ADMINISTRACIÓN DE SISTEMAS UNIX

Antonio Villalón Huerta <toni@aiind.upv.es>
Sergio Bayarri Gausi <sergio@aiind.upv.es>

Mayo, 2005

Índice General

Capítulo 1

Introducción: el superusuario

1.1 Conceptos básicos para administradores

El administrador de una máquina Unix se conoce habitualmente como `root` (éste es su nombre de entrada al sistema o *login*), y la principal característica distintiva respecto al resto de usuarios de una máquina es su **UID**, igual a **0** (el GID también es 0 en ciertos Unices). Este hecho le va a permitir evitar todos los controles del sistema operativo en cuanto a permisos de ficheros, acceso al *hardware*, protección de memoria, etc. Por tanto, el `root` o superusuario es (o ha de ser) la única entidad del sistema con acceso total a los recursos del entorno. Aunque en principio cualquier usuario con UID 0 en el archivo de contraseñas tendrá privilegios de administrador, en la mayoría de los casos es conveniente por seguridad y estabilidad de la máquina que este tipo de usuario sea único y con *login* `'root'`.

El `root` ha de suministrar al resto de usuarios un **entorno fiable, seguro y estable** en el cual puedan desarrollar sus trabajos de forma efectiva. Aunque a muchas máquinas Unix se les asigna un administrador elegido entre todos los usuarios, esto no suele ser una buena opción a la larga: el superusuario ha de ser una persona técnicamente preparada para desarrollar todas las actividades propias de esta tarea (que veremos más adelante), y por supuesto ha de tener unos conocimientos del sistema muy superiores al del resto de usuarios. Aparte de poseer conocimientos técnicos, ha de tener una gran responsabilidad, ya que su papel suele ser también normativo: tiene que definir unas reglas de convivencia y aprovechamiento del sistema para regular su uso.

Suele ser una regla habitual – y recomendable – entre administradores el crear para sí mismos un usuario común, sin ningún privilegio especial, para trabajar de forma normal en el sistema. Así, las entradas a la máquina como superusuario se minimizan, y sólo se realizan para efectuar tareas privilegiadas, como copias de seguridad, actualización de software o creación de nuevos usuarios. Es importante recordar que sólo debemos utilizar un acceso privilegiado para las tareas que lo requieran, no para el resto: no hay más que pensar en que, si tenemos un pequeño error como usuario `'root'`, podemos destruir por completo el sistema.

1.2 El entorno del superusuario

El *shell* o intérprete de órdenes del sistema Unix proporciona para el administrador un valor especial de la variable de entorno *$PS1*, '#', en lugar de los habituales '**$**', '%' o '>', proporcionados al resto de usuarios. Esto no da ningún privilegio especial al administrador (recordemos que todos sus privilegios provienen del UID 0), y cualquier usuario puede modificar el valor de su *$PS1* para igualarlo a '#'; sin embargo es útil para recordar a la persona encargada del sistema que está trabajando sin ningún tipo de restricción, por lo que un error en sus acciones puede ser fatal para la máquina.

Otras variables de entorno que suelen diferir del resto de usuarios son el directorio *$HOME*, que para el `root` es `/` o `/root/` generalmente, y también el *$PATH*, que incluirá ciertos directorios como `/sbin/` o `/usr/sbin/`, donde se encuentran mandatos para la administración del sistema que no son necesarios para un usuario sin privilegios.

En la práctica totalidad de sistemas Unix, las conexiones a la máquina como `root` mediante *telnet* u otro tipo de acceso remoto están limitadas a una serie de terminales denominadas 'seguras', terminales que generalmente son la propia consola del equipo y sus terminales virtuales (es decir, restringimos las conexiones `root` al monitor físico del ordenador); con esto se evita que el `root` pueda conectar desde cualquier lugar de la red, y su *password* sea capturado por un potencial intruso. La forma de definir qué terminales son las autorizadas para el administrador varía entre diferentes Unices; por ejemplo, en SunOS se definen en el archivo `/etc/ttytab`, en Solaris e IRIX en `/etc/default/login`, en OSF/1 en `/etc/securettys`, etc. En el caso de HP-UX y Linux, la lista de terminales seguras se encuentra en el fichero `/etc/securetty`; como hemos dicho, es conveniente que sólo se permita una conexión `root` desde el monitor físico del ordenador, para evitar así el movimiento de una contraseña de superusuario por la red. Sin embargo, también se permite la administración remota: en casos de urgencia, es posible para el `root` entrar al sistema como un usuario normal, y obtener privilegios de superusuario mediante la orden `su`:

```
rosita:~$ id
uid=1000(toni) gid=100(users) groups=100(users)
rosita:~$ su
Password:
rosita:/home/toni# id
uid=0(root) gid=0(root) groups=0(root)
rosita:/home/toni# exit
exit
rosita:~$ id
uid=1000(toni) gid=100(users) groups=100(users)
rosita:~$
```

