para nosotros) 172.17.1.0 hasta 172.17.20.0. El punto que conecta todas estas plantas (*backbone*) tiene su propia dirección, por ejemplo 172.17.1.0.

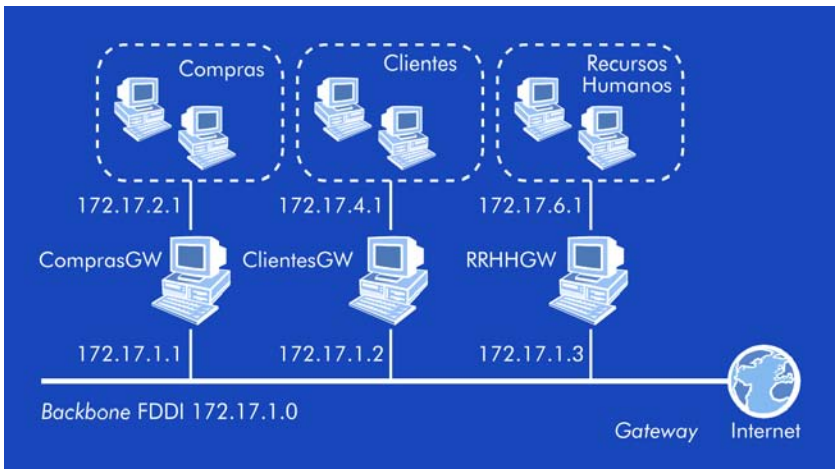
ANOTACIONES

Estas subredes comparten el mismo IP de red, mientras que el tercero es utilizado para identificar cada una de las subredes dentro de ella (por eso se utilizará una máscara de red 255.255.255.0).

El segundo concepto (*routing*), representa el modo en que los mensajes son enviados a través de las subredes. Por ejemplo, sean tres departamentos con subredes Ethernet: compras (subred 172.17.2.0), clientes (subred 172.17.4.0), recursos humanos, RR.HH., (subred 172.17.6.0) y el *backbone* con FFDI (subred 172.17.1.0). Para encaminar los mensajes entre los ordenadores de las tres redes, se necesitarán tres *gateways* que tendrán cada uno dos interfaces de red para cambiar entre Ethernet y FFDI. Éstos serán CromprasGW (IPs:172.17.2.1 y 172.17.1.1), CLientesGW (IPs:172.17.4.1 y 172.17.1.2) y RRHHGW (IPs:172.17.6.1 y 172.17.1.3), es decir, una IP hacia el lado de la subnet y otra hacia el *backbone*.

Cuando se envían mensajes entre máquinas de compras, no es necesario salir al *gateway*, ya que el protocolo TCP/IP encontrará la máquina directamente. El problema está cuando la máquina *Compras0* quiere enviar un mensaje a RRHH3. El mensaje debe circular por los dos *gateways* respectivos. Cuando Compras0 "ve" que RRHH3 está en otra red, envía el paquete a través del *gateway* ComprasGW, que a su vez se lo enviará a RRHHGW y que a su vez se lo enviará a RRHH3. La ventaja de las subredes es clara, ya que el tráfico entre todas las máquinas de compras, por ejemplo, no afectará a las máquinas de clientes o RR.HH. (si bien significa un planteamiento más complejo y caro a la hora de diseñar, y construir la red).

Figura 14. Configuración de segmentos y *gateways* en una Intranet



ANOTACIONES

IP utiliza una tabla para hacer el *routing* de los paquetes entre las diferentes redes y en la cual existe un *routing* por defecto asociado a la red 0.0.0.0. Todas las direcciones que coinciden con ésta, ya que ninguno de los 32 bits son necesarios, son enviados por el *gateway* por defecto (*default gateway*) hacia la red indicada. Sobre comprasGW, por ejemplo, la tabla podría ser:

Dirección	Máscara	Gateway	Interfaz
172.17.1.0	255.255.255.0	-	fddi0
172.17.4.0	255.255.255.0	172.17.1.2	fddi0
172.17.6.0	255.255.255.0	172.17.1.3	fddi0
0.0.0.0	0.0.0.0	172.17.2.1	fddi0
172.17.2.0	255.255.255.0	-	eth0

El '-' significa que la máquina está directamente conectada y no necesita *routing*. El procedimiento para identificar si se realiza el *routing* o no, se lleva a cabo a través de una operación muy simple con dos AND lógicos (*subred* AND *mask* y *origen* AND *mask*) y una comparación entre los dos resultados. Si son iguales no hay *routing*, sino que se debe enviar la máquina definida como *gateway* en cada máquina para que ésta realice el *routing* del mensaje.

Por ejemplo, un mensaje de la 172.17.2.4 hacia la 172.17.2.6 significará:

```
172.17.2.4 AND 255.255.255.0 = 172.17.2.0 y
172.17.2.6 AND 255.255.255.0 = 172.17.2.0
```

Como los resultados son iguales, no habrá *routing*. En cambio, si hacemos lo mismo con 172.17.2.4 hacia 172.17.6.6 podemos ver que habrá un *routing* a través del 172.17.2.1 con un cambio de interfaz (*eth0* a *ffdi0*) a la 172.17.1.1 y de ésta hacia la 172.17.1.2 con otro cambio de interfaz (*fddi0* a *eth0*) y luego hacia la 172.17.6.6. El *routing*, por defecto, se utilizará cuando ninguna regla satisfaga la coincidencia. En caso de que dos reglas coincidan, se utilizará aquella que lo haga de modo más preciso, es decir, la que menos ceros tenga. Para construir las tablas de *routing*, se puede utilizar el comando *route* durante el arranque de la máquina, pero si es necesario utilizar reglas más complejas (o *routing* automá-

ANOTACIONES

tico), se puede utilizar el *routing information protocol* (RIP) o entre sistemas autónomos el *external gateway protocol* (EGP) o también el *border gateway protocol* (BGP). Estos protocolos se implementan en el comando *gated*.

Para instalar una máquina sobre una red existente, es necesario, por lo tanto, disponer de la siguiente información obtenida del proveedor de red o de su administrador: dirección IP del nodo, dirección de la red IP, dirección de *broadcast*, dirección de máscara de red, dirección de *router*, dirección del DNS.

Si se construye una red que nunca tendrá conexión a Internet, se pueden escoger las direcciones que se prefieran, pero es recomendable mantener un orden adecuado en función del tamaño de red que se desee tener y para evitar problemas de administración dentro de dicha red. A continuación, se verá cómo se define la red y el nodo para una red privada (hay que ser cuidadoso, ya que si se tiene la máquina conectada a la red, se podría perjudicar a otro usuario que tuviera asignada esta dirección): dirección de nodo 192.168.110.23, máscara de red 255.255.255.0, parte de red 192.168.110., parte de nodo .23, dirección de red 192.168.110.0, dirección de *broadcast* 192.168.110.255.

6.4. ¿Cómo se debe configurar la red?

6.4.1. Configuración de la interfaz (NIC, *network interface controller*)

Una vez cargado el *kernel* de GNU/Linux, éste ejecuta el comando *init* que a su vez lee el archivo de configuración /etc/inittab y comienza el proceso de inicialización. Generalmente, el *inittab* tiene secuencias tales como: *si::sysinit:/etc/init.d/boot* , que representa el nombre del archivo de comandos (*script*) que controla las secuencias de inicialización. Generalmente este *script* llama a otros *scripts*, entre los cuales se encuentra la inicialización de la red.

ANOTACIONES

Ejemplo

En Debian se ejecuta etc/init.d/network para la configuración de la interfaz de red y en función del nivel de arranque; por ejemplo, en el 2 se ejecutarán todos los ficheros S* del directorio /etc/rc2.d (que son enlaces al directorio /etc/initd), y en el nivel de apagado, todos los K* del mismo directorio. De este modo, el *script* está sólo una vez (/etc/init.d) y de acuerdo a los servicios deseados en ese estado se crea un enlace en el directorio correspondiente a la configuración del nodo-estado.

Los dispositivos de red se crean automáticamente cuando se inicializa el hardware correspondiente. Por ejemplo, el controlador de Ethernet crea las interfaces *eth*[0..*n*] secuencialmente cuando se localiza el hardware correspondiente.

Nota

Consultar *man ifconfig* para las diferentes opciones del comando.

A partir de este momento, se puede configurar la interfaz de red, lo cual implica dos pasos: asignar la dirección de red al dispositivo e inicializar los parámetros de la red al sistema. El comando utilizado para ello es el *ifconfig* (*interface configure*). Un ejemplo será:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
```

Lo cual indica configurar el dispositivo *eth0* con dirección IP 192.168.110.23 y máscara de red 255.255.255.0. El *up* indica que la interfaz pasará al estado activo (para desactivarla debería ejecutarse *ifconfig eth0 down*). El comando asume que si algunos valores no se indican, son tomados por defecto. En este caso, el *kernel* configurará esta máquina como Tipo-C y configurará la red con 192.168.110.23 y la dirección de *broadcast* con 192.168.110.255. Por ejemplo:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
```

Existen comandos como el *ifup* e *ifdown*, que permite configurar-desconfigurar la red en forma más simple utilizando el archivo /etc/network/interfaces para obtener todos los parámetros necesarios (consultar *man interfaces* para su sintaxis).

ANOTACIONES

6.4.2. Configuración del Name Resolver

El siguiente paso es configurar el *name resolver* que convierte nombres tales como *pirulo.remix.com* en 192.168.110.23. El archivo /etc/resolv.conf es el utilizado para tal fin. Su formato es muy simple (una línea de texto por sentencia). Existen tres palabras clave para tal fin: *domain* (dominio local), *search* (lista de dominios alternativos) y *name server* (la dirección IP del *Domain Name Server*).

Ejemplo

Ejemplo de resolv.conf:

```
domain remix.com
search remix.com piru.com
name server 192.168.110.1
name server 192.168.110.65
```

Un archivo importante es el /etc/*host*.conf, que permite configurar el comportamiento del *name* resolver. Su importancia reside en indicar dónde se resuelve primero la dirección o el nombre de un nodo. Esta consulta puede ser realizada al servidor DNS o a tablas locales dentro de la máquina actual (/etc/hosts).

Ejemplo

Ejemplo de /etc/host.conf:

```
order hosts,bind
multi on
```

Esta configuración indica que primero se verifique el /etc/hosts antes de solicitar una petición al DNS y también indica (2.ª línea) que retorne todas las direcciones válidas que se encuentren en /etc/hosts. Por lo cual, el archivo /etc/hosts es donde se colocan las direcciones locales o también sirve para acceder a nodos sin tener que consultar al DNS.

La consulta es mucho más rápida, pero tiene la desventaja de que si el nodo cambia, la dirección será incorrecta. En un sistema correctamente configurado, sólo deberán aparecer el nodo local y una entrada para la interfaz *loopback*.

ANOTACIONES

Ejemplo

Ejemplo de /etc/hosts:

```
127.0.0.1 localhost loopback
192.168.1.2 pirulo.remix.com
```

Para el nombre de una máquina pueden utilizarse alias, que significa que esa máquina puede llamarse de diferentes maneras para la misma dirección IP. En referencia a la interfaz *loopback,* éste es un tipo especial de interfaz que permite realizar a nodo conexiones consigo misma (por ejemplo, para verificar que el subsistema de red funciona sin acceder a la red). Por defecto, la dirección IP 127.0.0.1 ha sido asignada específicamente al *loopback* (un comando telnet 127.0.0.1 conectará con la misma máquina). Su configuración es muy fácil (la realizan generalmente los *script* de inicialización de red).

Ejemplo

Configuración del *loopback*:

```
ifconfig lo 127.0.0.1
route add host 127.0.0.1 lo
```

En la versión 2 de la biblioteca GNU existe un reemplazo importante con respecto a la funcionalidad del archivo host.conf. Esta mejora incluye la centralización de información de diferentes servicios para la resolución de nombres, lo cual presenta grandes ventajas para el administrador de red. Toda la información de consulta de nombres y servicios ha sido centralizada en el archivo /etc/nsswitch.conf, el cual permite al administrador configurar el orden y las bases de datos de modo muy simple. En este archivo cada servicio aparece uno por línea con un conjunto de opciones donde, por ejemplo, la resolución de nombres de nodo es una de ellas. En éste se indica que el orden de consulta de las bases de datos para obtener el IP del nodo o su nombre será primero el servicio de DNS (que utilizará el archivo /etc/resolv.conf para determinar la IP del nodo DNS) y en caso de que no pueda obtenerlo, utilizará el de las bases de datos local (/etc/*hosts*). Otras opciones para ello podrían ser *nis, nisplus* que son otros servicios de infor-

ANOTACIONES

mación que serán descritos en unidades posteriores. También se puede controlar por medio de acciones (entre []) el comportamiento de cada consulta, por ejemplo:

```
hosts: xfn nisplus dns [NOTFOUND = return] files
```

Esto indica que cuando se realice la consulta al DNS, si no existe un registro para esta consulta, retorne al programa que la hizo con un cero. Puede utilizarse el '!' para negar la acción, por ejemplo:

```
hosts dns [!UNAVAIL = return] files
```

Ejemplo

Ejemplo de nsswitch.conf:

```
...
hosts: dns files
...
networks: files
```

6.4.3. Configuración del *routing*

Otro aspecto que hay que configurar es el *routing*. Si bien existe el tópico sobre su dificultad, generalmente se necesitan unos requerimientos de *routing* muy simples. En un nodo con múltiples conexiones, el *routing* consiste en decidir dónde hay que enviar y qué se recibe. Un nodo simple (una sola conexión de red) también necesita *routing*, ya que todos los nodos disponen de un *loopback* y una conexión de red (por ejemplo, Ethernet, PPP, SLIP, ...). Como se explicó anteriormente, existe una tabla llamada *routing table* que contiene filas con diversos campos, pero con tres campos sumamente importantes: **dirección de destino**, **interfaz** por donde saldrá el mensaje y **dirección IP**, que efectuará el siguiente paso en la red (*gateway*).

Ejemplo

Consulta de tablas de *routing*:

```
route -n
```

o también

```
netstat -r
```

El comando *route* permite modificar esta tabla para realizar las tareas de *routing* adecuadas. Cuando llega un mensaje, se mira su dirección destino, se compara con las entradas en la tabla y se envía por la interfaz en la cual la dirección que mejor coincide con el destino del paquete. Si un *gateway* es especificado, se envía a la interfaz adecuada.

Consideremos, por ejemplo, que nuestro nodo está en una red clase C con dirección 192.168.110.0 y tiene una dirección 192.168.110.23; y

ANOTACIONES

el *router* con conexión a Internet es 192.168.110.3. La configuración será:

- Primero la interfaz:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
```

- Más adelante, indicar que todos los datagramas para nodo con direcciones 192.168.0.* deben ser enviados al dispositivo de red:

```
route add –net 192.1 ethernetmask 255.255.255.0 eth0
```

El *-net* indica que es una ruta de red pero también puede utilizarse *-host* 192.168.110.3. Esta configuración permitirá conectarse a todos los nodos dentro del segmento de red (192.1), pero ¿qué pasará si se desea conectar con otro nodo fuera de este segmento? Sería muy difícil tener todas las entradas adecuadas para todas las máquinas a las cuales se quiere conectar. Para simplificar esta tarea, existe el *default route*, que es utilizado cuando la dirección destino no coincide en la tabla con ninguna de las entradas. Una posibilidad de configuración sería:

```
route add default gw 192.168.110.3 eth0
```

(el *gw* es la IP o nombre de un *gateway* o nodo *router*).

6.4.4. Configuración del *inetd*

El siguiente paso en la configuración de red es la configuración de los servidores y servicios que permitirán a otro usuario acceder a la máquina local o a sus servicios. Los programas servidores utilizarán los puertos para escuchar las peticiones de los clientes, los cuales se dirigirán a este servicio como *IP:port*. Los servidores pueden funcionar de dos maneras diferentes: *standalone* (en el cual el servicio escucha en el puerto asignado y siempre se encuentra activo) o a través del *inetd*.

ANOTACIONES

El **inetd** es un servidor que controla y gestiona las conexiones de red de los servicios especificados en el archivo /etc/inetd.conf, el cual, ante una petición de servicio, pone en marcha el servidor adecuado y le transfiere la comunicación.

Dos archivos importantes necesitan ser configurados: /etc/services y /etc/inetd.conf. En el primero se asocian los servicios, los puertos y el protocolo y en el segundo que programas servidores responderán ante una petición a un puerto determinado. El formato de /etc/services es **name port/protocol aliases**, donde el primer campo es nombre del servicio, el segundo, el puerto donde atiende este servicio y el protocolo que utiliza, y el siguiente un alias del nombre. Por defecto existen una serie de servicios que ya están preconfigurados. A continuación se muestra un ejemplo de /etc/services (# indica que lo que existe a continuación es un comentario):

```
tcpmux 1/tcp # TCP port service multiplexer
echo 7/tcp
echo 7/udp
discard 9/tcp sink null
discard 9/udp sink null
systat 11/tcp users
...
ftp 21/tcp
ssh 22/tcp # SSH Remote Login Protocol
ssh 22/udp # SSH Remote Login Protocol
telnet 23/tcp
# 24 - private
smtp 25/tcp mail
...
```

El archivo /etc/inetd.conf es la configuración para el servicio maestro de red (*inetd server daemon*). Cada línea contiene siete campos separados por espacios: *service socket_type proto flags user server_path server_args*, donde *service* es el servicio descrito en la primera columna de /etc/services, *socket_type* es el tipo de *socket* (valores posibles *stream*, *dgram*, *raw*, *rdm*, o *seqpacket*), *proto* es el protocolo válido para esta entrada (debe coincidir con el de /etc/services), *flags* indica la

ANOTACIONES

acción que tomar cuando existe una nueva conexión sobre un servicio que se encuentra atendiendo a otra conexión (*wait* le dice a *inetd* no poner en marcha un nuevo servidor o *nowait* significa que inetd debe poner en marcha un nuevo servidor). *user* será el usuario con el cual se identificará quien ha puesto en marcha el servicio, *server_path* es el directorio donde se encuentra el servidor y *server_args* son argumentos posibles que serán pasados al servidor. Un ejemplo de algunas líneas de /etc/inetd.conf es (recordar que # significa comentario, por lo cual, si un servicio tiene # antes de nombre, significa que no se encuentra disponible):

```
...
telnet stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.telnetd
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd
# fsp dgram udp wait root /usr/sbin/tcpd /usr/sbin/in.fspd
shell stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rshd
login stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rlogind
# exec stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rexecd
...
```

En Debian Woody 3.0 r1, la funcionalidad de **inetd** ha sido reemplazada por **xinetd** (recomendable), el cual necesita el archivo de configuración /etc/xinetd.conf (ver al final de la unidad). Si se desea poner en marcha el servicio de *inetd*, se debe ejecutar (y crear los *links* adecuados en los directorios /etc/rcX.d) /etc/init.d/inetd.real start.

6.4.5. Configuración adicional: *protocols* y *networks*

Existen otros archivos de configuración que en la mayoría de los casos no se utilizan pero que pueden ser interesantes. El /etc/protocols es un archivo que relaciona identificadores de protocolos con nombres de protocolos, así, los programadores pueden especificar los protocolos por sus nombres en los programas.

Ejemplo

Ejemplo de /etc/protocols:

```
...
icmp 1 ICMP Internet control message
  protocol
```

ANOTACIONES

Ejemplo

```
igmp 2 IGMP Internet Group Management
ggp 3 GGP Gateway-gateway protocol
ipencap 4 IP-ENCAP IP
encapsulated in IP (officially "IP")
...
```

El archivo /etc/networks tiene una función similar a /etc/hosts, pero con respecto a las redes, indica nombres de red en relación con su dirección IP (el comando *route* mostrará el nombre de la red y no su dirección en este caso).

Ejemplo

Ejemplo de /etc/networks:

```
...
loopnet 127.0.0.0
localnet 192.168.0.0
amprnet 44.0.0.0
...
```

6.4.6. Aspectos de seguridad

Es importante tener en cuenta los aspectos de seguridad en las conexiones a red, ya que una fuente de ataques importantes se produce a través de la red. Ya se hablará más sobre este tema en la unidad correspondiente a seguridad; sin embargo, hay unas cuantas recomendaciones básicas que deben tenerse en cuenta para minimizar los riesgos inmediatamente antes y después de configurar la red de nuestro ordenador:

- No activar servicios en /etc/inetd.conf que no se utilizarán, insertar un # antes del nombre para evitar fuentes de riesgo.

- Modificar el archivo /etc/ftpusers para denegar que ciertos usuarios puedan tener conexión vía ftp con su máquina.

- Modificar el archivo /etc/securetty para indicar desde qué terminales (un nombre por línea), por ejemplo: tty1 tty2 tty3 tty4, se permite la conexión del superusuario (*root*).

ANOTACIONES

Desde las terminales restantes, *root* no podrá conectarse.

- Utilizar el programa **tcpd**. Este servidor es un *wrapper* que permite aceptar-negar un servicio desde un determinado nodo y se coloca en el /etc/inetd.conf como intermediario de un servicio. El **tcpd** verifica unas reglas de acceso en dos archivos:

  `/etc/hosts.allow` y `/etc/host.deny`

  Si se acepta la conexión, pone en marcha el servicio adecuado pasado como argumento (por ejemplo, la línea del servicio de ftp antes mostrada en inetd.conf:

```
ftp stream tcp nowait root /usr/sbin/tcpd/usr/sbin/in.ftpd.
```

  tcpd primero busca /etc/hosts.allow y luego /etc/hosts.deny. El archivo hosts.deny contiene las reglas de cuáles son los nodos que no tienen acceso a un servicio dentro de esta máquina. Una configuración restrictiva es **ALL**: **ALL**, ya que sólo se permitirá el acceso a los servicios desde los nodos declarados en /etc/hosts.allow.

- El archivo /etc/hosts.equiv permite el acceso a esta máquina sin tener que introducir una clave de acceso (*password*). Se recomienda no utilizar este mecanismo y aconsejar a los usuarios no utilizar el equivalente desde la cuenta de usuario a través de archivo *.rhosts*.

- En Debian es importante configurar /etc/security/access.conf, el archivo que indica las reglas de quién y desde dónde se puede conectar (*login*) a esta máquina. Este archivo tiene una línea por orden con tres campos separados por ':' del tipo *permiso: usuarios:origen*. El primero será un +o- (acceso denegado), el segundo un nombre de usuario/s, grupo o user@host, y el tercero un nombre de un dispositivo, nodo, dominio, direcciones de nodo o de redes, o ALL.

Ejemplo

Ejemplo de *access.conf*. Este comando no permite *root logins* sobre *tty1*:

```
ALL EXCEPT root:tty1
     ...
```

ANOTACIONES

Permite acceder a *u1*, *u2*, *g1* y todos los de dominio *remix.com:*

```
+:u1 u2 g1 .remix.com:ALL
```

6.4.7. Opciones del IP

Existen una serie de opciones sobre el tráfico IP que es conveniente mencionar. Su configuración se realiza a través de inicializar el archivo correspondiente en directorio /proc/*sys*/net/ipv4/. El nombre del archivo es el mismo que el del comando y para activarlos se debe poner un 1 dentro del archivo, 0 para desactivarlo.

Ejemplo

Por ejemplo, si se quiere activar *ip_forward*, se debería ejecutar:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Los más utilizados son: **ip_forward** utilizado para el *routing* entre interfaces o con IP Masquerading; **ip_default_ttl**, que es el tiempo de vida para un paquete IP (64 milisegundos por defecto) **ip_bootp_agent** variable lógica (BOOLEAN) que acepta paquetes (o no) con dirección origen del tipo 0.b.c.d y destino de este nodo, *broadcast* o *multicast*.

6.5. Configuración del DHCP

DHCP son las siglas de *Dynamic Host Configuration Protocol*. Su configuración es muy simple y sirve para que en lugar de configurar cada nodo de una red individualmente se pueda hacer de forma centralizada y su administración sea más fácil. La configuración de un cliente es simplemente utilizando el programa *linuxconf* y habilitando en *Networking, Basic Host Information, Select Enable, Set Config Mode DHCP*.

La configuración del servidor requiere un poco más de atención pero no presenta complicaciones. Primero, para que el servidor pueda

ANOTACIONES

servir a todos los clientes DHCP (incluido Windows), deben realizarse algunas cuestiones previas relacionadas con las direcciones de *broadcast*. Para ello, primero el servidor debe poder enviar mensajes a la dirección 255.255.255.255, lo cual no es válido en GNU/Linux. Para probarlo, ejecútese:

```
route add -host 255.255.255.255 dev eth0
```

Si aparece el siguiente mensaje *255.255.255.255: Unknown host*, debe añadirse la siguiente entrada en /etc/hosts: *255.255.255.255 dhcp* e intentar nuevamente:

```
route add -host dhcp dev eth0
```

La configuración de *dhcpd* se puede realizar con la interfaz gráfica de *linuxconf* o bien editar /etc/dhcpd.conf. Un ejemplo de este archivo es:

```
# Ejemplo de /etc/dhcpd.conf:
default-lease-time 1200;
max-lease-time 9200;
option domain-name "remix.com";
deny unknown-clients;
deny bootp;
option broadcast-address 192.168.11.255;
option routers 192.168.11.254;
option domain-name-servers 192.168.11.1, 192.168.168.11.2;
subnet 192.168.11.0 netmásk 255.255.255.0
  { not authoritative;
    range 192.168.11.1 192.168.11.254
      host marte {
        hardware ethernet 00:00:95:C7:06:4C;
        fixed address 192.168.11.146;
        option host-name "marte";}
      host saturno {
        hardware ethernet 00:00:95:C7:06:44;
        fixed address 192.168.11.147;
        option host-name "saturno";}
  }
```

Esto permitirá al servidor asignar el rango de direcciones 192.168.11.1 al 192.168.11.254 tal y como se describe cada nodo. Si no existe el segmento *host* { ... } correspondiente, se asignan aleatoriamente. Las IP son asignadas por un tiempo mínimo de 1.200 segundos y máximo de 9.200 (en caso de no existir estos parámetros, se asignan indefinidamente).

Antes de ejecutar el servidor, debe verificarse si existe el fichero /var/state/dhcp/dhcpd.leases (en caso contrario, habrá que crearlo con touch /var/state/dhcp/dhcpd.leases). Para ejecutar el servidor: /usr/sbin/dhcpd (o bien ponerlo en los *scripts* de inicialización). Con /usr/sbin/dhcpd -d -f se podrá ver la actividad del servidor sobre la consola del sistema. [Mou01, Rid00, KD00, Dra99]

6.6. IP *aliasing*

Existen algunas aplicaciones donde es útil configurar múltiples direcciones IP a un único dispositivo de red. Los ISP (*Internet Service Providers*) utilizan frecuentemente esta característica para proveer de características personalizadas (por ejemplo, de World Wide Web y FTP) a sus usuarios. Para ello, el *kernel* debe estar compilado con las opciones de Network Aliasing e IP (*aliasing support*). Después de instalado el nuevo *kernel*, la configuración es muy fácil. Los alias son anexados a dispositivos de red virtuales asociados con el nuevo dispositivo con un formato tal como:

dispositivo: *número virtual*

Por ejemplo:

```
eth0:0, ppp0:8
```

Consideremos que tenemos una red Ethernet que soporta dos diferentes subredes IP simultáneamente y que nuestra máquina desea tener acceso directo a ellas. Un ejemplo de configuración sería:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
route add -net 192.168.110.0 netmask 255.255.255.0 eth0
ifconfig eth0:0 192.168.10.23 netmask 255.255.255.0 up
route add -net 192.168.10.0 netmask 255.255.255.0 eth0:0
```

ANOTACIONES

Lo cual significa que tendremos dos IP 192.168.110.23 y 192.168.10.23 para la misma NIC. Para borrar un alias, agregar un '-' al final del nombre (por ejemplo, *ifconfig eth0:0- 0*). [Mou01, Ran03]

6.7. IP Masquerade

El IP Masquerade es un recurso para que un conjunto de máquinas puedan utilizar una única dirección IP. Esto permite que los nodos ocultos (es decir, los que utilizan una IP privada, por ejemplo, 198.162.10.1) puedan salir hacia Internet; pero no pueden aceptar llamadas o servicios del exterior directamente, sino a través de la máquina que tiene la IP real.

Esto significa que algunos servicios no funcionan (por ejemplo, *talk*) y otros deben ser configurados en modo PASV (pasivo) para que funcionen (por ejemplo, FTP). Sin embargo, WWW, telnet o irc funcionan adecuadamente. El *kernel* debe estar configurado con las siguientes opciones: *Network firewalls, TCP/IP networking, IP:forwarding/gatewaying, IP: masquerading*. Normalmente, la configuración más común es disponer de una máquina con una conexión SLIP o PPP y tener otro dispositivo de red (por ejemplo, una tarjeta Ethernet) con una dirección de red reservada. Como vimos, y de acuerdo a la RFC 1918, se pueden utilizar como IP privadas los siguientes rangos de direcciones (IP/Mask): 10.0.0.0/255.0.0.0, 172.16.0.0/255.240.0.0, 192.168.0.0/255.255.0.0. Los nodos que deben ser ocultados (*masqueraded*) estarán dentro de esta segunda red. Cada una de estas máquinas debería tener la dirección de la máquina que realiza el *masquerade* como *default gateway* o *router*. Sobre dicha máquina podemos configurar:

- *Network route* para Ethernet considerando que la red tiene un IP = 192.168.1.0/255.255.255.0:

```
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
```

- Default *route* para el resto de Internet:

```
route add default ppp0
```

ANOTACIONES

- Todos los nodos sobre la red 192.168.1/24 serán *masqueraded*:

```
ipchains -A forward -s 192.168.1.0/24 -j MASQ
```

- Si se utiliza iptables sobre un *kernel* 2.4 o superior:

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Consultar las referencias y en la unidad que trata de la de seguridad sobre información de *ipchains* e *iptables*. [Ran03, KD00]

6.8. NAT con el *kernel* 2.2 o superiores

El *IP Network Address Translation NAT* es el reemplazo que deja obsoleto las prestaciones de GNU/Linux *IP Másquerade* y que aporta nuevas prestaciones al servicio. Dentro de las mejoras introducidas en la pila de TCP/IP del núcleo 2.2 de GNU/Linux es que el NAT forma parte del *kernel*. Para utilizarlo, es necesario que el *kernel* se compile con: *CONFIG_IP_ADVANCED_ROUTER, CONFIG_IP_MULTIPLE_TABLES y CONFIG_IP_ROUTE_NAT*. Y si se necesita control exhaustivo de las reglas NAT (por ejemplo, para activar el cortafuegos *firewalling*) debe estar también *CONFIG_IP_FIREWALL y CONFIG_IP_ROUTE_FWMARK*. Para trabajar con estas nuevas características, es necesario usar el programa *ip* (se puede obtener en ftp://ftp.inr.ac.ru/ip_routing/). Entonces para trasladar direcciones de datagramas de entrada se puede utilizar:

```
ip route add nat <extaddr>[/<masklen>] via <intaddr>
```

Esto hará que un paquete de entrada destinado a *ext-addr* (la dirección visible desde fuera de Internet) se transcribe su dirección destino a *int-addr* (la dirección de su red interna por medio de su *gateway/firewall*). El paquete se encamina de acuerdo a la tabla local de *route*. Se pueden trasladar direcciones simples o bloques. Por ejemplo:

```
ip route add nat 240.0.11.34 via 192.109.0.2
ip route add nat 240.0.11.32/27 via 192.109.0.0
```

ANOTACIONES

El primero hace que la dirección interna 192.109.0.2 sea accesible como 240.0.11.34. El segundo reubica (*remapping*) el *block* 192.109.0.031 a 240.0.11.3263. En este caso se ha utilizado como ejemplo traslaciones a direcciones de la clase DE tal como 240.0.*.* con el fin de no utilizar ninguna dirección pública. El usuario deberá reemplazar estas direcciones (240.0.11.34 y 240.0.11.3263 por las correspondientes direcciones públicas a las que desee realizar la traslación. [Ran03]

6.9. ¿Cómo configurar una conexión DialUP y PPP?

Configurar una conexión *dial-up* sobre PPP en GNU/Linux es muy simple. PPP (*Point to Point Protocol*) que permite realizar *IP-Links* entre dos ordenadores con un módem (considerar que debe ser un módem soportado por GNU/Linux, ya que no todos, especialmente los internos o los conocidos como Winmodems, se pueden configurar, puesto que muchos de ellos necesitan software adicional para establecer la comunicación). [Vas00, Law03, Sec00].

Como pasos previos se debe disponer de la siguiente información: el *init-string* del módem (normalmente no es necesario pero si la necesita y no la tiene disponible, se puede utilizar ATZ, que funciona en la mayoría de los módems, o consultar listas especializadas de *init-string*).

Además, necesitará los datos del ISP: identificación de conexión (*login name*), clave (*password*) y número de teléfono. Direcciones de DNS serían aconsejables, pero es opcional en las versiones actuales de **pppd**. Verificar además que su módem está correctamente conectado. Con un módem externo se debe ejecutar *echo > /dev/ttyS0* y mirar las luces del módem por si tienen actividad. En caso contrario, intentar con *ttyS1* por si el módem está conectado al 2.º puerto serie. Con un módem interno consultar el manual de hardware soportado para ver si este módem puede ser reconocido por GNU/Linux, y en caso afirmativo, podría tener que reconfigurar el *kernel* para utilizarlo. También puede utilizar *cat /proc/pci* por si se encuentra en el bus PCI. [PPP00]

La forma más fácil de configurar ahora el módem es a través del paquete kppp (debe instalar los paquetes kdenetwork-ppp* y ppp*). Sobre

ANOTACIONES

una terminal, ejecútese */usr*/bin/kppp. Sobre la ventana, complétense las opciones siguientes:

Accounts ⇒ New Connection
Dial ⇒ Authentication = 'PAP/CHAP'
Store Password ⇒ yes
IP ⇒ Dynamic IP Address
Autoconfigure hostname ⇒ No
Gateway ⇒ Default *Gateway* ⇒ Assign the Default Route
DNS ⇒ Configuration Automatic ⇒ Disable existing DNS
Device ⇒ ttyS1(com1) o ttyS2 (com2)
Modem ⇒ Query Modem para ver los resultados (si no obtiene resultados, cambie el dispositivo ttySx).

Entremos *login* y *password,* y estaremos conectado a Internet (para verificar la conexión podría ejecutar*ping www.google.com* por ejemplo). Aquí se ha utilizado el paquete *kppp*, pero igualmente podría utilizarse *linuxconf* o *gnomeppp* indistintamente).

6.10. *Virtual Private Network* (VPN)

Una VPN (Virtual Private Network) es una red que utiliza como transporte de datos Internet, pero impide que los datos puedan ser accedidos por miembros externos a ella.

Esto significa tener una red con VPN nodos unidos a través de un túnel por donde viaja el tráfico y donde nadie puede interactuar con él. Es utilizada cuando se tienen usuarios remotos que acceden a una red corporativa para mantener la seguridad y privacidad de los datos. Para configurar una VPN aquí, se utilizará SSH (*Secure Shell Package*) y PPP (lo haremos sobre una línea PPP), pero existen otras alternativas para crear VPN (CIPE, IPSec, PPTP) que pueden consultarse en la bibliografía (se recomienda consultar VPN PPP-SSH HOWTO, por Scott Bronson, sobre las ventajas/desventajas de utilizar SSH+PPP para establecer una VPN). [Bro01, Wil02]

ANOTACIONES

Pasos preliminares:

- Para configurar una VPN, se debe tener un cliente y un servidor y la información sobre las dos máquinas que se quieren interconectar por el túnel VPN. Esta información será *nombre del servidor = = pirulo.remix.com*, *usuario en el servidor sobre el cual se ejecutará el software* (crear uno específico para este fin)= *vpn*, *IP del servidor = 192.168.11.1*, *dirección IP del cliente = 192.168.11.2*.

- Verificar que el PPP se encuentra activo. Éste puede ser compilado con el *kernel* (mirar/var/log/daemon.log) o cargado como un módulo (utilizar *lsmod* y *modprob* para comprobar su funcionamiento en caso de que sea un módulo).

- Si se tiene un cortafuegos (*firewall*) (en este ejemplo con *Ipchains* [Rus00], pero es análogo con *Iptables* sobre un *kernel* 2.4.x) se debe habilitar el tráfico SSH sobre en servidor en el puerto 22. En el cliente se utilizará la opción -P y un puerto no privilegiado entre 1.024 y 65.535. Por ello, la configuración en el servidor podría ser algo como (reemplazar *eth0* y *IPADDR* con la interfaz donde se dirigirá el tráfico SSH):

```
ipchains -A input -i eth0 -p tcp -d IPADDR_server 22 -j ACCEPT
ipchains -A output -i eth0 -p tcp ! -y -s IPADDR_server 22 -j ACCEPT
```

Y para el cliente:

```
ipchains -A input -i eth0 -p tcp ! -y sourceport 22 -d IPADDR_client 1024:65535 -j ACCEPT
ipchains -A output -i eth0 -p tcp -s IPADDR_client 1024:65535 - - destination-port 22 -j ACCEPT
```

ANOTACIONES

- Con el usuario **vpn** creado, desde el cliente, habrá que intentar conectarse al servidor para ver que todo funciona con:

  ```
  ssh pirulo.remix.com -l vpn
  ```

  Si todo está bien, se deberá obtener el *prompt* de la cuenta en el servidor.

- **pppd** necesita ser ejecutado como *root* y sobre el servidor se deberá ejecutar con usuario **vpn**; la forma más fácil de hacerlo es a

través del comando **sudo** (permite la ejecución de comandos como *root*). Para hacerlo, puede modificar el archivo /etc/sudoers o utilizar el comando *visudo*. Como *root*, en el servidor se debe ejecutar *visudo* y agregar las siguientes líneas a final:

```
Cmnd_Alias VPN = /usr/sbin/pppd y vpn ALL = NOPASSWD: VPN
```

Se puede comprobar si todo está bien con *su - vpn*. Si se reciben por pantalla símbolos y secuencias de letras-caracteres, significa que todo está bien (para abortar el comando es necesario esperar 30 segundos o conectarse desde otra terminal y abortarlo). Si algo está mal, se tendrá un mensaje de error parecido a:

```
/usr/sbin/pppd: Permission denied
```

- Sobre el cliente utilizar el *script* vpn-pppssh (del PPP-SSH HOWTO Scott Bronson en http://www.linuxdoc.org), salvar en (por ejemplo) /usr/local/bin/vpn, cambiar los atributos *chmod* 755 /usr/local/bin/vpn y cambiar las variables:

```
SERVER_HOSTNAME = pirulo.remix
SERVER_USERNAME = vpn
SERVER_IFIPADDR = 192.168.11.1
CLIENT_IFIPADDR = 192.168.11.2.
```

Sobre el cliente y como *root*, ejecutar: */usr/local/bin/vpn start*. Se recibirá un mensaje tal como:

```
Using interface ppp1 Connect: ppp1 ⇔ /dev/pts/1 local IP address 192.168.11.1 remote IP address 192.168.11.2.
```

Para probar que funciona, ejecutar *ping 192.168.11.2* para probar la interfaz del cliente y *ping 192.168.11.1* para probar realmente la VPN. Si esto funciona, ya se ha configurado una PPP-SSH VPN. Para finalizar la VPN ejecutar:

```
/usr/local/bin/vpn-pppssh stop
```

6.11. Configuraciones avanzadas y herramientas

Existe un conjunto de paquetes complementarios (o que sustituyen a los convencionales) y herramientas que o bien mejoran la seguridad de la máquina (recomendados en ambientes hostiles), o bien ayudan

ANOTACIONES

en la configuración de red (y del sistema en general) en forma más amigable.

Estos paquetes pueden ser de gran ayuda al administrador de red para evitar intrusos o usuarios locales que se exceden de sus atribuciones (generalmente, no por el usuario local, sino a través de una suplantación de identidad) o bien ayudar al usuario novel a configurar adecuadamente los servicios.

En este sentido, es necesario contemplar:

- **Configuración avanzada de TCP/IP**: a través del comando **sysctl** es posible modificar los parámetros del *kernel* durante su ejecución o en el inicio para ajustarlos a las necesidades del sistema. Los parámetros susceptibles de modificar son los que se encuentran en el directorio /proc/*sys*/ y se pueden consultar con *sysctl -a*. La forma más simple de modificar estos parámetros es a través de archivo de configuración /etc/syscntl.conf. Después de la modificación, se debe volver a arrancar la red:

  ```
  /etc/init.d/networking restart
  ```

  En este apartado veremos algunas modificaciones para mejorar las prestaciones de la red (mejoras según condiciones) o la seguridad del sistema (consultar las referencias para más detalles) [Mou01]:

```
net.ipv4.icmp_echo_ignore_all = 1
```

No responde paquetes ICMP como por ejemplo el comando *ping* que podría significar un ataque DoS (*Denial-of-Service*).

```
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

Evita congestiones de red no respondiendo el *broadcast*.

```
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.lo.accept_source_route = 0
net.ipv4.conf.eth0.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
```

Inhibe los paquetes de *IP source routing* que podrían representar un problema de seguridad (en todas las interfaces).

```
net.ipv4.tcp_syncookies = 1 net.ipv4.conf.all.accept_redirects = 0
```

Permite rechazar un ataque DoS por paquetes SYNC que consumiría todos los recursos del sistema forzando a hacer un *reboot* de la máquina.

```
net.ipv4.conf.lo.accept_redirects = 0
net.ipv4.conf.eth0.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
```

Útil para evitar ataques con CMP Redirect Acceptance (estos paquetes son utilizados cuando el *routing* no tiene una ruta adecuada) en todas las interfaces.

```
net.ipv4.icmp_ignore_bogus_error_responses = 1
```

Envía alertas sobre todos los mensajes erróneos en la red.

```
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.lo.rp_filter = 1
net.ipv4.conf.eth0.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
```

Habilita la protección contra el *IP Spoofing* en todas las interfaces.

```
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.lo.log_martians = 1
net.ipv4.conf.eth0.log_martians = 1
net.ipv4.conf.default.log_martians = 1
```

Generará *log* sobre todos los *Spoofed Packets*, *Source Routed Packets* y *Redirect Packets*.

Los siguientes parámetros permitirán que el sistema pueda atender mejor y más rápidos las conexiones TCP.

```
net.ipv4.tcp_fin_timeout = 40
```

Por defecto, 60.

```
net.ipv4.tcp_keepalive_time = 3600
```

Por defecto, 7.200.

```
net.ipv4.tcp_window_scaling = 0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_timestamps = 0
```

Por defecto, todos a 1 (habilitados).

- **Iptables**: las últimas versiones de GNU/Linux (*kernel* 2.4 o superiores) incluyen nuevos mecanismos para construir filtros de paquetes llamado *netfilter* [Mou01]. Esta nueva funcionalidad es gestionada por una herramienta denominada **iptables** que presenta mejores características que su predecesor (*ipchains*). Como se verá en la unidad correspondiente a seguridad, es sumamente fácil construir un *firewall* con esta herramienta para detectar y hacer frente a los ataques más comunes scans, DoS, IPMAC Spoofing, etc. Su activa-

ANOTACIONES

ción pasa primero por verificar que el *kernel* es 2.4 o superior, que el mismo está configurado para dar soporte de *ipfilter* (lo cual significará que se deberá recompilar el *kernel* para activar la opción *Network packet filtering* [*CONFIG_NETFILTER*], y todas las subopciones específicas). Las reglas específicas se deben activar durante el arranque (por ejemplo, a través del /etc/init.d y el enlace adecuado al directorio *rc* adecuado) y tiene un formato similar (consultar las referencias sobre las capacidades y la sintaxis completa) a:

```
iptables -A Type -i Interface -p protocol -s SourceIP --source-port Port
-d DestinationIP --destination-port Port -j Action
```

- **GnuPG**: esta herramienta permite encriptar datos para su posterior envío (por ejemplo correo electrónico) o almacenamiento, y también para generar firmas digitales (cumple con el estándar de la RFC2440) y no utiliza algoritmos patentados, lo cual significa más libertad en el Open Source pero pérdida de compatibilidad con otras herramientas (por ejemplo, PGP 2.0) que utilizan algoritmos como el IDEA y RSA. Para su compilación y/o instalación, seguir las indicaciones de sus autores en http://www.gnupg.org/. En primer lugar, se debe crear una par de claves (pública y privada) ejecutando como *root* el comando gpg *--gen-key* **dos** veces y contestando las preguntas realizadas por el mismo. Generalmente, estas claves se almacenarán en *~/root*. Lo siguiente es exportar (por ejemplo a una página web) la clave pública para que otros usuarios la puedan utilizar para encriptar los correos/información que sólo podrá ver el usuario que ha generado la clave pública. Para ello, habrá que utilizar *gpg --export -ao UID,* lo cual generará un archivo ASCII de la clave pública del usuario UID. Para importar una clave pública de otro usuario, se puede usar *gpg --import filename,* y para firmar una clave (significa indicarle al sistema que se está de acuerdo en que la clave firmada es de quien dice ser), se puede utilizar *gpg --sign-key UID* . Para verificar una clave, se puede utilizar *gpg --verify file/data* y para encriptar/desencriptar *gpg -sear UID file g* , *gpg -d file,* respectivamente. [Gnu03]

- **Logcheck**: una de las actividades de un administrador de red es verificar diariamente (más de una vez por día) los archivos *log* para detectar posibles ataques/intrusiones o eventos que puedan dar indicios sobre estas cuestiones. Esta herramienta selecciona

ANOTACIONES

(de los archivos *log*) información condensada de problemas y riesgos potenciales y luego envia esta información al responsable, por ejemplo, a través de un correo. El paquete incluye utilidades para ejecutarse de modo autónomo y recordar la última entrada verificada para las subsiguientes ejecuciones. Para información sobre la configuración/instalación, podéis consultar las referencias. [Log03]

- **PortSentry** y **Tripwire**: estas herramientas ayudan en las funciones del administrador de red en cuanto a seguridad se refiere. *PortSentry* permite detectar y responder a acciones de búsqueda de puertos (paso previo a un ataque o a un *spamming*) en tiempo real y tomar diversas decisiones respecto a la acción que se está llevando a cabo. *Tripwire* es una herramienta que ayudará al administrador notificando sobre posibles modificaciones y cambios en archivos para evitar posibles daños (mayores). Esta herramienta compara las diferencias entre los archivos actuales y una base de datos generada previamente para detectar cambios (inserciones y borrado) lo cual es muy útil para detectar posibles modificaciones de archivos vitales como por ejemplo, en archivos de configuración. Consultar las referencias sobre la instalación/configuración de estas herramientas. [Tri03]

- **Xinetd**: esta herramienta mejora notablemente la eficiencia y prestaciones de *inetd* y *tcp –wrappers*. Una de las grandes ventajas de **xinetd** es que puede hacer frente a ataques de DoA (*Denial–of–Access*) a través de mecanismos de control para los servicios basados en la identificación de direcciones del cliente, en tiempo de acceso y tiempo de conexión (*logging*). No se debe pensar que Xinetd es el más adecuado para todos los servicios (por ejemplo, FTP y SSH es mejor que se ejecuten solos como *daemons*), ya que muchos de ellos generan una gran sobrecarga al sistema y disponen de mecanismos de acceso seguros que no crean interrupciones en la seguridad del sistema. [Xin03]

La compilación y/o instalación es simple, sólo es necesario configurar dos archivos: /etc/xinetd.conf (el archivo de configuración de Xinetd) y /etc/rc.d/init.d/xinetd (el archivo de inicialización de Xinetd). El primer archivo contiene dos secciones: **defaults**, que es donde se encuentran los parámetros que se aplicarán a todos los servicios y **service**, que serán los servicios que se pondrán en marcha a través de Xinetd.

ANOTACIONES

En la sección *defaults* se pueden insertar parámetros tales como el número máximo de peticiones simultáneas de un servicio, el tipo de registro (*log*) que se desea tener, desde qué nodos se recibirán peticiones por defecto, el número máximo de peticiones por IP que se atenderán, o servicios que se ejecutarán como superservidores (*imapd* o *popd*), como por ejemplo:

```
default {
   instances = 20
   log_type = SYSLOG authpriv
   log_on_success = HOST
   log_on_failure = HOST
   only_from = 192.168.0.0/16
   per_source = 3
   enabled = imaps
   }
```

La sección *service*, una por cada servicio como por ejemplo:

```
service imapd {
   socket_type = stream
   wait = no
   user = root
   server = /usr/sbin/imapd
   only_from = 0.0.0.0/0 #allows every client
   no_access = 192.168.0.1
   instances = 30
   log_on_success += DURATION USERID
   log_on_failure += USERID
   nice = 2
   redirect = 192.168.1.1 993 #Permite redireccionar el tráfico del port 993
                              hacia el nodo 192.168.1.1
   bind = 192.168.10.4        #Permite indicar a cual interfaz está asociado el
                              servicio para evitar problemas de suplantación de
                              servicio.
}
```

El archivo /etc/init.d/xinetd permitirá poner en marcha el servidor (con el enlace adecuado, según el nivel de ejecución seleccionado, por ejemplo, 3, 4 y 5). Es conveniente cambiar los atributos de am-

bos archivos para garantizar que no son modificados o desactivados con: *chmod* 700 /etc/init.d/xinetd; *chown* 0.0 /etc/init.d/xconfig; *chmod* 400 /etc/xinetd.conf; chattr +i /etc/xinetd.conf.

- **Linuxconf**: es una herramienta de configuración y administración de un sistema GNU/Linux. Es aconsejable para usuarios inexpertos que desean tener una configuración de la red en condiciones sin preocuparse de los archivos/procesos que entran en juego. Es aconsejable en entornos sin demasiadas particularidades y para un usuario-administrador novel. A continuación, se muestra una ventana de la interfaz principal del *linuxconf*:

Figura 15.

- **Webmin**: herramienta (paquetes *webmin-core*, *webmin-dhcp*, *webmin-inetd*, *webmin-sshd*, ...) que permite a través de una interfaz web (es necesario tener por ejemplo el servidor Apache instalado), configurar y añadir aspectos relacionados con la red en sus apartados *Servers-Internet Services and Protocols* y *Hardware-Network Configuration*. Para ello, es necesario, una vez instalados los paquetes, ejecutar por ejemplo **mozilla** y lla-

ANOTACIONES

mar a la URL https://localhost:10000, que solicitará la aceptación del certificado SSL y el usuario (inicialmente *root*) y su clave (*passwd*). Las figuras 16 y 17 muestran dos ejemplos de esta interfaz.

Figura 16.

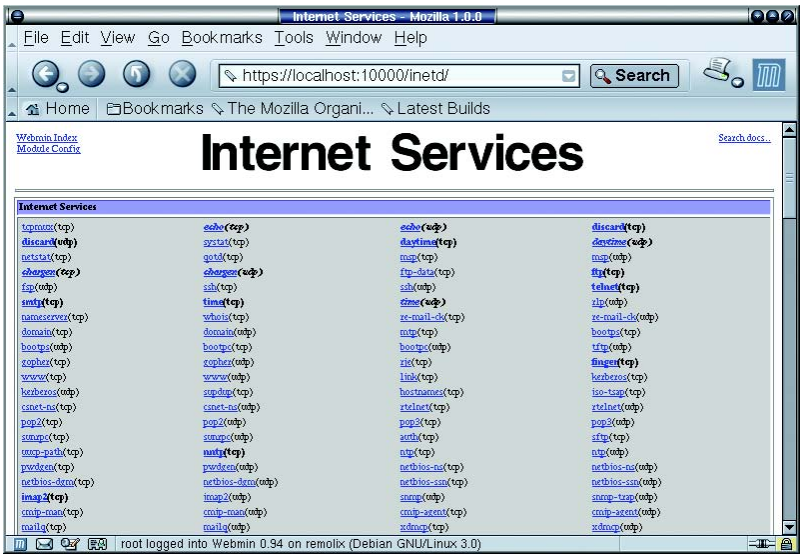
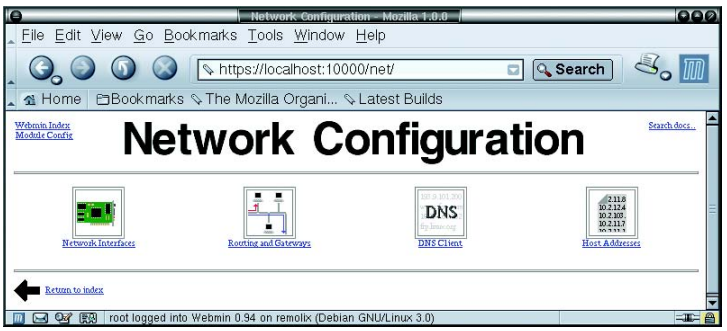


Figura 17.



- **Otras herramientas**: (algunas de ellas están recogidas en la unidad que trata acerca de la seguridad) Nmap (explorar y auditar con fines de seguridad una red), Nessus (evaluar la seguridad de una red de forma remota), Ethereal (analizador de protocolos de red), Snort (sistema de detección de intrusos, IDS), Netcat (utilidad simple y potente para depurar y explorar una red), TCPDump (monitorización de redes y adquisición de información), Hping2 (genera y envía paquetes de ICMP/UDP/TCP para analizar el funcionamiento de una red).

ANOTACIONES

6.12. Actividades para el lector

1) Definir los siguientes escenarios de red:

a) Máquina aislada.

b) Pequeña red local (4 máquinas, 1 *gateway*)

c) 2 redes locales segmentadas (2 conjuntos de 2 máquinas y un *router* cada una y un *gateway* general)

d) 2 redes locales interconectadas (dos conjuntos de 2 máquinas + *gateway* cada una)

e) 2 máquinas conectadas a través de una red privada virtual. Indicar la ventajas/desventajas de cada configuración, para qué tipo de infraestructura son adecuadas y qué parámetros relevantes se necesitan.

2) Configurar la red de la opción *a*, *b* y *d* del punto.

ANOTACIONES

7. Administración de servidores

La interconexión entre máquinas y las comunicaciones de alta velocidad han permitido que los recursos que se utilicen no estén en el mismo sitio geográfico del usuario. UNIX (y por supuesto GNU/Linux) es probablemente el máximo exponente de esta filosofía, ya que desde su inicio ha fomentado el compartir recursos y la independencia de 'dispositivos'. Esta filosofía se ha plasmado en algo común hoy en día que son los servicios. Un **servicio** es un recurso (que puede ser universal o no) y que permite bajo ciertas condiciones obtener información, compartir datos o simplemente procesar la información remotamente. Nuestro objetivo es analizar los servicios que permiten el funcionamiento de una red. Generalmente dentro de esta red existirá una máquina (o varias, según las configuraciones) que posibilitará el intercambio de información entre las restantes. Estas máquinas se denominan **servidores** y contienen un conjunto de programas que permiten que la información esté centralizada y sea fácilmente accesible. Estos servicios propician la reducción de costes y amplían la disponibilidad de la información, pero se debe tener en cuenta que un servicio centralizado presenta inconvenientes, ya que puede quedar fuera de servicio y dejar sin el servicio a todos los usuarios.

Una arquitectura de servidores debe tener los servicios replicados (*mirrors*) para solventar estas situaciones.

Los servicios se pueden clasificar en dos tipos: de **vinculación ordenador-ordenador** o de **relación hombre-ordenador**. En el primer caso se trata de servicios requeridos por otros ordenadores, mientras que en el segundo, son servicios requeridos por los usuarios (aunque hay servicios que pueden actuar en ambas categorías). Dentro del primer tipo se encuentran servicios de nombres, como el *Domain Name System* (DNS), el servicio de información de usuarios (NISYP), el directorio de información LDAP o los servicios de almacenamiento intermedio (*proxies*). Dentro de la segunda categoría, se contemplan servicios de conexión interactiva y ejecución remota (SSH, telnet),

ANOTACIONES

transferencia de ficheros (FTP), intercambio de información a nivel de usuario, como el correo electrónico (MTA, IMAP, POP), *news*, World Wide Web y archivos (NFS). Para mostrar las posibilidades de GNU/Linux Debian, se describirán cada uno de estos servicios con una configuración mínima y operativa, pero sin descuidar aspectos de seguridad y estabilidad.

7.1. *Domain Name System* (DNS)

La funcionalidad del servicio de DNS (como se explicó en la unidad dedicada a la administración de red) es convertir nombres de máquinas (legibles y fáciles de recordar por los usuarios) en direcciones IP o viceversa.

Ejemplo

A la consulta de cuál es el IP de *pirulo.remix.com,* el servidor responderá 192.168.0.1 (esta acción es conocida como *mapping*); del mismo modo, cuando se le proporcione la dirección IP, responderá con el nombre de la máquina (conocido como *reverse mapping*).

El *Domain Name System* (DNS) es una arquitectura arborescente para evitar duplicación de la información y facilitar la búsqueda. Por ello, un único DNS no tiene sentido sino como parte del árbol.

La aplicación que presta este servicio se llama *named*, se incluye en la mayoría de distribuciones de GNU/Linux (/usr/sbin/named) y forma parte de un paquete llamado *bind* (actualmente versión 9) coordinado por ISC (*Internet Software Consortium*). DNS es simplemente una base de datos, por lo cual es necesario que las personas que la modifiquen conozcan su estructura, ya que, de lo contrario el servicio quedará afectado. Como precaución, debe tenerse especial cuidado en guardar las copias de los archivos para evitar cualquier interrupción en el servicio. El paquete sobre Debian se encuentra como *bind* y *bind.doc*. [LN01, Deb03c, IET03]

ANOTACIONES

7.1.1. Servidor de nombres caché

En primer lugar se configurará un servidor de DNS para resolver consultas que actúe como caché para las consultas siguientes (*resolver, caching only server*). Es decir, la primera vez consultará al servidor adecuado porque se parte de una base de datos sin información, pero las veces siguientes responderá el servidor de nombres caché, con la correspondiente disminución del tiempo de respuesta. Para configurar el servidor de nombres caché, se necesita el archivo /etc/bind/named.conf (en Debian), que tiene el siguiente formato (se han respetado los comentarios originales dentro del archivo, indicados por //):

```
options {
   directory "/var/cache/bind";
   // query-source address * port 53;
   // forwarders {
   //      0.0.0.0;
   //      };
   auth-nxdomain no;   # conform to RFC1035
   };
   // prime the server with knowledge of the root servers}
zone "." {
      type hint;
      file "/etc/bind/db.root";
      };
   // be authoritative for the localhost forward and reverse zones, and for broadcast
      zones as per RFC 1912}
zone l.localhost" {
      type master;
      file "/etc/bind/db.local";
      };
zone "127.in-addr.arpa" {
      type master;
      file "/etc/bind/db.127";
      };
zone "0.in-addr.arpa" {
      type master;
      file "/etc/bind/db.0";
      };
zone "255.in-addr.arpa" {
      type master;
      file "/etc/bind/db.255";
      };
// add entries for other zones below here}
```

La sentencia *directory* indica dónde se encontrarán los archivos de configuración restantes (/var/cache/bind en nuestro caso). El archivo /etc/bind/db.root contendrá algo similar a lo siguiente (se muestra sólo las primeras líneas que no son comentarios indicados por ';'):

```
   ...
   ;formerly NS.INTERNIC.NET
                          3600000   IN   NS   A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.       3600000        A    198.41.0.4
   ; formerly NS1.ISI.EDU
.                         3600000        NS   B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.       3600000        A    128.9.0.107
   ; formerly C.PSI.NET
.                         3600000        NS   C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.       3600000        A    192.33.4.12
   ...
```

Este archivo describe los *root name servers* en el mundo. Estos servidores cambian, por lo que se debe actualizar periódicamente el archivo. Las siguientes secciones son las *zone*; por ejemplo, el archivo /etc/bind/db.127 para la *zone* 127.inaddr.arpa es (';' significa 'comentario'):

```
; BIND reverse data file for local loopback interface
   $TTL 604800 \\
   @       IN      SOA     ns.remix.bogus. root.remix.bogus. (
             1       ; Serial
             604800  ; Refresh
             86400   ; Retry
             2419200; Expire
             604800); Negative Cache TTL
   @       IN      NS      ns.remix.bogus.
   1.0.0   IN   PTR localhost.
```

Explicaremos su utilización más adelante. Lo siguiente es poner como *name server* en el /etc/resolv.conf:

```
search subdominio.su-dominio.es sudominio.es
   # por ejemplo search remix.bogus bogus
nameserver 127.0.0.1
```

Donde se deberán reemplazar los *subdominio.su-dominio.es* por los valores adecuados. La línea *search* indica qué dominios se buscarán para cualquier *host* que se quiera conectar (es posible sustituir *search* por

ANOTACIONES

domain, aunque tienen comportamientos diferentes) y el *nameserver* especifica la dirección de su *nameserver* (en este caso su propia máquina, que es donde se ejecutará el *named*). El *search* tiene el siguiente comportamiento: si un cliente busca la máquina *pirulo* primero se buscará *pirulo.subdominio.su-dominio.es*, luego *pirulo.sud-ominio.es* y finalmente, *pirulo*. Esto implica tiempo de búsqueda; ahora bien, si se tiene la seguridad de que *pirulo* está en *subdominio.su-dominio.es* no es necesario poner los restantes.

El paso siguiente es poner en marcha el *named* y mirar los resultados de la ejecución. Para poner en marcha el *daemon*, se puede hacer directamente con el *script* de inicialización /etc/init.d/bind9 start (en caso de que el *named* ya se esté ejecutando, hacer /etc/init.d/bind9 reload) o, si no, /usr/sbin/named. Mirando el *log* del sistema en /var/log/daemon.log veremos algo como:

```
Sep 1 20:42:28 remolix named[165]: starting BIND 9.2.1 \\
Sep 1 20:42:28 remolix named[165]: using 1 CPU \\
Sep 1 20:42:28 remolix named[167]: loading configuration from '/etc/bind/named.conf'
```

Aquí se indica el arranque del servidor y los mensajes de errores (si los hay), los cuales se deberán corregir y volver a empezar. Ahora se puede verificar la configuración con comandos como *nslookup* (original, fácil pero obsoleto según algunos autores), *host* o *dig* (recomendado). La salida de *dig -x 127.0.0.1* será algo como:

```
# dig -x 127.0.0.1
;; <<>> DiG 9.2.1 <<>> -x 127.0.0.1
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31245
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION: ;1.0.0.127.in-addr.arpa. IN PTR
;;ANSWER SECTION:
1.0.0.127.in-addr.arpa. 604800 IN PTR localhost.
;; AUTHORITY SECTION:
127.in-addr.arpa. 604800 IN NS ns.remix.bogus.
;; Query time: 1 msec
;; SERVER: 127.0.0.1 #53(127.0.0.1)
;; WHEN: Mon Sep 1 22:23:35 2003
;; MSG SIZE rcvd: 91
```

Donde se puede ver que la consulta ha tardado 1 milisegundo. Si se dispone de conexión a Internet, se podría buscar alguna máquina dentro de vuestro dominio y ver el comportamiento de vuestro servidor. En BIND9 existe el *lwresd* (*lightweight resolver daemon*), que es el *daemon* que provee servicios de nombres a clientes que utilizan la biblioteca de *BIND9 lightweight resolver*. Es esencialmente un servidor caché (como el que se ha configurado) el que realiza las consultas utilizando el *BIND9 lightweight resolver protocol* en lugar del protocolo DNS. Este servidor escucha por la interfaz 127.0.0.1 (por lo cual, sólo atiende a procesos de la máquina local) en UDP y el puerto 921. Las consultas de los clientes son decodificadas y resueltas utilizando el protocolo DNS. Cuando se obtienen las respuestas, el *lwresd* las codifica en el formato *lightweight* y las retorna al cliente que las ha solicitado.

7.1.2. *Forwarders*

En redes con una considerable carga, es posible equilibrar el tráfico utilizando la sección de *forwarders*. Si vuestro proveedor de red (ISP) tiene uno o más *nameservers* estables, es recomendable utilizarlos para descongestionar las consultas sobre su servidor. Para ello, debe quitarse el comentario (//) de cada línea de la sección *forwarders* del archivo /etc/bind/named.conf y reemplazar el 0.0.0.0 con las IP de los *nameservers* de su ISP. Esta configuración es recomendable cuando la conexión es lenta, por ejemplo, por módem.

7.1.3. Configuración de un dominio propio

DNS posee una estructura en árbol y el origen es conocido como '.' (ver /etc/bind/db.root). Bajo el '.' existen los TLD (*Top Level Domains*) como *org, com, edu, net,* etc. Cuando se busca en un servidor, si éste no conoce la respuesta, se buscará recursivamente en el árbol hasta encontrarla. Cada '.' en una dirección (por ejemplo, *pirulo.remix.com*) indica una rama del árbol de DNS diferente y un ámbito de consulta (o de responsabilidad) diferente que se irá recorriendo en forma recursiva de izquierda a derecha.

Otro aspecto importante, además del dominio, es el *in-addr.arpa* (*inverse mapping*), el cual también está anidado como los dominios

ANOTACIONES

y sirve para obtener nombres cuando se consulta por la dirección IP. En este caso, las direcciones son escritas al revés, en concordancia con el dominio. Si *pirulo.remix.com* es la 192.168.0.1, será escrita como 1.0.168.192, en concordancia con *pirulo.remix.com*.

A continuación, configuraremos el dominio propio *remix.bogus* en el archivo /etc/bind/db.127 [LN01]:

```
; BIND reverse data file for local loopback interface
$TTL 604800
@ IN SOA  ns.remix.bogus. root.remix.bogus. (
            1 ; Serial
       604800 ; Refresh
       86400  ; Retry
       2419200; Expire
     604800 ) ; Negative Cache TTL
@       IN   NS   ns.remix.bogus.
1.0.0   IN   PTR localhost.
```

Se debe tener en cuenta el '.' al final de los nombres de dominio. El origen de la jerarquía de una *zone* está especificada por la identificación de la zona, en nuestro caso *127.in-addr.arpa*. Este archivo (*db.127*) contiene 3 registros: SOA, NS, PTR. El **SOA** (*Start of Authority*) debe estar en todos los archivos de zona al inicio, después de **TTL**, y el símbolo @ significa el origen del dominio; **NS**, el servidor de nombres para el dominio, y **PTR** (*Domain Name Pointer*), que es el *host* 1 en la subred (127.0.0.1) y se denomina *local host*. Éste es el archivo serie 1 y el responsable del mismo es *root@remix.bogus* (último campo de la línea SOA). Ahora se podría reiniciar el *named* de la forma antes indicada y con el *dig -x 127.0.0.1*, ver su funcionamiento (que sería idéntico al mostrado anteriormente).

A continuación, habrá que añadir una nueva zona en el*named.conf*:

```
zone "remix.bogus" {
  type master;
  notify no;
  file "/etc/bind/remix.bogus";
  };
```

ANOTACIONES

Se debe recordar que en el *named.conf* los dominios van sin el '.' final. En el archivo *remix.bogus* se pondrán los *hosts* de los cuales seremos responsables:

```
; Zone file for remix.bogus
$TTL 604800
@ IN SOA  ns.remix.bogus. root.remix.bogus. (
          199802151; serial, todays date + todays serial
          604800   ; Refresh
          86400    ; Retry
          2419200  ; Expire
          604800 ) ; Negative Cache TTL
@         NS ns ; Inet Address of name server
          MX 10 mail.remix.bogus. ; Primary Mail Exchanger
localhostA    127.0.0.1
ns        A    192.168.1.2
mail      A    192.168.1.4
          TXT  "Mail Server"
ftp       A    192.168.1.5
          MX 10 mail
www       CNAME ftp
```

ANOTACIONES

Aquí aparece un nuevo registro **MX** que es el *Mail eXchanger*. Es el lugar donde se enviarán los correos electrónicos que lleguen, *alguien@remix.bogus,* y será a *mail.remix.bogus* (el número indica la prioridad si tenemos más de un MX). Recordar el '.' necesario siempre en los archivos de *zone* al final del dominio (si no se ponen, el sistema agrega el dominio SOA al final, lo cual transformaría, por ejemplo, *mail.remix.bogus* en *mail.remix.bogus.remix.bogus*, que es incorrecto). **CNAME** (*canonical name*) es la forma de dar a una máquina uno o varios alias. A partir de ahora se estaría en condiciones (después de /etc/init.d/bind9 reload) de probar, por ejemplo, *dig www.remix.bogus*.

El último paso es configurar la zona inversa, es decir, para que pueda convertir direcciones IP en nombres, por ejemplo, agregando una nueva zona:

```
zone "192.168.1.in-addr.arpa" {
  type master;
  notify no;
  file "/etc/bind/192.168.1";
  };
```

Y el archivo /etc/bind/192.168.1 similar al anterior:

```
$TTL 604800
@       IN   SOA ns.remix.bogus. root.remix.bogus. (
          199802151; serial, todays date + todays serial
          604800    ; Refresh
          86400     ; Retry
          2419200   ; Expire
          604800 ) ; Negative Cache TTL
@         NS ns.remix.bogus.
2       PTR  ns.remix.bogus
4       PTR  mail.remix.bogus
5       PTR  ftp.remix.bogus
```

Una vez creado un máster *server*, debe crearse un *slave server* por seguridad, que es idéntico al máster, excepto que la zona en lugar de *type master* deberá tener *slave* y la IP del *master*. Por ejemplo:

```
zone "remix.bogus" {
  type slave;
  notify no;
  masters {192.168.1.2; }
  };
```

7.2. NIS (YP)

Con el fin de facilitar la administración y dar comodidad al usuario en redes de diferentes tamaños que ejecutan GNU/Linux (o Sun o algún otro sistema operativo con soporte para este servicio) se ejecutan servicios de **Network Information Service, NIS** (o *yellow pages,* YP, en la definición original de Sun). GNU/Linux puede dar apoyo como cliente/servidor de NIS y pueden actuar como cliente (versión "beta") de NIS+, que es una versión más segura y optimizada de NIS. La infor-

ANOTACIONES

mación que se puede distribuir en NIS es: **usuarios** (*login names*), **palabras de acceso** (*passwords*) (/etc/passwd), **directorios** de usuario (*home directories*), información de **grupos** (*group information*) (/etc/group), lo cual presenta la ventaja de que, desde cualquier máquina cliente o desde el mismo servidor, el usuario se podrá conectar con la misma cuenta y *password* y al mismo directorio (aunque el directorio deberá ser montado anteriormente sobre todas las máquinas clientes por NFS o utilizando el servicio de *automount*). [Miq01, Kuk03]

La arquitectura NIS es del tipo cliente-servidor, es decir, existe un servidor que dispondrá de todas las bases de datos y unos clientes que consultarán estos datos en forma transparente para el usuario. Por ello, se debe considerar la configuración de servidores 'de refuerzo' (llamados secundarios) para que los usuarios no queden bloqueados ante la caída del servidor principal. Es por ello por lo que la arquitectura realmente se denomina de múltiples servidores (*master+mirrors*-clientes).

7.2.1. ¿Cómo iniciar un cliente local de NIS en Debian?

Un cliente local significa anexar el ordenador a un dominio NIS ya existente:

- Primero se debe verificar que se tienen instalados los paquetes *netbase* (red básica TCP/IP), *portmap* (servidor que convierte números RPC en puertos DARPA y es necesario para programas que ejecutan RPC, incluyendo NFS y NIS), y *nis* (específico). Se recomienda usar el comando *kpackage* o directamente con *apt-get* (se puede verificar si está instalado con *apt-cache pkgnames*) en modo texto. El procedimiento de instalación del paquete NIS solicitará un dominio (*NIS domainname*). Éste es un nombre que describirá el conjunto de máquinas que utilizarán el NIS (no es un nombre de *host*). Tener en cuenta que NISPirulo es diferente que Nispirulo como nombre de dominio. Para configurar este dominio, se puede utilizar el comando *nisdomainname*, dominio que se almacenará en /proc/sys/kernel/domainname.

ANOTACIONES

- En primer lugar se debe iniciar el servicio *portmap* con:

```
/etc/init.d/portmap start
```

Se puede comprobar si estos servicios están activos con *rpcinfo –p*.

- Si el servidor NIS no es local, se deberá utilizar el comando *ypbind*. El comando *ypbind* es utilizado para encontrar un servidor para el dominio especificado, ya sea mediante *broadcast* (no aconsejado por inseguro) o buscando al servidor indicado en el archivo de configuración /etc/yp.conf (recomendable). El archivo /etc/yp.conf tiene la siguiente sintaxis:

– *domain nisdomain server hostname*: indica utilizar el *hostname* para el dominio *nisdomain*. Se podrían tener más de una entrada de este tipo para un único dominio.

– *domain nisdomain broadcast*: indica utilizar *broadcast* sobre la red local para descubrir un servidor de dominio *nisdomain*.

– *ypserver hostname*: indica utilizar *hostname* como servidor. Es recomendable utilizar esta línea (*ypserver*) donde se deberá introducir la dirección IP del servidor NIS. Si se indica el nombre asegúrese que se puede encontrar por DNS la IP o que la misma figura en el archivo /etc/*hosts* ya que, de otro modo, el cliente se bloqueará.

- Inicie el servicio ejecutando:

```
/etc/init.d/nis stop
```

y luego:

```
/etc/init.d/nis start
```

A partir de este momento, el cliente NIS estará funcionando (se puede confirmar con *rpcinfo -u localhost ypbind*, que mostrará las dos versiones del protocolo activo) o se puede utilizar el comando *ypcat mapname* (por ejemplo, *ypcat passwd*, que mostrará los usuarios

ANOTACIONES

NIS definidos en el servidor) donde la relación *mapnames* a tablas de la base de datos NIS están definidos en /var/yp/nicknames.

7.2.2. ¿Qué recursos se deben especificar para utilizar en NIS?

Consideraremos que tenemos instalada una de las últimas distribuciones de Debian (por ejemplo, 3.0 *Woody* r1), que soporta la Libc6, y se quiere que los usuarios en una máquina cliente puedan acceder a la información del servidor. En este caso, se debe orientar la consulta del *login* a las bases de datos adecuadas haciendo:

1) Verificar el fichero /etc/nsswitch.conf y asegurarse de que las entradas *passwd*, *group*, *shadow* y *netgroup* son similares a:

```
passwd: compat
group: compat
shadow: compat
...
netgroup: nis
```

Ver *man nsswitch.conf* para la sintaxis de este archivo.

2) Agregar la siguiente línea en las máquinas clientes NIS en el fichero /etc/passwd al final del archivo (indicará que si el usuario no es local, se lo preguntará al servidor de NIS):

`+::::::` (un '+' y seis ':')

Debe tenerse en cuenta que en el /etc/passwd se puede utilizar el + y el – delante de cada nombre de usuario en el /etc/passwd para incluir/excluir el *login* de estos usuarios (*override*). Si se está utilizando *passwords* con shadow (más seguro, ya que no permite que un usuario normal pueda ver el *password* encriptado de otros usuarios) deberá incluir la siguiente línea al final del archivo /etc/shadow:

`+:::::::::` (un '+' y ocho ':')

3) Debe añadir también la siguiente línea al final de /etc/group:

`+:::` (un '+' y tres ':')

ANOTACIONES

4) Las búsquedas de *hosts* (*hosts lookups*) se realizarán mediante DNS (y no por NIS), por lo cual, para aplicaciones Libc6 en el fichero /etc/nsswitch.conf habrá que cambiar la entrada *hosts* por la siguiente línea: *hosts: files dns*. O, si se prefiere hacer por NIS, *hosts: files nis*. Para aplicaciones Libc5, se deberá modificar el fichero /host.conf poniendo *order hosts,dns* o *order hosts,nis* según desee).

Con esta configuración se podrá realizar una conexión local (sobre el cliente NIS) a un usuario que no esté definido en el fichero /etc/passwd, es decir, un usuario definido en otra máquina (*ypserver*).

Por ejemplo, se podría hacer *ssh l user localhost,* donde *user* es un usuario definido en *ypserver*.

7.2.3. ¿Cómo se debe ejecutar un *master* NIS *server*?

Consideramos que sobre la máquina se tiene instalado el paquete *nis* y el *portmap* (este último en funcionamiento) y que las bases de datos del NIS están creadas (ver el siguiente apartado):

- Habrá que asegurarse de que en el /etc/hosts se encuentran todas las máquinas que formarán parte del dominio en el formato FQDN (*Fully Qualified Domain Name*), que es donde se indican el *IP*, el nombre y dominio y el nombre sin dominio de cada máquina (por ejemplo, *192.168.0.1 pirulo.remix.com pirulo*). Esto es necesario sólo en el *server*, ya que el NIS no utiliza DNS.

- Además, existe en el archivo /etc/defaultdomain con el nombre del dominio escogido. No utilicéis vuestro dominio DNS para no incurrir en un riego de seguridad, excepto que configuréis adecuadamente los archivos /etc/ypserv.securenets (que indica con un par netmask/network desde qué sitio se podrán conectar los clientes) y /etc/ypserv.conf (que realiza un control más detallado porque indica qué *hosts* pueden acceder a qué mapas, por ejemplo: *passwd.byname* o *shadow.byname*).

ANOTACIONES

- Verificar que existe *NISSERVER = master* en /etc/default/nis.

- Por motivos de seguridad, se puede agregar el número de red local al archivo /etc/ypserv.securenets.

- Iniciar el servidor ejecutando el comando */etc/init.d/nis stop* y luego */etc/init.d/nis start*. Esta sentencia iniciará el *server* (*ypserv*) y el *password daemon* (yppasswdd), los cuales se podrán consultar si está activo con *ypwich -d domain*.

7.2.4. ¿Cómo se debe configurar un *server*?

La configuración del *server* se realiza mediante el comando */usr/lib/yp/ypinit -m*; sin embargo, es necesario verificar que existe el archivo /etc/networks, que es imprescindible para este *script*.

Si este archivo no existe, cread uno vacío con *touch /etc/networks*. También se puede hacer que sobre el servidor se ejecute el cliente *ypbind*; así, todos los usuarios entran por NIS, como se indicó anteriormente, modificando el fichero /etc/passwd donde todas las entradas normales antes de la línea +:::::: serán ignoradas por el NIS (sólo podrán acceder localmente), mientras que las posteriores podrán acceder por el NIS desde cualquier cliente. [Miq01]

Considerad que a partir de este momento los comandos para cambiar el *passwd* o información de los usuarios como *passwd, chfn, adduser* no son válidos. En su lugar, se deberán utilizar comandos tales como *yppasswd*, *ypchsh* e *ypchfn*. Si se cambian los usuarios o se modifican los archivos mencionados, habrá que reconstruir las tablas de NIS ejecutando el comando *make* en el directorio /var/yp para actualizar las tablas.

Tener en cuenta que Libc5 no soporta *shadow passwd*, por lo cual no se debe utilizar *shadow* con NIS si se tienen aplicaciones con Libc5. No habrá ningún problema si se tiene Libc6 que acepta NIS con soporte *shadow*.

ANOTACIONES

La configuración de un *slave server* es similar a la del *master* excepto que *NISSERVER* = *slave* en /etc/default/nis. Sobre el *master* se debe indicar que distribuya las tablas automáticamente a los *slaves* poniendo *NOPUSH* = *"false"* en el archivo /var/yp/Makefile.

Ahora se debe indicar al *master* quién es su esclavo ejecutando:

```
/usr/lib/yp/ypinit -m
```

y entrando los nombre de los *slave servers*. Esto reconstruirá los mapas, pero no enviará los archivos a los *slaves*. Para ello, sobre el *slave*, ejecutar:

```
/etc/init.d/nis stop
```

```
/etc/init.d/nis start
```

y, por último:

```
/usr/lib/yp/ypinit -s nombre_master_server.
```

Así el *slave* cargará las tablas desde el *master*.

También se podría poner en el directorio /etc/cron.d el archivo nis con un contenido similar a:

```
20 *   * * * root /usr/lib/yp/ypxfr_1perhour > /dev/null 2>&1
40 6   * * * root /usr/lib/yp/ypxfr_1perday  > /dev/null 2>&1
55 6,18* * * root /usr/lib/yp/ypxfr_2perday  > /dev/null 2>&1
```

Con lo cual, se asegurará de que todos los cambios del *master* serán transferidos al servidor NIS *slave*.

7.3. Servicios de conexión remota: telnet y ssh

7.3.1. Telnet y telnetd

Telnet es un comando (cliente) utilizado para comunicarse interactivamente con otro *host* que ejecuta el *daemon telnetd*. El comando *telnet* se puede ejecutar como *telnet host* o interactivamente como *telnet*, el cual pondrá el *prompt* "telnet>" y luego, por ejemplo: *open host*. Una

ANOTACIONES

vez establecida la comunicación, se deberá introducir el usuario y el *password* bajo el cual se desea conectar al sistema remoto. Se dispone de diversos comandos (en modo interactivo) tal como *open, logout, mode* (definir las características de visualización), *close, encrypt, quit, set, unset*, o puede ejecutar comando externos con '!'. Se puede utilizar el archivo /etc/telnetrc para definiciones por defecto, o *.telnetrc* para definiciones de un usuario particular (deberá estar en el directorio *home* de usuario).

El *daemon* telnetd es el servidor de protocolo telnet para la conexión interactiva. Telned es puesto en marcha generalmente por el *daemon inetd* y se recomienda incluir un *wrapper tcpd* (que utiliza las reglas de acceso en *host.allow* y el *host.deny*) en la llamada al *telnetd* dentro del archivo /etc/inetd.conf (por ejemplo, incluir una línea como *telnet stream tcp nowait telnetd.telenetd /usr/sbin/tcpd /usr/bin/in.telnetd*) para incrementar la seguridad del sistema (veáse la unidad dedicada a la seguridad). Se debe tener en cuenta que en Debian 3.0 *Woody* r1, la funcionalidad de *inetd* ha sido reemplazada por *xinetd* el cual requiere que se configure el archivo /etc/xinetd.conf (ver la unidad dedicada a la de administración de red). Si igualmente se quiere poner en marcha *inetd* a modo de pruebas se puede utilizar la sentencia */etc/init.d/inetd.real start*. Si el archivo /etc/uissue.net está presente, el *telnetd* mostrará su contenido al inicio de la sesión. También se puede utilizar /etc/security/access.conf para habilitar/deshabilitar *logins* de usuario, *host* o grupos de usuarios, según se conecten.

Se debe recordar que, si bien el par telnet-telnetd pueden funcionar en modo *encrypt* en las últimas versiones (transferencia de datos encriptados, aunque deben estar compilados con la opción correspondiente), es un comando que ha quedado en el olvido por su falta de seguridad aunque puede ser utilizado en redes seguras o situaciones controladas.

7.3.2. Ssh, Secure *shell*

Un cambio aconsejable hoy en día es utilizar ssh en lugar de telnet, rlogin o rsh. Estos comandos son inseguros por varias razones: la

ANOTACIONES

más importante es que todo lo que se transmite por la red, incluido *usernames* y *passwords*, es en texto plano (aunque existen versiones de *telnet-telnetd* encriptados, deben coincidir en que ambos los sean), cualquiera que tenga acceso a esa red o a algún segmento de la misma puede obtener toda esta información y luego suplantar la identidad del usuario. La segunda es que estos puertos (telnet, rsh,...) es el primer lugar donde un *cracker* intentará conectarse. El protocolo ssh (en su version OpenSSH) provee de un conexión encriptada y comprimida mucho más segura que, por ejemplo, telnet (es recomendable utilizar la versión 2 del protocolo). Todas las distribuciones actuales incorporan el cliente ssh y el servidor sshd por defecto.

ssh

Para ejecutar el comando, hacer:

`ssh -l login name host` o `ssh user@hostname`

A través de SSH se pueden encapsular otras conexiones como X11 o cualquier otra TCP/IP. Si se omite el parámetro -l, el usuario se conectará con el mismo usuario local y en ambos casos el servidor solicitará el *passwd* para validar la identidad del usuario. SSH soporta diferentes modos de autentificación (ver *man* ssh) basados en algoritmo RSA y clave pública.

Utilizando el comando *ssh-keygen -t rsa|dsa,* se pueden crear las claves de identificación de usuario. El comando crea en el directorio del *.shh* del usuario el fichero (por ejemplo, para el algoritmo de encriptación RSA) *id_rsa* y *id_rsa.pub* las claves privada y pública respectivamente. El usuario podría copiar la pública (*id_rsa.pub*) en la máquina remota en el directorio .ssh del usuario remoto, en el archivo *authorized_keys* . Este archivo podrá contener tantas claves públicas como sitios desde donde se quiera conectar a esta máquina en forma remota. La sintaxis es de una clave por línea y su funcionamiento es equivalente al archivo *.rhosts* (aunque las líneas tendrán un tamaño considerable). Después de haber introducido las claves públicas del usuario-máquina en este archivo, este usuario y desde esa máquina se podrá conectar sin *password*.

En forma normal (si no se han creado las claves), se le preguntará al usuario un *passwd*, pero como la comunicación será siempre encrip-

ANOTACIONES

tada, nunca será accesible a otros usuarios que puedan escuchar sobre la red. Para mayor información, consultar *man ssh*. Para ejecutar remotamente un comando, simplemente hacer:

```
ssh -l login name host_comando_remoto
```

Por ejemplo:

```
ssh -l user localhost ls -al).
```

sshd

El sshd es el servidor (*daemon*) para el ssh. Juntos reemplazan al *rlogin*, telnet, *rsh* y proveen una comunicación segura y encriptada en dos *hosts* inseguros en la red. Éste se arranca generalmente a través de los archivos de inicialización (/etc/init.d o /etc/rc) y espera conexiones de los clientes. El sshd de la mayoría de las distribuciones actuales soporta las versiones 1 y 2 del protocolo SSH. Cuando se instala el paquete, crea una clave RSA específica del *host*, y cuando el *daemon* se inicia, crea otra, la RSA para la sesión, que no se almacena en el disco y la cambia cada hora. Cuando un cliente inicia la comunicación, el cliente genera un número aleatorio de 256 bits que es encriptado con las dos claves del servidor y enviado. Este número se utilizará durante la comunicación como clave de sesión para encriptar la comunicación que se realizará a través de un algoritmo de encriptación estándar. El usuario puede seleccionar cualquiera de los disponibles ofrecidos por el servidor. Existen algunas diferencias (más seguro) cuando se utiliza la versión 2 del protocolo. A partir de este momento, se inician algunos de los métodos de autentificación de usuario descritos en el cliente o se le solicita el *passwd*, pero siempre con la comunicación encriptada. Para mayor información, consultar *man sshd*.

7.4. Servicios de transferencia de ficheros: FTP

El FTP (*File Transfer Protocol*) es un protocolo cliente/servidor (bajo TCP) que permite la transferencia de archivos desde y hacia un sistema remoto. Un servidor FTP es un ordenador que ejecuta el *daemon* FTPD.

ANOTACIONES

Algunos sitios que permiten la conexión anónima bajo el usuario *anonymous* son generalmente repositorios de software. En un sitio privado, se necesitará un usuario y un *passwd* para acceder.

7.4.1. Cliente ftp

Un cliente ftp permite acceder a servidores FTP y hay una gran cantidad de clientes disponibles. La utilización del ftp es sumamente simple, desde la línea de comando, ejecutar:

```
ftp nombre ftp host server
```

O también ftp, y luego en forma interactiva:

```
open nombre ftp host server
```

El servidor solicitará un *username* y un *password* (si acepta usuario anónimos, se introducirá *anonymous* como usuario y nuestra dirección de *e-mail* como *password*) y a partir del *prompt* del comando (después de algunos mensajes), podremos comenzar a transferir ficheros.

El protocolo permite transferencia en modo ASCII o binarios. Es importante decidir el tipo de fichero que hay que transferir porque una transferencia de un binario en modo ASCII quedará inutilizada. Para cambiar de un modo a otro, se debe ejecutar el comando *ascii* o *binary*. Comandos útiles del cliente ftp son el *ls* (navegación en el directorio remoto), *get nombre_del_fichero* (para descargar ficheros) o *mget* (que admite *), *put nombre_del_fichero* (para enviar ficheros al servidor) o *mput* (que admite *); en estos dos últimos se debe tener permiso de escritura sobre el directorio del servidor. Se pueden ejecutar comandos locales si antes del comando se inserta un ´!´. Por ejemplo !CD /tmp significará que los archivos que bajen a la máquina local se descargarán en /tmp. Para poder ver el estado y el funcionamiento de la transferencia, el cliente puede imprimir marcas, o *ticks*, que se activan con los comandos *hash* y *tick*. Existen otros comandos que se pueden consultar en la hoja del manual (*man ftp*) o haciendo *help* dentro del cliente.

ANOTACIONES

Contamos con numerosas alternativas para los clientes, por ejemplo en modo texto: ncftp, lukemftp, lftp, cftp, yafc Yafc, o en modo gráfico: gFTP, WXftp, LLNL XFTP, guiftp. [Bor00]

7.4.2. Servidores FTP

El servidor tradicional de UNIX se ejecuta a través del puerto 21 y es puesto en marcha por el *daemon inetd* (es conveniente incluir el *wrapper tcpd* con las reglas de acceso en *host.allow* y el *host.deny* en la llamada al ftpd por el *inetd* para incrementar la seguridad del sistema (podéis consultar la unidad dedicada a la de seguridad) o a través del *xinetd* (recomendable). Cuando recibe una conexión, verifica el usuario y el *password* y lo deja entrar si la autentificación es correcta. Un FTP *anonymous* trabaja en forma diferente, ya que el usuario sólo podrá acceder a un directorio definido en el archivo de configuración y al árbol subyacente, pero no hacia arriba, por motivos de seguridad. Este directorio generalmente contiene directorios pub/, bin/, etc/, y lib/ para que el *daemon* de ftp pueda ejecutar comandos externos para peticiones de *ls*. El *daemon* ftpd soporta los siguientes archivos para su configuración:

- /etc/ftpusers: lista de usuarios que no son aceptados sobre el sistema, un usuario por línea. /etc/ftpchroot: lista de usuarios a los que se les cambiará el directorio base chroot cuando se conecten. Necesario cuando deseamos configurar un servidor anónimo.

- /etc/ftpwelcome: anuncio de bienvenida.

- /etc/motd: noticias después del *login*.

- /etc/nologin: mensaje que se muestra después de negar la conexión.

- /var/log/ftpd: *log* de las transferencias.

Si queremos en algún momento inhibir la conexión al ftp, se puede hacer incluyendo el archivo /etc/nologin. El FTPD muestra su conte-

ANOTACIONES

nido y termina. Si existe un archivo *.message* en un directorio, el ftpd lo mostrará cuando se acceda al mismo.

La conexión de un usuario pasa por cinco niveles diferentes:

1) tener una contraseña válida,

2) no aparecer en la lista de /etc/ftpusers,

3) tener un *shell* estándar válido,

4) si aparece en /etc/ftpchroot, se le cambiará al directorio *home* (incluido si es *anonymous* o ftp),

5) si el usuario es *anonymous* o ftp, deberán tener una entrada en el /etc/passwd con *user ftp*, pero podrán conectarse especificando cualquier *passwd* (por convención se utiliza la dirección de *e-mail*).

Es importante tener en cuenta que los usuarios que sólo estén habilitados para utilizar el servicio ftp no dispongan de un *shell* a la entrada correspondiente de dicho usuario en /etc/passwd para impedir que este usuario tenga conexión, por ejemplo, por ssh o telnet. Para ello, cuando se cree el usuario, habrá que indicar, por ejemplo:

```
useradd-d/home/nteum -s /bin/false nteum
```

y luego:

```
passwd nteum
```

Lo cual indicará que el usuario nteum no tendrá *shell* para una conexión interactiva (si el usuario ya existe, se puede editar el fichero /etc/passwd y cambiar el último campo por /bin/false). Luego se debe agregar como última línea /bin/false en /ect/shells. En [Mou01] se describe paso a paso cómo crear tanto un servidor ftp seguro con usuarios registrados como un servidor ftp *anonymous* para usuarios no registrados.

Dos de los servidores no estándares más comunes son el WUFTPD (http://www.wuftpd.org) y el ProFTPD (http://www.proftpd.org). [Bor00, Mou01]

ANOTACIONES

7.5. Servicios de intercambio de información a nivel de usuario

7.5.1. El *Mail Transport Agent* (MTA)

Un **MTA** (*Mail Trasnport Agent*) se encarga de enviar/recibir los correos desde un servidor de *e-mail* hacia/desde Internet, que implementa el protocolo SMTP (*Simple Mail Transfer Protocol*). Debian utiliza por defecto *exim*, ya que es más fácil de configurar que otros paquetes MTA, como son *smail* o *sendmail* (este último es uno de los precursores). *exim* presenta características avanzadas tales como rechazar conexiones de sitios de SPAM conocidos, posee defensas contra *junk mails* o *mail bombing* y es extremadamente eficiente en el procesamiento de grandes cantidades de correos. Su ejecución se realiza a través de *inetd* en una línea en el archivo de configuración /etc/inetd.conf con parámetros bs para configuraciones normales (o *xinetd*).

exim utiliza un archivo de configuración en /etc/exim/exim.conf, que puede ser modificado manualmente, pero es recomendable hacerlo con un *shell script* llamado *eximconfig*, para poder configurar *exim* en forma interactiva. Los valores de la configuración dependerán de la situación de la máquina; sin embargo, su conexión es sumamente fácil, ya que el mismo *script* sugiere valores por defecto. No obstante, en /usr/doc/exim pueden encontrarse ejemplos de configuración típicas.

Se puede probar si la configuración es válida con *exim -bV* y, si hay errores en el archivo de configuración, el programa los mostrará por pantalla o, si todo está correcto, sólo pondrá la versión y fecha. Para probar si puede reconocer un buzón (*mailbox*) local, utilizad:

```
exim -v -bt usuario_local
```

Donde se mostrarán las capas de transporte utilizadas y la dirección local del usuario. También se puede hacer este test con un usuario remoto reemplazando usuario local por una dirección remota para ver su comportamiento. Luego intentad enviar un correo local y remotamente, pasando directamente los mensajes a *exim* (sin utilizar un agente por ejemplo, *mailx*), tecleando por ejemplo (todo junto):

ANOTACIONES

```
exim postmaster@SuDominio
From: user@dominio
To: postmaster@SuDominio
Subject: Test Exim
Mensaje de prueba
^D
```

A continuación, se pueden analizar los archivos de traza *mainlog* y *paniclog* en /var/log/exim/ para ver su comportamiento y cuáles son los mensajes de error generados. Obviamente, también podéis conectaros al sistema como *postmaster* y leer los correos para ver si todo es correcto. La otra forma consiste en ejecutarlo en modo debug utilizando como parámetro *-dNro*, donde *Nro* es el nivel de *debug* (19). El parámetro normal con el cual se debe poner en marcha es *exim -bs,* ya sea por *inetd* o por *xinetd*. También es posible ejecutarlo como *daemon* a través de */etc/init.d/exim start* en sistemas que necesiten prestaciones elevadas al tratamiento de los correos. Consultar la documentación (incluida en Debian el paquete *exim-doc-html*) para configurar filtros, verificación de *hosts*, de *sender*, etc. Es interesante también instalar el paquete *eximon*, que es un monitor del *exim* y permite al administrador ver la cola de correos, *logs* y realizar diferentes acciones con los mensajes en cola para ser distribuidos (*freezing, bouncing, thawing,* ...).

7.5.2. *Internet Message Access Protocol* POP (IMAP)

Este servicio soportado por el *daemon imapd* (los actuales soportan el protocolo IMAP4rev1) son de utilidad cuando se necesita consultar un archivo de correo electrónico (*mail file*) que se encuentra en una máquina diferente a la máquina desde la cual se accede (remotamente) y a través de un cliente de correo electrónico tal como Mozilla Mail, por ejemplo. El servicio se presta a través de los puertos 143 (*imap2*) o 993 (*imaps*) cuando soporta encriptación por SSL. Si se utiliza *inetd*, este servidor se pone en marcha a través de una línea en /etc/inetd.conf como:

```
imap2 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
imap3 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
```

En este ejemplo se llama al *wrapper tcpd* que funciona con *hosts.allow* y *hosts.deny* para incrementar la seguridad. Las aplicaciones más populares son *uw-imapd* (Universidad de Washington e instalado por defecto en Debian) o su versión segura *uw-imapd-ssl*, *cyrus-imap* o *courier-imap*. Para probar que el servidor *imap* funciona, se podría utilizar un cliente, por ejemplo, *mozilla -mail* y crear una cuenta para un usuario local y configurarlo adecuadamente para que se conecte sobre la máquina local, verificando el funcionamiento de *imap*.

Sobre Debian, la versión de *imap* ha sido compilada para soportar MD5 como método de autentificación de los usuarios remotos, para encriptar los *passwords* de conexión y evitar suplantación de identidad por *sniffing* en la red (el cliente utilizado para conectarse al servidor *imap* también debe soportar el método de autentificación por MD5). El método es muy simple y seguro, pero el servidor debe conocer los *passwords* en texto plano de los usuarios de correo, el cual deberá ser guardado con mucho cuidado, ya que si no, quien pueda acceder a él tendrá todos los *passwords* del sistema de correo de todos los usuarios. Es por ello, por lo que representa un modo más seguro que el de *passwords* imap en texto plano (por el puerto 143), pero no es perfecta. La forma de inicializar el MD5 es utilizando un fichero (que será la base de dato) /etc/cram-md5.pwd y cuidar que las protecciones sean 0400 (es decir, hacer *chmod 0400 /etc/crammd5.pwd*). El formato de este archivo es simple, una línea por cada usuario con el siguiente formato:

```
usuario<tab>password
```

donde las líneas que comienzan por '#' son interpretadas como comentarios. Cada entrada en la base debe tener una entrada válida en el fichero /etc/passwd. Debe tenerse en cuenta que, por seguridad, ambas contraseñas (la del crammd5.pwd y la del /etc/passwd) deben ser diferentes. Del fichero /etc/passwd sólo se obtienen el UID, GID y directorio raíz del usuario.

Por las razones dadas anteriormente es que se utiliza la versión SSL (*Secure Socket Layer*) del protocolo *imap* que al igual que ssh se basa en encriptar la comunicación a través de un certificado del *host* (el cliente utilizado para conectarse al servidor también debe soportar

ANOTACIONES

este método de conexión por ejemplo, *mozilla -mail*). Para configurar el servidor *imaps*, instalad el paquete *uw-imap-dssl* de Debian que es el servidor *imap* con soporte SSL (*imaps* en el puerto 993). La instalación genera un certificado autofirmado válido por un año y lo almacena en /etc/ssl/certs/imapd.pem. Este certificado se puede reemplazar por uno firmado por una compañía certificadora o se puede generar uno propio con OpenSSL. Es conveniente dejar sólo la entrada *imaps* en el archivo /etc/inetd.conf y quitar las entradas *imap2* e *imap3* si únicamente se quiere que el acceso a *imap* sea por SSL.

Otro protocolo de similares características que ha tenido mucha popularidad en el pasado, pero que hoy se ha visto superado por IMAP, es POP (Post Office Protocol) version 2 y 3. Su instalación y puesta en marcha es análoga a la de IMAP. Existen multitud de servidores POP, pero los más comunes son *courier-pop, cyrus-pop3d, ipopd* (Universidad de Washington), *qpopper, solid-pop3d*.

7.5.3. *News*

Las *news* o grupos de discusión son soportados a través del protocolo **NNTP**. Instalar un servidor de *news* es necesario cuando se desea leer *news* fuera de línea, cuando se quiere tener un repetidor de los servidores centrales o se quiere un propio servidor master de *news*. Los servidores más comunes son INN o CNEWS, pero son paquetes complejos y destinados a grandes servidores. Leafnode es un paquete USENET que implementa servidor TNP, especialmente indicado para sitios con grupos reducidos de usuarios, pero donde se desea acceder a gran cantidad de grupos de noticias. Este servidor se instala en la configuración básica de Debian y se puede reconfigurar con *dpkg-reconfigure leafnode*, parámetros como los servidores centrales, el tipo de conexión, etc. Este *daemon* se pone en marcha desde *inetd* en forma similar al *imap* (o con *xinetd*). *Leafnode* soporta filtros a través de expresiones regulares indicadas (del tipo *^Newsgroups:. * [,] alt.flame$*) en /etc/news/leafnode/filters, donde para cada mensaje se compara la cabecera con la expresión regular y, si existe coincidencia, el mensaje es rechazado.

La configuración de este servidor es simple y todos los archivos deben ser propiedad de un usuario *news* y con permiso de escritura (verificar

ANOTACIONES

que dicho propietario existe en /etc/passwd). Todos los archivos de control, *news* y configuración se encuentran en /var/spool/news excepto la configuración del propio servidor que está en el fichero /etc/news/leafnode/config. En la configuración existen algunos parámetros obligatorios que deben ser configurados (por ejemplo, para que el servidor pueda conectarse con los servidores maestros). Ellos son *server* (servidor de *news* desde donde se obtendrán y enviarán las *news*) y *expire* (número de días que un *thread* o sesión ha sido leída y se borrará). Tenemos, asimismo, un conjunto de parámetros opcionales de ámbito general o específicos del servidor que podrían configurarse. Para más información, consultad la documentación (*man leafnode* o /usr/doc/leafnode/README.Debian).

Para verificar el funcionamiento del servidor, se puede hacer:

```
telnet localhost nntp
```

y, si todo funciona correctamente, saldrá la identificación del servidor y se quedará esperando un comando, como prueba, se puede introducir *help* (para abortar, hacer Ctrl+ (y luego Quit).

7.5.4. World Wide Web (httpd)

Apache es uno de los servidores más populares y de altas prestaciones de HTTP (*HyperText Transfer Protocol*). Apache tiene un diseño modular y soporta extensiones dinámicas de módulos durante su ejecución. Es altamente configurable en el número de servidores y de módulos disponibles y soporta diversos mecanismos de autentificación, control de acceso, *metafiles*, *proxy caching* , servidores virtuales, etc. Con módulos (incluidos en Debian) es posible tener PHP3, Perl, Java Servlets, SSL y otras extensiones (podéis consultar la documentación en http://www.apache.org).

Apache está diseñado para ejecutarse como un proceso *daemon* standalone. En esta forma crea un conjunto de procesos hijo que manejarán las peticiones de entrada. También puede ejecutarse como Internet *daemon* a través de *inetd,* por lo cual se pondrá en marcha cada vez que se reciba una petición. La configuración del servidor pue-

ANOTACIONES

de ser extremadamente compleja según las necesidades (consultad la documentación), sin embargo, aquí veremos una configuración mínima aceptable. Los archivos de configuración se encuentran en /etc/apache y son httpd.conf (archivo principal de configuración), *srm.conf*, *access.conf* (estos dos últimos son mantenidos por compatibilidad y su funcionalidad está en el anterior), *mime.conf* (formatos MIME) y *magic* (número de identificación de archivos). Los archivos *log* se encuentran en /var/log/apache y son *error.log* (registra los errores en las peticiones del servidor), *access.log* (registro de quién y a qué ha accedido) y *apache.pid* (*pid* del proceso).

Apache se pone en marcha desde el *script* de inicio /etc/init.d/apache y los /etc/rcX.d, pero puede controlarse manualmente mediante el comando *apachectl*. También se puede utilizar el comando *apacheconfig* para configurar el servidor. Los directorios por defecto (en Debian) son:

- /var/www: directorio de documentos HTML.
- /usr/lib/cgibin: directorio de ejecutables (*cgi*) por el servidor.
- http://server.dominio/ user: páginas personales de los usuarios.
- /home/user/public.html: directorio de páginas personales.

El archivo por defecto que se lee de cada directorio es *index.html*.

Una vez instalados los paquetes *apache* y *apache-common*, Debian configura básicamente el servidor y lo pone en marcha. Se puede comprobar que funciona abriendo un *browser* (por ejemplo, el Konqueror, y poniendo en la barra de URL http://localhost, lo cual cargará la página /var/www/index.html).

Configuración manual de httpd.conf

Vamos a ver algunos de los parámetros más importantes en la configuración de Apache (el ejemplo está tomado de la versión 1.3 de Apache y existen algunos cambios menores si se utiliza la versión 2).

ANOTACIONES

`ServerType standalone`	Recomendado, más eficiente
`ServerRoot /etc/apache`	Donde están los archivos de configuración
`Port 80`	Donde el servidor escuchará las peticiones
`User www-data`	*User* y *group* con los cuales se ejecutará el servidor (importante por seguridad) deben ser usuarios válidos (pueden estar *locked*)
`Group www-data`	
`ServerAdmin webmaster@pirulo.remix.com`	Dirección de usuario que atenderá los errores
`ServerName pirulo.remix.com`	Nombre del servidor enviado a los usuarios -debe ser un nombre válido en /etc/host o DNS-
`DocumentRoot /var/www`	Directorio donde estarán los documentos
`Alias /icons/ /usr/share/apache/icons/`	Donde se encuentran los iconos
`ScriptAlias /cgibin/ /usr/lib/cgibin/`	Donde se encuentran los *script* CGI

7.6. Servicio de Proxy: Squid

Un servidor Proxy (PS) se utiliza para salvar ancho de banda de la conexión de red, mejorar la seguridad e incrementar la velocidad para obtener páginas de la Red (*web-surfing*).

Squid es uno de los principales PS, ya que es OpenSource, acepta ICP (características que le permiten intercambiar *hints* con otros PS), SSL (para conexiones seguras entre *proxies*) y soporta objetos FTP, Gopher, HTTP y HTTPS (seguro). Su funcionamiento es simple, almacena los objetos más solicitados en memoria RAM y los menos en una base de datos en el disco. Los servidores Squid, además, pueden configurarse de forma jerárquica para formar un árbol de *proxies* dependiendo de las necesidades. Existen dos configuraciones posibles:

1) Como acelerador de httpd para lograr más prestaciones al servicio de web,

2) Como *proxy-caching server* para permitir a los usuarios de una corporación utilizar el PS para salir hacia Internet.

En el primer modo, actúa como *proxy* inverso, es decir, acepta una petición del cliente, sirve el objeto si lo tiene y si no, lo solicita y se lo pasa

ANOTACIONES

al cliente cuando lo tiene, almacenándolo para la vez siguiente. En la segunda opción se puede utilizar como control y para restringir los sitios donde se puede conectar en Internet o autorizar el acceso a determinadas horas del día. Una vez instalado (paquete *squid* en Debian, también se puede instalar *squid-cgi*, *squidguard* o *squidtaild*) se generan tres archivos: /etc/squid.conf (configuración), /etc/init.d/squid (inicialización) y /etc/logrotate.d/squid (de control de los *logs*).

7.6.1. Squid como acelerador de http

En este modo, si el servidor de web está en la misma máquina donde está el PS, se deberá reconfigurar para que atienda peticiones del puerto 81 (en Apache, cambiar *Port 80* por *Port 81* en *httpd.conf*). El archivo de configuración (/etc/squid.conf) contiene una gran cantidad de entradas, pero aquí solo veremos las indispensables [Mou01]:

```
http_port 80                    Donde escucha httpd
icp_port 0                      Donde escucha ICP
hierarchy_stoplist cgi-bin \?
acl QUERY urlpath_regex cgi-bin \?
no_cache deny QUERY
cache_mem 100 MB                Memoria para objetos en curso
redirect_rewrites_host_header off
cache_replacement_policy lru
memory_replacement_policy lru
cache_dir ufs /var/spool/squid 100 16 256  Tipo y lugar donde está la Base de Datos de
                                           caché de disco
emulate_httpd_log on
acl all src 0.0.0.0/0.0.0.0     Acceso para todos
http_access allow all           y a todo
cache_mgr root                  Mail responsable
cache_effective_user proxy      UID
cache_effective_group proxy     GID
httpd_accel_host 192.168.1.1    Servidor real de Web
httpd_accel_port 81             Puerto
logfile_rotate 0
log_icp_queries off
buffered_logs on
```

De este modo, la opción *httpd_accel_host* desactiva la posibilidad de que se ejecute como *proxy-caching*.

7.6.2. Squid como *proxy-caching*

De esta manera se habilita el *squid* para que controle el acceso a Internet, cuándo accederán y a qué accederán. En este caso, el archivo de configuración deberá incluir la siguientes modificaciones/agregados en /etc/squid.conf:

```
acl localnet src 192.168.1.0/255.255.255.0
acl localhost src 127.0.0.1/255.255.255.255
acl Safe_ports port 80 443 210 70 21 102565535
acl CONNECT method CONNECT
acl all src 0.0.0.0/0.0.0.0
http_access allow localnet
http_access allow localhost
http_access deny !Puerto seguros
http_access deny CONNECT
http_access deny all
cache_emulate_httpd_log on
```

La gran diferencia con el otro modo son las líneas *acl*, donde, en este caso, se permitirá a los clientes de la clase C 192.168.1.0 acceder al PS, también el *localhost* IP y otros puertos que podrán acceder a Internet 80= http, 443 = https, 210 = wais, 70 = gopher, and 21 = ftp, además, se niega el método *connect* para evitar que desde fuera se puedan conectar al PS y luego se niegan todos los IP y puertos sobre el PS. [Mou01]

7.7. OpenLdap (Ldap)

LDAP significa *Lightweight Directory Access Protocol* y es un protocolo para acceder a datos basados en un servicio X.500. Éste se ejecuta sobre TCP/IP y el directorio es similar a una base de datos que contiene informa-

ANOTACIONES

ción basada en atributos. El sistema permite organizar esta información de manera segura y utilizando réplicas para mantener su disponibilidad, asegurando la coherencia y la verificación sobre los datos accedidos-modificados.

El servicio se basa en el modelo cliente-servidor, donde existe un servidor o más de uno que contienen los datos; cuando un cliente se conecta y solicita información, el servidor responde con los datos o un puntero a otro servidor donde podrá extraer más información, pero el cliente sólo verá un directorio de información global. [Mou01, Mal03]

Para importar y exportar información entre servidores ldap, o para describir una serie de cambios que serán aplicados al directorio, el formato utilizado se llama LDIF (LDAP *Data Interchange Format*). LDIF almacena la información en jerarquías orientadas a objetos que luego serán transformadas al formato interno de la base de datos. Un archivo LDIF tiene un formato similar a:

```
dn: o = UOC, c = SP
o: UOC
objectclass: organization
dn: cn = Pirulo Nteum, o = UOC, c = SP
cn: Pirulo Nteum
sn: Nteum
mail: nteum@yahoo.com
objectclass: person
```

Cada entrada es identificada por un nombre indicado como DN (*Distinguished Name*). El DN consiste en el nombre de la entrada más una serie de nombres que lo relacionan con la jerarquía del directorio y donde existe un *objectclass* que define los atributos que pueden ser utilizados en esta entrada. LDAP provee un conjunto básico de clases de objetos: *grupos* (incluye listas desordenadas de objetos individuales o grupos de objetos), *localizaciones* (tales como países y su descripción), *organizaciones* y *personas*. Una entrada puede, además, pertenecer a más de una clase de objeto, por ejemplo, un individuo es definido por

ANOTACIONES

la clase *persona*, pero también puede ser definido por atributos de las clases *inetOrgPerson*, *groupOfNames*, y organization. La estructura de objetos del servidor (llamado *schema*) determina cuáles son los atributos permitidos para un objeto de una clase (los cuales se definen en /etc/ldap/schema como *opeldap.schema, corba.schema, nis.schema, inetorgperson. schema*, etc.).

Todos los datos son representados como un par *atributo = valor* donde atributo es descriptivo de la información que contiene, por ejemplo, el atributo utilizado para almacenar el nombre de una persona es *commonName*, o *cn*, es decir, para una persona llamada Pirulo, Nteum, será representado por *cn: Pirulo Nteum* y llevará asociado otros atributos de la clase persona como *givenname: Pirulo surname: Nteum mail: pirulo@europe.com*. En las clases existen atributos obligatorios y optativos y cada atributo tiene una sintaxis asociada que indica qué tipo de información contiene el atributo, por ejemplo, *bin* (*binary*), *ces* (*case exact string*, debe buscarse igual), *cis* (*case ignore string*, puede ignorarse M-m durante la búsqueda), *tel* (*telephone number string*, se ignoran espacios y '-'), *dn* (*distinguished name*). Un ejemplo de un archivo en formato LDIF podría ser:

```
dn:           dc = UOC, dc = com
objectclass:  top
objectclass:  organizationalUnit

dn:           ou = groups, dc = UOC, dc = com
objectclass:  top
objectclass:  organizationalUnit
ou:           groups

dn:           ou = people, dc = UOC, dc = com
objectclass:  top
objectclass:  organizationalUnit
ou:           people
dn:           cn = Pirulo Nteum, ou = people, dc = UOC, dc = com
cn:           Pirulo Nteum sn: Nteum
objectclass:  top
objectclass:  person
objectclass:  posixAccount
objectclass:  shadowAccount
```

```
  uid:pirulo
  userpassword:{crypt}p1pss2ii(0pgbs*do&@ = )eksd
  uidnumber:104
  gidnumber:100
  gecos:Pirulo Nteum
  loginShell:/bin/bash
  homeDirectory: /home/pirulo
  shadowLastChange:10877
  shadowMin: 0
  shadowMax: 999999
  shadowWarning: 7
  shadowInactive: -1
  shadowExpire: -1
  shadowFlag: 0

dn:          cn = unixgroup, ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: posixGroup
cn: unixgroup
  gidnumber: 200
  memberuid: pirulo
  memberuid: otro-usuario
```

Las líneas largas pueden ser continuadas debajo comenzando por un espacio o un tab (formato LDIF). En este caso, se ha definido la base DN para la institución *dc = UOC*, *dc = com*, la cual contiene dos subunidades: *people* y *groups*. Luego se ha descrito un usuario que pertenece a *people* y a *group*. Una vez preparado el archivo con los datos, debe ser importado al servidor para que esté disponible para los clientes LDAP. Existen herramientas para transferir datos de diferentes bases de datos a formato LDIF. [Mal03]

Sobre Debian, se debe instalar el paquete **slapd** que es el servidor de OpenLdap. Durante la instalación realizará una serie de preguntas como: *Método de instalación del directorio:* auto; *extensiones al directorio [domain-host,sitio,institución]: host, domain, password del Adm; replicar cambios locales a otros servidores:* no. Esta instalación generará un archivo de configuración en /etc/ldap/slapd.conf y la base de datos sobre /var/lib/ldap. También existe otro archivo /etc/ldap/ldap.conf (o puede

ANOTACIONES

existir el ~/.ldaprc), que es el archivo de configuración utilizado para inicializar valores por defecto cuando se ejecutan clientes *ldap*. En éste se indica cuál es la base de datos, cuál es el servidor ldap, parámetros de seguridad, tamaño de la búsqueda, etc.

El archivo de configuración del servidor /etc/ldap/slapd.conf (ver *man slap.con*) está compuesto por diferentes secciones, cada una de ellas indicada por una de las siguientes directivas: *global, backend specific* y *database specific,* y en ese orden. La directiva *global* es de carácter general y se aplica a todos los *backends* (bases de datos) y definen cuestiones generales tales como los permisos de acceso, atributos, tiempos de espera, *schemas* , etc. La directiva *backend specific* define los atributos al *backend* específico que define (*bdb*, *dnssrv*, *ldbm*, ...), y el *database specific* los atributos específico para esa base de datos que define. Para poner en marcha el servidor, ejecutad:

`/etc/init.d/slapd start` (o `stop` para pararlo)

El sistema durante la instalación habrá creado los en laces adecuados para ejecutarlo después del inicio.

7.7.1. Creación y mantenimiento de la base de datos

Existen dos métodos para insertar datos en la base de datos de LDAP. El primero es fácil y adecuado para pequeñas cantidades de datos, es interactivo y se deben utilizar herramientas tales como ldapadd (o cualquier otra como Ldap Browser http://www.iit.edu/~gawojar/ldap/) para insertar nuevas entradas. El segundo se debe trabajar fuera de línea, es el adecuado para grandes BD y se utiliza el comando *slapadd* incluido con *slapd*. Por ser más general, describiremos sintéticamente el segundo método, donde primero se debe verificar que contiene los siguientes atributos en *slapd.conf*: *suffix* (*top* del directorio, por ejemplo, *suffix "o = UOC, c = SP"*); *directory /var/lib/ldap* (directorio donde se crearán los índices y que pueda escribir *slapd*). Se debe además verificar que la base de datos contiene las definiciones de los índices que se desean:

```
index cn,sn,uid
index objectClass pres,eq
```

ANOTACIONES

Una vez definido el *slapd.conf*, se debe ejecutar el comando:

```
slapadd -l entrada -f configuración [-d nivel] [-n entero | -b sufijo]
```

Los argumentos son:

- *-l:* archivo en formato LDFI.
- *-f:* archivo de configuración del servidor, donde se indican cómo crear los índices.
- *-d:* nivel de depuración.
- *-n:* Nro de base de datos, si se tiene más de una.
- *-b:* especifica qué base de datos hay que modificar.

Existen otros comandos con *slapd* tales como *slapindex*, que permite regenerar los índices, y *slapcat*, que permite volcar la BD a un archivo en formato LDIF.

7.8. Servicios de archivos (NFS)

El sistema NFS permite a un servidor exportar un sistema de archivo para que puedan ser utilizados en forma interactiva desde un cliente. El servicio se compone de un servidor nfsd y un cliente (*mountd*) que permiten compartir un sistema de archivo (o parte de él) a través de la red.

El servidor (en Debian) se pone en marcha a través de los *scripts* *nfscommon* y *nfs-kernel-server* en /etc/init.d (y los enlaces adecuados en /etc/rcX.d).

El servidor utiliza un archivo (/etc/exports) para gestionar el acceso y control sobre los sistemas de archivo que se accederán remotamente. Sobre el cliente, el *root* (u otro usuario a través de *sudo*) puede montar el sistema remoto a través del comando:

```
mount IP server: filesystem_origen directorio_local
```

ANOTACIONES

y a partir de este momento el *filesystem_origen* se verá dentro de directorio local (éste debe existir antes de ejecutar el *mount*). Esta tarea en el cliente se puede automatizar utilizando el archivo de *mount* automático (/etc/fstab) incluyendo una línea; por ejemplo:

```
pirulo.remix.com:/usr/local /pub nfs rsize = 8192,wzise = 8192,timeo = 14
```

Esta sentencia indica que se montará el directorio /usr/local del *host pirulo.remix.com* en el directorio local /pub. Los parámetros *rsize, wzise* son los tamaños de bloques de lectura y escritura, *timeo* es el *timeout* de RPC (si no se especifican estos tres valores, se toman los que incluye por defecto).

El archivo /etc/exports sirve de ACL (lista de control de acceso) de los sistemas de archivo que pueden ser exportados a los clientes. Cada línea contiene un *filesystem* por exportar seguido de los clientes que lo pueden montar, separados por espacios en blanco. A cada cliente se le puede asociar un conjunto de opciones para modificar el comportamiento (consultar *man exports* para un lista detallada de las opciones). Un ejemplo de esto podría ser:

```
# Ejemplo de /etc/exports
  /           /master(rw) trusty(rw,no_root_squash)
  /projects   proj*.local.domain(rw)
  /usr        *.local.domain(ro) @trusted(rw)
  /pub        (ro,insecure,all_squash)
```

La primera línea exporta el sistema de archivos entero (/) a *master* y *trusty* en modo lectura/escritura. Además, para *trusty* no hay *uid squashing* (el *root* del cliente accederá como *root* a los archivos *root* del servidor, es decir los dos *root* son equivalentes a pesar de ser de máquinas diferentes, es indicado para máquinas sin disco). La segunda y tercera líneas muestra ejemplo de '*' y de *netgroups* (indicados por @). La cuarta línea exporta el directorio /pub a cualquier máquina en el mundo, sólo de lectura, permite el acceso de clientes NFS que no utilizan un puerto reservado para el NFS (opción *insecure*) y todo se ejecuta bajo el usuario *nobody* (opción *all squash*).

ANOTACIONES

7.9. Actividades para el lector

1) Configurar un servidor DNS como caché y con un dominio propio.

2) Configurar un servidor/cliente NIS con dos máquinas exportando los directorios de usuario del servidor por NFS.

3) Configurar un servidor SSH para acceder desde otra máquina sin *passwd*.

4) Configurar un servidor Apache para visualizar las hojas personales de los usuarios.

5) Crear y configurar un sistema de correo electrónico a través de Exim y un servidor IMAP para recibir correos desde el exterior y poder leerlos desde una máquina remota con el cliente Mozilla Mail.

Nota

Otras fuentes de referencias e información:

[Deb03c, LPD03b, Ibi03]

ANOTACIONES

8. Administración de datos

Un aspecto importante de un sistema operativo es dónde y cómo se guardan los datos. Cuando la disponibilidad de estos datos debe ser eficiente, es necesario la utilización de bases de datos (DB).

Una **base de datos** es un conjunto estructurado de datos que pueden ser organizados de manera simple y eficiente por un manejador de dicha base. Las bases de datos actuales se denominan **relacionales**, ya que los datos pueden ser almacenados en diferentes tablas que facilitan su gestión y administración. Para ello y con el fin de estandarizar el acceso a las bases de datos se utiliza un lenguaje denominado **SQL** (*Structured Query Language*), que permite una interacción flexible, rápida e independiente de las aplicaciones a las bases de datos.

La forma más utilizada en la actualidad consiste en acceder a una base de datos desde una aplicación que ejecuta código SQL. Por ejemplo, es muy común acceder a una DB a través de una página web que contiene código PHP o Perl (los más comunes). Cuando se solicita una página por un cliente, se ejecuta el código PHP, Perl incrustado en la página, se accede a la DB y se genera la página con su contenido estático y el contenido extraído de la DB que posteriormente se envía al cliente. Dos de los ejemplos más actuales de bases de datos son los aportados por ProstgreSQL y MySQL, que serán objeto de nuestro análisis.

Sin embargo, cuando se trabaja en el desarrollo de un software, existen otros aspectos relacionados con los datos, que son su validez y su ámbito (sobre todo si existe un conjunto de usuarios que trabajan sobre los mismos datos). El RCS (*Revision Control System*) y el CVS (*Control Version System*) controlan y administran múltiples revisiones de archivos, automatizando el almacenamiento, lectura, identificación y mezclado de diferentes revisiones. Estos programas son útiles cuando un texto es revisado frecuentemente e incluye código fuente, ejecutables, bibliotecas, documentación, gráficos, artículos y otros archivos. [Pos03e, Mys03, Ced97]

ANOTACIONES

8.1. PostgreSQL

En el lenguaje de bases de datos (DB), PostgreSQL utiliza un modelo cliente servidor [Pos03a, Pos03b, Pos03c]. Una sesión de PostgreSQL consiste en una serie de programas que cooperan:

- Un proceso servidor que maneja los archivos de la DB acepta conexiones de los clientes y realiza las acciones solicitadas por éstos sobre la DB. El programa servidor es llamado en PostgreSQL *postmaster*.

- La aplicación del cliente (*frontend*) es la que solicita las operaciones que hay que realizar en la DB y que pueden ser de lo más variadas; por ejemplo: herramientas en modo texto, gráficas, servidores de web, etc.

Generalmente, el cliente y el servidor se encuentran en diferentes *hosts* y se comunican a través de una conexión TCP/IP. El servidor puede aceptar múltiples peticiones de diferentes clientes, activando para cada nueva conexión un proceso que lo atenderá en exclusiva de un modo transparente para el usuario. Existe un conjunto de tareas que pueden ser llevadas a cabo por el usuario o por el administrador, según convenga, y que pasamos a describir a continuación.

8.1.1. ¿Cómo se debe crear una DB?

La primera acción para verificar si se puede acceder al servidor de DB es crear una base de datos. El servidor PostgreSQL puede manejar muchas DB y es recomendable utilizar una diferente para cada proyecto. Para crear una base de datos, se utiliza el comando *createdb* desde la línea de comandos del sistema operativo. Este comando generará un mensaje *CREATE DATABASE* si todo es correcto. Es importante tener en cuenta que para esta acción debe haber un usuario habilitado para crear una base de datos. Se verá en el apartado de instalación (8.1.4), que existe un usuario, el que instala la base de datos, que tendrá permisos para crear bases de datos y crear nuevos usuarios que a su vez puedan crear bases de datos. Generalmente (y en Debian) este usuario es *postgres* por

ANOTACIONES

defecto. Por ello, antes de hacer el *createdb*, se debe hacer un *postgres* (si se es *root,* no es necesaria ninguna palabra clave, pero si se es otro usuario, necesitaremos la palabra clave de *postgres*) y luego se podrá realizar el *createdb*.

Ejemplo

Crear DB nteumdb:

```
createdb nteumdb
```

Si no encontráis el comando, puede ser que no esté bien configurado el camino o que la DB esté mal instalada. Se puede intentar con todo el camino (/usr/local/pgsql/bin/createdb nteumdb), que dependerá de la instalación que se haya hecho, o consultar referencias para la solución de problemas. Otros mensajes de error serían *could not connect to server* cuando el servidor no está arrancado o *CREATE DATABASE: permission denied* cuando no se tienen privilegios para crear la DB. Para eliminar la base de datos, se puede utilizar *dropdb nteumdb*.

8.1.2. ¿Cómo se puede acceder a una DB?

Una vez creada la DB, se puede acceder a ella de diversas formas:

- Ejecutando un comando interactivo llamado *psql*, que permite editar y ejecutar comandos SQL.
- Ejecutando una interfaz gráfica como PgAccess o alguna *suite* que tenga soporte ODBC para crear y manipular DB.
- Escribiendo un aplicación utilizando algunos de los lenguajes soportados, por ejemplo, PHP, Perl, Java, ... (ver PostgreSQL 7.3 Programmer's Guide).

Nota

Para poder acceder a la DB, el servidor de base de datos deberá estar en funcionamiento. Cuando se instala Postgresql se crean los enlaces adecuados para que el servidor se inicie en el arranque del ordenador. Para más detalles, consultad el apartado de instalación (8.1.4.).

Por simplicidad utilizaremos *psql* para acceder a la DB, por lo que se deberá introducir *psql nteumdb*: saldrán unos mensajes con la versión e información y un *prompt* similar a *nteumdb* ⇒. Se pueden ejecutar algunos de los comandos SQL siguientes:

`SELECT version();` o también `SELECT current date;`

ANOTACIONES

psql también tienen comandos que no son SQL y comienzan por ´\´ por ejemplo \h (lista tsodos los comandos disponibles) o \q para terminar.

Ejemplo

Acceder a la DB nteumdb:

```
psql nteumdb ↵
nteumdb ⇒
```

8.1.3. El lenguaje SQL

No es la finalidad de este apartado hacer un tutorial sobre SQL, pero se analizarán unos ejemplos para ver las capacidades de este lenguaje. Son ejemplos que vienen con la distribución de PostgreSQL en el directorio DirectorioInstalacion/src/tutorial, para acceder a ellos, cambiad al directorio de PostgreSQL (cd DirectorioInstalación/*src*/tutorial) y ejecutad *psql -s nteumdb* y después, dentro \i basics.sql. El parámetro \i lee los comandos del archivo especificado (*basic.sql* en nuestro caso).

PostgreSQL es una base de datos relacional (*Relational Database Management System*, RDBMS), lo cual significa que maneja los datos almacenados en tablas. Cada tabla tiene un número determinado de filas y de columnas y cada columna tiene un tipo específico de datos. La tablas se agrupan en una DB y un único servidor maneja esta colección de DB (todo el conjunto se denomina *agrupación de bases de datos – database cluster*).

Para crear, por ejemplo, una tabla con *psql*, ejecutad:

```
CREATE TABLE tiempo (
   ciudad      varchar(80),
   temp_min    int,
   temp_max    int,
   lluvia      real,
   dia         date
   );
```

ANOTACIONES

Ejemplo

Crear tabla. Dentro de *psql*:

```
CREATE TABLE NombreTB (var1 tipo,
   var2 tipo,...);
```

El comando termina cuando se pone ';' y se pueden utilizar espacios en blanco y *tabs* libremente. *varchar(80)* especifica una estructura de datos que puede almacenar hasta 80 caracteres (en nuestro caso).

Ejemplo

Un segundo ejemplo podría ser:

```
CREATE TABLE ciudad (
   nombre    varchar(80),
   lugar     point
);
```

El *point* es un tipo específico de PostgreSQL. Para borrarlo:

```
DROP TABLE nombre_tabla;
```

Para introducir datos, se pueden utilizar dos formas, la primera es poniendo todos los datos de la tabla y la segunda, indicando las variables y los valores que se desean modificar:

```
INSERT INTO tiempo VALUES ('Barcelona', 16, 37, 0.25, '2003-08-11');
INSERT INTO tiempo (ciudad, temp_min, temp_max, lluvia, dia) VALUES
     ('Barcelona', 16, 37, 0.25, '2003-08-11');
```

Esta forma puede ser sencilla para unos pocos datos, pero cuando hay que introducir gran cantidad de datos, se pueden copiar desde un archivo con la sentencia:

`COPY tiempo FROM '/home/user/tiempo.txt';` (este archivo debe estar en el servidor, no en el cliente).

Para mirar una tabla, podríamos hacer:

```
SELECT * FROM tiempo;
```

donde el * significa todas las columnas.

Nota

Se recomienda ver el capítulo 3 de PostgreSQL 7.3 sobre características avanzadas (*Views, Foreign Keys, Transactions, Inheritance*). [Pos03d]

ANOTACIONES

Ejemplo

Introducir datos en tabla

Dentro de *psql*:

```
INSERT INTO NombreTB (valorVar1,
  ValorVar2,...);
```

Datos desde un archivo

Dentro de *psql*:

```
COPY NombreTB FROM 'NombreArchivo';
```

Ejemplo

Visualizar datos. Dentro de *psql*:

```
SELECT * FROM NombreTB;
```

Ejemplo

Ejemplos de comandos más complejos serían:

- Visualiza la columna ciudad después de realizar la operación:

```
SELECT ciudad, (temp_max+temp_min)/2 AS
  temp_media, date FROM tiempo;
```

- Visualiza todo donde se cumple la operación lógica:

```
SELECT * FROM tiempo WHERE
  city = 'Barcelona'
  AND lluvia \verb+>+ 0.0;
```

- Unión de tablas:

```
SELECT * FROM tiempo, ciudad WHERE
  ciudad = nombre;
```

- Funciones, máximo en este caso:

```
SELECT max(temp_min) FROM tiempo;
```

ANOTACIONES

- Funciones anidadas:

```
SELECT ciudad FROM tiempo WHERE
  temp_min = (SELECT max(temp_min)
  FROM tiempo);
```

- Modificación selectiva:

```
UPDATE tiempo SET temp_max = temp_max 2,
  temp_min = temp_min 2 WHERE
  dia > '19990128';
```

- Borrado del registro:

```
DELETE FROM tiempo WHERE
  ciudad = 'Sabadell';
```

8.1.4. Instalación PostgreSQL

Este paso es necesario para los administradores de la DB [Pos03a]. Dentro de las funciones del administrador de DB se incluye en la instalación del software, inicialización y configuración, administración de los usuarios, DBs y tareas de mantenimiento de la DB.

La instalación de la base de datos se puede realizar de dos modos, a través de los binarios de la distribución, lo cual no presenta ninguna dificultad, ya que los *scripts* de distribución realizan todos los pasos necesarios para tener la DB operativa, o a través del código fuente, que será necesario compilar e instalar. En el primer caso, se puede utilizar (Debian) el *kpackage* o el *apt-get*. Para el segundo caso, se recomienda siempre ir al origen (o a un repositorio espejo de la distribución original). Es importante tener en cuenta que la instalación desde el código fuente quedará luego fuera de la DB de software instalado y se perderán los beneficios de administración de software que presente por ejemplo *apt-cache* o *apt-get*.

Instalación desde el código fuente paso a paso

- Primero se debe obtener el software del sitio:

  ftp://postgresql.org/pub/postgresql-7.3.tar.gz

ANOTACIONES

Y se descomprime:

```
gunzip postgresql-7.3.tar.gz
tar xf postgresql-7.3.tar
```

- Cambiarse al directorio *postgresql* y configurarlo con *./configure*.

- Compilarlo con *gmake*, verificar la compilación con *gmake check* e instalarlo con*gmake install* (por defecto, lo hará en /usr/local/pgsql).

Postinstalación

Inicializar las variables, en *bash*, *sh*, *ksh*:

```
LD_LIBRARY_PATH = /usr/local/pgsql/lib;
PATH = /usr/local/pgsql/bin:$PATH;
export LD_LIBRARY_PATH PATH;
```

o bien, en *csh*:

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib;
set path = (/usr/local/pgsql/bin $path)
```

Es recomendable poner esta inicialización en los *scripts* de configuración del usuario, por ejemplo /etc/profile o .bashrc para el *bash*. Para tener acceso a los manuales, se debe inicializar la variable MANPATH de la misma forma:

```
MANPATH = /usr/local/pgsql/man:$MANPATH; export MANPATH
```

- Una vez instalada la DB, se deberá crear un usuario que manejará las bases de datos (es conveniente crear un usuario diferente del *root* para que no tenga conexión con otros servicios de la máquina), por ejemplo, el usuario postgres utilizando el comando *useradd*, por ejemplo.

- A continuación, se debe crear un área de almacenamiento para las bases de datos (espacio único) sobre el disco que será un di-

ANOTACIONES

rectorio, por ejemplo /usr/local/pgsql/data. Para ello, ejecutar el comando *initdb -D /usr/local/pgsql/data*, conectado como el usuario creado en el punto anterior. Puede recibir un mensaje que no puede crear el directorio por falta de privilegios, por lo cual se deberá crear el directorio primero y luego indicarle a la DB cuál es; como *root,* hay que hacer, por ejemplo:

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su postgres
initdb -D /usr/local/pgsql/data
```

- Iniciar el servidor (que se llama *postmaster*), para ello, utilizar:

```
postmaster -D /usr/local/pgsql/data
```

para ejecutarla en modo activo (*foreground*); y para ejecutarlo en modo pasivo (*background*) utilizar:

```
postmaster -D /usr/local/pgsql/data > logfile 2>&1 &.
```

Las redirecciones se hacen para almacenar los errores del servidor. El paquete también incluye un *script* (**pg_ctl**) para no tener que conocer toda la sintaxis de postmaster para ejecutarlo:

```
/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data
```

- Para abortar la ejecución del servidor, se puede hacer de diferentes formas, con el *pg-ctl*, por ejemplo, o bien directamente con:

```
kill -INT 'head -1 /usr/local/pgsql/data/postmaster.pid'
```

Usuarios de DB

Los usuarios de la DB son completamente distintos de los usuarios del sistema operativo. En algunos casos podría ser interesante mantener una correspondencia, pero no es necesario. Los usua-

Nota

Crear, borrar usuarios:

`createuser [ opciones] nombre`

`dropuser [opciones] nombre`

ANOTACIONES

rios son para todas las DB que controla dicho servidor, no para cada DB. Para crear un usuario, ejecutad la sentencia:

```
SQL CREATE USER nombre
```

Para borrar usuarios:

```
DROP USER nombre
```

También se puede llamar a los programas *createuser* y *dropuser* desde la línea de comandos. Existe un usuario por defecto llamado **postgres** (dentro de la DB), que es el que permitirá crear los restantes (para crear nuevos usuarios desde *psql -U postgres* si el usuario de sistema operativo con el que se administra la DB no es *postgres*).

Un usuario de DB puede tener un conjunto de atributos en función de lo que puede hacer:

- **Superusuario**: este usuario no tiene ninguna restricción. Por ejemplo, podrá crear nuevos usuarios; para ello, ejecutar:

  ```
  CREATE USER nombre CREATEUSER
  ```

- **Creador de DB**: tiene permiso para crear DB. Para crear un usuario de estas características utilizar el comando:

  ```
  CREATE USER nombre CREATEDB
  ```

- *Password*: sólo es necesario si por cuestiones de seguridad se desea controlar el acceso de los usuarios cuando se conecten a una DB. Para crear un usuario con contraseña, se puede utilizar:

  ```
  CREATE USER nombre PASSWORD 'palabra_clave'
  ```

  donde *palabra_clave* será la clave para ese usuario.

ANOTACIONES

A un usuario se le pueden cambiar los atributos utilizando el comando *ALTER USER*. También se pueden hacer grupos de usuarios que compartan los mismos privilegios con:

```
CREATE GROUP NomGrupo
```

Y para insertar usuarios en este grupo:

```
ALTER GROUP NomGrupo ADD USER Nombre1
```

O para borrar:

```
ALTER GROUP NomGrupo DROP USER Nombre1
```

Ejemplo

Operaciones con grupo dentro de **psql**:

```
CREATE GROUP NomGrupo;
ALTER GROUP NomGrupo ADD USER Nom1,...;
ALTER GROUP NomGrupo DROP USER Nom1,...;
```

Cuando se crea una DB, los privilegios son para el usuario que la crea (y para el *superuser*). Para permitir que otro usuario utilice esta DB o parte de ella, se le deben conceder privilegios. Hay diferentes tipos de privilegios como *SELECT*, *INSERT*, *UPDATE*, *DELETE*, *RULE*, *REFERENCES*, *TRIGGER*, *CREATE*, *TEMPORARY*, *EXECUTE*, *USAGE*, y *ALL PRIVILEGES* (consultad las referencias para ver su significado). Para asignar los privilegios, se puede utilizar:

```
GRANT UPDATE ON objeto TO usuario
```

donde *usuario* deberá ser un usuario válido de PostgerSQL y *objeto*, una tabla, por ejemplo. Este comando lo deberá ejecutar el superusuario o el dueño de la tabla. El usuario *PUBLIC* puede ser utilizado como sinónimo de todos los usuarios y *ALL* como sinónimo de todos los privilegios. Por ejemplo, para quitar todos los privilegios a todos los usuarios de objeto, se puede ejecutar:

```
REVOKE ALL ON objeto FROM PUBLIC;
```

ANOTACIONES

8.1.5. Mantenimiento

Hay un conjunto de tareas que son responsabilidad del administrador de DB y que se deben realizar periódicamente:

1) **Recuperar el espacio**: se deberá ejecutar periódicamente el comando *VACUUM*, el cual recuperará el espacio de disco de filas borradas o modificadas, actualizará las estadísticas utilizadas por el planificador de PostgreSQL y mejorará las condiciones de acceso.

2) **Reindexar**: PostgreSQL en ciertos casos puede dar algunos problemas con la reutilización de los índices, por ello es conveniente utilizar *REINDEX* periódicamente para eliminar páginas y filas. También se puede utilizar **contrib/reindexdb** para reindexar una DB entera (se debe tener en cuenta que dependiendo del tamaño de las DB estos comandos pueden tardar un cierto tiempo).

3) **Cambio de archivos *log***: se debe evitar que los archivos de *log* sean de tamaño muy grande y difíciles de manejar. Se puede hacer fácilmente cuando se inicia el servidor con:

   ```
   pg_ctl start | logrotate
   ```

   logrotate renombra y abre un nuevo archivo de *log* y se puede configurar con /etc/logrotate.conf.

4) **Copia de seguridad y recuperación** (*backup* y *recovery*): existen dos formas de salvar los datos, por la sentencia SQL Dump o salvando el archivo de la DB. El primero será:

   ```
   pg_dump ArchivoDB > ArchivoBackup
   ```

   Para recuperar, se puede utilizar:

   ```
   psql ArchivoDB < ArchivoBackup
   ```

   Para salvar todas las DB del servidor, se puede ejecutar:

   ```
   pg_dumpall > ArchivoBackupTotal
   ```

   Otra estrategia es salvar los archivos de las bases de datos a nivel del sistema operativo, por ejemplo con:

   ```
   tar -cf backup.tar /usr/local/pgsql/data
   ```

ANOTACIONES

Existen dos restricciones que pueden hacer este método poco práctico:

- El servidor debe ser parado antes de salvar y de recuperar los datos.
- Deben conocerse muy bien todas las implicaciones a nivel archivo donde están todas las tablas, transacciones y demás, ya que si no, una DB puede quedar inútil. Además (generalmente), el tamaño que se salvará será mayor que el realizado con los métodos anteriores, ya que por ejemplo, con el *pg_dumb* no se salvan los índices, sino el comando para recrearlos.

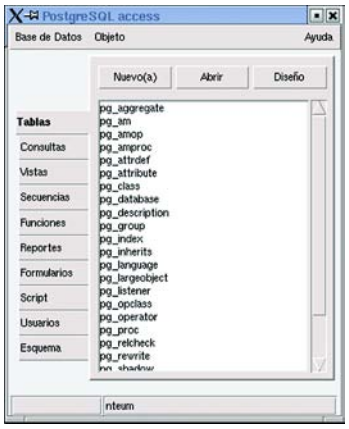
8.1.6. Pgaccess

La aplicación **pgaccess** [*NombreDB*] permite acceder y administrar una base de datos con una interfaz gráfica. La forma más fácil de acceder (por ejemplo, desde KDE) es desde una terminal, el administrador de DB deberá hacer, si no es el usuario *postgres*:

`xhost +`	Permite que otras aplicaciones puedan conectarse al display del usuario actual
`su postgres`	
`pgaccess [NombreDB]&`	Si se configura en 'Preferencias' abrirá siempre la última DB

La figura 18 muestra la interfaz de *pgaccess*.

Figura 18.



ANOTACIONES

En una sesión típica el administrador/usuario podría, en primer lugar, *Abrir Base de Datos* , indicando aquí, por ejemplo, *Port = 5432*, *Base de Datos = nteum*, (los otros parámetros no son necesarios si la base de datos es local) y luego *Abrir*. A partir de este momento, el usuario podrá trabajar con el espacio bidimensional seleccionando qué es lo que desea hacer en el eje *Y* (**Tablas**, **Consultas**, **Vistas**, etc.) y con ese elemento seleccionado, y escogiendo uno de ese tipo dentro de la ventana utilizar el eje *X* superior para *Nuevo* (*añadir*), *Abrir* o *Diseño*. Por ejemplo, si se selecciona en *Y Usuarios* y en *X, Nuevo*, la aplicación solicitará el nombre del usuario, su contraseña (con verificación), su validez en el tiempo y sus características (por ejemplo, *Crear DB*, *Crear otros usuarios*). En *Base de Datos* también se podría seleccionar *Preferencias* para, por ejemplo, cambiar el tipo de fuente y seleccionar la posibilidad de ver las tablas del sistema.

Las configuraciones personales de los usuarios quedarán registradas en el archivo ~/.pgaccessrc. La interfaz permite realizar/agilizar gran parte del trabajo del usuario/administrador y es recomendable para usuarios que se acaban de iniciar en PostgreSQL, ya que no necesitan conocer la sintaxis de la línea de comando como en **psql** (la propia aplicación solicitará mediante múltiples ventanas todas las opciones de un comando).

Una herramienta más simple es a través del módulo correspondiente de **webmin** (es necesario instalar los paquetes *webmin-core* y los módulos necesarios, por ejemplo en este caso *webmin-postgresql*). Durante la instalación, el *webmin* dará un aviso que el usuario principal será el *root* y utilizará la misma contraseña que el *root* del sistema operativo. Para conectarse, se podrá hacer, por ejemplo desde Mozilla, *https://localhost:10000* el cual solicitará aceptar (o denegar) la utilización del certificado para la comunicación SSL y, a continuación, mostrará todos los servicios que puede administrar, entre ellos PostgreSQL Data Base Server.

8.2. Mysql

MySQL [Mys03] es (según sus autores) la base de datos (DB) SQL abierta, es decir, software libre (Open Source) más popular, y es desarrollada y distribuida por MySQL AB (compañía comercial que ob-

ANOTACIONES

tiene sus beneficios de los servicios que provee sobre la DB). MySQL es un DBMS (*Database Management System*). Un DBMS es el que puede añadir y procesar los datos almacenados dentro de la DB. Al igual que PostgreSQL, MySQL es una base de datos relacional, lo que significa que almacena los datos en tablas en lugar de una única ubicación lo cual permite mayor velocidad y flexibilidad. Al ser software libre, cualquiera puede obtener el código, estudiarlo y modificarlo de acuerdo a sus necesidades sin pago alguno, ya que MySQL utiliza licencia GPL. MySQL provee en su página web un conjunto de estadísticas y prestaciones en comparación con otras DB para mostrar al usuario cuán rápida, fiable y fácil es de usar. La decisión de elegir una DB se debe hacer cuidadosamente en función de las necesidades de los usuarios y del entorno donde se utilizará esta DB.

8.2.1. Instalación

- Obtener desde http://www.mysql.com/ o desde cualquiera de los repositorios de software. Se pueden obtener los binarios y los archivos fuente para compilarlos e instalarlos.

- En el caso de los binarios, utilizar la distribución de Debian y seleccionar los paquetes *mysql-client*, *mysql-server* y *mysql-common*. La instalación, después de unas preguntas, creará un usuario *mysql* y una entrada en /etc/init.d/mysql para arrancar/parar el servidor en el *boot*. También se puede hacer manualmente haciendo:

  ```
  /etc/init.d/mysql start|stop
  ```

  Para acceder a la base de datos, se puede utilizar el monitor mysql desde la línea de comando. Si obtiene los binarios en otro formato (no Debian ni RPM), por ejemplo *gz* desde el sitio web de MySQL, deberá ejecutar los siguientes comandos para instalar la DB:

```
groupadd mysql                Crea el grupo
useradd -g mysql mysql        Crea el usuario
cd /usr/local                 Cambia de directorio
gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -  Descomprime
```

```
ln -s full-path-to-mysql-VERSION-OS mysql
                                  Crea un link simbólico
cd mysql                          Cambia de directorio
scripts/mysql_install_db          Instala la DB
chown -R root .                   Cambia los privilegios
chown -R mysql data               Idem
chgrp -R mysql .                  Idem
bin/mysqld_safe --user = mysql \verb+&+
                                  Arranca el servidor
bin/mysqld_safe --user = mysql \verb+&+
                                  Si se utiliza MySQL 4.x.
```

Esto crea el usuario/grupo/directorio, descomprime, e instala la DB en /usr/local/mysql.

- En caso de obtener el código fuente, los pasos son similares:

```
groupadd mysql
useradd -g mysql mysql
gunzip < mysql-VERSION.tar.gz | tar -xvf -
cd mysql-VERSION
./configure --prefix = /usr/local/mysql    Configura el código fuente
make                                       Crea el ejecutable
make install                               Instala binarios
scripts/mysql_install_db
chown -R root /usr/local/mysql
chown -R mysql /usr/local/mysql/var
chgrp -R mysql /usr/local/mysql
cp support-files/mymedium.cnf /etc/my.cnf
/usr/local/mysql/bin/mysqld_safe --user = mysql &
```

ANOTACIONES

Es importante prestar atención cuando se realiza el *configure*, ya que *prefix* = /usr/local/mysql es el directorio donde se instalará la DB y se puede cambiar para ubicar la DB en el directorio que se desee.

8.2.2. Postinstalación y verificación

Una vez instalada (ya sea de los binarios o del código fuente), se deberá verificar si el servidor funciona correctamente. En Debian puede hacer directamente:

```
/etc/init.d/mysql start        Inicia el servidor
```

`mysqladmin version`	Genera información de versiones y (c)
`mysqladmin variables`	Muestra los valores de las variables
`mysqladmin -u root shutdown`	Finaliza la ejecución del servidor
`mysqlshow`	Mostrará las DB predefinidas
`mysqlshow mysql`	Mostrará las tablas de la DB *mysql*

Si se instala desde el código fuente, antes de hacer estas comprobaciones se deben ejecutar los siguientes comandos para crear las bases de datos (desde el directorio de la distribución):

```
./scripts/mysql_install_db
cd DirectorioInstalacionMysql
./bin/mysqld_safe --user = mysql &
```

Si se instala de binarios (RPM, Pkg, ...), se debe hacer lo siguiente:

```
cd DirectorioInstalacionMysql
./scripts/mysql_install_db
./bin/mysqld_safe user = mysql &
```

El *script mysql_install_db* crea la DB *mysql* y *mysqld_safe* arranca el servidor mysqld. A continuación, se pueden probar todos los comandos dados anteriormente para Debian, excepto el primero que es el que arranca el servidor. Además, si se han instalado los tests, se podrán ejecutar con CD *sql-bench* y luego *run-all-tests*. Los resultados se encontrarán en el directorio *sql-bech/Results* para compararlos con otras DB.

8.2.3. El programa monitor (cliente) mysql

El cliente *mysql* se puede utilizar para crear y utilizar DB simples, es interactivo y permite conectarse al servidor, ejecutar búsquedas y visualizar los resultados. También funciona en modo *batch* (como un *script*) donde los comandos se le pasan a través de un archivo. Para ver todas las opciones del comando, se puede ejecutar *mysql --help*. Podremos realizar una conexión (local o remota) con el comando *mysql*, por ejemplo, para una conexión por la interfaz de red pero desde la misma máquina:

Nota

Cliente (*frontend*) mysql:
`mysql [NombreDB]`

```
mysql -h localhost -u mysql -p NombreDB
```

Si no se pone el último parámetro, ninguna DB es seleccionada.

ANOTACIONES

Nota

Para más información, podéis consultar la documentación, comandos y opciones. [Mys03]

Una vez dentro, el *mysql* pondrá un *prompt* (*mysql>*) y esperará a que le introduzcamos algún comando (propio y SQL), por ejemplo *help*. A continuación, daremos una serie de comandos para probar el servidor (recordar poner siempre el ';' para terminar el comando):

```
mysql> SELECT VERSION(), CURRENT_DATE;
```
Se puede utilizar mayúsculas o minúsculas.

```
mysql> SELECT SIN(PI()/4), (4+1)*5; Calculadora.
mysql> SELECT VERSION(); SELECT NOW();
```
Múltiples comandos en la misma línea.

```
mysql> SELECT
  -> USER()
  -> ,
  -> CURRENT_DATE;
```
O en múltiples líneas.

```
mysql> SHOW DATABASES;
```
Muestra las DB disponibles.

```
mysql> USE test
```
Cambia la DB.

```
mysql> CREATE DATABASE nteum; USE nteum;
```
Crea y selecciona una DB llamada *nteum*.

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
  -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```
Crea una tabla dentro de nteum.

```
mysql> SHOW TABLES;
```
Muestra las tablas.

```
mysql> DESCRIBE pet;
```
Muestra la definición de la tabla.

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```
Carga datos desde *pet.txt* en *pet*. El archivo *pet.txt* debe tener un registro por línea separado por *tabs* de los datos de acuerdo a la definición de la tabla (fecha en formato AAAA-MM-DD)

```
        mysql> INSERT INTO pet
  -> VALUES ('Marciano','Estela','gato','f','1999-03-30',NULL);
```
Carga los datos *in-line*.

```
mysql> SELECT * FROM pet;
```
Muestra los datos de la tabla.

```
mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Browser";
```
Modifica los datos de la tabla.

```
mysql> SELECT * FROM pet WHERE name = "Browser";
```
Muestra selectiva.

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```
Muestra ordenada.

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```
Muestra selectiva con funciones.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO marciano@localhost
   -> IDENTIFIED BY 'passwd' WITH GRANT OPTION;
```

Crea un usuario marciano en la DB. Lo debe hacer el root de la DB. O también se puede hacer directamente con:

```
mysql> INSERT INTO user (Host,User,Password)
   ->     VALUES('localhost','marciano','passwd');
```

8.2.4. Administración

Mysql dispone de un archivo de configuración en /etc/mysql/my.cnf (en Debian), al cual se le pueden cambiar las opciones por defecto a la DB, como por ejemplo, el puerto de conexión, usuario, contraseña de los usuarios remotos, archivos de *log*, archivos de datos, si acepta conexiones externas, etc. Con respecto a la seguridad, se deben tomar algunas precauciones:

1) No dar a nadie (excepto al usuario *root* de *mysql*) acceso a la tabla **user** dentro de la DB **mysql**, ya que aquí se encuentran las contraseñas de los usuarios que podrían ser utilizados con otros fines.

2) Verificar *mysql -u root*. Si se puede acceder, significa que el usuario *root* no tiene contraseña. Para cambiarlo, se puede hacer:

```
mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('new_password')
     -> WHERE user = 'root';
mysql> FLUSH PRIVILEGES;
```

Ahora, para conectarse como *root*:

```
mysql -u root -p mysql
```

3) Comprobar la documentación (punto 4.2) respecto a las condiciones de seguridad y del entorno de red para evitar problemas de ataques y/o intrusión.

Para hacer copias de la base de datos, se puede utilizar el comando:

```
mysqldump --tab = /DirectorioDestino --opt NombreDB
```

ANOTACIONES

O también:

```
mysqlhotcopy NombreDB /DirectorioDestino
```

Asimismo, se pueden copiar los archivos *.frm', *.MYD', y *.MYI con el servidor parado. Para recuperar, ejecutad:

```
REPAIR TABLE o myisamchk -r
```

lo cual funcionará en el 99% de las veces. En caso contrario, podría copiar los archivos salvados y arrancar el servidor. Existen otros métodos alternativos en función de lo que se quiera recuperar, como la posibilidad de salvar/recuperar parte de la DB (consultar punto 4.4 de la documentación). [Mys03]

8.2.5. Interfaces gráficas

Para Mysql hay gran cantidad de interfaces gráficas, entre las que destacamos MysqlCC (se puede obtener desde http://www.mysql.com/), Mysql-Navigator, o Webmin con el módulo para trabajar con Mysql (paquetes *webmin-core* y *webmin-mysql*).

Estos dos últimos se incluyen en la distribución Debian 3.0*Woody* r1. En el caso de MysqlCC, es una interfaz muy potente (análoga a Pgaccess), pero que se debe descargar y compilar (no presenta ninguna dificultad) desde el servidor de Mysql.

Con **Mysql-navigator**, es necesario insertar un comentario '#' en la línea (41) correspondiente a *skip-networking* para permitir que el servidor pueda aceptar conexiones de red, luego pararlo y arrancarlo nuevamente para que estas opciones puedan ser cargadas:

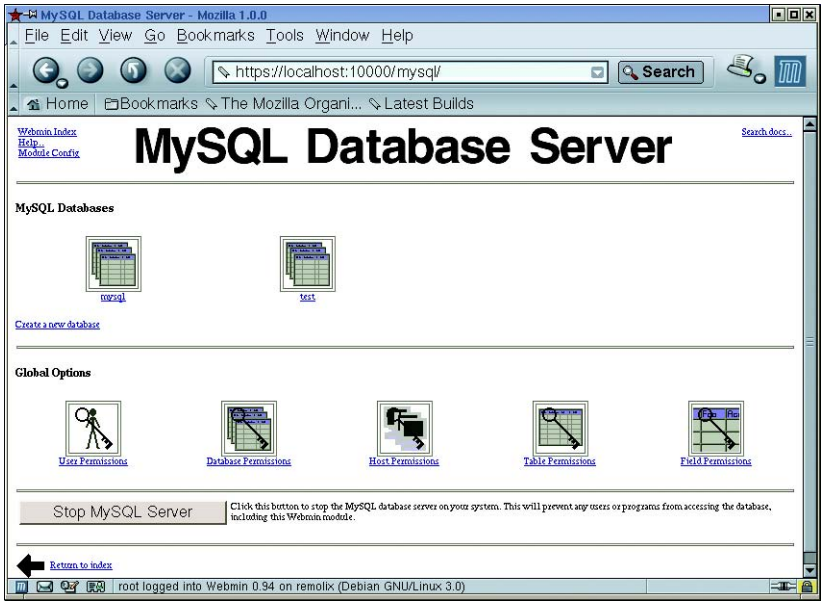
```
/etc/init.d/mysql stop|start
```

A continuación, se puede ejecutar **mysql-navigator** y en la opción *FileOpen* seleccionar *Host = localhost*, *User = mysql* y se verán todas las bases existentes. A partir de este momento, se puede seleccionar una DB y trabajar con ella. En forma análoga a PostgreSQL, **webmin**

ANOTACIONES

permite también trabajar con Mysql (es necesario instalar los paquetes *webmin-mysql* además del *webmin-core*). Durante la instalación, el webmin dará un aviso de que el usuario principal será el *root* y utilizará la misma contraseña que el *root* del sistema operativo. Para conectarse, se podrá hacer, por ejemplo desde Mozilla: *https://localhost:10000,* el cual solicitará aceptar (o denegar) la utilización del certificado para la comunicación SSL y a continuación mostrará todos los servicios que puede administrar, entre ellos Mysql Data Base *Server*. La figura 19 muestra la interfaz de **webmin** en el módulo de MysqlServer.