Acabamos de comentar que las conexiones como *root* se van a producir sobre todo desde la consola del sistema. Por defecto, algunos Unices como Linux, FreeBSD o SCO[1] nos van a ofrecer en modo texto varias terminales (por norma general 6, de *tty1* a *tty6*); podremos conmutar entre ellas pulsando *Alt-F[1..6]*. Si pulsamos *Alt-F7* llegamos a la salida de los sistemas de ventanas (para conmutar entre estos y el modo texto). Este conjunto de terminales virtuales en la consola del sistema es muy útil para máquinas sin entorno gráfico, en los que si sólo tuviéramos una terminal y esta se inutilizara por cualquier motivo, no podríamos trabajar en modo local. Aunque con seis terminales suele ser más que suficiente para la correcta administración del sistema, si en algún momento necesitamos más podemos conseguirlas mediante órdenes como `agetty`, `mingetty` o `ttymon` (dependiendo del Unix en que trabajemos), con las opciones adecuadas.

1.3 El papel del administrador

Las tareas de un administrador dependen habitualmente del tipo de sistema del cual se ocupe, y por tanto vienen marcadas por la organización de la máquina y de su entorno (empresa, usuarios, etc.). Aunque varían entre sistemas, ya que obviamente no es lo mismo administrar una máquina con mil usuarios que una con cincuenta, o proteger un sistema dedicado a las prácticas de alumnos que otro dedicado a trabajos militares, existen ciertas tareas comunes a todos los administradores, ya lo sean de un sencillo PC con un Unix gratuito o de un servidor de gama alta corriendo Solaris. Entre las más frecuentes podemos destacar:

[1] Solaris no las ofrece por defecto, pero pueden configurarse sin problemas en cualquier versión anterior a la 8.

- **Arranque y parada del sistema.** Se ha de realizar de una forma ordenada para evitar inconsistencias en el *kernel* o en el sistema de ficheros.
- **Gestión de los usuarios.** Añadir usuarios, eliminarlos, modificar sus privilegios, controlar sus actividades... e incluso aconsejarles en el manejo del sistema para que puedan trabajar de forma cómoda y eficiente.
- **Gestión de software.** El administrador es el encargado de actualizar o retirar programas del sistema, y por supuesto también de controlar su correcto funcionamiento (por ejemplo mediante parches de seguridad).
- **Gestión de periféricos.** Puesta a punto de terminales, impresoras, módems...
- **Gestión de sistemas de ficheros.** Montaje o desmontaje de unidades de disco (discos duros, disquetes, unidades de CDROM...), unidades de cinta, sistemas de ficheros remotos... y control de su uso por parte de los usuarios. El administrador también es responsable de la creación y actualización de las copias de seguridad del sistema.
- **Gestión del sistema de red.** Como administradores, hemos de configurar los diferentes dispositivos de red de nuestra máquina, así como el rutado correcto de los paquetes, los servicios ofrecidos, los servidores de nombres...
- **Seguridad de la máquina.** Sobre el `root` recae la mayor responsabilidad en cuanto a integridad, fiabilidad y privacidad de la máquina y la información contenida en ella. Aparte de posibles ataques externos, también se ha de controlar que los usuarios no realicen actividades a las que no están autorizados, creando unas normas que todos han de cumplir en el sistema.

Todas estas tareas van encaminadas a conseguir el **perfecto funcionamiento del sistema** durante el máximo tiempo posible, y también a poder **aprovechar todos los recursos y servicios** que un sistema Unix correctamente administrado puede ofrecer; en el presente curso vamos a ver, de forma introductoria, algunas de las tareas citadas anteriormente.

1.4 Comunicación con los usuarios.

No siempre es posible para un administrador poder hablar en persona con todos y cada uno de sus usuarios, ya sea por falta de tiempo o por la imposibilidad física de hablar con una persona que potencialmente puede estar trabajando en nuestro sistema desde cualquier punto del mundo. Por tanto, el sistema operativo Unix nos ofrece, como administradores, una serie de herramientas que nos van a permitir comunicarnos con las personas que trabajan en la máquina.

Aparte de los ya conocidos `write` o `talk` (que obviamente podemos utilizar como administradores, aunque no es lo común, ya que frecuentemente nos interesa contactar con un grupo más o menos amplio de usuarios, y no con uno sólo), disponemos de otros mecanismos de comunicación, algunos de ellos permitidos únicamente al `root` de la máquina, y que van a hacer llegar un cierto mensaje a un grupo determinado de usuarios.

La primera forma de comunicación con los usuarios que vamos a estudiar es el archivo `/etc/motd` (*Message Of The Day).* Es un archivo común de texto, que como administradores podemos editar y modificar a nuestra voluntad, y en el que se suele incluir un mensaje dirigido a todos los **usuarios del sistema que conectan habitualmente**. Cuando un usuario inicie sesión en la máquina, una vez se haya identificado con su *login* y *password*, le aparecerá en pantalla un volcado de este fichero.

Los mensajes más comunes que suelen aparecer en `/etc/motd` son aquellos que potencialmente interesan a todos los usuarios, o al menos a los que conectan al sistema de forma regular, y son del tipo

"El día 15, de 12:00 a 16:00, la máquina permanecerá cerrada por actualización de software. Disculpen las molestias."

"Se recuerda a los usuarios del grupo usr1 que no pueden sobrepasar los 3 MB. de disco utilizado por cada cuenta. Se borrarán los directorios que sobrepasen los 3'5 MB."

Teniendo definido un fichero `/etc/motd` en el sistema, cuando un usuario conecte, verá algo parecido a lo siguiente:

```
rosita login: toni
Password:
Last login: Fri Oct  3 18:36:37 from localhost

El dia 15, de 12:00 a 16:00, la maquina permanecera cerrada por
actualizacion de software. Disculpen las molestias.

rosita:~$
```

No debemos confundir el fichero `/etc/motd` con archivos como `/etc/issue` o `/etc/issue.net`; estos archivos se muestran **antes** de conectar al sistema, es decir, antes de que la máquina nos pregunte un nombre de usuario y una contraseña, por lo que no es recomendable poner en ellos mensajes para los usuarios. El contenido de estos ficheros es también un texto plano (podemos editar los archivos igual que `/etc/motd`) que se mostrará cuando alguien haga un *telnet* a nuestro sistema:

```
rosita:~# cat /etc/issue.net

                Bienvenidos a ROSITA

rosita:~# telnet rosita
Trying 192.168.0.1...
Connected to rosita.
Escape character is '^]'.

                Bienvenidos a ROSITA


rosita login:
```

El segundo mecanismo que tiene el administrador para comunicarse con sus usuarios es la orden `wall` *(Write ALL)*, que trabaja en tiempo real. Utilizando un mecanismo similar al de `write`, interrumpe la pantalla de todos los usuarios conectados y les envía un mensaje especificado en un fichero que `wall` recibe como argumento. Si este fichero no se especifica, `wall` recibe datos del teclado y los envía al pulsar Ctrl-D. Un ejemplo sería:

```
rosita:~# cat aviso.1
Vamos a apagar la maquina dentro de media hora.
rosita:~# wall aviso.1

Broadcast Message from root@rosita
        (/dev/tty1) at 04:35 ...

Vamos a apagar la maquina dentro de media hora.

rosita:~#
```

Equivalente a:

```
rosita:~# wall
Vamos a apagar la maquina dentro de media hora.
<ctrl-d>

Broadcast Message from root@rosita
        (/dev/tty1) at 04:35 ...

Vamos a apagar la maquina dentro de media hora.

rosita:~#
```

Como vemos, el mecanismo de `wall` interrumpe la pantalla de todos los usuarios conectados, incluido el propio `root`. Por tanto, los mensajes enviados con este mecanismo son aquellos que pueden interesar a los usuarios **conectados en ese momento** al sistema, pero no al resto: por ejemplo, a un usuario que no esté conectado, seguramente le importará poco que reiniciemos el sistema en un momento dado, y en caso de que le importe (puede tener procesos activos en la máquina) poco podrá hacer.

De los métodos que el sistema ofrece al administrador para comunicarse con sus usuarios, el tercero que queremos comentar es el envío de correo a varios usuarios al mismo tiempo. Obviamente esto lo utilizaremos para difundir un mensaje interesante para ciertos usuarios (por ejemplo, los pertenecientes a un grupo), que no necesariamente han de estar conectados al sistema en el momento en que enviamos el correo.

Para enviar correo a varios usuarios, en cualquier gestor (como `elm, mailx, mail o pine`) debemos separar los nombres de los usuarios con comas (o espacios, si invocamos desde el *prompt*), en el campo reservado al destinatario de un mensaje. Utilizando `mail`, por ejemplo, lo haríamos de la siguiente manera:

```
rosita:~# mail toni sergio monica
Subject: Cita para renovacion
El proximo lunes pasad por el despacho de 1400 a 1600 para renovar vuestros
datos en la maquina.
Saludos
.
EOT
rosita:~#
```

Así, haríamos llegar el mismo mensaje a los usuarios `toni, monica` y `sergio` de nuestro sistema, sin necesidad de escribir un correo para cada uno de ellos.