Figura 19.



8.3. *Source Code Control System* (CVS y RCS)

El ***Concurrent Versions System*** (**CVS**) es un sistema de control de versiones que permite mantener versiones antiguas de archivos (generalmente código fuente), guardando un registro (*log*) de quién, cuándo y por qué fueron realizados los cambios. A diferencia de otros sistemas, CVS no trabaja con un archivo/directorio por vez, sino que actúa sobre colecciones jerárquicas de los directorios que controla.

El CVS tiene por objetivo ayudar a manejar versiones de software y controla la edición concurrente de archivos fuente por múltiples au-

ANOTACIONES

tores. El ***Revision Control System*** (**RCS**) por su parte maneja múltiples revisiones de archivos automatizando su almacenamiento, apertura, identificación y mezcla de revisiones, pero, a diferencia del CVS, necesita un repositorio por cada proyecto de software. Ambos programas son sistemas de control de código indispensables cuando los programadores necesitan una historia completa de los cambios para poder volver atrás si es necesario.

El CVS utiliza RCS internamente como una capa de bajo nivel (por lo cual, es sumamente recomendable usar CVS) y además de su propia funcionalidad, presenta toda las prestaciones de RCS pero con notables mejoras en cuanto a la estabilidad, operación y mantenimiento (no tanto en la instalación y configuración). Entre ellas podemos destacar: funcionamiento descentralizado (cada usuario puede tener su propio árbol de código), edición concurrente, comportamiento adaptable mediante *shell scripts*, etc. Entre las ventajas de RCS, cabe destacar que es más simple de utilizar y administrar (obviamente con menor funcionalidad), necesita un área de trabajo centralizada donde todos deben trabajar desde ésta área y no permite la concurrencia (puede ser útil para pequeños sistemas de desarrollo).

No obstante, en el siguiente apartado veremos una breve descripción de la utilización de RCS que, como se mencionó, puede ser útil para pequeñas instalaciones, y a continuación se describirá con detalle la administración del CVS. [Ced97, CVS03, Vas03a, Kie97]

8.3.1. Revision Control System (RCS)

El Revision Control System (RCS) está formado por un conjunto de programas para las diferentes actividades del RCS: **rcs** (programa que controla los atributos de los archivos bajo RCS), **ci** y **co** (verifican la entrada y la salida de los archivos bajo el control de RCS), **ident** (busca en el RCS los archivos por palabras claves/atributos), **rcsclean** (limpia archivos no utilizados o que no han cambiado), **rcsdiff** (ejecuta el comando *diff* para comparar versiones), **rcsmerge** (une dos ramas (archivos) en un único archivo), y **rlog** (imprime los mensajes de *log*).

El formato de los archivos almacenados por RCS puede ser texto u otro formato, como por ejemplo binario (en este caso el programa *diff* debe poder utilizar archivos con datos de 8 bits). Un archivo RCS

ANOTACIONES

consiste en un archivo de revisión inicial llamado 1.1 y una serie de archivos de cambios, uno por cada revisión. Cada vez que se realiza una copia del repositorio hacia el directorio de trabajo con el comando **co** (obtiene una revisión de cada archivo RCS y lo pone en el archivo de trabajo) o se utiliza **ci** (almacena nuevas revisiones en el RCS), el número de versión se incrementa (por ejemplo, 1.2, 1.3, ...). Los archivos (generalmente) están en el directorio ./RCS y es necesario que el sistema operativo tenga instalados los comandos *diff* y *diff3* para que funcione adecuadamente.

Para compilarlo (en Debian no es necesario, ya que lo incluye por defecto), una vez obtenidos los archivos fuente desde alguna distribución (por ejemplo, ftp://sunsite.unc.edu/pub/Linux/devel/vc/rcs5.7.src.tar.gz), se deben ejecutar los siguientes comandos (como *root*):

`gunzip < rcs-5.7.src.tar.gz | tar -xvf -` Descomprime.

`cd rcs*`

`./configure` Configura el código fuente generando un *makefile* y un *conf.sh* según el sistema operativo que se vaya a instalar

`make install` Crea e instala binarios

Creando y manteniendo archivos

Con el comando **rcs** crearemos y modificaremos los atributos de los archivos (consultar *man rcs*). La forma más fácil de crear un repositorio es hacer en primer lugar un *mkdir rcs* en el directorio de originales e incluir los originales en el repositorio con:

`ci nombre_archivos_fuentes`

Se puede utilizar el * y siempre tener una copia de resguardo para evitar problemas. Esto creará las versiones de los archivos con nombre *./RCS/nombre_archivo* y solicitará un texto para describir el archivo. Luego, con *co RCS/nombre_archivo*, obtendremos una copia de trabajo desde el repositorio. Se puede bloquear o desbloquear este archivo para evitar modificaciones, respectivamente, con:

`rcs L nombre_archivo_de_trabajo` y `rcs U nombre_archivo_de_trabajo`

Con *rlog nombre_del_archivo* podremos ver la información sobre las diferentes versiones. [Kie97]

8.3.2. Concurrent Versions System (CVS)

En primer lugar se debe instalar el *Concurrent Versions System* (CVS) desde la distribución teniendo en cuenta que debemos tener instalado RCS y que deberemos instalar también OpenSSH si se lo quiere utilizar conjuntamente con CVS para acceso remoto. Las variables de entorno *EDITOR CVSROOT* deben estar inicializadas por ejemplo en /etc/profile (o en .bash profile):

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
```

Obviamente, los usuarios pueden modificar estas definiciones utilizando /.bash profile. Se debe crear el directorio donde estará el repositorio y configurar los permisos; como *root,* hay que hacer, por ejemplo:

```
export CVSROOT = /usr/local/cvsroot
groupadd cvs
useradd -g cvs -d \verb+$+CVSROOT cvs
mkdir \verb+$+CVSROOT
chgrp -R cvs \verb+$+CVSROOT
chmod o-rwx \verb+$+CVSROOT
chmod ug+rwx \verb+$+CVSROOT
```

Para inicializar el repositorio y poner archivo de código en él:

```
cvs -d /usr/local/cvsroot init
```

cvs init tendrá en cuenta no sobrescribir nunca un repositorio ya creado para evitar pérdidas de otros repositorios. Luego, se deberá agregar los usuarios que trabajarán con el CVS al grupo *cvs*; por ejemplo, para agregar el usuario *nteum*:

```
usermod -G cvs,nteum
```

ANOTACIONES

Ahora el usuario *nteum* deberá introducir sus archivos en el directorio del repositorio (/usr/local/cvsroot en nuestro caso) hacer:

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
export CVSREAD = yes
cd directorio_de_originales
cvs import NombreDelRepositorio vendor_1_0 rev_1_0
```

El nombre del repositorio puede ser un identificador único o también *usuario/projecto/xxxx* si es que el usuario desea tener organizados sus repositorios. Esto creará un árbol de directorios en CVSROOT con esa estructura.

Esto añade un directorio (/usr/local/cvsroot/*NombreDelRepositorio*) en el repositorio con los archivos que a partir de este momento estarán en el repositorio. Una prueba para saber si se ha almacenado todo correctamente es almacenar una copia en el repositorio y luego crear una copia desde allí y comprobar las diferencias. Por ejemplo, sean los originales en el *directorio_del_usuario/dir_org* y se desea crear un repositorio como *primer_cvs/proj*, se deberán ejecutar los siguientes comandos:

`cd dir_org`

Cambiar al directorio del código fuente original.

`cvs import -m "Fuentes originales" primer_cvs/proj usuarioX vers0`

Crea el repositorio en primer_cvs/proj con *ususarioX* y *vers0*.

`cd..`

Cambiar al directorio superior de *dir_org*.

`cvs checkout primer_cvs/proj`

Generar una copia del repositorio. La variable CVSROOT debe estar inicializada, si no, se deberá indicar todo el *path*.

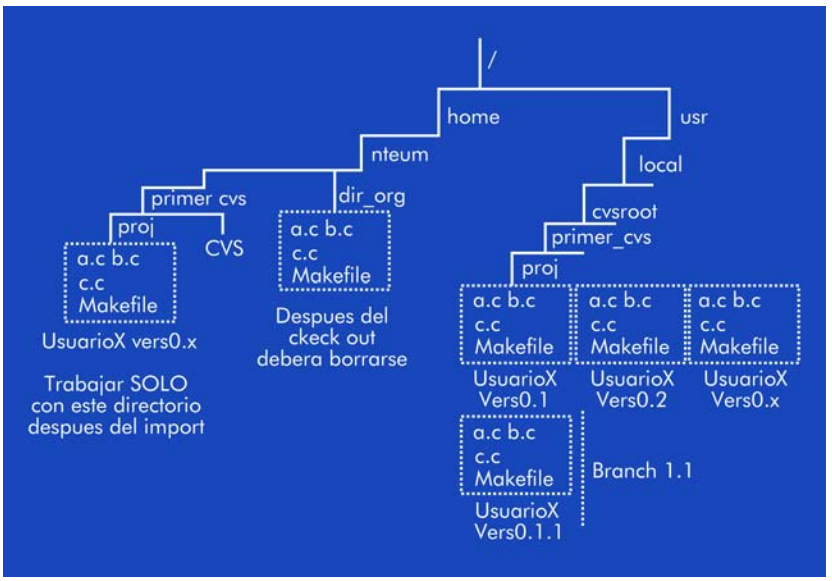
`diff -r dir_org primer_cvs/proj`

Muestra las diferencias entre uno y otro; que no debe haber ninguna excepto por el directorio primer_cvs/proj/CVS que ha creado el CVS.

`rm -r dir_org`

Borra los originales (realizar una copia de resguardo siempre por seguridad y para tener una referencia de dónde se inició el trabajo con el CVS)

Figura 20.



El hecho de borrar los originales no es siempre una buena idea; sólo en este caso, después de que se haya verificado que están en repositorio, para que por descuido no se trabaje sobre ellos y los cambios no queden reflejados sobre el CVS. Sobre máquinas donde los usuarios quieren acceder (por ssh) a un servidor CVS remoto, se deberá hacer:

```
export CVSROOT = ":ext:user@CVS.server.com:/home/cvsroot"
export CVS_RSH = "ssh"
```

Donde *user* es el *login* del usuario y *cvs.server.com* el nombre del servidor donde está CVS.

CVS ofrece una serie de comandos (se llaman con **cvs comd opciones** ...) para trabajar con el sistema de revisiones, entre ellos: *checkout*, *update*, *add*, *remove*, *commit* y *diff*.

El comando inicial *cvs checkout* ... crea su copia privada del código fuente para luego trabajar con ella sin interferir en el trabajo de otros usuarios (como mínimo se crea un subdirectorio donde estarán los archivos).

- **cvs update** se debe ejecutar del árbol privado cuando hay que actualizar sus copias de archivos fuentes con los cambios que otros programadores han hecho sobre los archivos del repositorio.

ANOTACIONES

- **cvs add file**... es un comando necesario cuando hay que agregar nuevos archivos en su directorio de trabajo sobre un módulo donde ya se ha hecho un *checkout* previamente. Estos archivos serán enviados al repositorio CVS cuando se ejecute el comando *cvs commit*.

- **cvs import** se puede usar para introducir archivos nuevos en el repositorio.

- **cvs remove file**... este comando se utilizará para borrar archivos del repositorio (una vez que se hayan borrado éstos del archivo privado). Este comando debe ir acompañado de un *cvs commit* para que los cambios sean efectivos, ya que se trata del comando que transforma todas las peticiones de los usuarios sobre el repositorio.

- **cvs diff file**... se puede utilizar sin que afecte a ninguno de los archivos implicados si se necesita verificar las diferencias entre repositorio y directorio de trabajo o entre dos versiones.

- **cvs tag -R "versión"** se puede utilizar para introducir un número de versión en los archivos de un proyecto y luego hacer un *cvs commit* y un *cvs checkout -r 'version' proyecto* para registrar una nueva versión.

Una característica interesante del CVS es poder aislar cambios de los archivos aislados en una línea de trabajo separada llamada *ramificación* (*branch*). Cuando se cambia un archivo sobre una rama, estos cambios no aparecen sobre los archivos principales o sobre otras ramas. Más tarde, estos cambios se pueden incorporar a otros ramas o al archivo principal (*merging*). Para crear una nueva rama, utilizar *cvs tag -b rel-1-0-patches* dentro del directorio de trabajo, lo cual asignará a la rama el nombre de *rel-1-0-patches*. La unión de ramas con el directorio de trabajo significa utilizar el comando *cvs update -j*. Consultar las referencias para mezclar o acceder a diferentes ramas.

Ejemplo de una sesión

Siguiendo el ejemplo de la documentación dada en las referencias, se mostrará una sesión de trabajo con CVS. Como CVS almacena to-

ANOTACIONES

das los archivos en un repositorio centralizado, se asumirá que el mismo ya ha sido inicializado anteriormente.

Consideremos que se está trabajando con un conjunto de archivos en C y un *makefile*, por ejemplo. El compilador utilizado es *gcc* y el repositorio es inicializado a *gccrep*.

En primer lugar, se debe obtener una copia de los archivos del repositorio a nuestra copia privada con:

```
cvs checkout gccrep
```

que creará un nuevo directorio llamado gccrep con los archivos fuente. Si se ejecuta *cd gccrep* y *ls*, se verá por ejemplo *CVS makefile a.c b.c c.c*, donde existe un directorio CVS que se crea para el control de la copia privada que normalmente no es necesario tocar.

Después de esto se podría utilizar un editor para modificar a.c e introducir cambios sustanciales en el archivo (ver en la documentación sobre múltiples usuarios concurrentes si se necesita trabajar con más de un usuario en el mismo archivo), compilar, volver a cambiar, etc.

Cuando se decide que se tiene una versión nueva con todos los cambios introducidos en *a.c* (o en los archivos que sea necesario), es momento de hacer una nueva versión almacenando a.c (o todos los que se han tocado) en el repositorio y hacer esta versión disponible al resto de usuarios: *cvs commit a.c*.

Utilizando el editor definido en la variable *CVSEDITOR* (o *EDITOR* si ésta no está inicializada) se podrá introducir un comentario que indique qué cambios se han hecho para que sirva de ayuda a otros usuarios o para recordar qué es lo que caracterizó a esta versión y luego poder hacer un histórico.

Si se decide eliminar los archivos (porque ya se terminó con el proyecto o porque no se trabajará más con él), una forma de hacerlo es a nivel de sistema operativo (*rm -r gccrep*), pero es mejor utilizar el propio *cvs* fuera del directorio de trabajo (nivel inmedia-

ANOTACIONES

to superior): *cvs release -d gccrep*. El comando detectará si hay algún archivo que no ha sido enviado al repositorio, y si lo hay y se borra, significa que se perderán todos los cambios, por ello preguntará si se desea continuar o no.

Para mirar las diferencias, por ejemplo, se ha modificado *b.c* y no se recuerda qué cambios se hicieron, se puede utilizar dentro del directorio de trabajo: cvs *diff b.c*. Éste utilizará el comando del sistema operativo *diff* para comparar la versión *b.c* con la versión que se tiene en el repositorio (siempre hay que recordar hacer un *cvs commit b.c* si se desea que estas diferencias sean transferidas al repositorio como una nueva versión).

Múltiples usuarios

Cuando más de una persona trabaja en un proyecto software con diferentes revisiones, es sumamente complicado porque habrá ocasiones en las que más de un usuario se quiera editar el mismo fichero simultáneamente. Una posible solución es bloquear el fichero o utilizar puntos de verificación reservados (*reserved checkouts*), lo cual sólo permitirá a un usuario editar el mismo fichero simultáneamente. Para ello, se deberá ejecutar el comando *cvs admin -l command* (ver *man* para las opciones).

CVS utiliza un modelo por defecto de puntos no reservados (*unreserved checkouts*), que permite a los usuarios editar simultáneamente un fichero de su directorio de trabajo. El primero de ellos que transfiera sus cambios al repositorio lo podrá hacer sin problemas, pero los restantes recibirán un mensaje de error cuando desean realizar la misma tarea, por lo cual, deberán utilizar comandos de *cvs* para transferir en primer lugar los cambios al directorio de trabajo desde el repositorio y luego actualizar el repositorio con sus propios cambios.

Consultad las referencias para ver un ejemplo de aplicación y otras formas de trabajo concurrente con comunicación entre usuarios. [Vas03a].

8.3.3. Interfaces gráficas

Contamos con un conjunto de interfaces gráficas como **tkcvs** [Tkc03] desarrollada en *Tcl/Tk* u otra también muy popular, **cervisia** [Cer03c].

ANOTACIONES

En el sitio web de CVS se pueden encontrar también un conjunto de *plugins* para CVS o para usuarios no técnicos, para lo que es recomendable utilizar **gCVS** [Ara03]. A continuación, se muestran dos de la interfaces gráficas mencionadas (*tkcvs* y *gcvs*):

Figura 21.

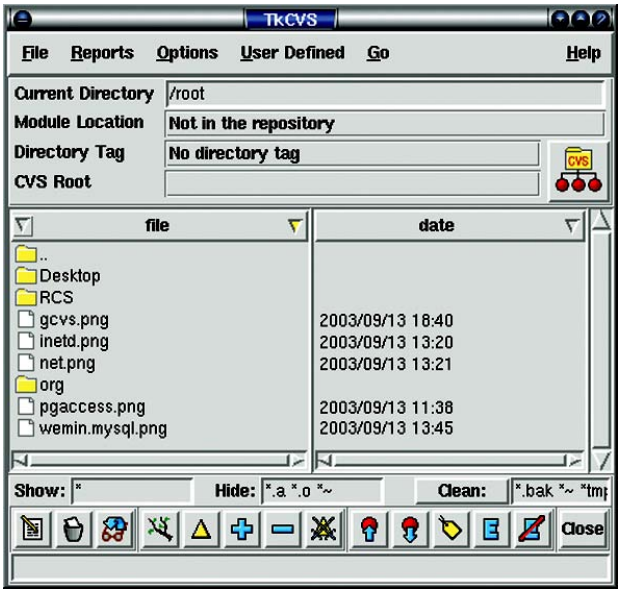
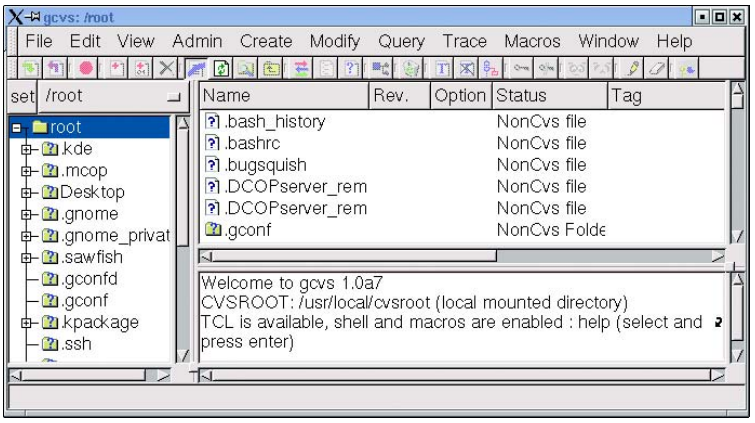


Figura 22.



8.4. Actividades para el lector

1) Definir en PostgreSQL una DB que tenga al menos 3 tablas con 5 columnas (de las cuales 3 deben ser numéricas) en cada tabla.

Generar un listado ordenado por cada tabla/columna. Generar un listado ordenado por el mayor valor de la columna *X* de todas las tablas. Cambiar el valor numérico de la columna *Y* con el valor numérico de la columna *Z* + valor de la columna *W*/2.

2) El mismo ejercicio anterior, pero con MySQL.

3) Configurar el CVS para hacer tres revisiones de un directorio donde hay 4 archivos .c y un *makefile*. Hacer una ramificación (*branch*) de un archivo y luego mezclarlo con el principal.

4) Simular la utilización de un archivo concurrente con dos terminales de Linux e indicar la secuencia de pasos para hacer que dos modificaciones alternas de cada usuario queden reflejadas sobre el repositorio CVS.

5) El mismo ejercicio anterior, pero uno de los usuarios debe conectarse desde otra máquina al repositorio.

Nota

Otras fuentes de referencias e información:

[Deb03c, LPD03b, Ibi03, Mou01]

ANOTACIONES

9. Administración de seguridad

El salto tecnológico de los sistemas de escritorio aislados, hasta los sistemas actuales integrados en redes locales e Internet, ha traído una nueva dificultad a las tareas habituales del administrador: el control de la seguridad de los sistemas.

La seguridad es un campo complejo, en el cual se mezclan tanto técnicas de análisis, con otras de detección o de prevención de los posibles ataques, como el análisis de factores "psicológicos", como el comportamiento de los usuarios del sistema o las posibles intenciones de los atacantes.

Los ataques pueden provenir de muchas fuentes y afectar desde a una aplicación o servicio, hasta a algún usuario, o a todos, o al sistema informático entero.

Los posibles ataques pueden cambiar el comportamiento de los sistemas, incluso hacerlos caer (inutilizarlos), o dar una falsa impresión de seguridad, que puede ser difícilmente detectable. Podemos encontrarnos con ataques de autentificación (obtener acceso por parte de programas o usuarios previamente no habilitados), escuchas (redirigir o pinchar los canales de comunicación y los datos que circulan), o sustitución (sustituir programas, máquinas o usuarios por otros, sin que se noten los cambios).

Una idea clara que hay que tener en mente es que es imposible poder conseguir una seguridad al 100%.

Nota

La seguridad absoluta no existe. Una falsa impresión de seguridad puede ser tan perjudicial como no tenerla. El área de la seguridad es muy dinámica, y hay que mantener actualizados constantemente los conocimientos.

Las técnicas de seguridad son un arma de doble filo, que fácilmente pueden darnos una falsa impresión de control del problema. La seguridad actual es un problema amplio, complejo y lo que es más importante, dinámico. Nunca podemos esperar o decir que la seguridad está

garantizada, sino que con bastante probabilidad será una de las áreas a la cual el administrador tendrá que dedicar más tiempo y mantener actualizados sus conocimientos sobre el tema.

En esta unidad examinaremos algunos tipos de ataques con que podemos encontrarnos, cómo podemos verificar y prevenir partes de la seguridad local, y también en entornos de red. Asimismo, examinaremos técnicas de detección de intrusos y algunas herramientas básicas que nos pueden ayudar en el control de la seguridad.

También es preciso mencionar que en esta unidad sólo podemos hacer una mera introducción a algunos de los aspectos que intervienen en la seguridad de hoy en día. Para cualquier aprendizaje real con más detalle, se recomienda consultar la bibliografía disponible, así como los manuales asociados a los productos o herramientas comentados.

9.1. Tipos y métodos de los ataques

La seguridad computacional en administración puede ser entendida como el proceso que ha de permitir al administrador del sistema prevenir y detectar usos no autorizados de éste. Las medidas de prevención ayudan a parar los intentos de usuarios no autorizados (los conocidos como *intrusos*), para acceder a cualquier parte del sistema. La detección ayuda a descubrir cuándo se produjeron estos intentos o, en el caso de llevarse a cabo, establecer las barreras para que no se repitan y poder recuperar el sistema si éste ha sido quebrantado.

Los intrusos (también conocidos coloquialmente como *hackers*, *crackers*, 'atacantes' o 'piratas', ...) normalmente desean obtener control sobre el sistema, ya sea para causar malfuncionamientos, corromper el sistema o sus datos, ganar recursos en la máquina, o simplemente utilizarlo para lanzar ataques contra otros sistemas, y así proteger su verdadera identidad y ocultar el origen real de los ataques. También está la posibilidad de examinar la información (o robarla) del sistema, el puro espionaje de las acciones del sistema o causar daños físicos en la máquina, ya sea reformatear el disco, cambiar datos, borrar o modificar software crítico, ...

Con respecto a los intrusos, hay que establecer algunas diferencias, que no suelen estar muy claras en los términos coloquiales. Normal-

ANOTACIONES

mente nos referimos a **hacker** [Him01], como aquella persona con grandes conocimientos en informática, más o menos apasionada por los temas de programación y seguridad informática y que, normalmente sin finalidad malévola, utiliza sus conocimientos para protegerse a sí mismo, introducirse en redes para demostrar sus fallos de seguridad, y, en algunos casos, como reconocimiento de sus habilidades.

Ejemplo

Un ejemplo sería la propia comunidad GNU/Linux, que debe mucho a sus *hackers*, ya que hay que entender el término *hacker* como experto en unos temas (más que como intruso en la seguridad).

Por otro lado encontraríamos a los **crackers**. Aquí es cuando se utiliza el término de manera más o menos despectiva, hacia aquellos que utilizan sus habilidades para corromper (o destrozar) sistemas, ya sea sólo por fama propia, por motivos económicos, por ganas de causar daño o simplemente por molestar; por motivos de espionaje tecnológico, actos de ciberterrorismo, etc. Asimismo, se habla de *hacking* o *cracking*, cuando nos referimos a técnicas de estudio, detección y protección de la seguridad, o por el contrario, de técnicas destinadas a causar daño rompiendo la seguridad de los sistemas.

Desafortunadamente, es bastante más fácil de lo que parece obtener acceso a un sistema, (ya sea desprotegido o parcialmente seguro). Los intrusos descubren permanentemente nuevas vulnerabilidades (llamadas a veces 'agujeros', o *exploits*), que les permiten introducirse en las diferentes capas de software. La complejidad cada vez mayor del software (y del hardware), hace que aumente la dificultad para testar de manera razonable la seguridad de los sistemas informáticos. El uso habitual de los sistemas GNU/Linux en red, ya sea la propia Internet o en redes propias con tecnología TCP/IP como las intranets, nos lleva a exponer nuestros sistemas a ser víctimas de ataques de seguridad. [Bur02][Fen02][Lin03e]

Lo primero que hay que tener en cuenta es romper el mito de la **seguridad informática**: simplemente **no existe**. Lo que sí que podemos conseguir es un cierto nivel de seguridad que nos haga sentirnos seguros dentro de ciertos parámetros. Pero como tal, sólo es una percep-

ANOTACIONES

ción de seguridad, y como todas las percepciones, puede ser falsa y podemos darnos cuenta en el último momento, cuando ya tengamos nuestros sistemas afectados. La conclusión lógica es que la seguridad informática exige un esfuerzo importante de constancia, realismo y aprendizaje prácticamente diario.

Tenemos que ser capaces de establecer en nuestros sistemas, unas políticas de seguridad que nos permitan prevenir, identificar y reaccionar ante los posibles ataques. Y tener presente que la sensación que podamos tener de seguridad no es más que eso, una sensación. Por lo tanto, no hay que descuidar ninguna de las políticas implementadas y hay que mantenerlas al día, así como nuestros conocimientos del tema.

Nota

Las amenazas afectan a la confidencialidad, integridad o accesibilidad de nuestros sistemas.

Los posibles ataques son una amenaza constante a nuestros sistemas y pueden comprometer su funcionamiento, así como los datos que manejamos; ante todo lo cual, siempre tenemos que definir una cierta política de requerimientos de seguridad sobre nuestros sistemas y datos. Las amenazas que podemos sufrir podrían afectar a los aspectos siguientes:

- **Confidencialidad**: la información debe ser accesible sólo a aquellos que estén autorizados; estamos respondiendo a la pregunta: ¿quién podrá acceder a la misma?
- **Integridad**: la información sólo podrá ser modificada por aquellos que estén autorizados: ¿qué se podrá hacer con ella?
- **Accesibilidad**: la información tiene que estar disponible para quienes la necesiten y cuando la necesiten, si están autorizados: ¿de qué manera, y cuándo se podrá acceder a ella?

Pasemos a mencionar una cierta clasificación (no exhaustiva) de los tipos de ataques habituales que podemos padecer:

- **Autentificación**: ataques en los que se falsifica la identidad del participante de manera que obtiene acceso a programas o servicios de los que no disponía en un principio.
- **Intercepción** (o **escucha**): mecanismo por el cual se interceptan datos, por parte de terceros, cuando éstos no estaban dirigidos a ellos.

ANOTACIONES

- **Falsificación** (o **reemplazo**): sustitución de algunos de los participantes –ya sea máquinas, software o datos– por otros falsos.

- **Robo** de recursos: uso de nuestros recursos sin autorización.

- O, simplemente, **vandalismo**: después de todo, suele ser bastante común la existencia de mecanismos que permiten interferir con el funcionamiento adecuado del sistema o servicios y causar molestias parciales, o el paro o cancelación de recursos.

Los métodos utilizados y las técnicas precisas pueden variar mucho (es más, cada día se crean novedades), y nos obligan, como administradores, a estar en contacto permanente con el campo de la seguridad para conocer a qué nos podemos enfrentar diariamente.

Para cada uno de los tipos de ataques mencionados, normalmente se pueden utilizar uno o más métodos de ataque, que a su vez pueden provocar también uno o más de los tipos de ataques.

Respecto a dónde se produzca el ataque, hemos de tener claro qué puede hacerse o cual será el objetivo de los métodos:

Nota

Los ataques pueden tener finalidades destructivas, inhabilitadoras o de espionaje, de nuestros componentes, ya sea hardware, software o sistemas de comunicación.

- **Hardware**: a este respecto, la amenaza está directamente sobre la accesibilidad, ¿qué podrá hacer alguien que tenga acceso al hardware? En este caso normalmente necesitaremos medidas "físicas", como controles de seguridad para acceder a los locales donde estén las máquinas para evitar problemas de robo o rotura del equipo con el fin de eliminar su servicio. También puede comprometerse la confidencialidad y la integridad si el acceso físico a las máquinas permite utilizar algunos de sus dispositivos como las disqueteras, o el arranque de las máquinas, o el acceso a cuentas de usuario que podrían estar abiertas.

- **Software**: si la accesibilidad se ve comprometida en un ataque, puede haber borrado o inutilización de programas, denegando el acceso. En caso de confidencialidad, puede provocar copias no autorizadas de software. En integridad podría alterarse el funcionamiento por defecto del programa, para que éste fallase en algunas situaciones o bien para que realice tareas que puedan ser interesantes de cara al atacante, o simplemente comprometer la in-

ANOTACIONES

tegridad de los datos de los programas: hacerlos públicos, alterarlos, o simplemente robarlos.

- **Datos**: ya sean estructurados, como en los servicios de base de datos, o gestión de versiones (como *cvs*), o simples archivos. Mediante ataques que amenacen la accesibilidad, pueden ser destruidos o eliminados, denegando así el acceso a los mismos. En el caso de la confidencialidad, estaríamos permitiendo lecturas no autorizadas y la integridad se vería afectada cuando se produzcan modificaciones o creación de nuevos datos.

- **Canal de comunicación** (en la red, por ejemplo): para los métodos que afecten a la accesibilidad, nos puede provocar destrucción o eliminación de mensajes, e impedir el acceso a la red. En confidencialidad, lectura y observación del tráfico de mensajes, desde o hacia la máquina. Y respecto a la integridad, cualquier modificación, retardo, reordenación, duplicación o falsificación de los mensajes entrantes y/o salientes.

9.1.1. Técnicas utilizadas en los ataques

Los métodos utilizados son múltiples y pueden depender de un elemento (hardware o software) o de la versión de éste. Por lo tanto, hay que mantener actualizado el software para las correcciones de seguridad que vayan apareciendo, y seguir las indicaciones del fabricante o distribuidor para proteger el elemento.

A pesar de ello, normalmente siempre hay técnicas o métodos de "moda", del momento actual, algunas breves indicaciones de estas técnicas de ataque (de hoy en día) son:

Nota

Las técnicas de los ataques son muy variadas y evolucionan constantemente en lo que a los detalles usados se refiere.

- **Bug exploits**: o explotación de errores o agujeros [CER03b] [Ins98][San03], ya sea de un hardware, software, servicio, protocolo o del propio sistema operativo (por ejemplo, en el *kernel*), y normalmente de alguna de las versiones de éstos en concreto. Normalmente, cualquier elemento informático es más o menos propenso a errores en su concepción, o simplemente a cosas que no se han tenido en cuenta o previsto. Periódicamente, se descubren agujeros (a veces se denominan *holes*, *exploits*, o simple-

mente *bugs*, ...), que pueden ser aprovechados por un atacante para romper la seguridad de los sistemas. Suelen utilizarse o bien técnicas de ataque genéricas, como las que se explican a continuación, o bien técnicas particulares para el elemento afectado. Cada elemento afectado tendrá un responsable –ya sea fabricante, desarrollador, distribuidor o la comunidad GNU/Linux– de producir nuevas versiones o parches para tratar estos problemas. Nosotros, como administradores, tenemos la responsabilidad de estar informados y de mantener una política de actualización responsable para evitar los riesgos potenciales de estos ataques. En caso de que no haya soluciones disponibles, también podemos estudiar la posibilidad de utilizar alternativas al elemento, o bien inhabilitarlo hasta que tengamos soluciones.

- **Virus**: programa normalmente anexo a otros y que utiliza mecanismos de autocopia y transmisión. Son habituales los virus anexados a programas ejecutables, a mensajes de correo electrónico, o incorporados en documentos o programas que permiten algún lenguaje de macros (no verificado). Son quizás la mayor plaga de seguridad de hoy en día.

  Los sistemas GNU/Linux están protegidos casi totalmente contra estos mecanismos por varias razones: en los programas ejecutables, tienen un acceso muy limitado al sistema, en particular a la cuenta del usuario. Con excepción del usuario *root*, con el cual hay que tener mucho cuidado con lo que éste ejecuta. El correo no suele utilizar lenguajes de macros no verificados (como en el caso de Outlook y Visual Basic Script en Windows, que es un agujero de entrada de virus), y en el caso de los documentos, estamos en condiciones parecidas, ya que no soportan lenguajes de macros no verificados (como el VBA en Microsoft Office).

  En todo caso, habrá que prestar atención a lo que pueda pasar en un futuro, ya que podrían surgir algunos virus específicos para Linux aprovechando algunos *bugs* o *exploits*. Un punto que sí que hay que tener en cuenta es el de los sistemas de correo, ya que si bien nosotros no generaremos virus, sí que podemos llegar a transmitirlos; por ejemplo, si nuestro sistema funciona como *router* de correo, podrían llegar mensajes con virus que podrían ser enviados a otros. Aquí se puede implementar alguna política

ANOTACIONES

de detección y filtrado de virus. Otra forma de azote de plagas que podría entrar dentro de la categoría de virus son los mensajes de *spam*, que si bien no suelen ser utilizados como elementos atacantes, sí que podemos considerarlos como problemas por su "virulencia" de aparición, y el coste económico que pueden causar (pérdidas de tiempo y recursos).

- **Worm** (o 'gusano'): normalmente se trata de un tipo de programas que aprovechan algún agujero del sistema para realizar ejecuciones de código sin permiso. Suelen ser utilizados para aprovechar recursos de la máquina, como el uso de CPU, bien cuando se detecta que el sistema no funciona o no está en uso, o bien si son malintencionados, con el objetivo de robar recursos o bien utilizarlos para parar o bloquear el sistema. También suelen utilizar técnicas de transmisión y copia.

- **Trojan Horse** (o 'caballos de Troya', o 'troyanos'): programas útiles que incorporan alguna funcionalidad, pero ocultan otras que son las utilizadas para obtener información del sistema o comprometerlo. Un caso particular puede ser el de los códigos de tipo móvil en aplicaciones web, como los Java, JavaScript o ActiveX; éstos normalmente piden su consentimiento para ejecutarse (ActiveX en Windows), o tienen modelos limitados de lo que pueden hacer (Java, JavaScript). Pero como todo software, también tienen agujeros y son un método ideal para transmitir troyanos.

- **Back Door** (o *trap door*, 'puerta trasera'): método de acceso a un programa escondido que puede utilizarse con fines de otorgar acceso al sistema o a los datos manejados sin que lo conozcamos. Otros efectos pueden ser cambiar la configuración del sistema, o permitir introducir virus. El mecanismo usado puede ser desde venir incluidos en algún software común, o bien en un troyano.

- **Bombas lógicas**: programa incrustado en otro, que comprueba que se den algunas condiciones (temporales, acciones del usuario, etc.), para activarse y emprender acciones no autorizadas.

- **Keyloggers**: programa especial que se dedica a secuestrar las interacciones con el teclado del usuario. Pueden ser programas individuales o bien troyanos incorporados en otros programas.

ANOTACIONES

Normalmente, necesitarían introducirse en un sistema abierto al que se dispusiese de acceso. La idea es captar cualquier introducción de teclas, de manera que se capturen contraseñas, interacción con aplicaciones, sitios visitados por la red, formularios rellenados, etc.

- **Scanner** (escaneo de puertos): más que un ataque, sería un paso previo, que consistiría en la recolección de posibles objetivos. Básicamente, consiste en utilizar herramientas que permitan examinar la red en busca de máquinas con puertos abiertos, sean TCP, UDP u otros protocolos, los cuales indican presencia de algunos servicios. Por ejemplo, escanear máquinas buscando el puerto 80 TCP, indica la presencia de servidores web, de los cuales podemos obtener información sobre el servidor y la versión que utilizan para aprovecharnos de vulnerabilidades conocidas.

- **Sniffers** ('husmeadores'): permiten la captura de paquetes que circulan por una red. Con las herramientas adecuadas podemos analizar comportamientos de máquinas: cuáles son servidores, clientes, qué protocolos se utilizan y en muchos casos obtener contraseñas de servicios no seguros. En un principio, fueron muy utilizados para capturar contraseñas de telnet, rsh, rcp, ftp, ... servicios no seguros que no tendrían que utilizarse (usar en su lugar las versiones seguras: ssh, scp, sftp). Tanto los *sniffers* (como los *scanners*) no son necesariamente una herramienta de ataque, ya que también pueden servir para analizar nuestras redes y detectar fallos, o simplemente analizar nuestro tráfico. Normalmente, tanto las técnicas de *scanners* como las de *sniffers* suelen utilizarse por parte de un intruso con el objetivo de encontrar las vulnerabilidades del sistema, ya sea para conocer datos de un sistema desconocido (*scanners*), o bien para analizar la interacción interna (*sniffer*).

- **Hijacking** (o 'secuestro'): son técnicas que intentan colocar una máquina de manera que intercepte o reproduzca el funcionamiento de algún servicio en otra máquina que ha pinchado la comunicación. Suelen ser habituales los casos para correo electrónico, transferencia de ficheros o web. Por ejemplo, en el caso web, se puede capturar una sesión y reproducir lo que el usuario está haciendo, páginas visitadas, interacción con formularios, etc.

- **Buffer overflows**: técnica bastante compleja que aprovecha errores de programación en las aplicaciones. La idea básica es aprovechar desbordamientos (*overflows*) en *buffers* de la aplicación, ya sean

ANOTACIONES

colas, *arrays*, etc. Si no se controlan los límites, un programa atacante puede generar un mensaje o dato más grande de lo esperado y provocar fallos. Por ejemplo, muchas aplicaciones C con *buffers* mal escritos, en *arrays*, si sobrepasamos el límite podemos provocar una sobreescritura del código del programa, causando malfuncionamiento o caída del servicio o máquina. Es más, una variante más compleja permite incorporar en el ataque trozos de programa (compilados C o bien *shell scripts*), que pueden permitir la ejecución de cualquier código que el atacante quiera introducir.

- **Denial of Service** ('ataque DoS'): este tipo de ataque provoca que la máquina caiga o que se sobrecarguen uno o más servicios, de manera que no sean utilizables. Otra técnica es la DDoS (*Distributed* DoS), que se basa en utilizar un conjunto de máquinas distribuidas para que produzcan el ataque o sobrecarga de servicio. Este tipo de ataques se suelen solucionar con actualizaciones del software, ya que normalmente se ven afectados aquellos servicios que no fueron pensados para una carga de trabajo determinada y no se controla la saturación. Los ataques DoS y DDoS son bastante utilizados en ataques a sitios web, o servidores DNS, los que ven afectados por vulnerabilidades de los servidores, por ejemplo, de versiones concretas de Apache o BIND. Otro aspecto por tener en cuenta es que nuestro sistema también podría ser usado para ataques de tipo DDoS, mediante control ya sea de un *backdoor* o un troyano.

Un ejemplo de este ataque (DoS), bastante sencillo, es el conocido como *SYN flood*, que trata de generar paquetes TCP que abren una conexión, pero ya no hacen nada más con ella, simplemente la dejan abierta; esto gasta recursos del sistema en estructuras de datos del *kernel*, y recursos de conexión por red. Si se repite este ataque centenares o miles de veces, se consigue ocupar todos los recursos sin utilizarlos, de modo que cuando algunos usuarios quieran utilizar el servicio, les sea denegado porque los recursos están ocupados. Otro caso conocido es el correo *bombing*, o simplemente reenvío de correo (normalmente con emisor falso) hasta que se saturan las cuentas de correo o el sistema de correo cae, o se vuelve tan lento que es inutilizable. Estos ataques son en cierta medida sencillos de realizar, con las herramientas adecuadas, y no tienen una solución fácil, ya que se aprovechan del funcio-

Nota

SYN flood, ver: http://www.cert.org/advisories/CA-1996-21.html

E-mail bombing, ver: http://www.cert.org/tech tips/email _bombing_ spamming.html

ANOTACIONES

namiento interno de los protocolos y servicios; en estos casos tenemos que tomar medidas de detección y control posterior.

- **spoofing**: las técnicas de *spoofing* engloban varios métodos (normalmente muy complejos) de falsificar tanto la información, como los participantes en una transmisión (origen y/o destino). Existen *spoofing* como los de:

– *IP spoofing*, falsificación de una máquina, permitiendo que genere trafico falso, o escuche tráfico que iba dirigido a otra máquina, combinado con otros ataques puede saltarse incluso protección de *firewalls*.

– *ARP spoofing*, técnica compleja (utiliza un DDoS), que intenta falsificar las direcciones de fuentes y destinatarios en una red, mediante ataques de las cachés de ARP, que poseen las máquinas, de manera que se sustituyan las direcciones reales por otras en varios puntos de una red. Esta técnica permite saltarse todo tipo de protecciones, incluidos *firewalls*, pero no es una técnica sencilla.

– *E-mail*, es quizás el más sencillo. Se trata de generar contenidos de correos falsos, tanto por el contenido como por la dirección de origen. En este tipo son bastante utilizadas las técnicas de lo que se denomina *ingeniería social*, que básicamente intenta engañar de una forma razonable al usuario, un ejemplo clásico son correos falsos del administrador de sistema, o bien, por ejemplo, del banco donde tenemos nuestra cuenta corriente, mencionando que ha habido problemas en las cuentas y que se tiene que enviar información confidencial, o la contraseña anterior para solucionarlo, o pidiendo que la contraseña se cambie por una concreta. Sorprendentemente, esta técnica consigue engañar a un número considerable de usuarios. Incluso con (ingeniera social de) métodos sencillos: algún *cracker* famoso comentaba que su método preferido era el teléfono. Como ejemplo, exponemos el caso de que una empresa de certificación (*verisign*), en que los *crackers* obtuvieron la firma privada de la empresa de software Microsoft, con sólo realizar una llamada diciendo que era de parte de la empresa, que se les había presentado un problema, y volvían a necesitar su

Nota

Ver el caso de Microsoft en: http://www.computerworld.com/softwaretopics/os/windows/story/0,10801,59099,00.html

clave, ... Resumiendo, unos niveles altos de seguridad informática se pueden ir al traste por una simple llamada telefónica.

Algunas recomendaciones generales (muy básicas) para la seguridad, podrían ser:

- Controlar un factor problemático, los **usuarios**: uno de los factores que puede afectar más a la seguridad es la confidencialidad de las contraseñas, y ésta se ve afectada por el comportamiento de los usuarios; esto facilita a posibles atacantes las acciones desde dentro del propio sistema. La mayoría de los ataques suelen venir de dentro del sistema, o sea, una vez el atacante ha ganado acceso al sistema.

  Entre los usuarios, está aquel que es un poco olvidadizo (o indiscreto), que bien olvida la contraseña cada dos por tres, lo menciona en conversaciones, lo escribe en un papel que olvida, o que está junto (o pegado) al ordenador o sobre la mesa de trabajo, o que simplemente lo presta a otros usuarios o conocidos. Otro tipo es el que pone contraseñas muy predecibles, ya sea su mismo *id* de usuario, su nombre, su DNI, el nombre de su novia, el de su madre, el de su perro, etc., cosas que con un mínimo de información pueden encontrarse fácilmente. Otro caso son los usuarios normales con un cierto conocimiento, que colocan contraseñas válidas, pero siempre hay que tener en cuenta que hay mecanismos que pueden encontrarlas (*cracking de passwords*, *sniffing*, *spoofing* ...). Hay que establecer una cierta "cultura" de seguridad entre los usuarios, y mediante técnicas obligarles a que cambien las contraseñas, no utilicen palabras típicas, las contraseñas deben ser largas (tener más de 2 o 3 caracteres), etc. Últimamente, en muchas empresas e instituciones se está implantando la técnica de hacer firmar un contrato al usuario de manera que se le obliga a no divulgar la contraseña o cometer actos de vandalismo o ataques desde su cuenta (claro que esto no impide que otros lo hagan por él).

- No utilizar ni ejecutar programas de los que no podamos garantizar su origen. Normalmente, muchos distribuidores utilizan mecanismos de comprobación de firmas para verificar que los paquetes de software son tales, como por ejemplo las sumas *md5*

ANOTACIONES

(comando *md5sum*) o la utilización de firmas GPG [Hat03d](comando *gpg*). El vendedor o distribuidor provee una suma *md5* de su archivo (o imagen de CD), y podemos comprobar la autenticidad de éste.

- No utilizar usuarios privilegiados (como *root*) para el trabajo normal de la máquina; cualquier programa (o aplicación) tendría los permisos para acceder a cualquier parte.

- No acceder remotamente con usuarios privilegiados ni ejecutar programas que puedan tener privilegios. Y más, si no conocemos, o hemos comprobado, los niveles de seguridad del sistema.

- No utilizar elementos que no sabemos cómo actúan ni intentar descubrirlo a base de repetidas ejecuciones.

Estas medidas pueden ser poco productivas, pero si no hemos asegurado el sistema, no podemos tener ningún control sobre lo que puede pasar, y aun así, nadie asegura que no se pueda colar algún programa malicioso que burlara la seguridad si lo ejecutamos con los permisos adecuados. O sea, que en general hemos de tener mucho cuidado con todo este tipo de actividades que supongan accesos y ejecución de tareas de formas más o menos privilegiadas.

9.1.2. Contramedidas

Respecto a las medidas por tomar sobre los tipos de ataques presentados, podemos encontrar algunas preventivas y de detección de lo que sucede en nuestros sistemas.

Veamos algunos tipos de medidas que podríamos tomar en los campos de prevención y detección de intrusos (se mencionan herramientas útiles, algunas las examinaremos más adelante):

- **Password cracking**: en ataques de fuerza bruta para romper las contraseñas, suele ser habitual intentar obtener acceso por *logins* de forma repetida; si se consigue entrar, la seguridad del usuario ha sido comprometida y se deja la puerta abierta a otros tipos de

ANOTACIONES

ataques como, por ejemplo, los *backdoor*, o simplemente la destrucción de la cuenta. Para prevenir este tipo de ataques, hay que reforzar la política de contraseñas, pidiendo una longitud mínima y cambios de contraseña periódicos. Una cosa que hay que evitar es el uso de palabras comunes en las contraseñas: muchos de estos ataques se hacen mediante la fuerza bruta, con un fichero de diccionario (con palabras en el idioma del usuario, términos comunes, argot, etc.). Este tipo de contraseñas serán las primeras en caer. También puede ser fácil obtener información del atacado, como nombres, DNI o su dirección, y usar estos datos para probar las contraseñas. Por todo lo cual, tampoco se recomiendan contraseñas con DNI, nombres (propios o de familiares, etc.), direcciones, etc. Una buena elección suelen ser contraseñas de entre 6 a 8 caracteres como mínimo, con contenido de caracteres alfabéticos, numéricos y algún carácter especial.

Aunque la contraseña esté bien elegida, ésta puede ser insegura si se utiliza en servicios no seguros. Por lo tanto, se recomienda reforzar los servicios mediante técnicas de encriptación que protejan las contraseñas y los mensajes. Y por el contrario, evitar (o no utilizar) todos aquellos servicios que no soporten encriptación, y consecuentemente, susceptibles de ser atacados con métodos, por ejemplo de *sniffers*; entre éstos podríamos incluir servicios telnet, ftp, rsh, rlogin (entre otros).

- **Bug exploits**: evitar disponer de programas que no se utilicen, sean antiguos, o no se actualicen (por estar obsoletos). Aplicar los últimos parches y actualizaciones que estén disponibles, tanto para las aplicaciones, como para el sistema operativo. Probar herramientas que detecten vulnerabilidades. Mantenerse al día de las vulnerabilidades que se vayan descubriendo.

- **Virus**: utilizar mecanismos o programas antivirus, sistemas de filtrado de mensajes sospechosos, evitar la ejecución de sistemas de macros (que no se puedan verificar). No hay que minimizar los posibles efectos de los virus, cada día se perfeccionan más, y técnicamente es posible realizar virus simples que puedan desactivar redes en cuestión de minutos (sólo hay que ver algunos de los virus recientes del mundo Windows).

Nota

Ver parches para el sistema operativo en:

http://www.debian.org/securit

http://www.redhat.com/solutions/security/news/

Ejemplo

Sobre vulnerabilidades, una buena herramienta es Nessus. Para descubrir nuevas vulnerabilidades, ver CERT en:

http://www.cert.org/advisories/

ANOTACIONES

- **Worm** (o gusano): controlar el uso de nuestras máquinas o usuarios en horas no previstas, y el control del tráfico de salida y/o entrada.

- **Trojan horse** (o caballos de Troya, o troyanos): verificar la integridad de los programas periódicamente, mediante mecanismos de suma o firmas. Detección de tráfico anómalo de salida o entrada al sistema. Utilizar *firewalls* para bloquear tráfico sospechoso. Una versión bastante peligrosa de los troyanos la forman los *rootkits* (comentados más adelante), que realizan más de una función gracias a un conjunto variado de herramientas. Para la verificación de la integridad, podemos utilizar mecanismos de sumas como (*md5* o *gpg*) o herramientas que automatizan este proceso, como Tripwire o AIDE.

- **Back door** (o **trap door**, puerta trasera): hay que obtener de los proveedores o vendedores del software la certificación de que éste no contiene ningún tipo de *backdoor* escondido no documentado, y por suspuesto aceptar el software proveniente sólo de sitios que ofrezcan garantías. Cuando el software sea de terceros, o de fuentes que podrían haber modificado el software original, muchos fabricantes (o distribuidores) integran algún tipo de verificación de software basado en códigos de suma o firmas digitales (tipo *md5* o *gpg*) [Hat03d]. Siempre que éstas estén disponibles, sería útil verificarlas antes de proceder a la instalación del software. También puede probarse el sistema intensivamente, antes de colocarlo como sistema de producción.

  Otro problema puede consistir en la alteración del software *a posteriori*. En este caso pueden ser también útiles los sistemas de firmas o sumas para crear códigos sobre software ya instalado y controlar que no se produzcan cambios en software vital. O bien copias de seguridad, con las que podemos hacer comparaciones para detectar cambios.

- **Bombas lógicas**: en este caso suelen ocultarse tras activaciones por tiempo o por acciones del usuario. Podemos verificar que no existan en el sistema trabajos no interactivos introducidos de tipo *crontab*, *at*, y otros procesos (por ejemplo, de tipo *nohup*), que dispongan de ejecución periódica, o estén en ejecución en segun-

ANOTACIONES

do plano desde hace mucho tiempo (comandos *w*, *jobs*). En todo caso, podrían utilizarse medidas preventivas que impidieran trabajos no interactivos a los usuarios, o que solamente los permitiesen a aquellos que lo necesitasen.

- **Keyloggers** y **rootKits**: en este caso habrá algún proceso intermediario que intentará capturar nuestras pulsaciones de teclas y las almacenará en algún lugar. Habrá que examinar situaciones donde aparezca algún proceso extraño perteneciente a nuestro usuario, o bien detectar si tenemos algún fichero abierto con el que no estemos trabajando directamente (por ejemplo, podría ser de ayuda *lsof*, ver *man*), o bien conexiones de red, si se tratase de *keylogger* con envío externo. Para probar un funcionamiento muy básico de un *keylogger* muy sencillo, puede verse el comando de sistema *script* (ver *man script*). El otro caso, el *rootkit* (que suele incluir también algún *keylogger*) suele ser un *pack* de unos cuantos programas con varias técnicas, y permite al atacante, una vez entra en una cuenta, utilizar diversos elementos como un *keylogger*, *backdoors*, troyanos (sustituyendo a comandos del sistema), etc., con tal de obtener información y puertas de entrada al sistema, muchas veces se acompaña de programas que realizan limpieza de los *logs*, para eliminar las pruebas de la intrusión. Un caso particularmente peligroso lo forman los *rootkits*, que se usan o vienen en forma de módulos de *kernel*, lo que les permite actual a nivel de *kernel*. Para su detección, sería necesario controlar que no haya tráfico externo que salga hasta una cierta dirección. Una herramienta útil para verificar los *rootkits* es *chrootkit*.

Nota

La herramienta *chrootkit* puede encontrarse en: http://www.chrootkit.org

- **Escáner** (escaneo de puertos): los escáneres suelen lanzar sobre uno o más sistemas bucles de escaneo de puertos conocidos para detectar los que quedan abiertos y aquellos servicios que están funcionando y que podrían ser susceptibles de ataques.

- **Sniffers** (husmeadores): evitar intercepciones e impedir así la posibilidad de que se introduzcan escuchas. Una técnica es la construcción hardware de la red, que puede dividirse en segmentos para que el tráfico sólo circule por la zona que se va a utilizar, poner *firewalls* para unir estos segmentos y poder controlar el tráfico de entrada y salida. Usar técnicas de encriptación para que los mensajes no puedan ser leídos e interpretados por alguien que

ANOTACIONES

escuche la red. Para el caso tanto de escáneres como *sniffers*, podemos utilizar herramientas como Ethereal y Snort, para realizar comprobaciones sobre nuestra red, o para el *port scanning* Nmap. En el caso de los *sniffers*, éstos pueden ser detectados en la red mediante la búsqueda de máquinas en modo Ethernet promiscuo (están a la escucha de cualquier paquete que circula), normalmente, la tarjeta de red sólo captura el tráfico que va hacia ella (o de tipo *broadcast* o multicast).

- **Hijacking** (o 'secuestro'): implementar mecanismos de encriptación en los servicios, requerir autentificación y, si es posible, que esta autentificación se renueve periódicamente. Controlar el tráfico entrante o saliente por *firewalls*. Monitorizar la red para detectar flujos de tráfico sospechosos.

- **Buffer overflows**: suelen ser comunes como *bugs* o agujeros del sistema, suelen solucionarse por actualización del software. En todo caso, pueden observarse por *logs* situaciones extrañas de caída de servicios que deberían estar funcionando.

- **Denial of Service** ('ataque DoS'), y otros como **SYN flood**, o correos *bombing* tomar medidas de bloqueo de tráfico innecesario en nuestra red (por ejemplo, por medio de *firewalls*). En aquellos servicios que se pueda, habrá que controlar tamaños de *buffer*, número de clientes por atender, *timeouts* de cierre de conexiones, capacidades del servicio, etc.

- **spoofing**: a) IP *spoofing*, b) ARP *spoofing*, c) correo electrónico. Estos casos necesitan una fuerte encriptación de los servicios, control por *firewalls*, mecanismos de autentificación basados en varios aspectos (por ejemplo, no basarse en la IP, si ésta pudiera verse comprometida), se pueden implementar mecanismos que controlen sesiones establecidas y se basen en varios parámetros de la máquina a la vez (sistema operativo, procesador, IP, dirección Ethernet, etc.). También monitorizar sistemas DNS, cachés de ARP, *spools* de correo, etc., para detectar cambios en la información que invaliden a anteriores.

- **Ingeniería social**: no es propiamente una cuestión informática, pero también es necesaria para que las personas no empeoren la

ANOTACIONES

seguridad. Medidas adecuadas como aumentar la información o educar a usuarios y técnicos en temas de seguridad: controlar qué personal dispondrá de información crítica de seguridad y en qué condiciones puede cederla a otros. Los servicios de ayuda y mantenimiento de una empresa pueden ser un punto crítico: controlar quién posee información de seguridad, y cómo la usa.

Respecto a usuarios finales, mejorar su cultura de contraseñas, evitar que la dejen apuntada en cualquier sitio, a la vista de terceros, o simplemente la divulguen.

9.2. Seguridad del sistema

Ante los posibles ataques, tenemos que disponer de mecanismos de prevención, detección y recuperación de nuestros sistemas.

Para la prevención local, hay que examinar los diferentes mecanismos de autentificación y permisos de accesos a los recursos para definirlos correctamente y poder garantizar la confidencialidad y la integridad de nuestra información. En este caso, nos estaremos protegiendo de atacantes que hayan obtenido acceso a nuestro sistema, o bien de usuarios hostiles que quieran saltarse las restricciones impuestas en el sistema.

Respecto a la seguridad en red, tenemos que garantizar que los recursos que ofrecemos (si proporcionamos unos determinados servicios) tienen los parámetros de confidencialidad necesarios y que los servicios no pueden ser usados por terceros no deseados, de modo que, un primer paso será controlar que los servicios ofrecidos sean los que realmente queremos, y que no estamos ofreciendo además otros servicios que no tenemos controlados. En el caso de servicios de los que nosotros somos clientes, también habrá que asegurar los mecanismos de autentificación, en el sentido de que accedemos a los servidores correctos y no existen casos de suplantación de servicios o servidores (normalmente bastante difíciles de detectar).

Respecto a las aplicaciones y a los mismos servicios, además de garantizar la correcta configuración de niveles de acceso mediante per-

ANOTACIONES

misos y de autentificación de los usuarios permitidos, tendremos que vigilar la posible explotación de *bugs* en el software. Cualquier aplicación, por muy bien diseñada e implementada que esté, puede tener un número más o menos alto de errores que pueden ser aprovechados para, con ciertas técnicas, saltarse las restricciones impuestas. En este caso, practicamos una política de prevención que pasa por mantener el sistema actualizado en la medida de lo posible, de manera que, o bien actualizamos ante cualquier nueva corrección, o bien somos conservadores y mantenemos aquellas versiones que sean más estables en cuestión de seguridad. Normalmente, esto significa verificar periódicamente unos cuantos sitios de seguridad para conocer los últimos fallos detectados en el software y las vulnerabilidades que se derivan de éstos, y que podrían exponer nuestros sistemas a fallos de seguridad, ya sea local o por red.

9.3. Seguridad local

La seguridad local [Peñ03] [Hat03b] es básica para la protección del sistema [Deb][Hat03c], ya que normalmente, tras un primer intento de acceso desde la red, es la segunda barrera de protección antes de que un ataque se haga con parte del control de la máquina. Además, la mayoría de los ataques acaban haciendo uso de recursos internos del sistema.

Nota

Diversos ataques, aunque vengan del exterior, tienen como finalidad conseguir el acceso local.

9.3.1. *Bootloaders*

Respecto a la seguridad local, ya en arranque se nos pueden presentar problemas debido al acceso físico que un intruso pudiera tener a la máquina.

Uno de los problemas ya está en el arranque del sistema. Si el sistema puede arrancarse de disco o CD, un atacante podría acceder a los datos de una partición Linux (o también Windows) tan sólo con montar el sistema de ficheros y podría colocarse como usuario *root* sin necesidad de conocer ninguna contraseña. En este caso, se necesita proteger el arranque del sistema desde la BIOS, por ejemplo, protegiendo el acceso por contraseña, de manera que no se permita el arranque desde CD o disquete. También es razonable actualizar

ANOTACIONES

la BIOS, ya que también puede tener fallos de seguridad. Además, hay que tener cuidado, porque muchos de los fabricantes de BIOS ofrecen contraseñas extras conocidas (una especie de *backdoor*), con lo cual no podemos depender de estas medidas en exclusiva.

El siguiente paso es proteger el *bootloader*, ya sea *lilo* o *grub*, para que el atacante no pueda modificar las opciones de arranque del *kernel* o modificar directamente el arranque (caso de *grub*). Cualquiera de los dos puede protegerse también por contraseñas.

En *grub*, el fichero /sbin/grub-md5-crypt pide la contraseña y genera una suma *md5* asociada. Después, el valor obtenido se introduce en /boot/grub/grub.conf. Bajo la línea *timeout*, se introduce:

```
password --md5 suma-md5-calculada
```

Para *lilo* se coloca, o bien una contraseña global con:

```
password = contraseña
```

o bien uno en la partición que queramos:

```
image = /boot/vmlinuz-version
  password = contraseña
  restricted
```

En este caso *restricted* indica además que no se podrán cambiar los parámetros pasados al *kernel* desde línea de comandos. Hay que tener cuidado de poner el fichero /etc/lilo.conf protegido a sólo lectura/escritura desde el *root* (*chmod* 600).

Otro tema relacionado con el arranque es la posibilidad de que alguien que tenga acceso al teclado reinicie el sistema, debido a que si se pulsa CTRL+ALT+DEL, se provoca una operación de *shutdown* en la máquina. Este comportamiento viene definido en /etc/inittab, con una línea como:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

ANOTACIONES

Si se comenta, esta posibilidad de reiniciar quedará desactivada. O por el contrario, puede crearse un fichero /etc/shutdown.allow, que permite a ciertos usuarios poder reiniciar.

9.3.2. *Passwords y shadows*

Las contraseñas típicas de los sistema UNIX iniciales (y de primeras versiones GNU/Linux), estaban encriptadas mediante unos algoritmos DES (pero con claves pequeñas, y una llamada de sistema se encargaba de encriptar y desencriptar, en concreto *crypt* (ver *man*).

Normalmente, se encontraban en el fichero /etc/passwd, en el segundo campo, por ejemplo:

```
user:sndb565sadsd:...
```

Pero el problema está en que este fichero es legible por cualquier usuario, con lo que un atacante podía obtener el fichero y utilizar un ataque de fuerza bruta, hasta que desencriptase *passwords* contenidos en el fichero, o bien mediante ataques de fuerza bruta por medio de diccionarios.

El primer paso es utilizar los nuevos ficheros /etc/shadow, donde se guarda ahora la contraseña. Este fichero es sólo legible por el *root*, y por nadie más. En este caso, en /etc/passwd aparece un asterisco (*) donde antes estaba la contraseña encriptada. Por defecto, las distribuciones de GNU/Linux actuales usan contraseñas de tipo *shadow* a no ser que se les diga que no las usen.

Un segundo paso es cambiar el sistema de encriptación de las contraseñas por uno más complejo y difícil de romper. Ahora, tanto Red Hat como Debian ofrecen contraseñas por *md5*; normalmente nos dejan escoger el sistema en tiempo de instalación. Hay que tener cuidado con las contraseñas *md5*, ya que si usamos NIS, podríamos tener algún problema; si no, todos los clientes y servidores usarán md5 para sus contraseñas. Las contraseñas se pueden reconocer en /etc/shadow porque vienen con un prefijo "1".

ANOTACIONES

Otras posibles actuaciones consisten en obligar a los usuarios a cambiar la contraseña con frecuencia (el comando *change* puede ser útil), imponer restricciones en el tamaño y el contenido de las contraseñas, y validarlas contra diccionarios de términos comunes.

Respecto a las herramientas, es interesante disponer de un *cracker* de contraseñas (o sea, un programa para obtener contraseñas), para comprobar la situación real de seguridad de las cuentas de nuestros usuarios, y forzar así el cambio en las que detectemos inseguras. Dos de las más utilizadas por administradores son John the Ripper y "crack". Pueden funcionar también por diccionario, con lo cual, será interesante disponer de algún diccionario ASCII de español (pueden encontrarse algunos por la red). Otra herramienta es "Slurpie"; en este caso se trata de una herramienta que puede probar varias máquinas a la vez.

Una cuestión que debe tenerse en cuenta es que siempre hagamos estas pruebas sobre nuestros sistemas. No hay que olvidar que los administradores de otros sistemas (o el proveedor de acceso o ISP) tendrán sistemas de detección de intrusos habilitados y podemos ser objeto de denuncia por intentos de intrusión, ya sea ante las autoridades competentes (unidades de delitos informáticos) o en nuestro ISP para que se nos cierre el acceso. Hay que tener mucho cuidado con el uso de herramientas de seguridad, que tienen siempre el doble filo de la navaja, por si son de seguridad o de intrusión.

9.3.3. *Suid y sticky bits*

Otro problema importante son algunos permisos especiales que son utilizados sobre ficheros o *script*.

El bit *sticky* se utiliza sobre todo en directorios temporales en que queremos que en algunos grupos (a veces no relacionados), cualquier usuario pueda escribir, pero sólo pueda borrar bien el propietario del directorio, o bien el propietario del fichero que esté en el directorio. Un ejemplo clásico de este bit, es el directorio temporal /tmp. Hay que vigilar que no haya directorios de este tipo, ya que pueden permitir que cualquiera escriba en ellos, por lo que habrá que comprobar que no haya más que los puramente necesarios como temporales. El bit se coloca mediante

ANOTACIONES

(*chmod +t dir*), y puede quitarse con *-t*. En un *ls* aparecerá como un directorio con permisos *drwxrwxrwt* (fijaos en la última *t*).

El bit *setuid* permite a un usuario ejecutar (ya sea un ejecutable o un *shell script*) con los permisos de otro usuario. Esto en algún caso puede ser de utilidad, pero es potencialmente peligroso. Es el caso, por ejemplo, de programas con *setuid* de *root*: un usuario, aunque no tiene permisos de *root*, puede ejecutar un programa con *setuid* que puede disponer de permisos internos de usuario *root*. Esto es muy peligroso en caso de *scripts*, ya que podrían editarse y modificarse para hacer cualquier cosa. Por lo tanto, hay que tener controlados estos programas, y en caso de que no se necesite el *setuid*, eliminarlo. El bit se coloca mediante *chmod +s*, ya sea aplicándolo al propietario (se llama entonces *suid*) o al grupo (se llama bit *sgid*); puede quitarse con *-s*. En el caso de visualizar con *ls*, el fichero aparecerá con *–rwSrw-rw* (fijaos en la *S*), si es sólo *suid*, en *sgid* la *S* aparecería tras la segunda *w*.

En caso de utilizar *chmod* con notación octal, se usan cuatro cifras, donde las tres últimas son los permisos clásicos *rwxrwxrwx* (recordad que hay que sumar en la cifra 4 para *r*, 2 *w*, y 1 para *x*), y la primera tiene un valor para cada permiso especial que se quiera (que se suman): 4 (para *suid*), 2 (*sgid*), y 1 (para *sticky*).

9.3.4. Habilitación de *hosts*

En el sistema hay unos cuantos ficheros de configuración especiales que permiten habilitar el acceso a una serie de *hosts* a algunos servicios de red, pero cuyos errores pueden permitir atacar después la seguridad local. Nos podemos encontrar con:

- *.rhosts* de usuario: permite que un usuario pueda especificar una serie de máquinas (y usuarios) que pueden usar su cuenta mediante comandos "r" (*rsh*, *rcp*, ...) sin necesidad de introducir la contraseña de la cuenta. Esto es potencialmente peligroso, ya que una mala configuración del usuario podría permitir entrar a usuarios no deseados, o que un atacante (con acceso a la cuenta del usuario) cambiase las direcciones en *.rhosts* para poder entrar cómodamente sin ningún control. Normalmente, no se tendría que

ANOTACIONES

permitir crear estos archivos, e incluso habría que borrarlos completamente y deshabilitar los comandos "r".

- /etc/hosts.equiv: es exactamente lo mismo que los ficheros *.rhosts* pero a nivel de máquina, especificando qué servicios, qué usuarios y qué grupos pueden acceder sin control de contraseña a los servicios "r". Además, un error como poner en una línea de ese fichero un "+", permite el acceso a "cualquier" máquina. Normalmente, hoy en día tampoco suele existir este fichero, y siempre existe la alternativa del servicio ssh a los "r".

- /etc/hosts.lpd: en el sistema de impresión LPD se utilizaba para poner las máquinas que podían acceder al sistema de impresión. Hay que tener mucho cuidado, si no estamos sirviendo, de deshabilitar completamente el acceso al sistema, y en caso de que sí lo estemos, restringir al máximo las máquinas que realmente hacen uso del mismo. O intentar cambiar a un sistema CUPS o LPRng, que tienen mucho más control sobre los servicios. El sistema LPD es un blanco habitual de ataques de tipo gusano o *buffer overflows*, y están documentados varios *bugs* importantes. Hay que estar atentos si utilizamos este sistema y el fichero *hosts.lpd*.

9.3.5. Módulos PAM

Los módulos PAM [Peñ03][Mor03] son un método que permite al administrador controlar cómo se realiza el proceso de autentificación de los usuarios para determinadas aplicaciones. Las aplicaciones tienen que haber sido creadas y enlazadas a las bibliotecas PAM. Básicamente, los módulos PAM son un conjunto de bibliotecas compartidas que pueden incorporarse a las aplicaciones como método para controlar la autentificación de sus usuarios. Es más, puede cambiarse el método de autentificación (mediante la configuración de los módulos PAM), sin que sea necesario cambiar la aplicación.

Los módulos PAM (las bibliotecas) suelen estar en el directorio /lib/security. Y la configuración de PAM está presente en el directorio /etc/pam.d, donde aparece un fichero de configuración de PAM por cada aplicación que está usando módulos PAM. Nos encontramos con la configuración de autentificación de aplicaciones y servicios como ssh, de *login* gráfico de X Window System, como xdm, gdm, kdm, xscreensaver, ... o, por

ANOTACIONES

ejemplo, del *login* del sistema (la entrada por identificador de usuario y contraseña).

La línea típica de estos ficheros tendría este formato:

```
module-type control-flag module-path arguments
```

donde se especifica:

a) tipo de módulo: si es un módulo que requiere que el usuario se autentifique (*auth*), o de acceso restringido (*account*); cosas que hay que hacer cuando el usuario entra o sale (*session*); o bien hay que actualizar la contraseña (*password*).

b) *flags* de control: especifican si es necesario (*required*), si es un requisito previo (*requisite*), si es suficiente (*sufficient*) o si es opcional (*optional*). Ésta es una sintaxis. Hay otra más actual que trabaja en parejas *valor* y *acción*.

c) la ruta (*path*) del módulo.

d) argumentos que se pasan al módulo (dependen de cada módulo).

Nota

Para saber más, se puede ver la "The Linux-PAM System Administrators' Guide":

http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-4.html

Un pequeño ejemplo del uso de módulos PAM (en una distribución Debian), puede ser su uso en el proceso de *login*:

```
auth     requisite   pam_securetty.so
auth     requisite   pam_nologin.so
auth     required    pam_env.so
auth     required    pam_unix.so nullok
account  required    pam_unix.so
session  required    pam_unix.so
session  optional    pam_lastlog.so
session  optional    pam_motd.so
session  optional    pam_mail.so standard noenv
password required    pam_unix.so nullok obscure min = 4 max = 8 md5
```

Esto especifica los módulos PAM necesarios para controlar la autentificación de usuarios en el *login* (entrada al sistema). Uno de los mó-

dulos *pam_unix.so* es el que realmente hace la verificación de la contraseña del usuario (viendo ficheros *passwd*, *shadow*, ...).

Otros controlan su sesión para ver cuándo ha entrado por última vez, o guardan cuándo entra y sale (para el comando *lastlog*), también hay un módulo que se encarga de verificar si el usuario tiene correo por leer (también hay que autentificarse), y otro que controla que cambie la contraseña (si está obligado a hacerlo en el primer *login* que haga) y que tenga de 4 a 8 letras, y puede utilizarse *md5* para la encriptación.

En este ejemplo podríamos mejorar la seguridad de los usuarios: el *auth* y el *password* permiten contraseñas de longitud nula: es el argumento *nullok* del módulo. Esto permitiría tener usuarios con contraseñas vacías (fuente de posibles ataques). Si quitamos este argumento, ya no permitimos contraseñas vacías en el proceso de *login*. Lo mismo puede hacerse en el fichero de configuración de *passwd* (en este caso, el comando de cambio del *password*), que también presenta el *nullok*. Otra posible acción es incrementar en ambos ficheros el tamaño máximo de las contraseñas, por ejemplo, con *max = 16*.

9.3.6. Alteraciones del sistema

Nota

CERT UNIX checklist: http://www.cert.org/tech_tips/usc20_full.html

Otro problema puede ser la alteración de comandos o configuraciones básicas del sistema, mediante la introducción de troyanos o *backdoors* en el sistema, por la simple introducción de software que sustituya o modifique ligeramente el comportamiento del software de sistema.

Un caso típico es la posibilidad de forzar el *root* para que ejecute comandos falsos de sistema; por ejemplo, si el *root* incluyese el "." en su variable de *PATH*, esto permitiría la ejecución de comandos desde su directorio actual, lo que habilitaría para colocar archivos que sustituyesen comandos del sistema y que serían ejecutados en primer término antes que los de sistema. El mismo proceso puede hacerse con un usuario, aunque por ser más limitados sus permisos, puede no afectar tanto al sistema, cuanto más a la propia seguridad del usuario. Otro caso típico es el de las pantallas de *login* falsas, pudiéndose sustituir el típico proceso de *login, passwd*, por un programa falso que almacene las contraseñas introducidas.

ANOTACIONES

En caso de estas alteraciones, será imprescindible usar políticas de auditoría de los cambios, ya sea por medio de cálculo de firmas o sumas (*gpg* o *md5*), o bien mediante algún software de control como Tripwire o AIDE. Para los troyanos podemos hacer diferentes tipos de detecciones, o bien utilizar herramientas como *chkrootkit*, si éstos viniesen de la instalación de algún *rootkit* conocido.

Nota

chkrootkit, ver: http://www.chkrootkit.org

9.4. Seguridad en red

9.4.1. Cliente de servicios

Como clientes de servicios, básicamente tenemos que asegurar que no pongamos en peligro a nuestros usuarios (o se pongan en peligro solos) por usar servicios inseguros. Evitar el uso de servicios que no utilicen encriptación de datos y contraseñas (ftp, telnet, correo no seguro). Utilizar en este caso técnicas de conexión encriptada, como SSH y SSL.

Nota

Como clientes de servicios, tendremos que evitar el uso de servicios inseguros.

Otro punto importante se refiere a la posible sustitución de servidores por otros falsos, o bien a técnicas de secuestro de sesiones. En estos casos tenemos que disponer de mecanismos de autentificación potentes que nos permitan verificar la autenticidad de los servidores (por ejemplo SSH y SSL tienen algunos de estos mecanismos). Y también tendremos que verificar la red, en busca de intrusos que intenten sustituciones de servidores, además de políticas correctas de filtrado de paquetes mediante *firewalls*, que nos permitan sacar nuestros paquetes de petición y usar los servidores adecuados, controlando los paquetes de entrada que recibamos como respuestas.

9.4.2. Servidor: inetd y xinetd

La configuración de los servicios de red [Mou02], tal como hemos visto, se realiza desde varios lugares [Ano99][Hat01][Peñ03]:

- En /etc/inetd.conf o el equivalente directorio/etc/xinetd.d: estos sistemas son una especie de "superservidores", ya que controlan varios servicios hijos y sus condiciones de arranque. El servicio *inetd*

ANOTACIONES

es utilizado en Debian, y *xinetd,* en Red Hat (y en Debian puede ser puesto opcionalmente en sustitución de *inetd*).

- Servidores iniciados en arranque: según el *runlevel* tendremos una serie de servicios arrancados. El arranque se originará en el directorio asociado al *runlevel*, por ejemplo, en Debian el *runlevel* por defecto es el 2, los servicios arrancados serán arrancados desde el directorio /etc/rc2.d, seguramente con enlaces a los *scripts* contenidos en /etc/init.d, en los cuales se ejecutará con el parámetro *start*, *stop*, *restart* según corresponda.

- Otros servicios de tipo RPC: asociados a llamadas remotas entre máquinas son, por ejemplo, utilizados en NIS y NFS. Puede examinarse cuáles hay con el comando *rpcinfo –p*.

Otros ficheros de apoyo (con información útil) son: /etc/services, que está formado por una lista de servicios locales o de red conocidos, junto con el nombre del protocolo (tcp, udp u otros), que se utiliza en el servicio, y el puerto que utiliza; /etc/protocols es una lista de protocolos conocidos; y /etc/rpc es una lista de servidores RPC, junto con los puertos usados. Estos ficheros vienen con la distribución y son una especie de base de datos que utilizan los comandos y herramientas de red para determinar los nombres de servicios, protocolos, o *rpc* y sus puertos asociados.

Una de las primeras acciones a realizar por el administrador será deshabilitar todos aquellos servicios que no esté utilizando o no tenga previsto utilizar, habrá que documentarse sobre la utilización de los servicios [Mou02], y qué software puede necesitarlos. [Neu00]

En el caso de /etc/inetd.conf, sólo tenemos que comentar la línea del servicio que hay que deshabilitar, colocando un "#" como primer carácter en la línea.

En el otro modelo de servicios, utilizado por defecto en Red Hat 9 (y opcional en Debian), el *xinetd*, la configuración reside en el fichero /etc/xinetd.conf, donde se configuran algunos valores por defecto de control de *logs*, y después la configuración de cada servicio hijo se hace a través de un fichero dentro del directorio /etc/xinetd.d. En cada fichero se define la información del servicio, equivalente a la

ANOTACIONES

que aparece en el *inetd.conf*, en este caso, para desactivar un servicio sólo hay que poner una línea *disable = yes* dentro del fichero del servicio. Xinetd tiene una configuración más flexible que *inetd*, al separar la configuración de los diferentes servicios en diferentes archivos, y tiene bastantes opciones para limitar las conexiones a un servicio, en el número o la capacidad de éstas; todo lo cual permite un mejor control del servicio y, configurando adecuadamente, podemos evitar algunos de los ataques por denegación de servicio. (DoS o DDoS)

Respecto al tratamiento de los servicios de los *runlevel* desde comandos de la distribución, ya mencionamos varias herramientas en la unidad de administración local que permitían habilitar o deshabilitar servicios. También existen herramientas gráficas como ksysV de KDE, o el *redhat-config-services* y *ntsysv* en Red Hat. Y a un nivel inferior, podemos ir al nivel de *runlevel* que queramos (/etc/rcx.d), y desactivar los servicios que queramos cambiando las *S* o *K* iniciales del *script* por otro texto: un método sería, por ejemplo: cambiar *S20ssh*, por *STOP_S20ssh*, y ya no arrancará; la próxima vez, cuando lo volvamos a querer, quitamos el prefijo y lo volvemos a tener activo.

Otro aspecto es el cierre de servicios no seguros, tradicionalmente, en el mundo UNIX se habían utilizado servicios de transferencia de archivos como ftp, de conexión remota como telnet, y comandos de ejecución (*login* o copia) remotos, muchos de los cuales comenzaban con la letra "r" (por ejemplo, *rsh*, *rcp*, *rexec*, ...). Otros peligros potenciales son los servicios *finger* y *rwhod*, que permitían obtener información desde la red de los usuarios de las máquinas; aquí el peligro estaba en la información que podía obtener un atacante y que le podía facilitar mucho el trabajo. Todos estos servicios no deberían ser usados actualmente debido a los peligros potenciales que implican. Respecto al primer grupo:

a) ftp, telnet, en las transmisiones por red **no encriptan** las contraseñas, y cualquiera puede hacerse con las contraseñas pde servicios o las cuentas asociadas (por ejemplo, mediante un *sniffer*).

b) *rsh*, *rexec*, *rcp* además presentan el problema de que bajo algunas condiciones ni siquiera necesitan contraseñas (por ejemplo, si se hace desde sitios validados en fichero *.rhosts*), con lo cual vuelven a ser inseguros, y dejar grandes puertas abiertas.

ANOTACIONES

La alternativa es usar clientes y servidores seguros que soporten la encriptación de los mensajes y autentificación de los participantes. Hay alternativas seguras a los servidores clásicos, pero actualmente la solución más usada es mediante la utilización del paquete OpenSSH (que puede combinarse también con OpenSSL para entornos web). OpenSSH ofrece soluciones basadas en los comandos *ssh*, *scp* y *sftp*, permitiendo sustituir a los antiguos clientes y servidores (se utiliza un *daemon* denominado sshd). El comando *ssh* permite las antiguas funcionalidades de telnet, *rlogin*, y *rsh* entre otros, y *scp* sería el equivalente seguro de *rcp*, y *sftp* del *ftp*.

Respecto a SSH, también hay que tener la precaución de usar *ssh version 2* (o simplemente *ssh2*). La primera versión tiene algunos *exploits* conocidos; hay que tener cuidado al instalar OpenSSH, y si no necesitamos la primera versión, instalar sólo soporte para SSH2.

Además, la mayoría de servicios que dejemos activos en nuestras máquinas tendrían después que ser filtrados a través de *firewall*, para asegurar que no fuesen utilizados o atacados por personas a los que no iban destinados.

9.5. Detección de intrusiones

Con los sistemas sistemas de detección de intrusos [Hat01] (IDS) se quiere dar un paso más adelante. Una vez hayamos podido configurar más o menos correctamente nuestra seguridad, el paso siguiente consistirá en una detección y prevención activa de las intrusiones.

Nota

Los sistemas IDS nos permiten la detección a tiempo de intrusos, usando nuestros recursos o explorando nuestros sistemas en busca de fallos de seguridad.

Los sistemas IDS crean procedimientos de escucha y generación de alertas al detectar situaciones sospechosas, o sea, buscamos síntomas de posibles accidentes de seguridad.

Los hay desde sistemas basados en la información local, por ejemplo, recopilan información de los *log* del sistema, vigilan cambios en los ficheros de sistema o bien en las configuraciones de los servicios típicos. Otros sistemas están basados en red e intentan verificar que no se produzcan comportamientos extraños, como por ejemplo los casos de *spoofing*, donde hay falsificaciones de direcciones conocidas; contro-

ANOTACIONES

lar tráfico sospechoso, posibles ataques de denegación de servicio, detectando tráfico excesivo hacia determinados servicios, controlando que no hay interfaces de red en modo promiscuo (síntoma de *sniffers* o capturadores de paquetes).

Ejemplo

Algunos ejemplos de herramientas IDS serían: Logcheck (verificación de *logs*), TripWire (estado del sistema mediante sumas *md5* aplicadas a los archivos), AIDE (una version libre de TripWire), Snort (IDS de verificación de estado de una red completa).

9.6. Protección mediante filtrado (*wrappers* y *firewalls*)

Los **TCP wrappers** [Mou02] son un software que actúa de intermediario entre las peticiones del usuario de servicio y los *daemons* de los servidores que ofrecen el servicio. Muchas de las distribuciones vienen ya con los *wrappers* activados y podemos configurar los niveles de acceso. Los *wrappers* se suelen utilizar en combinación con *inetd* o *xinetd*, de manera que protejan los servicios que ofrecen.

El *wrapper* básicamente sustituye al *daemon* (demonio) del servicio por otro que actúa de intermediario (llamado *tcpd*, normalmente en /usr/sbin/tcpd). Cuando éste recibe una petición, verifica el usuario y origen de ésta, para determinar si la configuración del *wrapper* del servicio permite o no utilizarlo. Además, incorpora facilidades de generar *logs*, o bien informar por correo de los posibles intentos de acceso y después ejecuta el *daemon* adecuado asignado al servicio.

Por ejemplo, supongamos la siguiente entrada en *inetd*:

```
finger stream tcp nowait nobody /usr/etc/in.fingerd in.fingerd
```

Ésta se cambia por:

```
finger stream tcp nowait nobody /usr/sbin/tcpd in.fingerd
```

de manera que, cuando llegue una petición, será tratada por el *daemon tcpd* que se encargará de verificar el acceso (para más detalles, ver página *man tcpd*).

Nota

Los *wrappers* nos permiten controlar la seguridad mediante listas de acceso a nivel de servicios.

También existe un método alternativo de *wrapper* TCP, que consiste en que la aplicación original se compile con la biblioteca de *wrappers*. Entonces la aplicación no tiene por qué estar en *inetd*, y podremos controlarla igual que el primer caso con la configuración que comentamos a continuación.

El sistema de *wrappers* se controla donde los ficheros /etc/hosts.deny, en que especificamos qué servicios denegamos y a quién, por medio de opciones, como un pequeño *shell* para guardar la información del intento, y el fichero /etc/hosts.allow, donde solemos colocar qué servicio vamos a utilizar, seguido de la lista de a quién dejamos entrar (más adelante, en el taller, veremos un pequeño ejemplo. También existen los comandos *tcpdchk*, que testan la configuración de los ficheros *hosts* (ver *man hosts_access* y *hosts_options*), para comprobar que son correctos, es decir, testa la configuración. El otro comando útil es *tcpdmatch*, al cual damos un nombre de servicio y un posible cliente (usuario, y/o *host*), y nos dice qué haría el sistema ante esta situación.

9.6.1. *Firewalls*

Un *firewall* (o cortafuegos) es un sistema o grupo de sistemas que refuerza las políticas de control de acceso entre redes. El *firewall* puede estar implementado en software, como una aplicación especializada corriendo en un computador individual, o bien puede tratarse de un dispositivo especial dedicado a proteger uno o más computadores.

En general dispondremos, o bien de la aplicación de *firewall* para proteger una máquina concreta conectada directamente a Internet (ya sea directa o por proveedor), o bien podemos colocar en nuestra red una o varias máquinas dedicadas a esta función, de modo que protejan nuestra red interna.

Técnicamente, la mejor solución es disponer de un computador con dos o más tarjetas de red que aíslen las diferentes redes (o segmentos de red) conectados, de manera que el software de *firewall* en la máquina (o si fuese un hardware especial) se encargue de conectar

ANOTACIONES

los paquetes de las redes y determinar cuáles pueden pasar o no, y a qué red.

Este tipo de *firewall* suele combinarse con un *router* para enlazar los paquetes de las diferentes redes. Otra configuración típica es la de *firewall* hacia Internet, por ejemplo con dos tarjetas de red: en una obtenemos/proporcionamos trafico a Internet y en la otra enviamos o proporcionamos el tráfico a nuestra red interna, pudiendo así eliminar el tráfico que no va destinado a nosotros, y también controlar el tráfico que se mueve hacia Internet, por si no queremos que se tenga acceso a algunos protocolos, o bien sospechamos que hay posibilidades de fugas de información por algunos ataques. Una tercera posibilidad es la máquina individual conectada con una única tarjeta de red hacia Internet directa o bien a través de un proveedor. En este caso, sólo queremos proteger nuestra máquina de ataques de intrusos, de tráfico no deseado o de que se vea comprometida al robo de datos.

Es decir, en estos casos podemos comprobar que el *firewall* –dependiendo de si es software o no, de si la máquina tiene una o varias tarjetas de red o de si protege a una máquina individual o a una red– puede tener configuraciones y usos diferentes.

El *firewall* en general permite definir al usuario una serie de políticas de acceso (cuáles son las máquinas a las que se puede conectar o las que pueden recibir información y el tipo de información) por medio del control de los puertos TCP/UDP permitidos de entrada (*incomming*) o de salida (*outcomming*). Algunos *firewalls* vienen con políticas preconfiguradas; en algún caso sólo dicen si se quiere un nivel de seguridad alto, medio o bajo; otros permiten personalizar las opciones totalmente (máquinas, protocolos, puertos, etc.).

Nota

Los *firewall* permiten establecer seguridad a nivel de paquetes y conexiones de comunicación.

Otra técnica a veces relacionada es NAT (*Network Address Translation*). Esta técnica proporciona una vía para ocultar las direcciones IP usadas en la red privada y las oculta de Internet, pero mantiene el acceso desde las máquinas. Uno de los métodos típicos es el denominado *masquerading*. Utilizando *NAT masquerading*, uno o varios dispositivos en la red pueden aparecer como una única dirección IP vistos desde fuera. Esto permite conectar varios computadores a un único dispositivo de conexión externa; por ejemplo, un caso de *router* ADSL

ANOTACIONES

en casa permite conectar varias máquinas sin necesidad de que el proveedor nos proporcione varias direcciones IP. Los *routers* ADSL suelen ofrecer algún tipo de *NAT masquerading*, y también posibilidades de *firewall*. Suele ser bastante común utilizar una combinación de ambas técnicas. En este caso, entra en juego, además de la configuración de la máquina del *firewall* (los casos vistos antes), la configuración de la red privada interna que queremos proteger.

9.6.2. Netfilter: IPtables

Nota

Netfilter, ver: http://www.netfilter.org

Ipchains, ver: http://www.netfilter.org/ipchains/

El *kernel* Linux (a partir 2.4) proporciona un subsistema de filtrado denominado Netfilter [Net03] que proporciona características de filtrado de paquetes y también NAT. Este sistema permite usar diferentes interfaces de filtrado, entre ellas, la más usada se denomina IPtables. El comando principal de control es *iptables*. Anteriormente [Hat03a], se proporcionaba otro filtrado denominado *ipchains* en los *kernel* 2.2 [Gre00], el sistema tenía una sintaxis diferente (aunque parecida). En los *kernel* 2.0 se utilizaba otro sistema denominado *ipfwadm*. Aquí (y en los ejemplos posteriores) vamos a tratar sólo con Netfilter/IPTables (es decir, con *kernels* 2.4).

Nota

IPTables aporta diferentes elementos como las tablas, *chains*, y las propias reglas.

La interfaz del comando *iptables* permite realizar las diferentes tareas de configuración de las reglas que afectan al sistema de filtrado: ya sea generación de *logs*, acciones de *pre* y *post routing* de paquetes, NAT, y *forwarding* (reenvío) de puertos.

Arranque del servicio con: */etc/init.d/iptables start*, si no estaba configurado ya en el *runlevel*.

El comando *iptables -L* lista las reglas activas en ese momento en cada una de las cadenas (*chains*). Si no se han configurado previamente, suelen ser de aceptar todos los paquetes de las cadenas (o *chains*) de *input* (entrada), *output* (salida) y *forward* (reenvío).

El sistema de IPTables tiene como nivel superior las tablas. Cada una contienen diferentes cadenas, que a su vez contienen diferentes reglas. Las tres tablas que existen son: **Filter**, **NAT** y **Mangled**. La primera sirve para las propias normas de filtrado, la segunda para realizar traslación de direcciones dentro de un sistema que utilice

ANOTACIONES

NAT, y la tercera, menos usada, sirve para especificar algunas opciones de control de los paquetes, y cómo controlarlos. Concretamente, si estamos con un sistema directamente conectado a Internet, utilizaremos tan sólo la tabla Filter. Si el sistema está en una red privada que tiene que pasar por un *router*, *gateway* o *proxy* (o combinación de éstos), seguramente dispondremos de un sistema de *NAT* o *IP masquerading;* si estamos configurando precisamente la máquina que permite acceso externo, tendremos que tocar la tabla NAT y la Filter. Si la máquina está en un sistema de red privada, pero es una de las máquinas internas, será suficiente con la tabla Filter, a no ser que sea un servidor el que haga NAT a otro segmento de red.

Si un paquete llega al sistema, en el *firewall* se mirará primero si existen reglas en la tabla NAT, por si hay que hacer traducciones de direcciones hacia la red interna (las direcciones normalmente no son visibles hacia fuera); después se mirarán las reglas de la tabla Filter para decidir si se van a dejar pasar los paquetes, o si no son para nosotros, y tenemos reglas *forward* para saber hacia dónde los reenviamos. Por el contrario, cuando nuestros procesos generan paquetes, las reglas *output* de la tabla Filter controlan si los dejamos salir o no, y si hubiese sistema NAT, las reglas efectuarían la traducción de direcciones de manera que se enmascarasen. En la tabla NAT suele haber dos cadenas: *prerouting* y *postrouting*. En la primera, las reglas han de decidir si hay que hacer algún *routing* del paquete y cuál será la dirección de destino. En el segundo, se decide finalmente si el paquete se pasa o no hacia el interior (la red privada, por ejemplo). Y también existe una cadena *output* para el tráfico que se genere localmente de salida a la red privada, ya que *prerouting* no lo controla (para más detalles, podéis ver *man iptables*).

A continuación comentaremos algunos aspectos y ejemplos de configuración de la tabla Filter (para las otras tablas, podéis consultar la bibliografía).

La configuración típica de la tabla Filter es de una serie de reglas que especifican qué se hace dentro de una determinada cadena (o *chain*), como las tres anteriores (*input, output* o *forward*). Normalmente, se especifica:

```
iptables -A chain -j target
```

ANOTACIONES

donde *chain* es *input, output* o *forward*, y *target* el destino que se le va a dar al paquete que se corresponda con la regla. La opción *-A* añade la regla a las existentes. Con esto hay que tener cuidado, ya que el orden importa. Hay que colocar las menos restrictivas al principio, puesto que, si primero ponemos una regla que elimine los paquetes, aunque haya otra regla, ésta no será tenida en cuenta. La opción *-j* permite decidir qué haremos con los paquetes, típicamente *accept* ('aceptar'), *reject* ('rechazar'), o *drop* (simplemente 'perderlo'). Es importante la diferencia entre *reject* y *drop*. Con el primero, rechazamos el paquete y normalmente informamos al emisario de que hemos rechazado el intento de conexión (normalmente por un paquete de tipo ICMP). Con el segundo (*drop*), simplemente perdemos el paquete como si nunca hubiese existido y no enviamos ningún tipo de respuesta. Otro *target* utilizado es *log*, para enviar el paquete al sistema de *log*. Normalmente en este caso hay dos reglas, una con el *log* y otra igual con *accept, drop* y *reject*, para permitir enviar al *log* la información de paquetes aceptados, rechazados o perdidos.

Al poner la regla, también puede usarse la opción *-I* (insertar) para indicar una posición, por ejemplo:

```
iptables -I INPUT 3 -s 10.0.0.0/8 -j ACCEPT
```

que nos dice que se coloque la regla en la cadena *input* en tercera posición; y se van a aceptar paquetes (*-j*) que provengan (con fuente, o *source*, *-s*) de la subred 10.0.0.0 con *netmask* 255.0.0.0. Con *-D* de forma parecida podemos borrar bien un número de regla, bien la regla exacta, como se especifica a continuación, bien borrando la primera regla de la cadena o la regla que mencionamos:

```
iptables -D INPUT 1
iptables -D INPUT -s 10.0.0.0/8 -j ACCEPT
```

También hay reglas que permiten definir una "política" por defecto de los paquetes (opción *-P*); se va ha hacer con todos los paquetes lo mismo. Por ejemplo, se suele decir que se pierdan todos los paquetes por defecto, y se habilitan luego los que interesan, y muchas veces

ANOTACIONES

también se evita que haya *forwarding* de paquetes si no es necesario (si no actuamos de *router*), esto podría ponerse como:

```
iptables -P INPUT DENY
iptables -P OUTPUT REJECT
iptables -P FORWARD REJECT
```

Todo lo cual establece unas políticas por defecto que consisten en denegar la entrada de paquetes, no permitir salir y no reenviar paquetes. Ahora se podrán añadir las reglas que conciernen a los paquetes que deseemos utilizar, diciendo qué protocolos, puertos y orígenes o destinos queremos permitir o evitar. Esto puede ser difícil, ya que tenemos que conocer todos los puertos y protocolos que utilice nuestro software o servicios. Otra táctica sería dejar sólo activos aquellos servicios que sean imprescindibles y habilitar con el *firewall* el acceso de los servicios a las máquinas deseadas.

Algunos ejemplos de estas reglas de la tabla Filter podrían ser:

1) `iptables -A INPUT -s 10.0.0.0/8 -d 192.168.1.2 -j DROP`

2) `iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with tcp-reset`

3) `iptables -I INPUT -p tcp --dport 113 -s 10.0.0.0/8 -j ACCEPT`

Donde:

1) Perdemos los paquetes que vengan de 10.x.x.x con destino a 192.168.1.2.

2) Rechazamos los paquetes tcp con destino al puerto 113, emitiendo una respuesta de tipo *tcp-reset*.

3) Los mismos paquetes que en 2) pero que provengan de 10.x.x.x serán aceptados.

Respecto a los nombres de protocolos y puertos, el sistema *iptables* utiliza la información proporcionada por los ficheros /etc/services y /etc/protocols, pudiéndose especificar la información (de puerto y

ANOTACIONES

protocolo) de forma numérica o bien por nombre (hay que tener cuidado en este caso de que la información de los ficheros sea correcta, que no hayan sido modificados, por ejemplo, por un atacante).

La configuración de *iptables* suele establecerse mediante llamadas consecutivas al comando *iptables* con las reglas. Esto crea un estado de reglas activas que pueden consultarse con *iptables -L*, si deseamos salvarlas para que sean permanentes, podemos hacerlo en Red Hat con:

```
/etc/init.d/iptables save
```

Y se guardan en:

```
/etc/sysconfig/iptables
```

En Debian puede hacerse:

```
/etc/init.d/iptables save nombre-reglas
```

Hay que tener cuidado de que exista previamente el directorio /var/*log*/iptables, que es donde se guardan los ficheros; *nombre-reglas* será un fichero en el directorio.

Con (*/etc/init.d/iptables load*) podemos cargar las reglas (en Debian hay que dar el nombre del fichero de reglas), aunque Debian soporta unos nombres por defecto de ficheros, que son *active* para las reglas normales (las que se usarán en un *start* del servicio) e *inactive* para las que quedarán cuando se desactive el servicio (se haga un *stop*). Otra aproximación comúnmente usada es la de colocar las reglas en un fichero *script* con las llamadas *iptables* que se necesiten y llamarlas por ejemplo colocándolas en el *runlevel* necesario, o con enlace hacia el *script* en /etc/init.d.

9.6.3. Paquetes de *firewalls* en las distribuciones

Respecto a herramientas de configuración más o menos automáticas del *firewall*, existen varias posibilidades, pero hay que tener en cuen-

ANOTACIONES

ta que no suelen ofrecer las mismas prestaciones que la configuración manual de *iptables* (que en la mayoría de casos sería el proceso recomendado). Algunas herramientas son:

- **lokkit**: en Red Hat, muy básico, sólo permite elegir al usuario un nivel de seguridad que desea (alto, medio o bajo). Después enseña los servicios que se ven afectados y podemos dejar, o no, pasar al servicio cambiando la configuración por defecto. El mecanismo utilizado por debajo es *iptables*. Puede verse la configuración final que realiza de reglas /etc/sysconfig/iptables que, a su vez, es leído por el servicio *iptables,* que se carga en arranque, o bien por parada o arranque mediante /etc/init.d/iptables con las opciones *start* o *stop*. En Debian también es posible instalarlo, pero deja la configuración de las reglas en /etc/defaults/lokkit-l, y un *script* en /etc/init.d/lokkit-l. También existe una versión gráfica llamada *gnome-lokkit*.

- **Bastille** [Pro03a]: programa de seguridad bastante completo y didáctico, ya que nos explica paso a paso diferentes recomendaciones de seguridad y si las queremos aplicar, así como la configuración del *firewall* (el programa es *interactiveBastille*). Funciona en varias distribuciones, tanto en Red Hat como en Debian.

- **fwbuilder**: una herramienta que permite construir las reglas del *firewall* de forma gráfica. Se puede utilizar en varios operativos (GNU/Linux tanto Red Hat como Debian, OpenBSD, MacOS), con diferentes tipos de *firewalls* (*iptables* incluido).

Nota

Ver: http://www.fwbuilder.org/

Normalmente, cada uno de estos paquetes utiliza un sistema de reglas que guarda en algún fichero propio de configuración, y que suele arrancar como servicio o como ejecución de *script* en el *runlevel* por defecto.

9.6.4. Consideraciones finales

Aunque dispongamos de *firewalls* bien configurados, hay que tener presente que no son una medida de seguridad absoluta, ya que hay ataques complejos que pueden saltarse el control, o bien falsear datos que creen confusión. Además, las necesidades de conectividad modernas obligan a veces a crear software que permita el *bypass* (cruce) de los *firewalls*:

Nota

Nunca se debe confiar en un único mecanismo o sistema de seguridad. Hay que establecer la seguridad del sistema a diferentes niveles.

- Tecnologías como IPP, protocolo de impresión utilizado por CUPS, o el WebDAV, protocolo de autoría y actualización de sitios web, permiten pasar (o es necesario que pasen) las configuraciones de los *firewalls*.

- A menudo se utiliza (por ejemplo, los anteriores protocolos y otros) una técnica denominada *tunneling*, que básicamente encapsula protocolos no permitidos, sobre la base de otros que sí que lo están; por ejemplo, si un *firewall* permite sólo paso de tráfico http (puerto 80 por defecto), es posible escribir un cliente y un servidor (cada uno a un lado diferente del *firewall*) que hablen cualquier protocolo conocido por ambos, pero que en la red es transformado en un protocolo http estándar, con lo cual, el tráfico puede cruzar el *firewall*.

- Los códigos móviles por web (ActiveX, Java, y JavaScript), cruzan los *firewalls*, y por lo tanto es difícil proteger los sistemas si éstos son vulnerables a los ataques contra agujeros descubiertos.

Así, aunque los *firewalls* son una muy buena solución a la mayor parte de la seguridad, siempre pueden tener vulnerabilidades y dejar pasar tráfico que se considere válido, pero que incluya otras fuentes posibles de ataques o vulnerabilidades. En seguridad **nunca** hay que considerar (y confiar en) **una única solución**, y esperar que nos proteja absolutamente; hay que examinar los diversos problemas, plantear soluciones que los detecten a tiempo y políticas de prevención que nos curen en salud, por lo que pueda pasar.

9.7. Herramientas de seguridad

Algunas de las herramientas pueden considerarse también herramientas de ataque. Por lo tanto, se recomienda probar estas herramientas sobre máquinas de nuestra red local o privada; nunca hacerlo con IP de terceros, ya que éstos podrían tomarlo como intrusiones y pedirnos responsabilidades, o a nuestro ISP, o avisar a las autoridades competentes para que nos investiguen o cierren nuestro acceso.

A continuación, comentamos brevemente algunas herramientas, así como algunos usos que pueden utilizar:

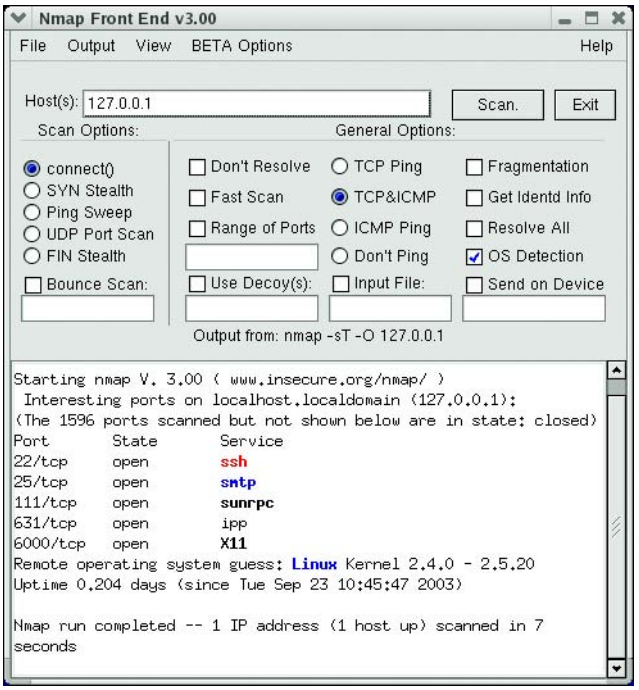
a) **TripWire**: esta herramienta mantiene una base de datos de sumas de comprobación de los ficheros importantes del sistema.