Capítulo 2

Arranque y parada de máquina

2.1 Arranque del sistema

Cuando encendemos el interruptor de la máquina, y una vez que el *firmware* cede el control al sistema operativo (algo dependiente de la arquitectura de nuestra máquina), se lanza en el equipo un proceso llamado **boot**. Este proceso realiza unas tareas de muy bajo nivel que escapan al contenido de este curso, como cargar el *kernel* del sistema operativo o inicializar estructuras internas de Unix. Antes de morir va a realizar una tarea muy importante: crear el proceso **INIT**.

INIT va a ser el primer proceso que para nosotros se va a ejecutar en el sistema (tendrá siempre el PID 1), y de él cuelgan todos los demás. Su misión va a ser situar a Unix en un modo determinado de operación (denominado *runlevel*) de los diferentes que nos ofrece el sistema operativo. Un *runlevel* no es más que una determinada **configuración software que permite la ejecución de una serie de procesos en la máquina**; generalmente, y por defecto, va a ser el modo multiusuario, que en algunos sistemas Unix corresponde al *runlevel* 3 (por ejemplo, en Linux Slackware) y en otros corresponde al *runlevel* 2 (por ejemplo, en Linux Debian o Solaris). Podemos determinar el *runlevel* en que se encuentra el sistema mediante la orden `runlevel` (Linux) o `who -r` (Solaris, AIX, HP-UX, Tru64...):

```
luisa:~# uname -a
Linux luisa 2.2.18 #7 Thu May 30 02:11:33 CET 2002 i586 unknown
luisa:~# runlevel
N 3
luisa:~#
```

Una vez INIT ha situado al sistema en un determinado *runlevel*, va a permanecer activo en la tabla de procesos hasta que se realice el apagado de la máquina, asegurándose del correcto funcionamiento del sistema, por ejemplo adoptando procesos *zombies* huérfanos y eliminándolos; si INIT muere, el sistema se desmorona.

Durante el proceso de arranque encontramos una de las diferencias más claras entre las dos grandes familias de sistemas Unix: BSD y System V. En la familia BSD, INIT ejecuta el archivo `/etc/rc`, un simple *shellscript* que lanza directamente algunos programas necesarios para el arranque de la máquina e invoca a otros *scripts* (`/etc/rc.local`, `/etc/netstart`...) para configurar todo lo necesario en un correcto inicio del sistema; en este caso, si queremos añadir o eliminar programas lanzados de forma automática en el arranque, no tenemos más que modificar las líneas correspondientes en `/etc/rc` (o en alguno de los *shellscripts* lanzados por él).

Por contra, en el arranque de la familia System V, INIT lee (**lee**, no **ejecuta**) el fichero `/etc/inittab`, desde el que se invocan utilidades (en función del sistema Unix pueden denominarse simplemente `rc` o `rc2`, `rc3`...) que se encargan de ejecutar pequeños *scripts* que lanzan diferentes tareas necesarias

para el arranque del sistema. Estos *scripts* a los que hacemos referencia se ubican en directorios como `/etc/rc.d/` o `/etc/` y se denominan `rc?.d/` (siendo `'?'` un número entre 0 y 6, correspondiente a cada *runlevel* posible); junto a estos directorios suele ubicarse uno llamado `init.d/`, donde se almacenan realmente los *scripts* de arranque del sistema (los de los directorios `rc?.d` suelen ser enlaces a ellos).

El nombre de cada uno de los *scripts* indica que se va a ejecutar durante la entrada al *runlevel* (si comienza por `'S'`) o durante la salida del mismo (si comienza por `'K'`); a continuación se emplaza un número que no indica más que el orden de ejecución, y finalmente una cadena identificativa. Para añadir un *script* al arranque, no tenemos más que ubicarlo en el directorio correspondiente y darle un nombre que comienze por `'S'` (recordemos que ha de ser mayúsculas, ya que Unix es *case sensitive*); si ya está en `init.d/` únicamente necesitamos enlazarlo, y si no, debemos crearlo. Para detener el arranque de un determinado subsistema, sólo tenemos que eliminar el fichero correspondiente.

Veamos un ejemplo que clarifique todo esto; en Solaris, todos los *scripts* de arranque del sistema se ubican en el directorio `/etc/init.d/`, y los que se ejecutan en cada *runlevel* están en `/etc/rc?.d/`: son simples enlaces a los anteriores. Así, existe un archivo `/etc/rc2.d/S88sendmail` que inicia el subsistema de correo al entrar en el nivel 2, y que no es más que un enlace a `/etc/init.d/sendmail`; de esta forma, si no queremos que este subsistema se arranque, no tenemos más que eliminar el fichero `S88sendmail`:

```
anita:~# rm -f /etc/rc2.d/S88sendmail
anita:~#
```

Si más tarde quisiéramos volver a automatizar esta tarea en el arranque, volveríamos a crear el enlace a `/etc/init.d/sendmail`:

```
anita:~# ln /etc/init.d/sendmail /etc/rc2.d/S88sendmail
anita:~#
```

En realidad, existe un tercer modo de arranque, ni System V ni BSD, que es el utilizado por AIX (el Unix de IBM, el que más características propietarias incorpora). AIX emplea una mezcla entre ambos modelos, de forma que existe un archivo `/etc/inittab` que INIT interpreta, al igual que hemos comentado para System V, pero no encontramos los directorios `rc?.d/` habituales de esta familia, sino que desde `inittab` directamente se invoca a los *shellscripts* necesarios para el arranque del sistema; de esta forma, para añadir un nuevo servicio al arranque de la máquina, lo más habitual es crear una entrada en `/etc/inittab` que lance dicho servicio (directa o indirectamente). Por ejemplo, si el *script* que inicia el servidor *web* se denomina `/etc/rc.httpd`, incluiremos una línea como la siguiente:

```
bruja:~# mkitab "apache:2:wait:/etc/rc.httpd >/dev/console 2>&1"
bruja:~#
```

Hoy en día muchos entornos Unix utilizan como modelo de arranque una mezcla entre *System V* y BSD, aunque cada día más se tiende al esquema *System V*. En cualquier caso, en todos los casos en los que existe, el fichero `/etc/inittab` del que INIT lee su configuración no es más que un archivo de texto que indica al proceso las tareas que tiene que ejecutar durante el arranque. Como superusuarios podemos crear nuevos niveles de operación o modificar los existentes, aunque esto no va a ser en absoluto recomendable para la estabilidad de la máquina a no ser que conozcamos en profundidad el sistema con el que trabajamos. Además, podremos cambiar el *runlevel* en que se encuentra la máquina desde la línea de mandatos, con la orden `init`.

En el caso de Slackware Linux (el utilizado durante el curso), tenemos un modelo muy similar al de AIX; podemos ver que en un arranque normal del sistema se van a invocar en primer lugar un par de *shellscripts* (`rc.S` y `rc.M`) situados en el directorio `/etc/rc.d/`. En este directorio vamos a

encontrar archivos que ofrecen al sistema una configuración determinada cada vez que se arranque o detenga la máquina, como el lanzamiento de ciertos demonios (un *daemon* es simplemente un programa que se ejecuta permanentemente en el sistema y espera peticiones de otros programas; por ejemplo `lpd` espera trabajos para imprimir). Un ejemplo habitual de fichero `/etc/inittab` en este caso es el siguiente:

```
rosita:~# cat /etc/inittab
# Default runlevel.
id:3:initdefault:
# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.S
# Script to run when going single user (runlevel 1).
su:1S:wait:/etc/rc.d/rc.K
# Script to run when going multi user.
rc:23456:wait:/etc/rc.d/rc.M
# What to do at the "Three Finger Salute".
ca::ctrlaltdel:/sbin/shutdown -t5 -rfn now
# Runlevel 0 halts the system.
l0:0:wait:/etc/rc.d/rc.0
# Runlevel 6 reboots the system.
l6:6:wait:/etc/rc.d/rc.6
# What to do when power fails (shutdown to single user).
pf::powerfail:/sbin/shutdown -f +5 "THE POWER IS FAILING"
# If power is back before shutdown, cancel the running shutdown.
pg:0123456:powerokwait:/sbin/shutdown -c "THE POWER IS BACK"
# If power comes back in single user mode, return to multi user mode.
ps:S:powerokwait:/sbin/init 5
# The getties in multi user mode on consoles an serial lines.
#
# NOTE NOTE NOTE adjust this to your getty or you will not be
#                able to login !!
#
# Note: for 'agetty' you use linespeed, line.
# for 'getty_ps' you use line, linespeed and also use 'gettydefs'
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
# Runlevel 4 used to be for an X-window only system, until we discovered
# that it throws init into a loop that keeps your load avg at least 1 all
# the time. Thus, there is now one getty opened on tty6. Hopefully no one
# will notice. ;^)
# It might not be bad to have one text console anyway, in case something
# happens to X.
x1:4:wait:/etc/rc.d/rc.4
# End of /etc/inittab
rosita:~#
```

Podemos fijarnos en que cada línea del fichero `/etc/inittab` está: formada por cuatro campos separados por el carácter `':'`, de esta forma:

código : runlevel : acción : proceso

Cada uno de estos campos tiene el siguiente significado:

código: Cadena de hasta cuatro caracteres que **identifica** a la tarea.

runlevel: Nivel o niveles de operación en el que se ha de ejecutar la tarea; responde a la pregunta **¿cuándo?**. Los niveles de operación habituales en Unix son los siguientes (el número suele cambiar entre sistemas):