ANOTACIONES

Puede servir como sistema IDS preventivo. Nos sirve para "tomar" una foto del sistema, poder comparar después cualquier modificación introducida y comprobar que éste no haya sido corrompido por un atacante. El objetivo aquí es proteger los archivos de la propia máquina, evitar que se produzcan cambios como, por ejemplo, los que podría haber provocado un *rootkit*. Así, cuando volvamos a ejecutar la herramienta, podemos comprobar los cambios respecto a la ejecución anterior. Hay que elegir un subconjunto de archivos que sean importantes en el sistema, o fuentes de posibles ataques. TripWire es propietario, pero hay una herramienta libre equivalente llamada AIDE.

b) **Nmap** [Ins03b]: es una herramienta de escaneo de puertos para redes grandes. Puede escanear desde máquinas individuales a segmentos de red. Permite diversos modelos de *scan*, dependiendo de las protecciones que tenga el sistema. También tiene técnicas que permiten determinar el sistema operativo que usan las máquinas remotas. Puede emplear diferentes paquetes de TCP y UDP para probar las conexiones. Existe una interfaz gráfica denominada *xnmap*.

Figura 23. xnmap analizando los servicios locales



c) **Ethereal**: es un analizador de protocolos y captura el tráfico de la red (es un *sniffer*). Permite visualizar el tráfico capturado, ver es-

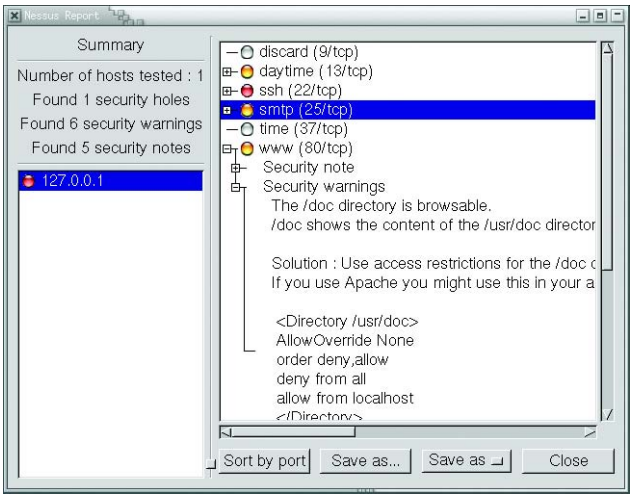
ANOTACIONES

tadísticas y datos de los paquetes individuales, y agruparlos, ya sea por origen, destino, puertos o protocolo. Puede incluso reconstruir el tráfico de una sesión entera de un protocolo TCP.

d) **Snort** [Sno03]: es un sistema IDS que permite realizar análisis de tráfico en tiempo real y guardar *logs* de los mensajes. Permite realizar análisis de los protocolos, búsquedas por patrones (protocolo, origen, destino, etc.). Puede ser usado para detectar diversos tipos de ataques. Básicamente, analiza el tráfico de la red para detectar patrones que puedan corresponder a un ataque. El sistema utiliza una serie de reglas para, o bien anotar la situación (*log*), o bien producir un aviso (*alert*) o descartar la información (*drop*).

e) **Nessus** [Nes03]: es un detector de vulnerabilidades conocidas (ya soporta más de 1.200) y asesora sobre cómo mejorar la seguridad para las detectadas. Es un programa modular que incluye una serie de *plugins* para los diferentes análisis. Utiliza una arquitectura cliente-servidor, con un cliente gráfico que muestra los resultados y el servidor que realiza las diferentes comprobaciones sobre las máquinas. Tiene capacidad para examinar redes enteras. Genera informes de resultados en forma de *reports* que pueden exportarse a diferentes formatos (por ejemplo, HTML).

Figura 24. Cliente de Nessus mostrando el informe de vulnerabilidades y posibles soluciones



Podemos encontrar muchas más herramientas de seguridad. Un buen sitio para consultar es http://www.insecure.org/tools.html, donde los

creadores del Nmap mantienen una lista de herramientas votadas por los usuarios.

9.8. Análisis *logs*

Dando una visión sobre los ficheros de *log* [Ano99][Fri02], podemos hacernos una idea rápida del estado global del sistema, así como de las últimas ocurrencias, y detectar accesos (o intentos) indebidos. Cuidado, los *logs*, si ha habido una intrusión real, pueden haber sido limpiados o falseados. La mayoría de archivos de *log* residen en el directorio /var/log.

Muchos de los servicios pueden poseer *logs* propios, que normalmente se establecen en la configuración (mediante el correspondiente fichero de configuración). La mayoría suelen utilizar las facilidades de *log* incorporadas en el sistema Syslog, mediante el demonio Syslogd. Su configuración reside en /etc/syslog.conf. Esta configuración suele establecerse por niveles de mensajes: existen diferentes tipos de mensajes según su importancia. Normalmente suelen aparecer niveles *debug, info, err, notice, warning, err, crit, alert, emerg*, en que más o menos ésta sería la ordenación de importancia de los mensajes (de menor a mayor). Normalmente, la mayoría de los mensajes se dirigen al *log* /var/log/messages, pero puede definirse que cada tipo vaya a ficheros diferentes y también se puede identificar quién los ha originado; típicamente el *kernel*, correo, *news*, el sistema de autentificación, etc.

Por lo tanto, cabe examinar (y en todo caso adaptar) la configuración de Syslog para conocer en qué *logs* podemos encontrar/generar la información. Otro punto importante es controlar su crecimiento, ya que según los que estén activos y las operaciones (y servicios) que se realicen en el sistema, los *logs* pueden crecer bastante. En Debian y Red Hat se controla a través de *logrotated*, un demonio que se encarga periódicamente de hacer copias y comprimirlas de los *logs* más antiguos, se puede encontrar su configuración general en /etc/logrotate.conf, algunas aplicaciones hacen configuraciones específicas que se pueden encontrar en el directorio /etc/logrotate.d.

ANOTACIONES

Comentamos en los siguientes puntos algunos ficheros de *log* que habría que tener en cuenta (quizás los más utilizados):

a) /var/log/messages: es el fichero de *log* por defecto del demonio Syslogd, pero habría que revisar su configuración, no sea que esté cambiado a otro sitio, o haya varios. Este fichero contiene una amplia variedad de mensajes de orígenes diversos (diferentes demonios, servicios o el mismo *kernel*); todo lo que se observase anómalo tendría que ser verificado. En caso de que haya habido una intrusión, mirar alrededor de la fecha de la intrusión.

b) /var/log/utmp: este fichero contiene información binaria para cada usuario que está actualmente activo. Es interesante para determinar quién está dentro del sistema. El comando *who* utiliza este fichero para proporcionar esta información.

c) /var/log/wtmp: cada vez que un usuario entra en el sistema, sale, o la máquina reinicia, se guarda una entrada en este fichero. Es un fichero binario de donde el comando *last* obtiene información, en que se menciona qué usuarios entraron o salieron, cuándo y dónde se originó la conexión. Puede ser útil para buscar dónde (en qué cuentas) se originó la intrusión o detectar usos de cuentas sospechosas. También existe una variación del comando llamada *lastb*, que lista los intentos de *login* que no pudieron validarse correctamente y se usa el fichero /var/log/btmp (puede ser necesario crearlo si no existe). Estos mismos fallos de autentificación también suelen enviarse al *log auth.log*. De modo parecido, el comando *lastlog* utiliza otro fichero /var/loglastlog para verificar cuál fue la última conexión de cada uno de los usuarios.

d) /var/log/secure: suelen utilizarse en Red Hat para enviar los mensajes de los *tcp wrappers* (o los *firewalls*). Cada vez que se establece una conexión a un servicio de *inetd*, o bien en el caso de Red Hat 9, para *xinetd* (con su propia seguridad) se añade un mensaje de *log* a este fichero. Pueden buscarse intentos de intrusos de servicios que no suelen usarse, o bien máquinas no familiares que se intentan conectar.

En el sistema de *logs*, otra cosa que habría que asegurar es que el directorio de logs /var/log, sea sólo escribible por el *root*. De otro modo, cualquier atacante podría falsificar la información de los *logs*. Por ejemplo, si el atacante consigue acceso a *root*, puede, y suele, borrar las pista de sus accesos.

ANOTACIONES

9.9. Taller: análisis de la seguridad mediante herramientas

Vamos a realizar algunos de los procesos descritos sobre un sistema Debian, para configurar mejor su seguridad.

Primero examinaremos qué ofrece nuestra máquina a la red. Para ello, utilizaremos la herramienta *nmap* como escaneador de puertos. Con el comando (desde *root*):

```
nmap -sTU -O localhost
```

obtenemos:

```
root@maquina:~# nmap -sUT -O localhost
starting nmap 3.27 ( www.insecure.org/nmap/ ) at 2003-09-17 11:31 CEST
Interesting ports on localhost (127.0.0.1):
(The 3079 ports scanned but not shown below are in state: closed)
Port          State         Service
9/tcp         open          discard
9/udp         open          discard
13/tcp        open          daytime
22/tcp        open          ssh
25/tcp        open          smtp
37/tcp        open          time
37/udp        open          time
80/tcp        open          http
111/tcp       open          sunrpc
111/udp       open          sunrpc
113/tcp       open          auth
631/tcp       open          ipp
728/udp       open          unknown
731/udp       open          netviewdm3
734/tcp       open          unknown
Remote operating system guess: Linux kernel 2.4.0-2.5.20
Uptime 2.011 days (since Mon Sep 15 11:14:57 2003)
Nmap run completed --1 IP address (1 host up) scanned in 9.404 seconds
```

Podemos observar que nos ha detectado un gran número de servicios abiertos (dependiendo de la máquina podría haber más: telnet, ftp, finger...), tanto en protocolos *tcp* como *udp*. Algunos servicios como *discard, daytime, time* pueden ser útiles en alguna ocasión, pero normalmente no tendrían que estar abiertos a red, ya que se consideran inseguros. SMTP es el servicio de reenvío, y *routing*, el de correo; si actuamos como *host* o servidor de correo, tendría que estar activo; pero si sólo leemos y escribimos correo mediante cuentas POP3 o IMAP, no tiene por qué estarlo.

Otra manera de detectar servicios activos sería mediante la búsqueda de puertos activos a la escucha, esto puede hacerse con *netstat -lut*.

El servicio *nmap* también puede aplicarse con el nombre DNS o IP de la máquina, así vemos lo que se vería desde el exterior (con *localhost* vemos lo que puede ver la propia máquina), o mejor incluso podríamos utilizar una máquina de una red externa (por ejemplo, un PC cualquiera conectado a Internet), para examinar lo que verían de nuestra máquina desde fuera.

Vamos ahora a ir a /etc/inetd.conf para desactivar estos servicios. Buscamos líneas como:

```
discard stream tcp nowait root internal
```

```
smtp stream tcp nowait mail /usr/sbin/exim exim -bs
```

y les colocamos un # al principio de la línea (sólo a aquéllas de los servicios que queramos desactivar y sepamos qué hacen realmente (consultar páginas *man*), o se recomiende su desactivación. Otro caso de desactivación recomendado sería el de los servicios de ftp, telnet, finger, ... y usar ssh para sustituirlos.

Ahora tenemos que reiniciar *inetd* para que vuelva a leer la configuración que hemos cambiado: */etc/init.d/inetd restart*.

ANOTACIONES

Volvemos a *nmap*:

```
22/tcp        open          ssh
80/tcp        open          http
111/tcp       open          sunrpc
111/udp       open          sunrpc
113/tcp       open          auth
631/tcp       open          ipp
728/udp       open          unknown
734/tcp       open          unknown
```

De lo que nos queda, tenemos el servicio ssh, que queremos dejar, y el servidor de web, que lo pararemos de momento:

```
/etc/init.d/apache stop
```

ipp es el servicio de impresión asociado a CUPS. En administración local vimos que había una interfaz web de CUPS que se conectaba al puerto 631. Si queremos tener una idea de a qué se dedica un puerto determinado podemos mirar en /etc/services:

```
root@maquina:    ~# grep 631 /etc/services
ipp 631/tcp       # Internet Printing Protocol
ipp 631/udp       # Internet Printing Protocol
```

Si no estamos actuando como servidor de impresión hacia el exterior, tenemos que ir a la configuración de CUPS y eliminar esa prestación (por ejemplo, colocando un *listen* 127.0.0.1:631, para que sólo escuche la máquina local), o limitar el acceso a las máquinas permitidas.

Nos aparecen también algunos puertos como desconocidos, en este caso el 728 y 734; esto indica que el sistema no ha podido determinar qué *nmap* está asociado al puerto. Vamos a intentar verlo nosotros. Para ello, ejecutamos sobre el sistema el comando *netstat*, que ofrece diferentes estadísticas del sistema de red, desde paquetes enviados y recibidos, y errores, hasta lo que nos interesa, que son las conexiones abiertas y quién las usa. Vamos a intentar buscar quién está usando los puertos desconocidos:

```
root@maquina:~# netstat -anp | grep 728
udp 0 0 0.0.0.0:728 0.0.0.0:* 552/rpc.statd
```

ANOTACIONES

Y si hacemos lo mismo con el 734, observamos también que quien ha abierto el puerto ha sido *rpc.statd*, que es un *daemon* asociado a NFS (en este caso el sistema tiene un servidor NFS). Si hacemos este mismo proceso con los puertos 111 que aparecían como *sunrpc*, observaremos que el *daemon* que hay detrás es *portmap* , que se usa en el sistema de llamadas RPC. El sistema de llamadas RPC (*Remote Procedure Call*) permite utilizar el mecanismo de llamadas remotas entre dos procesos que están en diferentes máquinas. *portmap* es un *daemon* que se encarga de traducir las llamadas que le llegan por el puerto a los números de servicios RPC internos que se tengan, y es utilizado por diferentes servidores como NFS, NIS, NIS+.

Los servicios RPC que se ofrecen, se pueden ver con el comando *rpcinfo*:

```
root@maquina:~# rpcinfo p
  programa vers   proto puerto
    100000 2 tcp   111  portmapper
    100000 2 udp   111  portmapper
    100024 1 udp   731  status
    100024 1 tcp   734  status
    391002 1 tcp 39797  sgi_fam
    391002 2 tcp 39797  sgi_fam
```

donde observamos los servicios RPC con algunos de los puertos que ya se habían detectado. Otro comando que puede resultar útil es *lsof* que, entre otras funciones, permite relacionar puertos con los servicios que los han abierto (por ejemplo: *lsof -i | grep 731*).

El *daemon portmap* es un poco crítico con la seguridad, ya que, en principio, no ofrece mecanismos de autentificación del cliente, pues se supone que se delegan en el servicio (NFS, NIS,...). Por lo tanto, *portmap* podría ser víctima de intentos de DoS que podrían provocar errores en los servicios o hacerlos caer. Normalmente, protegeremos *portmap* por medio de algún tipo de *wrapper* y/o *firewall*. Si no utilizamos, y no tenemos previsto utilizar, servicios NFS y NIS, lo mejor es desactivar completamente *portmap*, quitándolo del *runlevel* donde se active. También podemos pararlos momentáneamente con los *scripts* (en Debian):

```
/etc/init.d/nfs-common
/etc/init.d/nfs-kernel-server
/etc/init.d/portmap
```

ANOTACIONES

dándoles el parámetro *stop* para parar los servicios RPC (en este caso NFS).

A continuación, controlaremos la seguridad en base a un *wrapper* sencillo. Vamos a suponer que queremos dejar paso a través de ssh de una máquina determinada, llamémosla 1.2.3.4 (dirección IP). Vamos a cerrar *portmap* al exterior, ya que no tenemos NIS, y de NFS tenemos servidor pero no estamos sirviendo nada (podríamos cerrarlo, pero lo dejaremos para futuros usos). Vamos hacer un *wrapper* (suponemos que los TCP *wrappers* están ya instalados), modificando los ficheros *hosts.deny -i allow*. En /etc/*hosts*.deny:

```
ALL : ALL : spawn (/usr/sbin/safe_finger -l @%h \
| /usr/bin/mail -s "%c FAILED ACCESS TO %d!!" root) &
```

estamos denegando todos los servicios (cuidado, aquellos relacionados con *inetd*) (primer *all*) a todos (*all*), y la acción por tomar será averiguar quién ha pedido el servicio y en qué máquina, y vamos a enviar un correo al *root* informando del intento. Podríamos también escribir un fichero de *log*, ... Ahora, en /etc/*hosts*.allow:

```
sshd: 1.2.3.4
```

habilitamos el acceso por la máquina IP 1.2.3.4 en el servidor *sshd* (del ssh). También podríamos colocar el acceso a *portmap*, sólo haría falta una línea *portmap: la_ip*. Podemos colocar una lista de máquinas o bien subredes que puedan utilizar el servicio (ver *man hosts.allow*). Recordar que también tenemos los comandos *tcpdchk*, para comprobar que la configuración del *wrapper* esté correcta, y *tcpdmatch* para simular qué pasaría con un determinado intento, por ejemplo:

```
root@maquina:~# tcpdmatch sshd 1.2.3.4
warning: sshd: no such process name in /etc/inetd.conf
client: hostname maquina.dominio.es
client: address 1.2.3.4
server: process sshd
matched: /etc/hosts.allow line 13
access: grantedv
```

nos dice que sería concedido el acceso. Un detalle es que nos dice que *sshd* no está en el *inetd.conf*, y si lo verificamos, vemos que así es: no se activa por el servidor *inetd*, sino por *daemon* en el *runlevel* que estemos. Además (en Debian), éste es un caso de demonio que está compilado con las bibliotecas de *wrappers* incluidas (y, por lo tanto, no necesita *tcpd* para funcionar). En Debian hay varios *daemons* así: *ssh*, *portmap*, *in.talk*, *rpc.statd*, *rpc.mountd*, entre otros. Esto permite asegurar estos *daemons* mediante *wrappers*.

Otra cuestión por verificar son las conexiones actuales existentes. Con el comando *netstat -utp*, podemos listar las conexiones *tcp* , *udp* establecidas con el exterior, ya sean entrantes o salientes; así, en cualquier momento podemos detectar los clientes conectados y a quién estamos conectados. Otro comando importante (de múltiples funciones) es *lsof*, que puede relacionar ficheros abiertos con procesos o conexiones por red establecidas mediante *lsof -i*, pudiéndose así detectar accesos indebidos a ficheros.

También podríamos utilizar un *firewall* para procesos similares (o bien como mecanismo añadido). Comenzaremos viendo cómo están las reglas del *firewall* en este momento: (comando *iptables –L*)

```
root@aopcjj:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source          destination
Chain FORWARD (policy ACCEPT)
target prot opt source          destination
Chain OUTPUT (policy ACCEPT)
target prot opt source          destination
```

O sea, que el *firewall* no está poniendo ninguna restricción en este momento, y permite la entrada, salida y reenvío de todos los paquetes.

En este punto, podríamos añadir un *firewall* que nos permitiese una gestión más adecuada de paquetes que recibimos y enviamos, y que sería un control previo para mejorar la seguridad. Dependiendo de nuestras necesidades, estableceríamos las reglas necesarias de modo parecido a las que comentamos en los ejemplos de *firewalls* de la unidad.

ANOTACIONES

En caso de poner los *firewalls*, podemos considerar si usar este mecanismo como único y quitar los *wrappers*: podría hacerse, ya que los *firewalls* (en este caso mediante *iptables*) ofrecen un mecanismo muy potente que nos permite seguir a un paquete por tipo, por protocolo y por lo que está haciendo en el sistema. Un buen *firewall* podría ser más o menos suficiente, pero, por si acaso, más medidas de seguridad no vienen mal. Y en caso de que el *firewall* no estuviese bien diseñado y dejase escapar algunos paquetes, el *wrapper* sería la medida, a nivel de servicio, para parar los accesos no deseados. Por poner una metáfora que se suele utilizar, si nos planteásemos nuestro sistema como la defensa de un castillo medieval, el foso y primeras murallas serían el *firewall*, y la segunda muralla de contención, el *wrapper*.

9.10. Actividades para el lector

1) Supongamos que colocamos en nuestra máquina un sitio web, por ejemplo con Apache. Nuestro sitio está pensado para diez usuarios internos, pero no controlamos este número. Más adelante nos planteamos poner en Internet este sistema, ya que creemos que puede ser útil para los clientes, y lo único que hacemos es poner el sistema con una IP pública en Internet. ¿Qué tipo de ataques podría sufrir este sistema?

2) ¿Cómo podemos detectar los ficheros con suid en nuestro sistema? ¿Qué comandos serán necesarios? ¿Y los directorios con SUID o SGID? ¿Por qué es necesario, por ejemplo, que /usr/bin/*passwd* tenga bit de SUID?

3) Los ficheros *.rhosts*, como hemos visto, son un peligro importante para la seguridad. ¿Podríamos utilizar algún método automático que comprobase su existencia periódicamente? ¿Cómo?

4) Supongamos que queremos deshabilitar un servicio del cual sabemos que tiene el script /etc/init.d/servicio que lo controla: queremos desactivarlo en todos los *runlevels* en los que se presenta. ¿Cómo encontramos los *runlevels* en los que está? (por ejemplo, buscando enlaces al *script*).

ANOTACIONES

5) Examinar los servicios en activo en vuestra máquina. ¿Son todos necesarios? ¿Cómo habría que protegerlos o desactivarlos?

6) Practicar el uso de algunas de las herramientas de seguridad descritas (*nmap*, *nessus*, etc.).

7) ¿Qué reglas IPtables serían necesarias para una máquina en la que sólo queremos acceso por SSH desde una dirección concreta?

8) ¿Y si queremos únicamente un acceso al servidor web?

9.11. Otras fuentes de referencia e información

- [Deb] [Hat03c] Los sitios de seguridad de las distribuciones.
- [Peñ03] Imprescindible para Debian, muy buena descripción para seguir paso a paso la configuración de seguridad, [Hat03b] sería equivalente para Red Hat 9.
- [Mou02] Excelente referencia de seguridad para Red Hat (aplicable también para Debian).
- [Max99][Hat01] Libros seguridad Linux, cubriendo amplios aspectos y técnicas.
- [Lin03e] Pequeña guía (2 páginas) de seguridad.
- [Sei02] Guía paso a paso identificando los puntos clave que hay que verificar y los problemas que puedan surgir.
- [Net03] Proyecto Netfilter, y IPTables.
- [Ian03] Una lista de puertos TCP/IP.
- [Pro03a] [Sno03] [Ins03b] [Nes03] Algunas de las herramientas de seguridad más usadas.

ANOTACIONES

- [NSA03b] Versión de Linux con miras a la seguridad, producida por la NSA.

- [CER03a][Aus03][Ins03a][Inc03b] [NSA03a] Sitios de organismos de seguridad.

- [CER03b][Ins98][San03] Vulnerabilidades y *exploits* de los diferentes operativos.

- [NSA03a][FBI03][USA03] Algunas "policías" de cibercrimen en Estados Unidos.

ANOTACIONES

10. Configuración, sintonización y optimización

Un aspecto fundamental, una vez que el sistema está instalado, es la configuración y adaptación del sistema a las necesidades del usuario y que las prestaciones del sistema sean lo más adecuadas posible a las necesidades que de él se demandan. GNU/Linux es un sistema operativo-eficiente que permite un grado de configuración excelente y una optimización muy delicada de acuerdo a las necesidades del usuario. Es por ello por lo que, una vez realizada una instalación (o en algunos casos una actualización), deben hacerse determinadas configuraciones vitales en el sistema. Si bien el sistema "funciona", es necesario efectuar algunos (adaptación al entorno o sintonización) para permitir que estén cubiertas todas las necesidades del usuario/servicios que presta la máquina. Esta sintonización dependerá de dónde se encuentre funcionando la máquina y se llevará a cabo, en algunos casos para mejorar el rendimiento del sistema, y en otros (además) por cuestiones seguridad (ver unidad 3.9). Cuando está en funcionamiento, es necesario monitorizar el sistema para ver su comportamiento y actuar en consecuencia. Si bien es un aspecto fundamental, la sintonización de un operativo muchas veces se relega a la opinión de expertos o *gurús* de la informática; pero conociendo los parámetros que afectan al rendimiento, es posible llegar a buenas soluciones haciendo un proceso cíclico de análisis, cambio de configuración, monitorización y ajustes.

10.1. Aspectos básicos

Antes de conocer cuáles son las técnicas de optimización, es necesario enumerar las causas que pueden afectar a las prestaciones de un sistema operativo [Maj96]. Entre éstas, se pueden mencionar:

a) Cuellos de botella en los recursos: la consecuencia es que todo el sistema irá más lento porque existen recursos que no pueden satisfacer la demanda a la que se les somete. El primer paso para

ANOTACIONES

optimizar el sistema es encontrar estos cuellos de botella y por qué ocurren, conociendo sus limitaciones teóricas y prácticas.

b) Ley de Amdahl: según esta ley, "hay un límite de cuánto puede uno mejorar en velocidad una cosa si sólo se optimiza una parte de ella"; es decir, si se tiene un programa que utiliza el 10% de CPU y se optimiza reduciendo la utilización en un factor de 2, el programa mejorará sus prestaciones (*speedup*) en un 5%, lo cual puede significar un tremendo esfuerzo no compensado con los resultados.

c) Estimación del *speedup*: es necesario estimar cuánto mejorará para evitar esfuerzos y costes innecesarios. Se puede utilizar la ley anterior para valorar si es necesaria una inversión en tiempo o económica en el sistema.

d) Efecto burbuja: siempre se tiene la sensación de que cuando se encuentra la solución a un problema, surge otro. Una manifestación de este problema es que el sistema se mueve constantemente entre poblemas de CPU y problemas de entrada/salida, y viceversa.

e) Tiempo de repuesta frente a cantidad de trabajo: si se cuenta con veinte usuarios, mejorar en la productividad significará que todos tendrán más trabajo hecho al mismo tiempo, pero no mejores respuestas individualmente; podría ser que el tiempo de respuesta para algunos fuera mejor que para otros. Mejorar el tiempo de respuesta significa optimizar el sistema para que las tareas individuales tarden lo menos posible.

f) Psicología del usuario: dos parámetros son fundamentales: 1) el usuario estará insatisfecho generalmente cuando se produzcan variaciones en el tiempo de respuesta; y 2) el usuario no detectará mejoras en el tiempo de ejecución menores del 20%.

g) Efecto prueba: las medidas de monitorización afectan a las propias medidas. Se debe ir con cuidado cuando se realizan las pruebas por los efectos colaterales de los propios programas de medida.

h) Importancia de la media y la variación: se deben tener en cuenta los resultados, ya que si se obtiene una media de utilización de

ANOTACIONES

CPU del 50% cuando ha sido utilizada 100, 0, 0, 100, se podría llegar a conclusiones erróneas. Es importante ver la variación sobre la media.

i) Conocimientos básicos sobre el hardware del sistema por optimizar: para mejorar una cosa es necesario "conocer" si es susceptible de mejora. El encargado de la optimización deberá conocer básicamente el hardware subyacente (CPU, memorias, buses, cachés, entrada/salida, discos, vídeo, ...) y su interconexión para poder determinar dónde están los problemas.

j) Conocimientos básicos sobre el sistema operativo por optimizar: del mismo modo que en el punto anterior, el usuario deberá conocer aspectos mínimos sobre el sistema operativo que pretende optimizar, entre los cuales se incluyen conceptos como procesos y *threads* (creación, ejecución, estados, prioridades, terminación), llamadas al sistema, *buffers* de caché, sistema de archivos, administración de memoria y memoria virtual (paginación, *swap*) y tablas del *kernel*.

Nota

Para optimizar hay que tener en cuenta la saturación de los recursos. Ley de Amdahl: relaciona los conocimientos de software y hardware disponible, el tiempo de respuesta y el número de trabajos.

10.1.1. Monitorización sobre UNIX System V

Los sistemas compatibles UNIX SV utilizan los comandos *sar* y *sadc* para obtener estadísticas del sistema. Su equivalente en GNU/Linux (incluido en Debian) es *atsar* (y *atsadc*), que es totalmente equivalente a los que hemos mencionado. El comando *atsar* lee contadores y estadísticas del fichero /proc y las muestra por la salida estándar. La primera forma de llamar al comando es:

```
atsar opciones t [n]
```

donde muestra la actividad en *n* veces cada *t* segundos con una cabecera mostrando los contadores de actividad (el valor por defecto de *n* es 1).

La segunda forma de llamarlo es:

```
atsar -opciones -s time -e time -i sec -f file -n day#
```

ANOTACIONES

El comando extrae datos del archivo especificado por *-f* (por defecto /var/log/atsar/atsarxx, con *xx* el día del mes) y que fueron previamente guardados por *atsadc* (se utiliza para recoger los datos, salvarlos y procesarlos y en Debian está en /usr/lib/atsar). El parámetro *-n* puede ser utilizado para indicar el día del mes y *-s*, *-e* la hora de inicio-final, respectivamente. Para activar *atsadc*, por ejemplo, se podría incluir en /etc/cron.d/atsar una línea como la siguiente:

```
@reboot root test x /usr/lib/atsadc && /usr/lib/atsar/atsadc /var/log/atsar/atsa'date +\%d'
10,20,30,40,50 * * * * root test x /usr/lib/atsar/atsa1 && /usr/lib/atsar/atsa1
```

La 1.ª crea el archivo después de un reinicio. La 2.ª guarda los datos cada 10 minutos con el *shell script atsa1*, que llama al *atsadc*.

En *atsar*, las opciones se utilizan para indicar qué contadores hay que mostrar; algunos de ellos son:

Opción	Descripción
u	Utilización CPU
d	Actividad de disco
I	Número de interrupciones/s
v	Utilización de tablas en el *kernel*
y	Estadísticas de utilización de *ttys*
p	Información de paginación y actividad de *swap*
r	Memoria libre y ocupación de *swap*
l	Estadísticas de red
L	Información de errores de red
w	Estadísticas de conexiones IP
t	Estadísticas de TCP
U	Estadísticas de UDP
m	Estadísticas de ICMP
N	Estadísticas de NFS
A	Todas las opciones

Nota

Monitorizar con ***atsar***

- CPU: *atsar -u*
- Memoria: *atsar -r*
- Disco: *atsar -d*
- Paginación: *atsar -p*

A continuación se verán algunos ejemplos de utilización del *atsar* y el significado de la información que genera:

1) Utilización de CPU

```
atsar -u 4 5
Linux nteum 2.4.18 #1 SMP Sat Sep 13 19:49:55 CEST 2003 i686 09/23/2003
22:26:20 cpu %usr %sys %nice %idle pswch/s runq nrproc lavg1 lavg5 avg15 _cpu_
22:26:24 all 0 0 0 100 83 1 69 0.00 0.06 0.06
22:26:28 all 0 0 0 100 83 1 69 0.00 0.06 0.06
22:26:32 all 0 0 0 100 84 1 69 0.00 0.06 0.06
22:26:36 all 0 0 0 100 84 1 69 0.00 0.06 0.06
22:26:40 all 0 0 0 100 85 1 69 0.00 0.06 0.06
```

- *usr* y *sys* muestran el porcentaje de tiempo de CPU en el modo usuario con *nice* ≤ 0 (normales) y en el modo *kernel*.
- *nice*, lo mismo, pero con procesos de usuario con *nice* > 0 (menor prioridad que la media).
- *idle* indica el tiempo no utilizado de CPU por los procesos en estado de espera.
- *pswch/s* es el número de cambios de contextos.
- *runq* da el número de procesos que espera la CPU.
- *nrproc*, el número de procesos en el sistema.
- *lavg1,5,15* indica la carga media en la cola de ejecución en el último, en los últimos 5 y en los últimos 15 minutos.

En este caso *idle* = 100, lo cual significa que la CPU está ociosa (se puede observar *runq* = 1, por lo que no hay procesos por ejecutar y la carga es baja); si *idle* ≤ 10 y *runq* es elevado, debería pensarse en optimizar la CPU, ya que podría ser el cuello de botella del sistema.

2) Número de interrupciones por segundo

```
atsar -I 4 5
Linux nteum 2.4.18 #1 SMP Sat Sep 13 19:49:55 CEST 2003 i686 09/23/2003
23:09:04 cpu iq00 iq01 iq10 iq12 iq14 iq15 _intr/s_
23:09:08 all 98 6 10 34 1 0
23:09:12 all 100 9 7 20 0 2
23:09:16 all 100 2 4 10 0 3
```

Muestra la información de la frecuencia de interrupciones de los niveles activos que se encuentran en /proc/interrupts; por ejemplo, en este caso: 0 = *timer*, 1 = *teclado*, 10 = *USB*, 12 = *psmouse*, 14 = *ide0*, 15 = *ide1*.

3) Memoria y *swap*

```
atsar -r 4 5
Linux nteum 2.4.18 #1 SMP Sat Sep 13 19:49:55 CEST 2003 i686 09/23/2003
23:10:00 memtot memfree  memshared  buffers  cached   swptot   swpfree
23:10:01 249M   22324K   0K         16312K   96932K   251M     256960K
```

En este caso *memtot* es la memoria principal (MP) total; *memfree*, la libre; *memshared*, la compartida; *buffers* es la MP utilizada en *buffers*; cached, memoria principal utilizada en caché de páginas; *swptot,* el espacio total de *swap*, y *swpfree* el espacio libre de *swap* (donde M = 1.000 Kbytes). Es importante tener en cuenta que si no hay espacio en MP, las páginas de los procesos irán a parar al *swap,* donde debe haber sitio. Esto se debe contrastar con la utilización de CPU. También se debe controlar que el tamaño de los *buffers* sea adecuado y esté en relación con los procesos que están realizando operaciones de entrada/salida.

4) Utilización de las tablas del kernel

```
atsar -v 4 5
Linux nteum 2.4.18 #1 SMP Sat Sep 13 19:49:55 CEST 2003 i686 09/23/2003
23:10:35   superb-sz  inode-sz     file-sz    dquota-sz  flock-sz_curmax_
23:10:36   1/1        122180/1     107/8192   0/0        5/0
```

ANOTACIONES

En este caso, *superb-sz* es el número actual-máximo de *superblocks* mantenido por el *kernel* para los sistemas de archivos montados; *inode-sz*, el número actual máximo de *incore-inodes* en el *kernel* necesario, es de uno por disco como mínimo; *file-sz* actual-máximo número de archivos abiertos, *dquota-sz* actual-máximo ocupación de entradas de cuotas, *flock-sz* actual-máximo de archivos bloqueados.

Para las restantes opciones, consultar *man atsar*. Esta monitorización se puede completar con el comando *ps -edaflm* (*process status*) y el

comando *top*, que mostrarán la actividad y estado de los procesos en el sistema. A continuación, se muestran dos ejemplos de ambos comandos:

```
ps -edaflm
```

```
F    S UID        PID   PPID  C  PRI  NI  ADDR  SZ   WCHAN    STIME TTY    TIME      CMD
100  S root         1      0  0   69   0   -     318  141d8e   21:28 ?      00:00:04  init [2]
040  S root         2      1  0   69   0   -       0  121d85   21:28 ?      00:00:00  [keventd]
040  S root       169      1  0   69   0   -     509  141d8e   21:28 ?      00:00:00  /sbin/syslogd
040  S root       253      1  0   69   0   -     541  141d8e   21:28 ?      00:00:00  /usr/sbin/dhcpd-2.2.x -q
040  S daemon405           1  0   69   0   -     346  11d9bc   21:29 ?      00:00:00  /usr/sbin/atd
100  R root       954    544  0   73   0   -     671  -        23:56 pts/1  00:00:00  ps -edaflm
...
```

Donde los parámetros reflejan el valor indicado en la variable del *kernel* para este proceso, los más importantes desde el punto de vista de la monitorización son: *F flags* (en este caso 100 es con superprivilegios, 040 creado desde el inicio *daemon*), *S* es el estado (*D*: no-interrumpible durmiendo entrada/salida, *R*: ejecutable o en cola, *S*: durmiendo, *T*: en traza o parado, *Z*: muerto en vida, 'zombie'). *PRI* es la prioridad; *NI* es *nice*; *STIME*, el tiempo de inicio de ejecución; *TTY*, desde donde se ha ejecutado; *TIME*, el tiempo de CPU; CMD, el programa que se ha ejecutado y sus parámetros. Si se quiere salida con refresco (configurable), se puede utilizar el comando *top*, que muestra unas estadísticas generales (procesos, estados, carga, etc.), y después, información de cada uno de ellos similar al *ps*, pero se actualiza cada 5 segundos por defecto:

```
top
23:57:10 up 2:28, 1 user, load average: 0.00, 0.05, 0.15
68 processes: 67 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 11.5% user, 1.9% system, 0.4% nice, 86.2% idle
Mem: 255288K total, 233432K used, 21856K free, 16324
buffersK Swap: 258040K total, 1080K used, 256960K free,
               96200K cachedK 0m
 PID    USER        PRI   NI   SIZE    RSS    SHARE    STAT   %CPU   %MEM   TIME   COMMAND
 2000   root         13    0    956    956      748    R      0.9    0.3    0:00   top
    1   root          9    0    484    484      424    S      0.0    0.1    0:04   init
    2   root          9    0      0      0        0    SW     0.0    0.0    0:00   keventd
   94   daemon        9    0    432    432      356    S      0.0    0.1    0:00   portmap
  169   root          9    0    784    784      668    S      0.0    0.3    0:00   syslogd
```

```
172   root        9    0   532   532   384   S      0.0   0.2   0:00   klogd
176   root        9    0  2248  2248  1736   S      0.0   0.8   0:00   namedK
253   root        9    0   820   820   688   S      0.0   0.3   0:00   dhcpd-2.2.x
221   root        7   -5   708   708   568   S <    0.0   0.2   0:00   diald
337   postgres    9    0  1692  1692  1624   S      0.0   0.6   0:00   postmaster
419   root        9    0  1416  1372  1304   S      0.0   0.5   0:00   apache
...
```

Nota

Consultad el comando *man* para una descripción de los parámetros y características.

Debian Linux también incluye todo un conjunto de herramientas de monitorización equivalentes a *sar*, pero que son originarias de UNIX BSD y que presentan una funcionalidad similar, aunque desde diferentes comandos: *vmstat* (estadísticas de CPU, memoria y entrada/salida), *iostat* (estadísticas de discos y CPU), *uptime* (carga de CPU y estado general).

10.1.2. Optimizando el sistema

A continuación veremos algunas recomendaciones para optimizar el sistema en función de los datos obtenidos.

1) Resolver los problemas de memoria principal

Se debe procurar que la memoria principal pueda acoger un porcentaje elevado de procesos en ejecución, ya que si no es así, el sistema operativo podrá paginar e ir al *swap*; pero esto significa que la ejecución de ese proceso se degradará notablemente. Si se agrega memoria, el tiempo de respuesta mejorará notablemente. Para ello, se debe tener en cuenta el tamaño de los procesos (SIZE) en estado *R* y agregarle la que utiliza el *kernel*, que se puede obtener con el comando *dmesg*, que nos mostrará, por ejemplo:

```
Memory:  255048k/262080k available (1423k kernel core, 6644k reserved, 466k
         data, 240k init, Ok highmem
```

Luego se deberá comparar con la memoria física y analizar si el sistema está limitado por la memoria (se verá con *atsar -r* y *-p* mucha actividad de paginación).

Las soluciones para la memoria son obvias: o se incrementa la capacidad o se reducen las necesidades. Por el coste actual de la memoria,

es más adecuado incrementar su tamaño que emplear muchas horas para ganar un centenar de bytes por quitar, ordenar o reducir requerimientos de los procesos en su ejecución. Reducir los requerimientos puede hacerse reduciendo las tablas del *kernel*, quitando módulos, limitando el número máximo de usuarios, reduciendo los *buffers*, etc.; todo lo cual degradará el sistema (efecto burbuja) y las prestaciones serán peores (en algunos casos, el sistema puede quedar totalmente no operativo).

Otro aspecto que se puede reducir es la cantidad de memoria de los usuarios eliminando procesos redundantes y cambiando la carga de trabajo. Para ello, se deberán monitorizar los procesos que están durmiendo (*zombies*) y eliminarlos, o aquellos que no progresan en su entrada/salida (saber si son procesos activos, cuánto de CPU llevan gastado y si los 'usuarios están por ellos'). Cambiar la carga de trabajo es utilizar planificación de colas para que los procesos que necesitan gran cantidad de memoria se puedan ejecutar en horas de poca actividad (por ejemplo, por la noche lanzándolos con el comando *at*).

Nota

¿Dónde mirar?
1.º Memoria
2.º CPU
3.º Entrada/salida
4.º TCP/IP
5.º *Kernel*
6.º Varios

2) Mucha utilización de CPU

Básicamente nos la da el tiempo *idle* (valores bajos). Con *ps* o *top* se deben analizar qué procesos son los que 'devoran CPU' y tomar decisiones como: posponer su ejecución, pararlos temporalmente, cambiar la prioridad (menos conflictivo de todos, se puede utilizar el comando *renice* prioridad PID), optimizar el programa (para la próxima vez) o cambiar la CPU (o agregar otra). Como ya se ha mencionado, GNU/Linux utiliza el directorio /proc para mantener todas las variables de configuración del *kernel* que pueden ser analizadas y, en cierto caso, 'ajustadas', para lograr prestaciones diferentes o mejores.

Para ello, se debe utilizar el comando *systune dump > /tmp/sysfile* para obtener todas las variables y sus valores en el archivo /tmp/sysfile. Este archivo se puede editar, cambiar la variable correspondiente y luego utilizar el comando *systune -c /tmp/sysfile* para cargarlas nuevamente en el /proc. El comando *systune* también lee por defecto si no tiene la opción -c de /etc/systune.conf. En este caso, por ejemplo, se podría modificar (procédase con cuidado, porque el *kernel* puede

Nota

Consultar [?, Aiv02] para mayor información sobre la estructura, organización y definición de variables.

ANOTACIONES

quedar fuera de servicio) las variables de la categoría /proc/sys/vm (memoria virtual) o /proc/sys/kernel (configuración del *core* del *kernel*).

En este mismo sentido, también (para expertos o desesperados) se puede cambiar el tiempo máximo (*slice*) que el administrador de CPU (*scheduler*) del sistema operativo dedica a cada proceso en forma circular (si bien es aconsejable utilizar *renice* como práctica). Pero GNU/Linux, a diferencia de otros operativos, es un valor fijo dentro del código, ya que está optimizado para diferentes funcionalidades (pero es posible tocarlo). Se puede "jugar" (a su propio riesgo) con las variables *TICK_SCALE* (o *NICE_TO_TICKS*) en (Debian) kernel-source-2.4.18/kernel/sched.c (por defecto, se ajusta para que el *timeslice* sea aproximadamente de 50 milisegundos).

3) Reducir el número de llamadas

Otra práctica adecuada para mejorar las prestaciones es reducir el número de llamadas al sistema de mayor coste en tiempo de CPU. Estas llamadas son las invocadas (generalmente) por el*shell* *fork*() y *exec*(). Una configuración inadecuada de la variable *PATH* y debido a que la llamada *exec*() no guarda nada en caché, el directorio actual (indicado por ./), puede tener una relación desfavorable de ejecución. Para ello, siempre habrá que configurar la variable *PATH* con el directorio actual como última ruta. Por ejemplo, en `bash` (o en *.bashrc*) hacer: *export PATH = $PATH*. Si no es así, no está el directorio actual, o si está, rehacer la variable *PATH* para ponerlo como última ruta.

Se debe tener en cuenta que una alta actividad de interrupciones puede afectar a las prestaciones de la CPU con relación a los procesos que ejecuta. Mediante monitorización (*atsar -I*) se puede mirar cuál es la relación de interrupciones por segundo y tomar decisiones con respecto a los dispositivos que las causan. Por ejemplo, cambiar de módem por otro más inteligente o cambiar la estructura de comunicaciones si detectamos una actividad elevada sobre el puerto serie donde se encuentra conectado.

4) Mucha utilización de disco

Después de la memoria, un tiempo de respuesta bajo puede ser debido al sistema de discos. En primer lugar se debe verificar que se dis-

ANOTACIONES

ponga de tiempo de CPU (por ejemplo, *idle* > 20%) y que el número de entrada/salida sea elevado (por ejemplo, > 30 entrada/salida/s) utilizando *atsar -u* y *atsar -d*. Las soluciones pasan por:

a) En un sistema multidisco, planificar dónde se encontrarán los archivos más utilizados para equilibrar el tráfico hacia ellos (por ejemplo /home en un disco y /usr sobre otro) y que puedan utilizar todas las capacidades de la entrada/salida con caché y concurrente de GNU/Linux (incluso, por ejemplo, planificar sobre qué bus *ide* se colocan). Comprobar luego que existe un equilibrio del tráfico con *atsar -d* (o con *iostat*). En situaciones críticas se puede considerar la compra de un sistema de discos RAID que realizan este ajuste de forma automática.

b) Tener en cuenta que se obtienen mejores prestaciones sobre dos discos pequeños que sobre uno grande del tamaño de los dos anteriores.

c) En sistemas con un solo disco, generalmente se realizan, desde el punto de vista del espacio, cuatro particiones de la siguiente manera (desde fuera hacia dentro): /, *swap*, /usr, /home; pero que genera pésimas respuestas de entrada/salida porque si, por ejemplo, un usuario compila desde su directorio /home/user y el compilador se encuentra en /usr/bin, la cabeza del disco se moverá por toda su longitud. En este caso es mejor unir las particiones /usr y /home en una sola (más grande) aunque puede representar algunos inconvenientes en cuanto a mantenimiento.

d) Incrementar los *buffers* de caché de la entrada/salida (ver, por ejemplo: /proc/ide/hd...).

e) Si se utiliza un *ext2fs*, se puede utilizar el comando:

```
dumpe2fs -h /dev/hd...
```

para obtener información sobre el disco y:

```
tune2fs /dev/hd...
```

para cambiar algunos de los parámetros configurables del disco.

f) Obviamente, el cambio del disco por uno de mayor velocidad (RPM) siempre tendrá un impacto positivo en un sistema limitado por entrada/salida de disco. [Maj96]

ANOTACIONES

5) **Mejorar aspectos de TCP/IP**

Examinar la red con el comando *atsar* (o también con *netstat -i* o con *netstat -s | more*) para analizar si existen paquetes fragmentados, errores, *drops*, *overflows*, etc., que puedan estar afectando a las comunicaciones y con ello al sistema (por ejemplo, en un servidor de NFS, NIS, Ftp o Web). Si se detectan problemas, se debe analizar la red para considerar las siguientes actuaciones:

a) Fragmentar la red a través de elementos activos que descarten paquetes con problemas o que no son para máquinas del segmento.

b) Planificar dónde estarán los servidores para reducir el tráfico hacia ellos y los tiempos de acceso.

c) Ajustar parámetros del *kernel* (/proc/*sys*/net/), por ejemplo, para obtener mejoras en el *throughput* hacer:

```
echo 600 > /proc/sys/net/core/netdev max backlog
```

(por defecto 300). [?]

6) **Otras acciones sobre parámetros del kernel**

Existe otro conjunto de parámetros sobre el *kernel* que es posible sintonizar para obtener mejores prestaciones, si bien, teniendo en cuenta lo que hemos tratado anteriormente, se debe ir con cuidado, ya que podríamos causar el efecto contrario o inutilizar el sistema. Consultar en la distribución del código fuente en kernel-source-2.4.18/Documentation/sysctl los archivos *vm.txt*, *fs.txt*, *kernel.txt* y *sunrpc.txt*:

a) /proc/sys/vm: controla la memoria virtual (MV) del sistema. La memoria virtual permite que los procesos que no entran en memoria principal sean aceptados por el sistema pero en el dispositivo de *swap*, por lo cual, el programador no tiene límite para el tamaño de su programa (obviamente debe ser menor que el dispositivo de *swap*). Los parámetros susceptibles de sintonizar son: *bdflush*, *buffermem*, *freepages*, *kswapd*, *page-cluster*, *pagecache*, *pagetable_cache* (se pueden cambiar muy fácilmente con *gpowertweak*).

b) /proc/*sys*/fs: se pueden ajustar parámetros de la interacción *kernel*-FS tal como *file-max*.

c) /proc/sys/kernel, /proc/sys/sunrpc

Nota

Ver detalles en kernel.txt.

ANOTACIONES

7) Generar el *kernel* adecuado a nuestras necesidades

La optimización del *kernel* significa escoger los parámetros de compilación de acuerdo a nuestras necesidades. Es muy importante primero leer el archivo *readme* de /usr/src/linux. Una buena configuración del *kernel* permitirá que éste se ejecute más rápido, que se disponga de más memoria para los procesos de usuario y además resultará más estable. Hay dos formas de construir un *kernel*: **monolítico** (mejores prestaciones) o **modular** (basado en módulos ⇒ mejor portabilidad si tenemos un sistema muy heterogéneo y no se desea compilar un *kernel* para cada uno de ellos). Para compilar su propio *kernel* y adaptarlos a su hardware y necesidades, cada distribución tiene sus reglas (si bien el procedimiento es similar).

En este apartado trabajaremos con una distribución Debian (por diferentes razones y todas buenas) y comenzaremos por obtener el código fuente de la distribución, por ejemplo, *kernel-source-2.4.18,* y los ficheros *include* específicos para la arquitectura (por ejemplo, de la Debian Woody 3.0 r1), en este caso *kernel-headers-2.4.18-686*, que son los adecuados para PII-PIV. Para utilizar este último, sugerimos emplear el paquete *kernel-package* (a través del comando *make-kpkg*), el cual ha sido diseñado para facilitar la tarea de crear nuevas imágenes de los *kernels*.

Nota

Ver más detalles en:

/usr/share/doc/kernel-headers-2.4.18-686/Debian. README

/usr/share/doc/kernel-source-2.4.18/README.headers.gz

Aplicando *bunzip2* y *tar* a la distribución, se tendrá un directorio *kernel-source 2.4.18*. No se describirá el contenido de *Debian.README*, donde se explican paso a paso las tareas para configurar el nuevo *kernel*, pero se presentará un resumen:

```
cd /usr/src/kernel-source-2.4.18
make config
```

También *make menuconfig* o *make xconfig*. Para configurarlo, recoged primero información específica sobre su hardware (por ejemplo, procesador de la placa base, PCI, HD, *chipset*, CD-ROM, *floppy disk*, Ethernet, ratón, vídeo, etc.) y seleccionad las opciones más adecuadas para el hardware disponible.

```
make-kpkg clean Crear una imagen portable del kernel (.deb)
make-kpkg --revision= custom.1.0 kernel_image
```

ANOTACIONES

A partir de este paso, es necesario ser *root*, o *fakeroot* o *sudo*. Si se ha indicado al *bootloader*, habrá que usar *kernels* con *initrd kernels* (norma en casi todas las últimas imágenes de *kernel* oficiales); es necesario agregar *--initrd* a la línea anterior. Es preferible no utilizar este parámetro para evitar módulos de terceros a dicho *kernel*.

```
dpkg -i ../kernel-imageX.XXX_1.0_<arch>.deb
```

Instala el *kernel*.

```
shutdown -r now
```

Carga el nuevo *kernel*.

También es posible por el método tradicional, menos automático, de *make menuconfig*, *make dep*, *make bzImage* y *make modules*; de *make modules_install*, si se han configurado módulos; o bien de *mv /vmlinuz /vmlinuz.old*, *cp ...arch/i386/boot/bzImage /vmlinuz*, que salvará lo anterior y copiará lo nuevo. Modifique /etc/lilo.conf para agregar otra sección *image* con *vmlinuz.old*, por si el primero no funciona, poder reiniciar con el antiguo: ejecute el comando *lilo* y *shutdown -r now*.