0: Desconexión de máquina.
1: Monousuario, para tareas administrativas.
2: Multiusuario.
3: Multiusuario con sistemas de ficheros remotos.
4: Sistemas con sólo X-Window. En `/etc/inittab` podemos leer el problema que presenta este nivel si no se tiene una ventana de texto, por lo que 'tty6' también estará activa en el *runlevel* 4.
5: Desconexión y arranque.
6: Desconexión a ROM.
s: Monousuario, sólo puede trabajar el root desde consola.
S: Monousuario con consola remota.

acción: Indica la forma de ejecución de la tarea, respondiendo a la pregunta **¿Cómo?**. Entre las formas de ejecución más destacables encontramos:

initdefault: Indica en qué nivel de operación ha de entrar primero.
sysinit: Se ha de ejecutar durante la inicialización del sistema, sin importar el *runlevel*.
wait: El proceso se va a lanzar una única vez, al entrar en el *runlevel* indicado, e INIT esperará su finalización.
powerfail: Se ejecuta al detectar un fallo en alimentación.
powerokwait: Se ejecuta al recuperar la alimentación tras un fallo.
respawn: Se relanza el proceso si éste muere.
once: El proceso se ejecuta una única vez, al entrar al *runlevel* indicado.
ctrlaltdel: Se ejecuta si alguien ha pulsado esta combinación de teclas desde la consola del sistema. En algunos casos puede interesarnos no permitir el *shutdown* del sistema al pulsar esta combinación, por ejemplo si varios usuarios conectan desde consola, para evitar que cualquiera pueda resetear la máquina.

proceso: Indica al proceso la orden (o *shellscript*) a ejecutar en cada caso, respondiendo a la pregunta **¿Qué?**.

Como hemos comentado, los dos *shellscripts* invocados por INIT (`rc.S` y `rc.M`) van a encargarse de tareas rutinarias en cada arranque, como comprobar y montar los sistemas de ficheros o activar el área de *swap*, y también de invocar a otra serie de *scripts* en `/etc/rc.d/`, como `rc.cdrom` (buscará y montará una unidad de CD-ROM en el sistema), `rc.inet1` y `rc.inet2` (configuración del sistema de red), o `rc.local` (donde, como administradores, vamos a especificar comandos que nos interese ejecutar en el arranque del sistema donde trabajemos y que no vengan especificados en otros *scripts*, como cargar `gpm` para utilizar el ratón en consola, activar Num Lock, o actualizar la fecha del sistema con la de otro *host* de la red, mediante `netdate`).

Una vez ejecutados estos scripts de inicialización, y después de configurar ciertos parámetros del sistema, vemos que en un arranque normal (*runlevel 3*) se van a invocar una serie de terminales virtuales operativas desde consola, como ya dijimos en el capítulo anterior mediante la combinación de teclas *Alt-F[1..6]*. Esto se consigue, en el caso de Linux, mediante la orden `agetty` con las opciones adecuadas; hemos de fijarnos que la acción indicada en este caso es *respawn* (relanza el proceso cuando éste muere), para conseguir que tras una sesión en una de estas terminales virtuales (*tty1-tty6*) se vuelva a invocar a `agetty` y pueda ser posible una nueva conexión en esa terminal. La configuración de las terminales se encuentra en algunos Unix en el archivo `/etc/gettydefs`.

Tras todos estos pasos el sistema está preparado para empezar a recibir conexiones, al menos

desde consola. Para ello, se invoca al programa `/bin/login`, que se encarga de comprobar el *login* y *password* del usuario en el fichero `/etc/passwd`. Si todo es correcto, asigna unos valores a determinadas variables de entorno (como *$IFS*, el separador de campos de Unix) que no vamos a estudiar, **vuelca** el contenido de `/etc/motd` y **ejecuta** `/etc/profile`. Al usuario se le ofrece un *shell* especificado en el archivo de contraseñas (`/etc/passwd`), se le sitúa en su directorio *$HOME*, y entonces el sistema ha de estar dispuesto para recibir órdenes desde esta conexión.