8) **Otros optimizadores**

Existen algunas optimizaciones sugeridas por [Mou01]:

a) Editar:

```
/usr/src/kernel-source-2.4.18/include/linux/sem.h
```

y cambiar el número de semáforos disponibles. Donde dice *#define SEMMNI 128*, cambiarlo por *#define SEMMNI 512*.

b) Cambiar la longitud del *buffer* del *syslog* en:

```
/usr/src/kernel-source-2.4.18/kernel/printk.c
```

Donde dice *#define LOG_BUF_LEN (16384)* (esta variable aparecerá cuatro veces para diferentes procesadores con diferentes valores), cambiarla por *#define LOG_BUF_LEN (65536)*.

c) Editar (línea 20):

```
vi +20 /usr/src/kernel-source-2.4.18/Makefile
```

y agregar a la variable *HOSTCFLAGS* (línea 20) y a la variable *CFLAGS* (línea 91) *-O3 -funroll-loops* en lugar de *-O2*. Estos

ANOTACIONES

cambios (agresivos) pueden no funcionar con todos los *kernels*. No intentéis forzar su funcionamiento, ya que podría quedar el *kernel* inestable (similar a otros sistemas operativos muy comunes).

10.1.3. Optimizaciones de carácter general

Existen una serie de optimizaciones de índole general que pueden mejorar las prestaciones del sistema:

1) **Bibliotecas estáticas o dinámicas**: cuando se compila un programa, se puede hacer con una biblioteca estática (*libr.a*), cuyo código de función se incluye en el ejecutable o con una dinámica (*libr.so.xx.x*), donde se carga la biblioteca en el momento de la ejecución. Si bien las primeras garantizan código portable y seguro, consumen más memoria. El programador deberá decidir cuál es la adecuada para su programa incluyendo *-static* en la opciones del compilador (no ponerlo significa dinámicas) o *--disable-shared*, cuando se utiliza el comando *configure*. Es recomendable utilizar (casi todas las distribuciones nuevas lo hacen) la biblioteca estándar *libc.a* y *libc.so* de versiones 2.2.x o superiores (conocida como Libc6) que reemplaza a las anteriores.

2) **Selección del procesador adecuado**: generar código ejecutable para la arquitectura sobre la cual correrán las aplicaciones. Algunos de los parámetros más influyentes del compilador son: *-march* (por ejemplo, *march* = i686 o *-march* = k6) haciendo simplemente *gcc -march = i686*, el atributo de optimización *-O1,2,3* (*-O3* generará la versión más rápida del programa, *gcc -O3 -march = i686*) y los atributos *-f* (consultar la documentación para los diferentes tipos). Existe un archivo general de configuración del compilador, con lo cual los cambios se aplicarán a todos los ejecutables generados a partir de aquel momento) en /usr/lib/gcclib/i386linux/2.95.4/specs que al final existen unas líneas similares a [Mou01]:

```
*cpp_cpu_default:
...
*cc1_cpu:
%{!mcpu*: %{m386:-mcpu = i386 -march = i386}
   %{m486:-mcpu = i486 -march = i486}
      %{mpentium:-mcpu = pentium}
         %{mpentiumpro:mcpu = pentiumpro}}
```

Nota

Ver: *man gcc*.

Nota

Optimizaciones generales
Utilizar Libc6 (2.2.x)
Seleccionar procesador
Optimizar disco (avanzados)

ANOTACIONES

Que se puede sustituir (hacer una copia de seguridad antes) si tenemos un i686, PPro, PII, PIII, o Athlon (consultar las referencias para otros procesadores/parámetros) por:

```
*cpp_cpu_default: D__tune_i686__
...
*cc1_cpu:
%{!mcpu*: -O3 -march = i686 -funroll-loops -fomit-frame-pointer
%{m386:-mcpu = i386 -march = i386} %{m486:-mcpu = i486 -march = i486}
  %{mpentium:-mcpu = pentium} %{mpentiumpro:-mcpu = pentiumpro}}
```

3) **Optimización del disco**: en la actualidad, la mayoría de ordenadores incluye disco UltraDMA (100) por defecto; sin embargo, en una gran cantidad de casos no están optimizados para extraer las mejores prestaciones. Existe una herramienta (*hdparm*) que permite sintonizar el *kernel* a los parámetros del disco tipo IDE. Se debe tener cuidado con esta utilidad, sobre todo en discos UltraDMA (verificar en el BIOS que los parámetros para soporte por DMA están habilitados), ya que pueden inutilizar el disco. Consultar las referencias y la documentación ([Mou01] y *man hdparm*) sobre cuáles son (y el riesgo que comporta) las optimizaciones más importantes, por ejemplo: *-c3*, *-d1*, *-X34*, *-X66*, *-X12*, *-X68*, *-mXX*, *-a16*, *-u1*, *-W1*, *-k1*, *-K1*. Cada opción significa una optimización y algunas son de altísimo riesgo, por lo que habrá que conocer muy bien el disco. Para consultar los parámetros optimizados, se podría utilizar *hdparm -vtT /dev/hdX* (donde *X* es el disco optimizado) y la llamada a *hdparm* con todos los parámetros se puede poner en /etc/init.d para cargarla en el *boot*.

10.1.4. Configuraciones complementarias

Existen más configuraciones complementarias desde el punto de vista de la seguridad que de la optimización, pero son necesarias sobre todo cuando el sistema está conectado a una intranet o a Internet. Estas configuraciones implican las siguientes acciones [Mou01]:

a) **Impedir que se pueda arrancar otro sistema operativo**: si alguien tiene acceso físico a la máquina, podría arrancar con otro sistema operativo preconfigurado y modificar el actual, por lo que

ANOTACIONES

se debe inhibir desde el BIOS del ordenador el *boot* por *floppy* o CD-ROM y poner un *password* de acceso (recordar el *password* del BIOS, ya que, de otro modo, podría causar problemas cuando se quisiera cambiar la configuración).

b) **Configuración y red**: es recomendable desconectar la red siempre que se deseen hacer ajustes en el sistema. Se puede quitar el cable o deshabilitar el dispositivo con */etc/init.d/networking stop* (*start* para activarla de nuevo) o con *ifdown eth0* (*ifup eth0* para habilitarla) para un dispositivo en concreto.

c) **Modificar los archivos de /etc/security**: de acuerdo a las necesidades de utilización y seguridad del sistema. Por ejemplo:

`'access.conf'` — Quién puede hacer un *login* al sistema.

Formato:

```
permisssion : users    : origins
+ o -       : usuarios : desde donde
-:ALL EXCEPT root:tty1
```
Impide el acceso a todos *no-root* sobre *tty1*.

```
-:ALL EXCEPT user1 user2 user3:console
```
Impide el acceso excepto *user1,2,3* pero el último sólo desde *consola*.

```
-:user1:ALL EXCEPT LOCAL .uoc.edu
```

`'group.conf'`: Acceso a servicios restringidos.

Formato:

```
services;ttys;users;times;groups
xsh;tty*&!ttyp*;user1;Al00002400;floppy
```
Ejemplo de cómo, ejecutando *xsh* en *tty** (cualquier *ttyXXX*), el usuario *user1* tiene acceso al *floppy* a través del grupo *floppy* todos los días (*Al* = *all days*) a todas horas (0000-2400). Ver sintaxis en: /etc/security/time.conf.

```
xsh; tty* ;user2;!Wk0900-1800;games, sound
```
Otro ejemplo de cómo *user2* tiene acceso a los juegos a través de *xsh* sobre *tty** fuera de horas de trabajo (*Wk* = *working days*).

`'limits.conf'`: Limita los recursos a los usuarios y evita ataques por DoS.

Formato:

```
domain type item value
domain: usuario, grupo o *
type: soft y hard
item: core (max tamaño del core), data (max tamaño del data), fsize (max tamaño archivo),
      memlock (max memoria bloqueada), nofile (max file open), rss (max conjunto residente),
      stack (max stack), cpu (max tiempo CPU minutos), nproc (max N.º procesos), as (max address
      space),
      maxlogins (max logings), priority (max prioridad)
```

```
* soft   core      0
* hard   rss       10000
@grupo1  hard      nproc       20
@grupo2  soft      nproc       30
@grupo2  hard      nproc       50
ftp      hard      nproc       0
@gru     -         maxlogins   4
```

Nota

Configurar:
Arranque Bios
/etc/security
Passwd óptimos
T. Max. Inactividad
Exports y Fsatb
lilo
SuidSgid
.rhosts y Xaccess
Usuarios noutilizados

d) **Mantener la seguridad del password de root**: utilizar como mínimo 6 caracteres, con uno, por lo menos, en mayúsculas o algún carácter '.-_', que sea no trivial; asimismo, es recomendable activar el envejecimiento para forzar a cambiarlo periódicamente, así como también limitar el número de veces con *password* incorrecto. Cambie además el parámetro *min = x* la entrada en /etc/pam.d/passwd para indicar el número mínimo de caracteres que se utilizarán en el *password* (*x* es el número de caracteres).

e) **No acceder al sistema como root**: crear una cuenta como *sysadm* y trabajar con ella. Si se accede remotamente, habrá siempre que utilizar shh para conectarse al *sysadm* y en caso de ser necesario, realizar un *su* - para trabajar como *root*.

f) **Tiempo máximo de inactividad**: inicializar la variable *TMOUT*, por ejemplo a 360 (valor expresado en segundos), que será el tiempo máximo de inactividad que esperará el *shell* antes de bloquearse; se puede poner en los archivos de configuración del *shell* (por ejemplo, /etc/profile, /.bashrc, ...). En caso de utilizar entornos gráficos (KDE, Gnome, etc.), activar el salvapantallas con *password*.

g) **Configuración del NFS en forma restrictiva**: en el /etc/exports exportar sólo lo necesario, no utilizar comodines (*wildcards*), permitir sólo el acceso de lectura y no permitir el acceso de escritura por *root*,

ANOTACIONES

por ejemplo, con */directorio_exportado host.domain.com (ro, root_squash)*.

h) Evitar arranques desde el *lilo* con parámetros: se puede iniciar el sistema como *linux single,* lo cual arrancará el sistema operativo en modo de usuario único. Configurar el sistema para que el arranque de este modo siempre sea con *password*. Para ello, en el archivo /etc/inittab verificar que existe la línea:

```
S:wait:/sbin/sulogin
```

y que tiene habilitado el /bin/sulogin. Además, el archivo /etc/lilo.conf debe tener los permisos adecuados para que nadie lo pueda modificar excepto el *root* (*chmod 600 /etc/lilo.conf*). Para prevenir cambios accidentales, cambiar el atributo de bloqueo con *chattr +i /etc/lilo.conf* (utilizar *-i* cuando se desee cambiar). Este archivo permite una serie de opciones que es conveniente considerar: *timeout* = 00, si el sistema tiene sólo un sistema operativo para que haga el *boot* inmediatamente, *restricted*, para evitar que puedan insertar comandos en el momento del *boot* como *linux init = /bin/sh,* y tener acceso como *root* sin autorización; en este caso, debe ir acompañado de password = *palabra-de-password*; si sólo se pone *password*, solicitará el *password* para cargar la imagen del *kernel*.

Nota

Consultar man *install-mbr* para reconfigurar el *master boot record* (MBR).

i) Control de la combinación Ctrl-Alt-Delete. Para evitar que se pueda apagar la máquina desde el teclado, insertar un comentario (#) en la primera columna de la línea siguiente:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

del archivo /etc/inittab. Activar los cambios con *telinit q*.

j) Evitar servicios no ofrecidos: bloquear el archivo /etc/services para no admitir servicios no contemplados bloqueando el archivo con *chattr +i /etc/services*.

k) Conexión del *root:* modificar el archivo /etc/securetty que contiene las TTY y VC (*virtual console*) en que se puede conectar el *root* dejando sólo una de cada, por ejemplo, *tty1* y *vc*/1, y si es necesario conectarse como *sysadm* y hacer un *su*.

l) Eliminar usuarios no utilizados: borrar los usuarios/grupos que no sean necesarios, incluidos los que vienen por defecto (por ejemplo, *operator*, *shutdown*, *ftp*, *uucp* , *games*, ...), y dejar sólo los ne-

ANOTACIONES

cesarios (*root*, *bin*, *daemon*, *sync*, *nobody*, *sysadm*) y los que se hayan creado con la instalación de paquetes o por comandos (lo mismo con /etc/group). Si el sistema es crítico, podría considerarse bloquear (*chattr +i file*) los archivos /etc/passwd, /etc/shadow, /etc/group, /etc/gsahdow para evitar su modificación (cuidado con esta acción, porque no permitirá cambiar posteriormente los *passwd*).

m) Montar las particiones en forma restrictiva: utilizar en /etc/fstab atributos para las particiones tales como **nosuid** (no permite suplantar el usuario o grupo sobre la partición), **nodev** (no interpreta dispositivos de caracteres o bloques sobre esta partición) y **noexec** (no permite la ejecución de archivos sobre esta partición). Por ejemplo:

```
/tmp /tmp ext2 defaults,nosuid,noexec 0 0
```

También es aconsejable montar el /boot en una partición separada y con atributos **ro**.

n) Protecciones varias: cambiar a 700 las protecciones de los archivos de /etc/init.d (servicios del sistema) para que sólo el *root* pueda modificarlos, arrancarlos o pararlos y modificar los archivos /etc/issue y /etc/issue.net para que no den información (sistema operativo, versión, ...) cuando alguien se conecta por telnet, ssh, etc.

o) SUID y SGID: un usuario podrá ejecutar como propietario un comando si tiene el bit SUID o SGID activado, lo cual se refleja como una 's' SUID (-rwsr-xr-x) y SGID (-r-xr-sr-x). Por lo tanto, es necesario quitar el bit (*chmod a-s file*) a los comandos que no lo necesitan. Estos archivos pueden ser buscados con:

```
find / -type f -perm -4000 o -perm -2000 –print
```

Proceded con cuidado respecto a los archivos que quita el SUID-GUID porque el comando podría quedar inútil.

p) Archivos sospechosos: buscar periódicamente archivos con nombres no usuales, ocultos o sin un uid/gid válido, como '...' (tres puntos), '.. ' (punto punto espacio), '..^G', para ello, habrá que utilizar:

```
find / -name ".*" -print | cat –v      o          find / name ".." -print
```

Para buscar uid/gid no válidos, utilizar: *find / -nouser o -nogroup* (cuidado, porque algunas instalaciones se hacen con un usuario que luego no está definido y que el administrador debe cambiar).

q) Conexión sin *password*: no permitir el archivo *.rhosts* en ningún usuario a no ser que sea estrictamente necesario (se recomienda utilizar ssh con clave pública en lugar de métodos basados en *.rhosts*).

r) X *Display manager*: modificar el archivo /etc/X11/xdm/Xaccess para indicar los *hosts* que se podrán conectar a través de XDM y evitar que cualquier *host* pueda tener una pantalla de *login*.

10.1.5. Monitorización

En este apartado veremos algunas de las herramientas más interesantes (por orden alfabético) que incorpora GNU/Linux (Debian) para la monitorización de sistema. No es una lista exhaustiva, sino una selección de las más utilizadas (se recomienda ver el *man ¡herramienta¿* para mayor información):

- **atsar**, **ac, sac, sysstat, isag**: herramientas de auditoría tales como *ac*, *last*, *accton*, *sa*, *atsar* o *isag* (*Interactive System Activity Grapher*) para la auditoría de recursos *hw* y *sw*.

- **aris**: monitor de *logs* del sistema para detectar incidentes de seguridad.

- **arpwatch; mon**: monitor de actividad ethernet/FDDI que indica si hay cambios en tablas MACIP; monitor de servicios de red.

- **diffmon, fcheck**: generación de informes sobre cambios en la configuración del sistema y monitorización del sistemas de ficheros para detectar intrusiones.

- **genpower**: monitor para gestionar los fallos de alimentación.

- **gkrellm**: monitorización gráfica de CPU, procesos (memoria), sistemas de ficheros y usuarios, disco, red, Internet, *swap*, etc.

ANOTACIONES

- **lcap, systune**: retira capacidades asignadas al *kernel* en el fichero /proc/sys/kernel y adapta según las necesidades con *systune*.

- **ksensors**: (*lm-sensors*): monitor de placa base (temperatura, alimentación, ventiladores, etc.).

- **log-analysys**: analizador de *logs* y *syslogs*.

- **lsof, fam**: monitor de archivos abiertos y *file alteration monitor*.

- **mgm**: medidor de recursos totalmente configurable.

- **powertweak y gpowertweak**: monitorización y modificación de diferentes parámetros del hardware, *kernel*, red, VFS, o VM (permite modificar algunos de los parámetros mostrados anteriormente sobre /proc).

- **pstree, fuser, killall**: monitor (y ejecutor) del /proc (árbol de procesos, procesos utilizando archivos, eliminación de procesos por nombre respectivamente).

- **qps, gtop, tkps, lavaps** (de más a menos amigable): monitores de procesos de varios tipos (generalmente utilizan información de /proc) y permiten ver recursos, *sockets*, archivos, entorno y otra información que éstos utilizan, así como administrar sus recursos/estados.

- **swatch**: monitor para la actividad del sistema a través de archivos de *log*.

- : **vtgrab**: monitor de pantalla de otra máquina similar a VNC.

- **whowatch**: herramienta en tiempo real para la monitorización de usuarios.

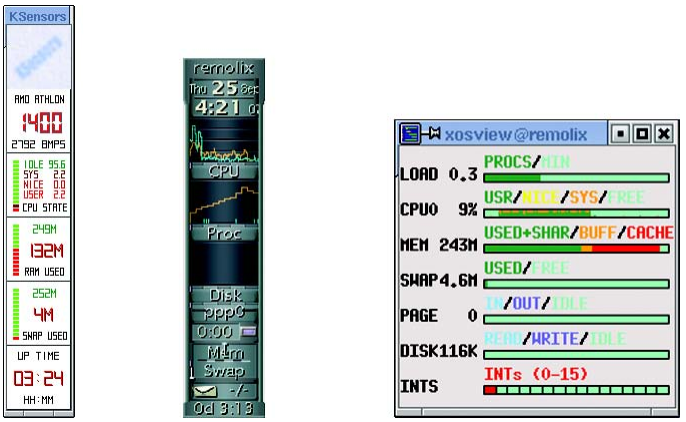
- **wmnd, dmachinemon**: monitor de tráfico de red y monitorización de un *cluster* por red.

- **xosview, si**: monitor de recursos gráfico y System Information.

ANOTACIONES

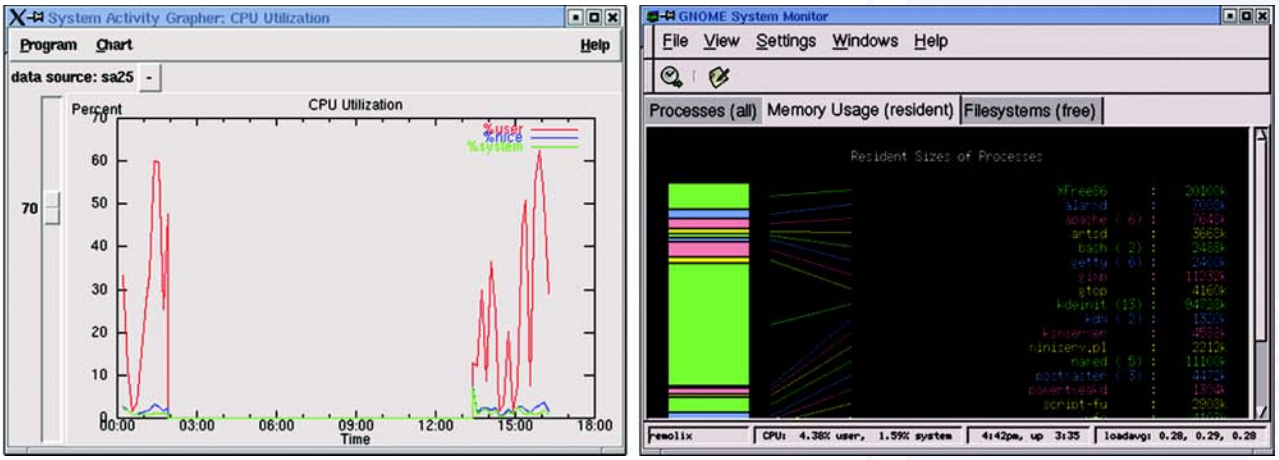
La figura 25 muestra las interfaces de ksensors, gkrellm y xosview, que presentan los resultados de la monitorización en tiempo real que están realizando.

Figura 25



A continuación se muestran las interfaces gráficas de isag y gtop. La interfaz isag obtiene la información generada por *systat* en /etc/cron.d/sysstat a través del comando *sa1* y *sa2,* en este caso, y es el acumulativo del día; mientras que *gtop* muestra una de sus posibles vistas con la ubicación proceso, memoria e información complementaria de CPU.

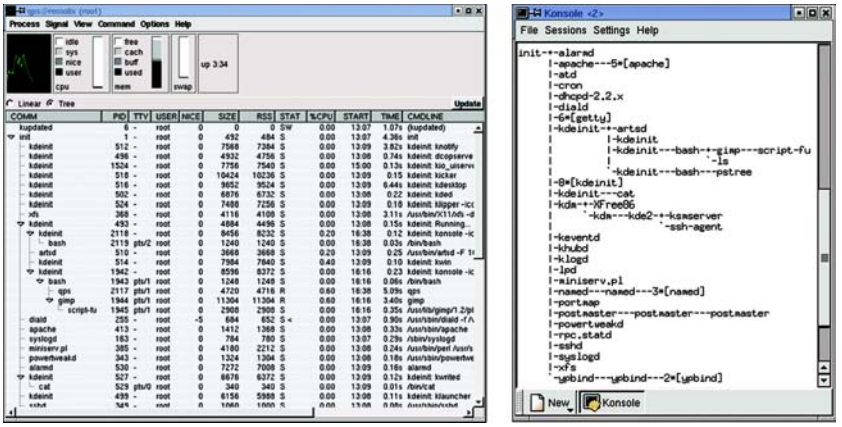
Figura 26.



Como herramienta para control de procesos se ha escogido *qps*, cuya interfaz se muestra a continuación junto con la misma información en formato texto dada por pstree. En la primera, se puede apre-

ciar también una monitorización de los recursos del sistema en forma cuantitativa.

Figura 27.



10.2. Actividades para el lector

Nota

Otras fuentes de referencias e información:
[Deb03c, LPD03b, Ibi03]

1) Realizar una monitorización completa del sistema con las herramientas que consideréis adecuadas y hacer un diagnóstico de la utilización de recursos y cuello de botella que podrían existir en el sistema. Simular la carga en el sistema del código de sumdis.c dado en la unidad que trata acerca de los *clusters*. Por ejemplo, utilizar:

```
sumdis 1 2000000
```

2) Cambiar los parámetros del *kernel* y del compilador y ejecutar el código mencionado en el punto anterior (*sumdis*.c) con, por ejemplo:

```
time ./sumdis 1 1000000
```

con ambos *kernels* y extraer conclusiones sobre los resultados.

ANOTACIONES

11. *Clustering*

Se denomina *cluster* (agrupación) de ordenadores a un grupo de ordenadores que trabajan con un fin común. Estos ordenadores agrupan hardware, redes de comunicación y software para trabajar conjuntamente como si fueran un único sistema. Existen muchas razones atrayentes para realizar estas agrupaciones, pero la principal es poder realizar el procesamiento de la información de forma más eficiente y rápida como si fuera un único sistema. Generalmente, un *cluster* trabaja sobre una red de área local (LAN) y permite una comunicación eficiente, si bien las máquinas se encuentran dentro de un espacio físico próximo. Una concepción mayor del concepto es la llamada *grid*, donde el objetivo es el mismo, pero implica agrupaciones de ordenadores unidos por redes de área extensa (WAN). Algunos autores consideran el *grid* como un *cluster* de *clusters* en un sentido 'global'. Si bien cada vez más la tecnología y los costes permiten estas aproximaciones, los esfuerzos y la complejidad de utilización de decenas o centenares (en algunos casos, miles) es muy grande. Sin embargo, las ventajas en tiempo de cómputo hacen que, aun así, este tipo de soluciones para el cómputo de altas prestaciones (HPC, *High Performance Computing*) sean consideradas muy atractivas y en constante evolución. En esta unidad se mostrarán algunas de las aproximaciones más difundidas y utilizadas. [RE98, Die98, Pro03b, Pro03d, Pro03e, Glo03a]

Nota

Un *cluster* es un conjunto de ordenadores en una LAN que trabajan con un objetivo común.

Las *grids* son agrupaciones de ordenadores unidos por redes de área extensa (WAN).

11.1. Introducción al HPC

Los avances en la tecnología han significado procesadores rápidos, de bajo coste y redes altamente eficientes, lo cual ha favorecido un cambio de la relación precio/prestaciones en favor de la utilización de sistemas de procesadores interconectados en lugar de un único procesador de alta velocidad. Este tipo de arquitectura se puede clasificar en dos configuraciones básicas:

- *Tighly coupled systems:* son sistemas donde la memoria es compartida por todos los procesadores (*shared memory systems*) y la

ANOTACIONES

memoria de todos ellos se 've' (por el programador) como una única memoria.

- *Loosely couple systems:* no comparten memoria (cada procesador posee la suya) y se comunican por mensajes pasados a través de una red (*message passing systems*).

En el primer caso son conocidos como sistemas paralelos de cómputo (*parallel processing system*) y en el segundo como sistemas distribuidos de cómputo (*distributed computing systems*). En este último caso podemos decir que un sistema distribuido es una colección de procesadores interconectados por una red donde cada uno tiene sus propios recursos (memoria y periféricos) y se comunican intercambiando mensajes por la red.

La historia de los sistemas informáticos es muy reciente (se puede decir que comienza en la década de los sesenta). En un principio eran sistemas grandes, pesados, caros, de pocos usuarios expertos, no accesibles, lentos. En la década de los setenta, la evolución permitió mejoras sustanciales llevadas a cabo por tareas interactivas (*interactive jobs*), tiempo compartido (*time sharing*), terminales y con una considerable reducción del tamaño. La década de los ochenta se caracteriza por un aumento notable de las prestaciones (hasta hoy en día) y una reducción del tamaño en los llamados *microcomputers*. Su evolución ha sido a través de las estaciones de trabajo (*workstations*) y los avances en redes (LAN de 10 Mbits/s y WAN de 56 Kbytes/s en 1973 a LAN de 1Gbit/s y WAN con ATM, *asynchronous transfer mode* de 1.2 Gbits/s en la actualidad), que es un factor fundamental en las aplicaciones multimedia actuales y de un futuro próximo. Los sistemas distribuidos, por su parte, comenzaron su historia en la década de los setenta (sistemas de 4 u 8 ordenadores) y su salto a la popularidad lo hicieron en la década de los noventa.

Si bien su administración/instalación/mantenimiento es compleja porque continúan creciendo, las razones básicas de su popularidad son el incremento de prestaciones que presentan en aplicaciones intrínsecamente distribuidas (aplicaciones que por su naturaleza son distribuidas), la información compartida por un conjunto de usuarios, compartir recursos, la alta tolerancia a los fallos y la posibilidad

ANOTACIONES

de expansión incremental (capacidad de agregar más nodos para aumentar las prestaciones y en forma incremental).

En los próximos apartados veremos algunos de los sistemas más comunes de procesamiento paralelo/distribuido así como los modelos de programación utilizados para generar código capaz de utilizar estas prestaciones.

11.1.1. Beowulf

Beowulf [RE98, RE02, Beo03, Rad03] es una arquitectura multiordenador que puede ser utilizada para aplicaciones paralelas/distribuidas (APD). El sistema consiste básicamente en un servidor y uno o más clientes conectados (generalmente) a través de Ethernet y sin la utilización de ningún hardware específico. Para explotar esta capacidad de cómputo, es necesario que los programadores tengan un modelo de programación distribuido que, si bien a través de UNIX es posible (*socket*, *rpc*), puede significar un esfuerzo considerable, ya que son modelos de programación a nivel de *systems calls* y lenguaje C, por ejemplo; pero este modo de trabajo puede ser considerado de bajo nivel.

Nota

Varias opciones:
- Beowulf
- OpenMosix
- Grid (Globus)

La capa de software aportada por sistemas tales como *Parallel Virtual Machine* (PVM) y *Message Passing Interface* (MPI) facilitan notablemente la abstracción del sistema y permite programar APD de modo sencillo y simple. La forma básica de trabajo es maestro-trabajadores (*master-workers*), en que existe un servidor que distribuye la tarea que realizarán los trabajadores. En grandes sistemas (por ejemplo, de 1.024 nodos) existe más de un maestro y nodos dedicados a tareas especiales como, por ejemplo, entrada/salida o monitorización.

Una de las principales diferencias entre Beowulf y un *Cluster of workstations* (COW) es que Beowulf se 've' como una única máquina donde los nodos se acceden remotamente, ya que no disponen de terminal (ni de teclado), mientras que un COW es una agrupación de ordenadores que pueden ser utilizados tanto por los usuarios de la COW, como por otros usuarios en forma interactiva a través de su pantalla y teclado. Hay que considerar que Beowulf no es un software que transforma el código del usuario en distribuido ni

ANOTACIONES

afecta al *kernel* del sistema operativo (como por ejemplo Mosix). Simplemente, es una forma de agrupación (*cluster*) de máquinas que ejecutan GNU/Linux y actúan como un superordenador. Obviamente, existe gran cantidad de herramientas que permiten obtener una configuración más fácil, bibliotecas o modificaciones al *kernel* para obtener mejores prestaciones, pero es posible construir un *cluster* Beowulf a partir de un GNU/Linux estándar y de software convencional. La construcción de un *cluster* Beowulf de dos nodos, por ejemplo, se puede llevar a cabo simplemente con las dos máquinas conectadas por Ethernet mediante un *hub*, una distribución de GNU/Linux estándar (Debian), el sistema de archivos compartido (NFS) y tener habilitados los servicios de red como rsh o ssh. En estas condiciones, se puede argumentar que se dispone de un *cluster* simple de dos nodos.

¿Dónde están los beneficios del cómputo paralelo? Veremos esto con un ejemplo [RE98]. Sea un programa para sumar números (por ejemplo, 4 + 5 + 6 ...) llamado *sumdis*:

```
#include <stdio.h>
  int main (int argc, char** argv)
    {
    float inicial, final, resultado, tmp;
    if (argc < 2) {
      printf ("Uso: %s N.° inicial N.° final\n",argv[0]);
      exit(1); }
    else {
      inicial = atol (argv[1]);
      final = atol (argv[2]);
      resultado = 0.0; }
    for (tmp = inicial; tmp <= final; tmp++){
      resultado + = tmp; }
    printf("%f\n", resultado)
    return 0; }
```

Si miramos la ejecución de este programa con, por ejemplo:

`time ./sumdis 1 1000000` (10^6)

se podrá observar que el tiempo en una máquina Debian 2.4.18 con AMD athjon 1.400 MHz 256 Mb RAM es (aproximadamente)

real = 0,013 s y *user* = 0,010 s, es decir, 13 ms en total y 10 ms en zona de usuario. Si, en cambio, hacemos:

`time ./suma 1 16000000` (16×10^6)

el tiempo será *real* = 182 ms, es decir, 14 veces más, lo cual, si se considera 160.000.000 (160×10^6), el tiempo será del orden de decenas de minutos.

La idea del cómputo distribuido es: si disponemos de un *cluster* de 4 máquinas (nodo1-nodo4) con un servidor y donde el fichero se comparte por NFS, sería interesante dividir la ejecución a través de *rsh* (no recomendable, pero como ejemplo es aceptable), de modo que el primero sume de 1 a 40.000.000, el segundo de 40.000.001 a 80.000.000, el tercero de 80.000.001 a 120.000.000 y el cuarto de 120.000.001 a 160.000.000. Los siguientes comandos muestran una posibilidad. Consideramos que el sistema tiene el directorio /home compartido por NFS y el usuario (*nteum*) que ejecutará el *script* tiene adecuadamente configurado el *.rhosts* para acceder sin contraseña a su cuenta. Además, si se ha activado el *tcpd* en /etc/inetd.conf en la línea de *rsh*, debe existir la correspondiente en el /etc/hosts.allow, que permita acceder a las cuatro máquinas del *cluster*:

`mkfifo salida` Crea una cola *fifo* en /home/nteum}

```
./distr.sh \& time cat salida | awk '{total + = $1 } END {printf "%lf", total}'
```

Ejecuta el comando *distr.sh;* se recolectan los resultados y se suman mientras se mide el tiempo de ejecución

El *shell script distr.sh* puede ser algo como:

```
rsh nodo1 /home/nteum/sumdis 1 40000000 > /home/nteum/salida < /dev/null &
rsh nodo2 /home/nteum/sumdis 40000001 80000000 > /home/nteum/salida < /dev/null &
rsh nodo3 /home/nteum/sumdis 80000001 120000000 > /home/nteum/salida < /dev/null &
rsh nodo4 /home/nteum/sumdis 120000001 160000000 > /home/nteum/salida < /dev/null &
```

Podremos observar que el tiempo se reduce notablemente y no exactamente en forma lineal, pero muy próxima. Obviamente, este ejemplo es muy simple y sólo con fines demostrativos. Los programadores utilizan bibliotecas que les permiten realizar el tiempo de ejecución, la creación y comunicación de procesos en un sistema distribuido (por ejemplo, PVM y MPI).

11.1.2. ¿Cómo hay que programar para aprovechar la concurrencia?

Existen diversas maneras de expresar la concurrencia en un programa. Las dos más comunes son:

1) Utilizando *threads* (o procesos).

2) Utilizando procesos en diferentes procesadores que se comunican por mensajes (MPS, *Message Passing System*).

Ambos métodos pueden ser implementados sobre diferentes configuraciones hardware (memoria compartida o mensajes), pero los sistemas MPS presentan el problema de latencia y velocidad de los mensajes en la red, lo que puede resultar un factor negativo. Sin embargo, con el avance de las tecnologías de red estos sistemas han crecido en popularidad (y en cantidad). Un mensaje es sumamente simple:

```
send(destino,msg), recv(origen,msg)
```

Las API más comunes hoy en día son PVM y MPI y además no limitan la posibilidad de utilizar *threads* (aunque a nivel local) y tener concurrencia entre procesamiento y entrada/salida. En cambio, en una máquina de memoria compartida (SHM, *Shared Memory*) sólo es posible utilizar *threads* y tiene el problema grave de la escalabilidad, ya que todos los procesadores utilizan la misma memoria y el número de procesadores en el sistema está limitado por el ancho de banda de la memoria.

En resumen, podemos concluir:

1) Proliferación de máquinas multitareas (multiusuario) conectadas por red con servicios distribuidos (NFS y NIS YP).

ANOTACIONES

2) Son sistemas heterogéneos con sistemas operativos de tipo NOS (*Networked Operating System*) que ofrecen una serie de servicios distribuidos y remotos.

3) La programación de aplicaciones distribuidas se puede efectuar a diferentes niveles:

a) Utilizando un modelo cliente-servidor y programando a bajo nivel (*sockets*).

b) El mismo modelo, pero con API de "alto" nivel (PVM, MPI).

c) Utilizando otros modelos de programación como, por ejemplo, programación orientada a objetos distribuidos (RMI, CORBA, Agents...).

PVM, *Parallel Virtual Machine*

PVM [Pro03e] es una API que permite generar, desde el punto de vista de la aplicación, una colección dinámica de ordenadores, que constituyen una máquina virtual (VM). Las tareas pueden ser creadas dinámicamente (*spawned*) y/o eliminadas (*killed*) y cualquier tarea PVM puede enviar un mensaje a otra. No existe un límite en el tamaño o número de mensajes (según las especificaciones, aunque pueden existir combinaciones hardware/sistema operativo que den como resultado limitaciones en el tamaño del mensaje) y el modelo soporta: tolerancia a fallos, control de recursos, control de procesos, heterogeneidad en las redes y en los *hosts*.

El sistema (VM) dispone de herramientas para el control de recursos (agregar o quitar *hosts* de la máquina virtual), control de procesos (creación/eliminación dinámica de procesos), diferentes modelos de comunicación (*blocking send*, *blocking/nonblocking receive*, *multicast*), grupos de tareas dinámicos (una tarea puede anexarse a un grupo o no dinámicamente) y tolerancia a fallos (la VM detecta el fallo y se puede reconfigurar).

La estructura de PVM se basa por un lado en el *daemon* (*pvm3d*) que reside en cada máquina y se interconectan utilizando UDP y por el otro, la biblioteca de PVM (*libpvm3.a*), que contiene todas las rutinas para enviar/recibir mensajes, crear/eliminar procesos, grupos, sincronización, ... y que utilizará la aplicación distribuida.

ANOTACIONES

PVM dispone de una consola (*pvm*) que permite poner en marcha el *daemon*, crear la VM, ejecutar aplicaciones, etc. Es recomendable instalar el software desde la distribución, ya que su compilación requiere cierta 'dedicación'.

Para realizar una aplicación paralela/distribuida en PVM, se puede partir de la versión serie o mirando la estructura física del problema y determinar qué partes pueden ser concurrente (independientes). Las partes concurrentes serán candidatas a reescribirse como código paralelo. Además, se debe considerar si es posible reemplazar las funciones algebraicas por sus versiones paralelizadas (por ejemplo, ScaLapack, Scalable Linear Algebra Package, disponibles en Debian como *scalapack-pvm | mpich-test | dev, scalapack1-pvm | mpich* según sean para PVM o MPI). También es conveniente averiguar si hay alguna aplicación similar paralela (http://www.epm.ornl.gov/pvm) que pueda orientarnos sobre el modo de construcción de la aplicación paralela.

Paralelizar un programa no es una tarea fácil, ya que se debe tener en cuenta la ley de Amdahl.

La **ley de Amdahl** afirma que el incremento de velocidad (*speedup*) está limitado por la fracción de código (*f*) que puede ser paralelizado:

$$speedup = 1/(1 - f)$$

Esta ley implica que una aplicación secuencial $f = 0$ y el $speedup = 1$, con todo el código paralelo $f = 1$ y $speedup$ = infinito (!), con valores posibles, 90% del código paralelo significa un $speedup = 10$, pero con $f = 0{,}99$ el $speedup = 100$. Esta limitación se puede evitar con algoritmos escalables y diferentes modelos de aplicación:

a) Maestro-trabajador: el maestro inicia a todos los trabajadores y coordina su trabajo y entrada/salida.

b) *Single Process Multiple Data* (SPMD): mismo programa que se ejecuta con diferentes conjuntos de datos.

c) Funcional: varios programas que realizan una función diferente en la aplicación.

ANOTACIONES

Nota

Ley de Amdahl:
$speedup = 1/(1f)$
f = fracción código paralelo

Con la consola *pvm* y a través del comando *add* se puede configurar la VM añadiendo todos los nodos. En cada uno de ellos debe existir el directorio ~/pvm3/bin/LINUX con los binarios de la aplicación. Deben estar declaradas la variables *PVM ROOT* = Directorio, donde se encuentra lib/LINUX/libpvm3.a y *PVM ARCH* = LINUX, que se pueden poner, por ejemplo, en el archivo /.cshrc. El *shell* por defecto del usuario (generalmente un usuario NIS o si no, en cada máquina debe existir el mismo usuario con la misma contraseña) debe ser el *csh* (si utilizamos el *rsh* como medio de ejecución remota) y debe estar configurado el archivo /.rhosts para permitir el acceso sin *password* a cada nodo. El paquete PVM incorpora un *rsh-pvm* que se puede encontrar en /usr/lib/pvm3/bin como *rsh* específico para PVM (ver la documentación), ya que existen algunas distribuciones que no lo incluyen por motivos de seguridad.

Como ejemplo de programación PVM, se muestra un programa del tipo servidor-clientes, donde el servidor crea los hijos, les envía datos, éstos circulan los datos una determinada cantidad de veces entre los hijos (el de izquierda recibe un dato, lo procesa y se lo envía al de la derecha) mientras el padre espera que cada hijo termine.

Nota

Compilar PVM:
gcc -O -I/usr/include/ -o output output.c -lpvm3

Ejemplo en PVM: *master.c*

Para compilar en Debian:

```
gcc -O -I/usr/include/ -o master master.c -lpvm3
```

Ejecución (1.º ejecutar el *daemon pvmd* con *pvm → quit*): *master* ↵

```
#include <stdio.h>
#include "pvm3.h"
#define SLAVENAME "/home/nteum/pvm3/cliente"
main() {
  int mytid, tids[20], n, nproc, numt, i, who, msgtype,loops;
  float data[10]; int n_veces;
  if( pvm_parent() ==PvmNoParent ){
    /*Retorna si es el proceso padre o hijo*/
    loops = 1;
    printf("\n Cuántos hijos (120)? ");
```

```
    scanf("%d", &nproc);
    printf("\n Cuántos bucles de comunicación hijo-hijo (1 - 5000)? ");
    scanf("%d", &loops); }
      /*Redirecciona la entrada/salida de los hijos al padre*/
  pvm_catchout(stdout);
      /*Crea los hijos*/
  numt = pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
/   *Inicia un nuevo proceso, 1.° :ejecutable hijo, 2.° : argv,
    3.° :opciones, 4.° :donde, 5.° :N.° copias, 6.° :matriz de id*/
  printf("Resultado del Spawn: %d \n", numt);

    /*Ha podido?*/
  if( numt < nproc ){
    printf("Error creando los hijos. Código de error:\n");
    for( i = numt ; i<nproc ; i++ ) {printf("Tid %d %d\n",i,tids[i]); }
    for( i = 0 ; i<numt ; i++ ){ pvm_kill( tids[i] ); }
    /*Mata los procesos con id en tids*/
    pvm_exit(); exit();
    /*Termina*/ }

  /*Inicio del programa de padre, inicializa los datos*/
  n = 10;
  for( i = 0 ; i<n ; i++ ){data[i] = 2.0;}
    /*Broadcast con datos iniciales a los slave*/}
  pvm_initsend(PvmDataDefault);.
    /*Borra el buffer y especifica el message encoding*/
  pvm_pkint(&loops, 1, 1);
    /*Empaqueta un dato en el buffer, 2.° N.°, 3*:stride*/
  pvm_pkint(&nproc, 1, 1);
  pvm_pkint(tids, nproc, 1);
  pvm_pkint(&n, 1, 1);
  pvm_pkfloat(data, n, 1);
  pvm_mcast(tids, nproc, 0);
    /*Multicast en el buffer a los tids y espera el resultado de los hijos */
  msgtype = 5;
  for( i = 0 ; i < nproc ; i++ ){
    pvm_recv( -1, msgtype );
    /*Recibe un mensaje, -1 :de cualquiera, 2.ª :tag del msg*/
    pvm_upkint( &who, 1, 1 );
    /*Desempaqueta*/
    printf("Terminó %d\n",who);
    }
  pvm_exit();
}
```

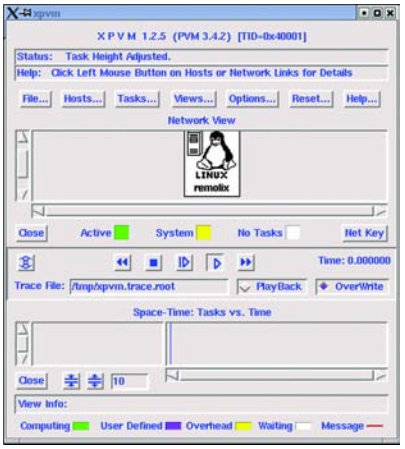
Ejemplo de PVM: *cliente.c*

Para compilar:

```
gcc -O -I/usr/share/pvm3/include/ -L/usr/share/pvm3/lib/LINUX
-o cliente cliente.c -lpvm3
#include <stdio.h>
#include "pvm3.h"
   main() {
      int mytid;/*Mi task id*/
      int tids[20];/*Task ids*/
      int n, me, i, nproc, master, msgtype, loops; float data[10];
      long result[4]; float work();

      mytid = pvm_mytid();
      msgtype = 0;
      pvm_recv( -1, msgtype );
      pvm_upkint(&loops, 1, 1);
      pvm_upkint(&nproc, 1, 1);
      pvm_upkint(tids, nproc, 1);
      pvm_upkint(&n, 1, 1);
      pvm_upkfloat(data, n, 1);

        /*Determina qué hijo es (0 -- nproc-1) */
      for( i = 0; i < nproc ; i++ )
        if( mytid == tids[i] ){ me = i; break; }
        /*Procesa y pasa los datos entre vecinos*/
      work( me, data, tids, nproc, loops );

        /*Envía los datos al máster */
      pvm_initsend( PvmDataDefault );
      pvm_pkint( &me, 1, 1 );
      msgtype = 5;
      master = pvm_parent(); /*Averigua quién lo creó*/
      pvm_send( master, msgtype);
      pvm_exit();
   }

float work(me, data, tids, nproc, loops)
   int me, *tids, nproc; float *data; {
   int i,j, dest; float psum = 0.0, sum = 0.1;
   for (j = 1; j <= loops; j++){
      pvm_initsend( PvmDataDefault );
      pvm_pkfloat( &sum, 1, 1 );
      dest = me + 1;
      if( dest == nproc ) dest = 0;
      pvm_send( tids[dest], 22 );
      i = me - 1;
      if (me == 0 ) i = nproc-1;
      pvm_recv( tids[i], 22 );
      pvm_upkfloat( &psum, 1, 1 );
   }
}
```

El programador cuenta con la gran ayuda de una interfaz gráfica (ver figura 28) que actúa como consola y monitor de PVM llamada *xpvm* (en Debian XPVM), que permite configurar la VM, ejecutar procesos, visualizar la interacción entre tareas (comunicaciones), estados, información, etc.

Figura 28.



MPI, *message passing interface*

La definición de la API de MPI [Pro03b, Pro03c] ha sido el trabajo resultante del MPI Fourum (MPIF), que es un consorcio de más de 40 organizaciones. MPI tiene influencias de diferentes arquitecturas, lenguajes y trabajos en el mundo del paralelismo como son: WRC (Ibm), Intel NX/2, Express, nCUBE, Vertex, p4, Parmac y contribuciones de ZipCode, Chimp, PVM, Chamaleon, PICL. El principal objetivo de MPIF fue diseñar una API, sin relación particular con ningún compilador ni biblioteca, tal que permitiera la comunicación eficiente (*memory-to-memory copy*), cómputo y comunicación concurrente y descarga de comunicación, siempre y cuando exista un coprocesador de comunicaciones. Además, que soportara el desarrollo en ambientes heterogéneos, con interfaz C y F77 (incluyendo C++, F90), donde la comunicación fuera fiable y los fallos resueltos por el sistema. La API también debía tener interfaz para diferentes entornos (PVM, NX, Express, p4, ...), disponer una implementación adaptable a diferentes plataformas con cambios insignificantes y que no interfiera con el sistema operativo (*thread-safety*). Esta API fue diseñada especialmente para programadores que utilizaran el *Message Passing Paradigm* (MPP) en C y F77 para aprovechar la característica más relevante: la portabilidad. El MPP se puede ejecutar sobre máquinas multiprocesadores, redes de WS e incluso sobre máquinas de memoria compartida. La versión MPI1 (la versión más extendida) no

ANOTACIONES

soporta creación (*spawn*) dinámica de tareas, pero MPI2 (en creciente evolución) sí que lo hace.

Muchos aspectos han sido diseñados para aprovechar las ventajas del hardware de comunicaciones sobre SPC (*scalable parallel computers*) y el estándar ha sido aceptado mayoritariamente por los fabricantes de hardware paralelo y distribuido (SGI, SUN, Cray, HPConvex, IBM, Parsystec, ...). Existen versiones freeware (por ejemplo, *mpich*) (que son totalmente compatibles con las implementaciones comerciales realizadas por los fabricantes de hardware) e incluyen comunicaciones punto a punto, operaciones colectivas y grupos de procesos, contexto de comunicaciones y topología, soporte para F77 y C y un entorno de control, administración y *profiling*. Pero existen también algunos puntos no resueltos como son: operaciones SHM, ejecución remota, herramientas de construcción de programas, depuración, control de *threads*, administración de tareas, funciones de entrada/salida concurrentes (la mayor parte de estos problemas de falta de herramientas están resueltos en la version 2 de la API MPI2). El funcionamiento en MPI1, al no tener creación dinámica de procesos, es muy simple, ya que de tantos procesos como tareas existan, autónomos y ejecutando su propio código estilo MIMD (*Multiple Instruction Multiple Data*) y comunicándose vía llamadas MPI. El código puede ser secuencial o *multithread* (concurrentes) y MPI funciona en modo threadsafe, es decir, se pueden utilizar llamadas a MPI en *threads* concurrentes, ya que las llamadas son reentrantes.

Para la instalación de MPI se recomienda utilizar la distribución, ya que su compilación es sumamente compleja (por las dependencias que necesita de otros paquetes). Debian *Woody* incluye la versión Mpich 1.2.1 (septiembre de 2000) que implementa el estándar MPI 1.2 y algunas partes de MPI 2 (como, por ejemplo, entrada/salida paralela). Además, esta misma distribución incluye otra implementación de MPI llamada LAMMPI versión 6.5.3 (paquetes *lam** y documentación en //usr/doc/lam-runtime/release.html). Las dos implementaciones son iguales, excepto por algunas pequeñas diferencias. Toda la información sobre Mpich se puede encontrar (después de instalar los paquetes *mpich**) en /usr/doc/mpi. Mpich necesita *rsh* para ejecutarse en otras máquinas, lo cual significa insertar en el directorio del usuario un archivo ~/.rhosts con líneas del siguiente formato: *host username* para permitir a *username* entrar en *host* sin *password* (al igual que PVM). Se debe tener en cuenta que hay que instalar el paquete rshserver sobre todas la máquinas y si se tiene *tcpd* en /etc/inetd.conf sobre *rsh.d*, se debe habilitar los *hosts* en /etc/hosts.allow. Además, se deberá tener montado

ANOTACIONES

el directorio del usuario por NFS en todas las máquinas y en el fichero /etc/mpich/machines.LINUX se deberá poner el nombre (*hostname*) de todas las máquinas que forman el *cluster* (una máquina por línea, por defecto aparece *localhost*). Además, el usuario deberá tener como *shell* por defecto el *Csh*.

Sobre Debian se puede instalar el paquete *update-cluster* para ayudar en su administración. La instalación sobre Debian *Woody* de Mpich utiliza *ssh* en lugar de *rsh* por motivos de seguridad, si bien existe un enlace de *rsh* → *ssh* por compatibilidad. La única diferencia es que se deberán utilizar los mecanismos de validación de *ssh* para la conexión sin contraseña a través de los ficheros correspondientes. De lo contrario, por cada proceso que se ejecute se deberá introducir la contraseña antes de la ejecución. Para permitir la conexión entre máquinas sin contraseña con *ssh*, se deberá hacer, por ejemplo, como usuario *nteum*:

`ssh-keygen -t rsa` Genera una llave pública (*id_rsa.pub*) y otra privada (*id_rsa*) para el algoritmo RSA en el directorio "*/.ssh*. No introducir la contraseña, o se pedirá en la conexión.

```
cd ./ssh
cp id_rsa.pub authorized_keys
```

authorized_keys es equivalente a *.rhosts* y se debe copiar en cada máquina en el mismo directorio; deberá tener todas las llaves públicas de los clientes que se quieren conectar sin contraseña.

El formato de *authorized_keys* es similar:

```
ssh rsa  AAAAB3NzaC1yc2EAAAABIwAAAIEA2Ju2ghfv2FRabF/BD3WpLeBFHdb7v/
         yay6Hni7KBHZka1BocKyp1jVyhufD071Y41hvgCTx6JR5rxJUx58LvYmAXksBeU8np2
         4aaHpHMQbN2lRQhJ15HRXhpMOU4ZZOFgahI9ltA1Fqgf9jIiRj1aJsaG3Mv7VDNQ4/
         6UHrmSqE = nteum@locahost
```

Para probar, se puede hacer *ssh localhost* y se debe poder entrar sin contraseña. Tener en cuenta que si se instala Mpich y LAM-MPI, el *mpirun* de Mpich se llamará *mpirun.mpich* y el *mpirun* será el de LAM-MPI. Es importante recordar que *mpirun* de LAM utilizará el *daemon lamboot* para formar la topología distribuida de la VM.