Aunque no es algo relativo estrictamente al arranque de la máquina, vamos a hablar del archivo `/etc/profile`, que se ejecuta cada vez que se abre una conexión, ya sea desde consola o remota, mediante `telnet` u otros mecanismos, como `ssh` o *SSL-telnet*. El archivo `/etc/profile` va a ser el encargado de inicializar una serie de variables de entorno comunes a todos los usuarios, como *$PATH, $MANPATH, $PS1* y *$PS2*. También podríamos definir en este fichero una especie de filtro para evitar el acceso por *telnet* desde ciertos lugares (si no disponemos de *TCPWrappers*, un software de seguridad más avanzado), enviar un mensaje a ciertos usuarios (por ejemplo en función del grupo al que pertenecen), etc. En definitiva, cualquier cosa que necesitemos como administradores, siempre teniendo en cuenta que el archivo se va a ejecutar al inicio de cada sesión para todos los usuarios, incluido por supuesto el *root*. Un ejemplo típico del contenido de este fichero puede ser el siguiente:

```
rosita:~# cat /etc/profile
# commands common to all logins
export OPENWINHOME=/usr/openwin
export MINICOM="-c on"
export MANPATH=/usr/local/man:/usr/man/preformat:/usr/man:/usr/X11/man:
/usr/openwin/man
export HOSTNAME="`cat /etc/HOSTNAME`"
export LESSOPEN="|lesspipe.sh %s"
PATH="$PATH:/usr/X11/bin:/usr/andrew/bin:$OPENWINHOME/bin:/usr/games:
/usr/local/bin/varios:."
LESS=-MM
# I had problems using 'eval tset' instead of 'TERM=', but you might
# want to
# try it anyway. I think with the right /etc/termcap it would work
# great.
# eval `tset -sQ "$TERM"`
if [ "$TERM" = "" -o "$TERM" = "unknown" ]; then
 TERM=linux
fi
#PS1='`hostname`:`pwd`# '
if [ "$SHELL" = "/bin/pdksh" -o "$SHELL" = "/bin/ksh" ]; then
 PS1="! $ "
elif [ "$SHELL" = "/bin/zsh" ]; then
 PS1="%m:%~%# "
elif [ "$SHELL" = "/bin/ash" ]; then
 PS1="$ "
else
 PS1='\h:\w$ '
fi
PS2='> '
ignoreeof=10
export PATH DISPLAY LESS TERM PS1 PS2 ignoreeof
umask 022
# set up the color-ls environment variables:
if [ "$SHELL" = "/bin/zsh" ]; then
  eval `dircolors -z`
```

```
elif [ "$SHELL" = "/bin/ash" ]; then
  eval `dircolors -s`
else
  eval `dircolors -b`
fi
fortune
echo
rosita:~#
```

2.2 Parada del sistema

Para detener correctamente un sistema Unix hemos de situar el proceso INIT en un determinado *runlevel*, generalmente 0 o 6 (podemos comprobarlo simplemente echando un vistazo a `/etc/inittab`). Para ello, el sistema ofrece al superusuario varios mandatos que pueden ser invocados desde el *shell*, y que se suelen encontrar en el directorio `/sbin/`, como `shutdown`, `halt` o `reboot`.

Tanto `halt` como `reboot` detienen el sistema; la diferencia entre ellos es que la segunda orden lo reinicializa. En principio sólo vamos a poder invocar estas órdenes cuando nos encontremos en los *runlevels* 0 o 6; si nos encontramos en otro nivel de operación habremos de utilizar `shutdown` para poder asegurar la estabilidad del sistema. Aunque en la mayoría de Unices no suelen existen problemas por esta causa, ya que INIT detecta el estado actual del sistema, y si no es alguno de los anteriores, invoca de forma automática a `shutdown` con los parámetros adecuados, es recomendable que nos acostumbremos a detener nuestros sistemas utilizando la orden `shutdown`.

`shutdown` va a detener la máquina de una forma ordenada, siguiendo unos pasos definidos. En primer lugar, notifica el hecho a todos los usuarios conectados (mediante `wall`) y bloquea el proceso de `login` creando el fichero `/etc/nologin`; siempre que este archivo exista (tanto durante la parada de la máquina como durante su funcionamiento normal), únicamente el administrador va a poder conectar al sistema. Posteriormente invoca a INIT en un *runlevel* 0 (para simplemente detener el sistema), 6 (para reinicializarlo) o incluso 1 (monousuario, para realizar tareas administrativas). Entonces INIT ejecuta el *script* correspondiente (leído de `/etc/inittab`), que suele encargarse de eliminar todos los procesos de la máquina, notificar el evento en el fichero de *log* correspondiente, desmontar los sistemas de ficheros que existan, desactivar el área de *swap* y, según se haya invocado la orden, detener el sistema o reinicializarlo.

La forma habitual (que se da en el 90% de los casos) de invocar a `shutdown` es

```
shutdown -r/-h now
```

Con el parámetro `-r` hacemos un *reboot*, y con `-h` un *halt* (simplemente detenemos el sistema, sin reinicializar).

2.3 El intérprete de órdenes

Hemos visto antes que el proceso *login* nos asigna un intérprete de mandatos, leído de nuestra entrada en `/etc/passwd`. Ya sabemos que un *shell* no es más que un programa del sistema que nos va a permitir relacionarnos con la máquina, introduciéndole órdenes desde el teclado; es la interfaz de usuario que nos ofrece cualquier sistema operativo.