El *daemon lamboot* ha sido diseñado para que los usuarios puedan ejecutar programas distribuidos sin tener permisos de *root* (también permite ejecutar programas en una VM sin llamadas a MPI). Por ello, para ejecutar el *mpirun* se deberá hacer como un usuario diferente de *root* y ejecutar antes *lamboot*. El **lamboot** utiliza un archivo de configuración en /etc/lam para la definición por defecto de los nodos (ver *bhost**) y consultar la documentación para mayor información. [Lam03]

Para compilar programas MMPI, se puede utilizar el comando *mpicc* (por ejemplo, *mpicc o test test.c*), que acepta todas las opciones de *gcc* aunque es recomendable utilizar (con modificaciones) algunos de los *makefiles* que se hallan en los ejemplos /usr/doc/mpich/examples. También se puede utilizar *mpireconfig Makefile,* que utiliza como entrada el archivo *Makefile.in* para generar el *makefile* y es mucho más fácil de modificar. Después se podrá hacer:

```
mpirun -np 8 programa
```

o bien:

```
mpirun.mpich -np 8 programa
```

donde *np* es el número de procesos o procesadores en que se ejecutará el programa (8, en este caso). Se puede poner el número que se desee, ya que Mpich intentará distribuir los procesos en forma equilibrada entre todas las máquinas de /etc/mpich/machines.LINUX. Si hay más procesos que procesadores, Mpich utilizará las características de intercambio de tareas de GNU/Linux para simular la ejecución paralela. En Debian y en el directorio /usr/doc/mpich-doc (un enlace a /usr/share/doc/mpich-doc) se encuentra toda la documentación en diferentes formatos (comandos, API de MPI, etc.).

Ejemplo

Compilar MPI:

```
mpicc -O
-o output output.c
```

A continuación veremos dos ejemplos (que se incluyen con la distribución Mpich 1.2.1 en el directorio /usr/doc/mpich/examples). Srtest es un programa simple para establecer comunicaciones entre procesos punto a punto, y *cpi* calcula valor de π en forma distribuida (por integración).

Ejemplo

Ejecutar Mpich:

```
mpirun.mpich np
N.º procesos output
```

Comunicaciones punto a punto: `srtest.c`

Para compilar: `mpicc .O .o srtest srtest.c`

ANOTACIONES

Ejecución Mpich: `mpirun.mpich -np N.º_procesos srtest` (solicitará la contraseña [*N.º procesos* - 1] veces si no se tiene el acceso directo por *ssh*).

Ejecución LAM: `mpirun np N.º_procesos srtest` (debe ser un usuario diferente de *root*).

```
#include "mpi.h"
#include <stdio.h>
#define BUFLEN 512
int main(int argc, char *argv[]) {
   int myid, numprocs, next, namelen;
   char buffer[BUFLEN], processor_name[MPI_MAX_PROCESSOR_NAME];
   MPI_Status status;
   MPI_Init(&argc,&argv);
      /* Debe ponerse antes de otras llamadas MPI, siempre */
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
      /*Integra el proceso en un grupo de comunicaciones*/
MPI_Get_processor_name(processor_name,&namelen);
      /*Obtiene el nombre del procesador*/
   fprintf(stderr,"Proceso %d sobre %s\n", myid, processor_name);
   strcpy(buffer,"Hola Pueblo");
   if (myid ==numprocs1) next = 0;
   else next = myid+1;
   if (myid ==0) {
        /*Si es el inicial, envía string de buffer*/.
      printf("%d Envío '%s' \n",myid,buffer);
      MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
        /*Blocking Send, 1 o :buffer, 2 o :size, 3 o :tipo, 4 o :destino, 5 o :tag, 6 o
:contexto*/
      /*MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, MPI_PROC_NULL, 299,MPI_COMM_WORLD);*/
      printf("%d recibiendo \n",myid);
        /* Blocking Recv, 1 o :buffer, 2 o :size, 3 o :tipo, 4 o :fuente, 5 o :tag, 6 o
:contexto, 7 o :status*/
      MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
      printf("%d recibió '%s' \n",myid,buffer) }
   else {
      printf("%d recibiendo \n",myid);
      MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,status);
        /*MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_PROC_NULL, 299,MPI_COMM_WORLD,&status);*/
      printf("%d recibió '%s' \n",myid,buffer);
      MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
      printf("%d envió '%s' \n",myid,buffer);}
      MPI_Barrier(MPI_COMM_WORLD);
        /*Sincroniza todos los procesos*/
      MPI_Finalize();
        /*Libera los recursos y termina*/
   return (0);
}
```

Cálculo de PI distribuido: *cpi.c*

Para compilar: *mpicc O o cpi cpi.c*.

Ejecución Mpich: `mpirun.mpich -np N.º procesos cpi`
(solicitará la contraseña (N.º procesos – 1) veces si no se tiene el acceso directo por *ssh*).

Ejecución LAM: `mpirun np N.º procesos cpi`
(debe ser un usuario diferente de *root*).

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
   double f( double );
   double f( double a) { return (4.0 / (1.0 + a*a)); }
   int main( int argc, char *argv[] ) {
      int done = 0, n, myid, numprocs, i;
      double PI25DT = 3.14159265358979323846264 3;
      double mypi, pi, h, sum, x;
      double startwtime = 0.0, endwtime;
      int namelen;
      char processor_name[MPI_MAX_PROCESSOR_NAME];
      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD,&numprocs); /*Indica el número de procesos en el grupo*/
      MPI_Comm_rank(MPI_COMM_WORLD,&myid); /*Id del proceso*/
      MPI_Get_processor_name(processor_name,&namelen); /*Nombre del proceso*/
      fprintf(stderr,"Proceso %d sobre %s\n", myid, processor_name);
      n = 0;
      while (!done) {
         if (myid ==0) { /*Si es el primero...*/
            if (n ==0) n = 100; else n = 0;
            startwtime = MPI_Wtime();} /* Time Clock */
         MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); /*Broadcast al resto*/
            /*Envía desde el 4.º arg. a todos los procesos del grupo
            Los restantes que no son 0 copiarán el buffer desde 4 o arg -proceso 0-*/
            /*1.º :buffer, 2.º :size, 3.º :tipo, 5.º :grupo */
         if (n == 0) done = 1;
         else {
            h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i + = numprocs) {
               x = h * ((double)i - 0.5);
               sum + = f(x); }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
               /* Combina los elementos del Send Buffer de cada proceso del grupo
               usando la operación MPI_SUM y retorna el resultado en el Recv Buffer.
```

```
            Debe ser llamada por todos los procesos del grupo usando los mismos argumentos*/
            /*1.° :sendbuffer, 2.° :recvbuffer, 3.° :size, 4.° :tipo, 5.° :oper,
            6.° :root, 7.° :contexto*/
         if (myid == 0){ /*Sólo el P0 imprime el resultado*/
            printf("Pi es aproximadamente %.16f, el error es %.16f\n", pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("Tiempo de ejecución = %f\n", endwtime-startwtime); }
         }
      }
      MPI_Finalize(); /*Libera recursos y termina*/
      return 0;
}
```

Al igual que en PVM existe XPVM, en MPI existe una aplicación análoga (más sofisticada) llamada XMPI (en Debian *xmpi*). También es posible instalar una biblioteca, *libxmpi3*, que implementa el protocolo XMPI para analizar gráficamente programas MPI con más detalles que los ofrecidos por *xmpi*.

11.2. OpenMosix

OpenMosix [ea02, Pro03d] es un paquete software que transforma un conjunto de máquinas conectadas por red bajo GNU/Linux en un *cluster*. Éste equilibra la carga automáticamente entre los diferentes nodos del *cluster* y los nodos pueden unirse o dejar el *cluster* sin interrumpir el servicio. La carga se distribuye entre los nodos teniendo en cuenta la velocidad de la conexión y la CPU. OpenMosix forma parte del *kernel* (a través de un Linux Kernel Patch) y mantiene total compatibilidad con GNU/Linux, los programas de usuario, archivos y recursos. Otra característica de OpenMosix es que incorpora un potente y optimizado sistema de archivos (oMFS) para aplicaciones de HPC (*High Performance Computing*). En Debian se puede instalar OpenMosix desde *openmosix-dev* (bibliotecas y *headers*), *kernel-pacth-openmosix* (OpenMosix *patch*), *openmosix* (herramientas de administración). También de manera análoga se puede instalar **mosix** (ver la documentación para las diferencias, sobre todo de licencias, entre Mosix y OpenMosix).

OpenMosix utiliza un archivo de configuración que se encuentra generalmente en /etc (ver documentación por antiguas versiones de

ANOTACIONES

este archivo) y se llama *openmosix.map*, el cual debe estar en cada nodo. Su formato es muy simple y cada línea tiene tres campos:

OpenMosix-Nodo_ID IP-Address(o hostname) Range-size

Un ejemplo sería:

```
1 node1          1
2 node2          1
3 node3          1
4 192.168.1.1    1
5 192.168.1.2    1
```

También se puede utilizar un rango donde el ID y la IP se incrementan respectivamente con: 1 192.168.1.1 4. Hay que procurar tener en cada nodo la misma configuración y la misma versión de OpenMosix. Para ejecutar OpenMosix, se debe hacer en cada nodo:

```
setpe -w -f /etc/openmosix.map
```

También se puede utilizar el *script* OpenMosix (copiándolo de *userspace-tools* a /etc/init.d) para arrancarlo durante el *boot*.

El sistema de archivos oMFS permite el acceso remoto a todos los archivos en el *cluster* como si ellos estuvieran localmente montados. Los sistemas de archivos (FS) de sus otros nodos pueden ser montados sobre /mfs y, por lo tanto, los archivos de /home sobre el nodo 3 se verán en cada máquina en /mfs/3/home.

Todos los UIDs (User IDs) y GIDs (Group IDs) del FS sobre cada nodo del *cluster* deberán se iguales (podría utilizarse OpenLdap para este fin).

Para montar el oMFS, se deberá modificar /etc/fstab con una entrada como:

```
mfs_mnt /mfs mfs dfsa = 1 0 0
```

ANOTACIONES

Para habilitarlo o para inhibirlo:

```
mfs_mnt /mfs mfs dfsa = 0 0 0
```

Después, el FS de cada nodo se verá en mfs/[openMosixNode ID]/. Una vez instalado, se podría ejecutar varias veces un *script* muy simple como, por ejemplo (ver *Howto* de OpenMosix):

```
awk 'BEGIN {for(i = 0;i<10000;i++)for(j = 0;j<10000;j++);}'
```

Y, a continuación, observar el comportamiento con *mosmom* o con *openmosixview* (recomendado). OpenMosix posee un *daemon* (*omdiscd*), que permite configurar automáticamente el *cluster* eliminando la necesidad de editar y configurar /etc/openmosix.map. Este *daemon* utiliza *multicast* para indicarles a los otros nodos que él también es un nodo OpenMosix, por lo cual, una vez arrancado el *omdiscd*, este *daemon* se unirá al *cluster* de forma automática. Para ello, es necesario tener el *default routing* (GW) de la red bien configurado. Una vez ejecutado (*omdiscd*), se generarán una serie de mensajes que indicarán el estado del *cluster* y la configuración. Con el comando *showmap* se podrá ver la nueva configuración generada por *omdiscd*. OpenMosix provee un conjunto de herramientas para que el administrador pueda configurar y sintoniza el *cluster* OpenMosix. Estas tareas se pueden realizar con herramientas en el espacio de usuario (*migrate*, *mon*, *mosctl*, *mosrun*) o a través de la interfaz /proc/hpc. Es importante tener en cuenta que hasta OpenMosix versión 2.4.16 la interfaz se llamaba /proc/mosix y desde la versión 2.4.17 se llama /proc/hpc.

A continuación, se presenta un resumen de las herramientas de configuración que se ejecutan en espacio de usuario, para las /proc/hpc consultar las referencias:

- **migrate** [PID] [OpenMosix ID]: envía una petición de migración a un proceso.

- **mon**: es un monitor con interfaz de texto que muestra información sobre el *cluster* a través de un diagrama de barras.

ANOTACIONES

- **mosctl**: es la herramienta de configuración de OpenMosix. A través de las opciones (stay, lstay, block, quiet, mfs, expel, bring, gettune, getyard, getdecay) se le pueden indicar a los procesos si pueden migrar o no, la utilización MFS, obtener información sobre la carga, equilibrio de la misma, etc.

- **mosrun** [h | OpenMosix ID | list of OpenMosix IDs] command [arguments]: ejecuta un comando sobre un nodo determinado.

11.3. *Metacomputers, grid computing*

Los requerimientos de cómputo necesarios para ciertas aplicaciones son tan grandes que requieren miles de horas para poder ejecutarse en entornos de *clusters*. Tales aplicaciones han promovido la generación de ordenadores virtuales en red, *metacomputers* o *grid computers*. Esta tecnología ha permitido conectar entornos de ejecución, redes de alta velocidad, bases de datos, instrumentos, etc., distribuidos geográficamente. Esto permite obtener una potencia de procesamiento que no sería económicamente posible de otra forma y con excelentes resultados. Ejemplos de su aplicación son experimentos como el *I-WAY networking* (el cual conecta superordenadores de 17 sitios diferentes) en América del Norte, o DataGrid, CrossGrid en Europa o IrisGrid en España. Estos *metacomputers* o *grid computers* tienen mucho en común con los sistemas paralelos y distribuidos (SPD), pero también difieren en aspectos importantes. Si bien están conectados por redes, éstas pueden ser de diferentes características, no se puede asegurar el servicio y están localizadas en dominios diferentes. El modelo de programación y las interfaces deben ser radicalmente diferentes (con respecto a la de los sistemas distribuidos) y adecuadas para el cómputo de altas prestaciones. Al igual que en SPD, las aplicaciones de *metacomputing* requieren una planificación de las comunicaciones para lograr las prestaciones deseadas; pero dada su naturaleza dinámica, son necesarias nuevas herramientas y técnicas. Es decir, mientras que el *metacomputing* puede formarse con la base de los SPD, para éstos es necesario la creación de nuevas herramientas, mecanismos y técnicas. [FK03]

11.3.1. Diferentes arquitecturas de cómputo

Si se tiene en cuenta sólo el aspecto de potencia de cálculo, podemos ver que existen diversas soluciones en función del tamaño y las ca-

ANOTACIONES

racterísticas del problema. En primer lugar, se podría pensar en un superordenador (servidor), pero presentan problemas como falta de escalabilidad, equipos y mantenimiento costoso, cómputo de picos (pasan mucho tiempo desaprovechados) y problemas de fiabilidad. La alternativa económica es un conjunto de ordenadores interconectados por una red de altas prestaciones (Fast Ethernet –LAN– o Myrinet –SAN) lo cual formaría un *cluster* de estaciones dedicado a computación paralela/distribuida (SPD) con un rendimiento muy alto (relación coste/rendimiento 3 a 15 veces). Pero estos sistemas presentan inconvenientes tales como coste elevado de las comunicaciones, mantenimiento, modelo de programación, etc. Sin embargo, es una solución excelente para aplicaciones de grano medio o HTC, *High Time Computing* ('computación de alta productividad'). Otro concepto interesante es el de *intranet computing,* que significa la utilización de los equipos de una red local (por ejemplo, una red de clase C) para ejecutar trabajos secuenciales o paralelos con la ayuda de una herramienta de administración y carga; es decir, es el siguiente paso a un *cluster* y permite la explotación de potencia computacional distribuida en una gran red local con las consiguientes ventajas, al aumentar el aprovechamiento de los recursos (ciclos de CPU a bajo coste), mejora de la escalabilidad y administración no demasiado compleja. Para este tipo de soluciones existe software tal como Sun Grid Engine de Sun Microsystems [Sun03], Condor de la Universidad de Wisconsin (ambos gratuitos) [Uni03] o LSF de Platform Computing (comercial) [Pla03].

La opción del *intranet computing* presenta algunos inconvenientes tales como la imposibilidad de gestionar recursos fuera del dominio de administración. Algunas de las herramientas mencionadas (Condor, LSF o SGE) permiten la colaboración entre diferentes subnodos del sistema, pero todos ellos deben tener la misma estructura administrativa, las mismas políticas de seguridad y la misma filosofía en la gestión de recursos. Si bien presenta un paso adelante en la obtención de cómputo a bajo precio, sólo gestionan la CPU y no los datos compartidos entre los subnodos. Además, los protocolos e interfaces son propietarios y no están basados en ningún estándar abierto, no se pueden amortizar los recursos cuando están desaprovechados ni se puede compartir recurso con otras organizaciones. [Beo03, Ext03, Die02, Die98]

El crecimiento de los ordenadores entre 1986 y 2000 se ha multiplicado por 500 y las redes por 340.000, pero las predicciones auguran que entre los años 2001 y 2010 los computadores se

ANOTACIONES

multiplicarán sólo por 60 y, en cambio las redes, por 4.000. Esto indica la pauta de la siguiente arquitectura para HPC: cómputo distribuido en Internet o *Grid Computing* (GC) o *metacomputing*. Por lo tanto, *grid computing* es una nueva tecnología emergente cuyo objetivo es compartir recursos en Internet de forma uniforme, transparente, segura, eficiente y fiable. Esta tecnología es complementaria a las anteriores, ya que permite interconectar recursos en diferentes dominios de administración respetando sus políticas internas de seguridad y su software de gestión de recursos en la intranet. Según unos de sus precursores, Ian Foster, en su artículo "What is the Grid? A Three Point Checklist" (2002), un *grid* es un sistema que:

1) coordina recursos que no están sujetos a un control centralizado,

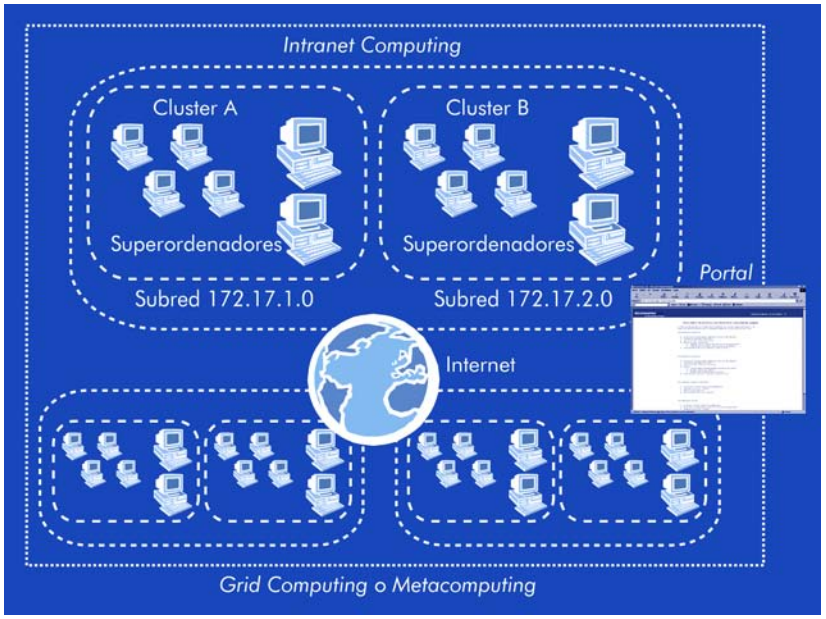
2) utiliza protocolos e interfaces estándares, abiertas y de propósitos generales, y

3) genera calidades de servicio no triviales.

Entre los beneficios que presenta esta nueva tecnología, se pueden destacar el alquiler de recursos, la amortización de recursos propios, gran potencia sin necesidad de invertir en recursos e instalaciones, colaboración/compartición entre instituciones y organizaciones virtuales, etc.

La figura 29 da una visión de todos estos conceptos. [Llo03]

Figura 29.



ANOTACIONES

11.3.2. Globus

El proyecto Globus [Glo03a, Glo03b] es uno de los más representativos en este sentido, ya que es el precursor en el desarrollo de un *toolkit* para el *Metacomputing* o *Grid Computing* y que proporciona avances considerables en el área de la comunicación, información, localización y planificación de recursos, autentificación y acceso a los datos. Es decir, Globus permite compartir recursos localizados en diferentes dominios de administración, con diferentes políticas de seguridad y gestión de recursos y está formado por un paquete software (*middleware*), que incluye un conjunto de bibliotecas, servicios y API.

La herramienta *globus* (*Globus toolkit*) está formada por un conjunto de módulos con interfaces bien definidas para interactuar con otros módulos y/o servicios. La funcionalidad de estos módulos es la siguiente:

- **Localización y asignación de recursos**: permite comunicar a las aplicaciones cuáles son los requerimientos y ubicar los recursos que los satisfagan, ya que una aplicación no puede saber dónde se encuentran los recursos sobre los cuales se ejecutará.

- **Comunicaciones**: provee de los mecanismos básicos de comunicación, que representan un aspecto importante del sistema, ya que deben permitir diversos métodos para que se pueda utilizar eficientemente por las aplicaciones. Entre ellos se incluyen paso de mensajes (*message passing*), llamadas a procedimientos remotos (RPC), memoria compartida distribuida, flujo de datos (*stream-based*) y *multicast*.

- **Servicio de información (***Unified Resource Information Service***)**: provee un mecanismo uniforme para obtener información en tiempo real sobre el estado y la estructura del metasistema donde se están ejecutando las aplicaciones.

- **Interfaz de autentificación**: son los mecanismos básicos de autentificación para validar la identidad de los usuarios y los recursos. El módulo genera la capa superior que luego utilizarán servicios locales para acceder a los datos y los recursos del sistema.

ANOTACIONES

- **Creación y ejecución de procesos**: utilizado para iniciar la ejecución de las tareas que han sido asignadas a los recursos pasándoles los parámetros de ejecución y controlando la misma hasta su finalización.

- **Acceso a datos**: es el responsable de proveer un acceso de alta velocidad a los datos almacenados en archivos. Para DB, utiliza tecnología de acceso distribuido o a través de CORBA y es capaz de obtener prestaciones óptimas cuando accede a sistemas de archivos paralelos o dispositivos de entrada/salida por red, tales como los HPSS (*High Performance Storage System*).

La estructura interna de Globus está sustentada por una base formada por las capas de seguridad, comunicaciones y detección de fallos y por tres pilares básicos tales como *administración de recursos,* descubrimiento y monitorización, y administración de datos. Sobre estas columnas se soportan las aplicaciones de alto nivel y herramientas.

Nivel	Descripción	Paquete *Globus*
superior	Herramientas de alto nivel	Comandos *Globus, MPICHG2, condorG, nimrod/G, gridWay*
Pilar 3	Administración de datos	GASS, GridFTP, GRC y GRM
Pilar 2	Descubrimiento y monitorización	MDS, GIIS y GRIS
Pilar 1	Administración de recursos	GRAM y DUROC
Base 2	Detección de fallos	HBM
Base 1	Comunicación	Globus I/O, GSI
Base O	Seguridad	GSI y CAS

Las siglas reflejan lo siguiente:

- **GSI** (*Globus Security Infrastructure*) implementa *Transport Layer Security* (TLS), *Secure Socket Layer* (SSL), basado en OpenSSL y *Public Key Infrastructure* (PKI) con certificados X.509 y *Generic Security Services* API (GSSAPI).

- **CAS**: *Community Authorization Service* (CAS)

- **Globus I/O**: *Transport Control Protocol* (TCP) *User Datagram Protocol* (UDP) File I/O.

ANOTACIONES

- **HBM**: *Heart Beat Monitor*

- **GRAM** (*Globus Resource Allocation Manager*): incluye como gestores locales: PBS, LSF, NQE, LoadLeveler, *UNIX fork* y el *Resource Specification Language* (RSL).

- **DUROC y MDS**: *Dynamic User Runtime On-line Co-allocator* y *Monitoring and Discovery Service*.

- **GIIS** (*Globus Index Information Service*): incluye *Lightweight Directory Access Protocol* (LDAP), basado en OpenLDAP.

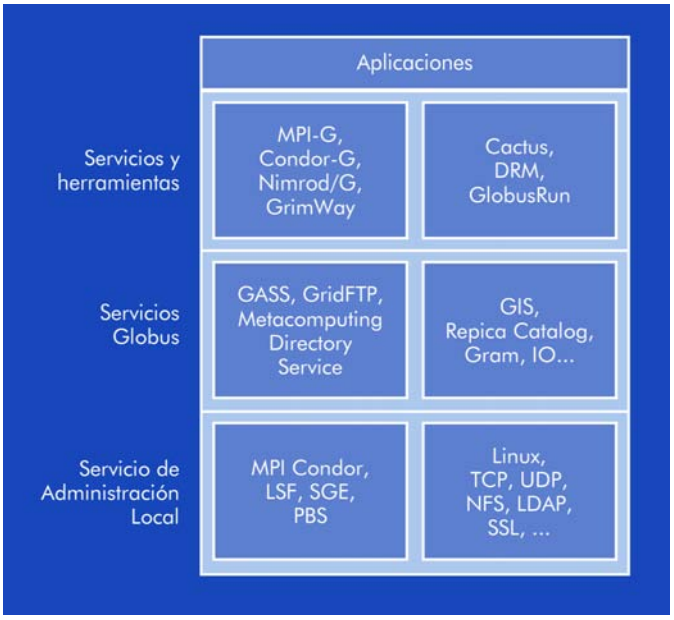
- **GRIS y GASS**: *Globus Resource Information Service y Global Access to Secondary Storage*.

- **GridFTP**: *Grid File Transfer Protocol*.

- **GRC y GRM**: *Globus Replica Catalog* y *Globus Replica Management*.

La figura 30 muestra la estructura de Globus y su interacción con el resto del sistema.

Figura 30.



ANOTACIONES

11.3.3. Software, instalación y administración de Globus

El sitio web de **'The Globus Alliance'** es http://www.globus.org [Glo03a]. Aquí se pueden encontrar tanto el código fuente, como toda la documentación necesaria para transformar nuestra intranet como parte de un *grid*. Formar parte de un *grid* significa ponerse de acuerdo y adoptar las políticas de todas las instituciones y empresas que forman parte de él. En España existen diferentes iniciativas basadas en Globus. Una de ellas es *IrisGrid* [Llo03], a la cual es posible unirse para obtener las ventajas de esta tecnología. Para mayor información, consultar: http://www.rediris.es/irisgrid/.

El primer paso para tener *globus* operativo es obtener el software (en este momento es Globus Toolkit 3.0.2), llamado GT3. Este software implementa los servicios con una combinación entre C y Java (los componentes C sólo se pueden ejecutar en plataformas UNIX GNU/Linux generalmente) y es por ello por lo que el software se divide por los servicios que ofrece. En función del sistema que se quiera instalar se deberán obtener unos paquetes u otros.

Suponiendo que el entorno de Globus todo son máquinas GNU/Linux, se debería seleccionar la opción 'All services', que contiene todos los programas y aplicaciones para UNIX. En 'All services' encontramos dos opciones: *fuentes* que deberán compilarse (GPT *Source Installation Package*) y *ejecutables para Linux* (GPT *Linux Installation Package*), por lo cual se escogerá este último. Este paquete contiene el código binario del GT3, así como Globus Toolkit 2.4.3 Linux binary bundle (incluidos ejemplos) y se deberá ejecutar install-gt3 /directorio/donde/instalar (leer primero los párrafos posteriores). Ténganse en cuenta las consideraciones sobre algunas distribuciones, por ejemplo, Red Hat 9.

A continuación, se darán algunas indicaciones de cómo inicializar y ejecutar algunos ejemplos con GT3 (para mayor información, podéis consultar *User's Guide Core Framework Globus Toolkit 3.0*). La distribución de ejecutables no requiere ninguna herramienta externa para ejecutar los ejemplos (excepto una VM de Java), pero es recomendable bajarse las herramientas que se utilizan en la distribución (*ant*) para construir y probar ejemplos. Siguiendo la nomenclatura de la documentación, se utilizará [*pgsa root*] como el directorio donde se

ANOTACIONES

tiene la distribución y coincide con [*ogsa java root*] en el caso de la distribución de binarios (los ejemplos se hallarán en [*ogsa root*]/*samples*). Las herramientas externas requeridas son:

- J2SE 1.3.1 SDK o JRE (este último si no se necesita compilar ningún archivo fuente, sólo ejecutar).
- Jakarta Ant 1.5 con *optional.jar* (recomendado para los binarios).
- JAAS library (sólo si se instala J2SE 1.3.1).

Para la distribución en binarios de GNU/Linux, se puede instalar como opción el Jakarta Tomcat 4.1.24 (consultar el sitio web http://jakarta.apache.org/tomcat), o la versión 4.0.6; pero no es necesario, ya que GT3 incluye un servicio de web básico para pruebas. La distribución incluye herramientas externas como Apache Axis, Java CoG Kit, Apache Xerces (JAXP 1.2) y Apache-XML-Security-J.

Una vez instaladas las herramientas (J2SE, Jakarta y JAAS) y los binarios (installgt3), se pueden probar los ejemplos que se incluyen. Para ello debéis ejecutar:

- Inicializar el servicio contenedor de *grid* (arrancará en el puerto 8080):

```
globus-start-container -p 8080
```

- Ejecutar el cliente con:

```
globus-service-browser
http://localhost:8080/ogsa/services/core/registry/ ContainerRegistryService?WSDL
```

NOTACIONES

Si se desea construir desde el código fuente o los ejemplos de la distribución de binarios se necesitará el *Jakarta Ant*. Para ello:

1) descargarse Ant 1.5 desde http://jakarta.apache.org/ant.
2) descomprimir en un directorio e inicializar la variable ANT_HOME a este directorio.

3) obtener la biblioteca *junit.jar* desde http://www.junit.org/ e incluirla en ANT_HOME/lib (sólo es necesario si se quieren compilar los ejemplos de la distribución de fuentes).

4) incluir ANT_HOME/bin en la variable *PATH*.

Para ejecutar los ejemplos utilizando *Jakarta Ant*, hacer:

1) Ejecutar el servicio contenedor con:

```
ant startContainer [-Dservice.port =  port].
```

2) Ejecutar la demo cliente con:

```
ant gui [-Dservice.port = port]
```

Por defecto, se ejecuta en el puerto 8080. Para parar el contenedor, se puede ejecutar ant stopContainer.

Consultar *User's Guide Core Framework Globus Toolkit 3.0* para la instalación de Tomcat y los diferentes ejemplos que incluye GT3, así como la configuración de los servicios de alto nivel y las capacidades de *logging-debugging* de la arquitectura. Es recomendable leer también el *GT3 Admin Guide Installation* y el *GT3 Admin Guide Configuration* disponibles en el sitio web de Globus.

11.4. Actividades para el lector

1) Instalar PVM sobre un nodo y ejecutar el programa *master.c* y *cliente.c* dados como ejemplos y observar su comportamiento a través de *xpmv*.

2) Instalar y configurar Mpich sobre un nodo; compilar y ejecutar el programa *cpi.c*.

3) Instalar y configurar LAM-MPI sobre un nodo; compilar y ejecutar el programa *cpi.c*. y observar su comportamiento a través de *xmpi*.

Nota

Otras fuentes de referencias e información:

[Deb03c, LPD03b, Ibi03, Mou01]

ANOTACIONES

Bibliografía

[Aiv02] Tigran Aivazian. "Linux Kernel 2.4 Internals". *The Linux Documentation Project* (guías), 2002.

[Ano99] Anónimo. *Maximum Linux Security: A Hacker's Guide to Protecting Your Linux Server and Network*. Sams, 1999.

[AR01] Jonathan Corbet Alessandro Rubini. *Linux Device Drivers 2nd Editon.* O'Reilly, 2001.

[Ara03] Arachne. "Interfaz Gcvs para CVS", 2003.
http://www.arachne.org/software/gcvs

[Arc03] Roberto Arcomano. "KernelAnalysis-HOWTO". *The Linux Documentation Project, 2003.*

[Aus03] CERT Australia. "Australian CERT", 2003.
http://www.auscert.org.au/

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

[Bai03] Edward C. Bailey. *RedHat Maximum RPM*, 2003.
http://www.redhat.com/docs/books/max-rpm/index.html

[Ban01] Tobby Banerjee. "Linux Installation Strategies HOWTO". *The Linux Documentation Project*, 2001.

[Bar03] Barrapunto. *barrapunto site*, 2003.
http://barrapunto.com

[Beo03] Beowulf.org. *Beowulf Web Site*, 2003.
http://www.beowulf.org

[Bor00] Matthew Borowski. "FTP". *The Linux Documentation Project*, 2000.

[Bro01] Scott Bronson. "VPN PPP-SSH". *The Linux Documentation Project*, 2001.

ANOTACIONES

[Bul03a] Bulma. "Bulma Linux User Group", 2003.
http://bulmalug.net

[Bul03b] Bulma. "Lista revistas Linux", 2003.
http://bulmalug.net/body.phtml?nIdNoticia = 1435

[Bur02] Hal Burgiss. "Security QuickStart HOWTO for Linux". *The Linux Documentation Project*, 2002.

[Ced97] Cederqvist. "Version Management with CVS", 1997.
http://www.cvshome.org

[CER03a] CERT. "CERT site", 2003.
http://www.cert.org

[CER03b] CERT. "CERT vulnerabilidades", 2003.
http://www.cert.org/nav/index red.html

[Cer03c] Cervisia. "Interfaz Cervisia para CVS", 2003.
http://cervisia.sourceforge.net

[Cis00] Cisco. "TCP/IP White Paper", 2000.
http://www.cisco.com

[Com01] Douglas Comer. *TCP/IP Principios básicos, protocolos y arquitectura*. Prentice Hall, 2001.

[Coo03] Mendel Cooper. "Advanced bashScripting Guide". *The Linux Documentation Project* (guías), 2003.

[CVS03] CVShome.org. "CVS Home", 2003.
http://www.cvshome.org

[DB03] Marco Cesati Daniel Bovet. *Understanding the Linux Kernel 2nd edition*. O'Reilly, 2003.

[Deb] Debian. "Sitio Seguridad de Debian".
http://www.debian.org/security/

[Deb02] Debian. "APT-HOWTO", 2002.
http://www.debian.org/doc/manuals/apt-howto/index.en.html

[Deb03a] Debian. "Software Libre vs Software Abierto", 2003.
http://www.debian.org/intro/free.es.html

ANOTACIONES

[Deb03b] Comunidad Debian. "Distribución Debian", 2003.
http://www.debian.org

[Deb03c] Debian.org. "Debian Home", 2003.
http://www.debian.org

[Die98] Hank Dietz. "Linux Parallel Processing". *The Linux Documentation Project*, 1998.

[Die02] Hank Dietz. "Linux Parallel Processing" , 2002.
http://yara.ecn.purdue.edu/pplinux/PPHOWTO/pphowto.html

[Dis03] Distrowatch. "Distribuciones Linux disponibles", 2003.
http://www.distrowatch.com

[Dra99] Joshua Drake. "Linux Networking". *The Linux Documentation Project*, 1999.

[ea02] Kris Buytaert y otros. "The OpenMosix". *The Linux Documentation Project*, 2002.

[Ext03] ExtremeLinux.org. "Extreme Linux Web Site", 2003.
http://www.extremelinux.org

[FBI03] FBI. "Brigada del FBI para cibercrimen", 2003
http://www.emergency.com/fbinccs.htm

[Fen02] Kevin Fenzi. "Linux security HOWTO". *The Linux Documentation Project*, 2002.

[FK03] Ian Foster; Carl Kesselmany. "Globus: A Metacomputing Infrastructure Toolkit", 2003
http:/www.globus.org

[Fre03] Freshmeat. "Freshmeat site", 2003
http://freshmeat.org

[Fri02] AEleen Frisch. *Essential System Administration*. O'Reilly, 2002.

[FSF03] FSF. "Free Software Foundation y Proyecto GNU", 2003
http://www.gnu.org

[G00] Mike G. "BASH Programming - Introduction HOWTO". *The Linux Documentation Project*, 2000.

ANOTACIONES

[Gar98] Bdale Garbee. *TCP/IP Tutorial.* N3EUA Inc., 1998.

[Glo03a] Globus. GT3 "Admin Guide Installation" y "Admin Guide Configuration", 2003
http://www.globus.org

[Glo03b] Globus. "User's Guide Core Framework Globus Toolkit 3.0", 2003
http://www.globus.org

[Gnu03] Gnupg.org. *GnuPG Web Site*, 2003.
http://www.gnupg.org/

[Gon00] Guido Gonzato. "From DOS/Windows to Linux HOWTO". *The Linux Documentation Project*, 2000.

[Gor03] Paul Gortmaker. "The Linux BootPrompt HOWTO". *The Linux Documentation Project*, 2003.

[Gre00] Mark Grennan. "Firewall and Proxy Server HOWTO". *The Linux Documentation Project*, 2000.

[GT03] Dirk Allaert Grant Taylor. "The Linux Printing HOWTO". *The Linux Documentation Project*, 2003.

[Hat01] Brian Hatch. *Hacking Linux Exposed*. McGraw-Hill, 2001.

[Hat03a] Red Hat. "Configuring Firewall", 2003.
http://www.redhat.com/support/resources/networking/firewall.html

[Hat03b] Red Hat. "Red Hat 9 Security Guide", 2003.
http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/security-guide/

[Hat03c] Red Hat. "Sitio Seguridad de Red Hat", 2003.
http://www.redhat.com/solutions/security/

[Hat03d] Red Hat. *Utilización firmas GPG en Red Hat*, 2003.
http://www.redhat.com/solutions/security/news/publickey.html

[Hen03] Bryan Henderson. "Linux Loadable Kernel Module HOWTO". *The Linux Documentation Project*, 2003.

[Him01] Pekka Himanen. *La ética del hacker y el espíritu de la era de la información*. Destino, 2001.

ANOTACIONES

[Hin00] Martin Hinner. "Filesystems HOWTO". *The Linux Documentation Project*, 2000.

[His03] HispaLinux. "Comunidad Hispana de Linux", 2003.
http://www.hispalinux.org

[Ian03] Iana. "Lista de puertos TCP/IP", 2003.
http://www.iana.org/assignments/port-numbers

[Ibi03] Ibiblio.org. "Linux Documentation Center", 2003.
http://www.ibiblio.org/pub/Linux/docs/HOWTO/

[IET03] IETF. "Repositorio de Request For Comment desarrollados por Internet Engineering Task Force (IETF) en el Network Information Center (NIC)", 2003.
http://www.cis.ohio-state.edu/rfc/

[Inc03a] Red Hat Inc. "Distribución Red Hat", 2003.
http://www.redhat.com

[Inc03b] Incidents.org. "vulnerabilidades Incidents", 2003.
http://isc.incidents.org

[Ins98] Insecure.org. "Vulnerabilidades y exploits", 1998.
http://www.insecure.org/sploits.html

[Ins03a] Insecure.org. "Insecure.org site", 2003.
http://www.insecure.org

[Ins03b] Insecure.org. "Nmap", 2003.
http://www.insecure.org/nmap/index.html

[Joh98] Michael K. Johnson. "Linux Information Sheet". *The Linux Documentation Project*, 1998.

[Jou03] Linux Journal. *Linux Journal [Revista Linux]* , 2003.
http://www.linuxjournal.com

[Kan01] Ivan Kanis. "Multiboot with GRUB Mini-HOWTO". *The Linux Documentation Project*, 2001.

[Kat01] Jonathan Katz. "Linux + Windows HOWTO". *The Linux Documentation Project*, 2001.

ANOTACIONES

[KD00] Olaf Kirch; Terry Dawson. *Linux Network Administrator's Guide*. O'Reilly Associates. Y comoe-*book* (free) en Free Software Foundation, Inc., 2000: http://www.tldp.org

[Ker02] Kernelhacking.org. "Kernel Hacking Doc Project", 2002. http://www.kernelhacking.org

[Ker03a] Kernelnewbies.org. "Kernel Newbies", 2003. http://www.kernelnewbies.org

[Ker03b] Kernel.org. "Linux Kernel Archives", 2003. http://www.kernel.org

[Kie97] Robert Kiesling. "The RCS (Revision Control System)". *The Linux Documentation Project*, 1997.

[Koe01] Kristian Koehntopp. "Linux Partition HOWTO". *The Linux Documentation Project*, 2001.

[Kuk03] Thorsten Kukuk. "The Linux NIS(YP)/NYS/NIS+". *The Linux Documentation Project*, 2003.

[Lam03] LamMPI.org. "LAM (Local Area Multicomputer)", 2003. http://www.lam-mpi.org

[Law03] David Lawyer. "Linux Módem". *The Linux Documentation Project*, 2003.

[Lev02] Bozidar Levi. *UNIX administration*. CRC Press, 2002.

[Lev03] Eric Levenez. "UNIX History", 2003. http://www.levenez.com/UNIX

[Lin03a] Gazeta Linux. *Gazeta, Revista Linux Online*, 2003. http://www.gazetalinux.com

[Lin03b] Linuxbase.org. *FHS Standard*, 2003. http://www.pathname.com/fhs

[Lin03c] Linuxbase.org. *Linux Standards Base project*, 2003. http://www.linuxbase.org

[Lin03d] LinuxISO. *Images CD de distribuciones Linux*, 2003. http://www.linuxiso.com

ANOTACIONES

[Lin03e] Linuxsecurity.com. *Linux Security Reference Card*, 2003.
http://www.linuxsecurity.com/docs/QuickRefCard.pdf

[lkm03] lkml. *Linux Kernel Mailing List*, 2003.
http://www.tux.org/lkml

[Llo03] Ignacio Martín Llorente. *Estado de la Tecnología Grid y la Iniciativa IrisGrid*, 2003.
http://www.rediris.es/irisgrid

[LN01] Nicolai Langfeldt; Jamie Norrish. "DNS". *The Linux Documentation Project*, 2001.

[Log03] Logcheck. "Logckeck Web Site" , 2003.
http://www.psionic.com/abacus/logcheck/

[LPD03a] LPD. *The Linux Documentation Project*, 2003.
http://www.tldp.org

[LPD03b] LPD. *The Linux Documentation Project*, 2003.
http://www.tldp.org

[LPD03c] LPD. *Linux Documentation Project en español*, 2003.
http://www.es.tldp.org

[Luc03] Lucas. "Comunidad Linux de México", 2003.
http://lucas.linux.org.mx

[Mag03] Linux Magazine. *Linux Magazine [Revista Linux]*, 2003.
http://www.linuxmagazine.com

[Maj96] Amir Majidimehr. *Optimizing UNIX for Performance*. Prentice Hall, 1996.

[Mal96] Fred Mallett. *TCP/IP Tutorial*. FAME Computer Education, 1996.

[Mal03] Luiz Ernesto Pinheiro Malère. "Ldap". *The Linux Documentation Project*, 2003.

[Miq01] Miquel, S. "NIS Debian". *Sobre Debian Woody*, /usr/doc/nis/nis.debian.howto, 2001.

[Mor03] Daniel Morill. *Configuración de sistemas Linux*. Anaya Multimedia, 2003.

ANOTACIONES

[Mou01] Gerhard Mourani. *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc., 2001.

[Mou02] Gerhard Mourani. "Securing and Optimizing Linux: The Ultimate Solution". *The Linux Documentation Project* (guías), 2002.

[Mur02] Gary Lawrence Murphy. *Kernel Book Project*, 2002.
http://kernelbook.sourceforge.net

[Mys03] Mysql. "Reference Manual Version 4.1.1-alpha", 2003.
http://www.mysql.com/

[Nes03] Nessus.org. "Nessus", 2003.
http://www.nessus.org

[Net03] Netfilter.org. *Proyecto Netfilter/IPtables*, 2003.
www.netfilter.org

[Neu00] Christopher Neufeld. "Setting Up Your New Domain Mini-HOWTO". *The Linux Documentation Project*, 2000.

[New03] Newsforge. "Newsforge site", 2003.
http://newsforge.org

[NSA03a] NSA. "NIST site (NSA)", 2003.
http://csrc.nist.gov/

[NSA03b] NSA. "Security Enhanced Linux", 2003.
http://www.nsa.gov/selinux/index.html

[O'K00] Greg O'Keefe. "From Power Up To bash Prompt HOWTO". *The Linux Documentation Project*, 2000.

[OSD03a] OSDL. "Open Source Developement Laboratories", 2003.
http://www.osdl.org

[OSD03b]OSDN. "Open Source Development Network", 2003.
http://osdn.com

[OSI03a] OSI. "Listado de licencias Open Source", 2003.
http://www.opensource.org/licenses/index.html

[OSI03b] OSI. "Open Source Definition", 2003.
http://www.opensource.org/docs/definition.php

ANOTACIONES

[OSI03c] OSI. "Open Source Iniciative", 2003.
http://www.opensource.org

[Peñ03] Javier Fernández-Sanguino Peña. "Securing Debian Manual", 2003.

http://www.*Debian*.org/doc/manuals/*securingDebianhowto*/index.en.html

[Pla03] Plataform. "LSF", 2003. http://www.platform.com

[Pos03a] PostgreSQL.org. "PostgreSQL Administrator's Guide 7.3", 2003.
http://www.postgresql.org/docs/7.3/

[Pos03b] PostgreSQL.org. "PostgreSQL Programmer's Guide 7.3", 2003.
http://www.postgresql.org/docs/7.3/

[Pos03c] PostgreSQL.org. "PostgreSQL Reference Manual 7.3", 2003.
http://www.postgresql.org/docs/7.3/

[Pos03d] PostgreSQL.org. "PostgreSQL User's Guide 7.3", 2003.
http://www.postgresql.org/docs/7.3/

[Pos03e] PostgreSQL.org. "PostgreSQL Web Site", 2003.
http://www.postgresql.org

[PPP00] Linux PPP. "Corwin Williams, Joshua Drake and Robert Hart". *The Linux Documentation Project*, 2000.

[Pra03] Joseh Pranevich. "The Wonderful World of Linux 2.6", 2003.
http://www.kniggit.net/wwol26.html

[Pri02] Steven Pritchard. "Linux Hardware HOWTO". *The Linux Documentation Project*, 2002.

[Pro02] GNU Project. "Grub Manual", 2002.
http://www.gnu.org/manual/grub/index.html

[Pro03a] Bastille Project. "Bastille", 2003.
http://bastille-linux.sourceforge.net/

[Pro03b] Mpich Project. "MPI", 2003.
http://www.mcs.anl.gov:80/mpi/

ANOTACIONES

[Pro03c] Mpich Project. "Mpich MPI Freeware", 2003.
http://www.mcs.anl.gov/mpi/*mpich*

[Pro03d] OpenMosix Project. "OpenMosix", 2003.
http://openMosix.sourceforge.net

[Pro03e] PVM Project. "PVM Web Site", 2003.
http://www.epm.ornl.gov/pvm/pvm home.html

[PS02] Ricardo Enríquez Pio Sierra. *Open Source*. Anaya Multimedia, 2002.

[Qui01] Ellie Quigley. *Linux shells by Example*. Prentice Hall, 2001.

[Rad03] Jacek Radajewski. "Beowulf Links", 2003.
http://www.sci.usq.edu.au/staff/jacek/beowulf

[Ran03] David Ranch. "Linux IP Másquerade". *The Linux Documentation Project*, 2003.

[Ray97] Eric Raymond. "La catedral y el bazar", 1997.
http://es.tldp.org/Otros/catedralbazar/cathedral-es-paper00.html

[Ray02a] Eric Raymond. "UNIX and Internet Fundamentals". *The Linux Documentation Project*, 2002.

[Ray02b] Eric Steven Raymond. "The Linux Installation HOWTO". *The Linux Documentation Project*, 2002.

[RE98] Jacek Radajewski; Douglas Eadline. "Beowulf". *The Linux Documentation Project*, 1998.

[RE02] Jacek Radajewski; Douglas Eadline. "Beowulf: Installation and Administration".
http://www.sci.usq.edu.au/staff/jacek/beowulf, 2002.

[Rid00] Daniel Lopez Ridruejo. "The Linux Networking Overview". *The Linux Documentation Project*, 2000.

[Rus00] Rusty Russell. "Linux IPCHAINS". *The Linux Documentation Project*, 2000.

ANOTACIONES

[S+02] Michael Schwartz y otros. *Multitool Linux - Practical Uses for Open Source Software*. Addison Wesley, 2002.

[Sal94] Peter H. Salus. "25 aniversario de UNIX". *Byte España* (n.º 1, noviembre), 1994.

[San03] Sans. "Top20 de vulnerabilidades", 2003.
http://www.sans.org/top20/

[Sec00] Andrés Seco. "Diald". *The Linux Documentation Project*, 2000.

[Sei02] Kurt Seifried. "Securing Linux, Step by Step", 2002.

http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html

[Sko03a] Miroslav Skoric. "LILO mini-HOWTO". *The Linux Documentation Project*, 2003.

[Sko03b] Miroslav Skoric. "Linux+WindowsNT mini-HOWTO". *The Linux Documentation Project*, 2003.

[Sla03] Slashdot. "Slashdot site", 2003.
http://slashdot.org

[Smi02] Rod Smith. ***Advanced Linux Networking***. Addison Wesley, 2002.

[Sno03] Snort.org. *Snort*, 2003.
http://www.snort.org

[Sou03] Sourceforge. "Sourceforge site", 2003.
http://sourceforge.org

[Sta02] Richard Stallman. "Discusión por Richard Stallman sobre la relación de GNU y Linux", 2002.
http://www.gnu.org/gnu/linux-and-gnu.html

[Stu01] Michael Stutz. "The Linux Cookbook: Tips and Techniques for Everyday Use". *The Linux Documentation Project* (guías), 2001.

[Sun02] Rahul Sundaram. "The dosemu HOWTO". *The Linux Documentation Project*, 2002.

ANOTACIONES

[Sun03] Sun. "Sun Grid Engine". http://www.sun.com/gridware, 2003.

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

[Tkc03] Tkcvs. "Interfaz Tkcvs para CVS", 2003. http://www.tkcvs.org

[Tri03] Tripwire.com. *Tripwire Web Site*, 2003. http://www.tripwire.com/

[Tum02] Enkh Tumenbayar. "Linux SMP HOWTO". *The Linux Documentation Project*, 2002.

[Uni03] Wisconsin University. *Condor Web Site.* http://www.cs.wisc.edu/condor, 2003.

[USA03] Dep Justicia USA. "Division del Departamento de justicia de USA para el cibercrimen", 2003. http://www.usdoj.gov/criminal/cybercrime/

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

[Vas00] Alavoor Vasudevan. "Modem-Dialup-NT". *The Linux Documentation Project*, 2000.

[Vas03a] Alavoor Vasudevan. "CVS-RCS (Source Code Control System)". *The Linux Documentation Project*, 2003.

[Vas03b] Alavoor Vasudevan. "The Linux Kernel HOWTO". *The Linux Documentation Project*, 2003.

[W+02] Matt Welsh y otros. *Running Linux 4th edition*. O'Reilly, 2002.

[War03] Ian Ward. "Debian and Windows Shared Printing mini-HOWTO". *The Linux Documentation Project*, 2003.

[Wil02] Matthew D. Wilson. "VPN". *The Linux Documentation Project*, 2002.

[Woo00] David Wood. "SMB HOWTO". *The Linux Documentation Project*, 2000.

ANOTACIONES

[Xin03] **Xinetd.org.** "Xinetd Web Site", 2003.
http://www.xinetd.org/

[Zan01] **Renzo Zanelli.** *Win95 + WinNT + Linux multiboot using LILO mini-HOWTO. The Linux Documentation Project*, 2001.

ANOTACIONES

GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

ANOTACIONES

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

ANOTACIONES

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent.

An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

ANOTACIONES

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

ANOTACIONES

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

ANOTACIONES

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

ANOTACIONES

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

ANOTACIONES

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

ANOTACIONES

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the

ANOTACIONES

translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

ANOTACIONES

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

ANOTACIONES


UOC
La universidad virtual
Formación de Posgrado