En nuestro sistema Unix podemos tener instalados diversos intérpretes de órdenes; en el archivo `/etc/shells` tenemos los disponibles para cada usuario, incluido el *root*. Por defecto se asigna el *Bourne Again Shell (*`bash`*)*, el más cómodo, que incluye lo mejor de todos los anteriores intérpretes, como el *C-Shell (*`csh`*)* o el *Korn Shell (*`ksh`*)*. `bash` incluye rodillo de mandatos, histórico de órdenes (en el archivo `.bash_history` del directorio *$HOME* de cada usuario), uso de tabuladores para

completar órdenes, etc.

Cualquier intérprete de Unix nos ofrece un entorno de programación completo con herramientas similares a las de la mayoría de lenguajes de programación secuencial (*if, for, while...*). También nos va a ofrecer una serie de órdenes, denominadas órdenes internas, que no encontramos como archivos ejecutables en el sistema, y que por supuesto no va a ser necesario cargar en memoria para poder ejecutarlas; un ejemplo típico de este tipo de instrucciones es `exit`: no encontraremos ningún ejecutable en todos los sistemas de ficheros de Unix que se denomine así.

En Unix existe el mandato `chsh`, que permite a cualquier usuario modificar el *shell* que el sistema le asigna por defecto (recordemos que este dato está grabado en `/etc/passwd`). En determinados clones, `chsh` no va a comprobar que el *shell* sea válido (esto es, que sea una entrada de `/etc/shells`). Esto puede suponer un problema: imaginemos que un usuario inexperto cambia su *shell* de entrada a `/bin/login`; sería imposible para él volver a entrar al sistema mediante conexión remota hasta que el administrador modifique la información de `/etc/passwd`. Por tanto, desde hace algún tiempo la mayoría de Unices del mercado (Linux entre ellos) hacen la comprobación oportuna.

Analizando el funcionamiento de `chsh`, vemos que una de las cosas que realiza es modificar el fichero de contraseñas para cambiar el *shell* de entrada de un usuario; cualquier usuario que lo ejecute está escribiendo en un archivo sobre el que no tiene permiso de escritura, por lo que es inmediato deducir que el ejecutable `chsh` está setuidado. Puede ser interesante para nosotros resetear el bit de *setuid* del archivo (`chmod -s`) para así conseguir un mayor nivel de seguridad del sistema, ya que, como veremos más adelante, es conveniente mantener al mínimo el número de archivos *setuidados* en la máquina. Si lo hacemos así y algún usuario quiere cambiar su intérprete de órdenes de forma automática cada vez que entre al sistema puede incluir en su *.profile* la orden `exec` invocando a un nuevo intérprete; como veremos más adelante, al hablar de procesos en un sistema Unix, este mandato sustituye en memoria el proceso que lo invoca por el ejecutable cuyo nombre recibe como parámetro. De esta forma, si por ejemplo el intérprete asignado a un usuario es `/bin/sh` y desea trabajar habitualmente con `/bin/ksh`, este usuario puede, sin ningún privilegio especial, introducir una línea en su `.profile` como la siguiente:

```
luisa:~$ cat .profile
exec /bin/ksh
luisa:~$
```

Cada vez que el usuario inicie una sesión su *shell* de entrada ejecutará el fichero `.profile`, con lo que será sustituido por el nuevo intérprete.

Capítulo 3

Procesos

3.1 Conceptos básicos

El sistema operativo Unix, por ser multitarea, ha de multiplexar el procesador entre los diferentes ejecutables cargados en memoria. Ha de asignar y retirar la CPU a los programas sin que estos se percaten de ello. Por tanto, no basta sólo con tener el binario cargado en un área de memoria para conseguir una correcta ejecución, sino que se le ha de añadir un entorno que contiene información adicional (valor de registros, ficheros abiertos...). Al conjunto formado por el ejecutable en memoria y este entorno se le denomina **imagen**, y contiene toda la información para ejecutarse a intervalos de tiempo (*quantums*) en el procesador.

Cuando una de las diferentes imágenes consigue la CPU se le denomina **proceso**. Un proceso es, por lo tanto, el aspecto dinámico de una imagen. Como en un intervalo de tiempo más o menos corto todas las imágenes se van a ejecutar, se suelen confundir ambos términos, asimilando el concepto de imagen al de proceso.

La mayoría de órdenes (excepto las internas al *shell*, como `cd, exit` o la asignación de valores a variables de entorno) utilizados en una sesión Unix son procesos. El *shell* mismo es un proceso del sistema, que se va a ejecutar durante toda la sesión. Cuando un usuario introduce una orden, el intérprete de mandatos se encarga de crear un nuevo proceso (con la llamada `fork()`), que será hijo suyo; el *shell*, a su vez, es hijo del proceso `login`. El proceso creado con `fork()` es idéntico al padre, por lo que ha de ejecutar la primitiva `exec()` para sustituir su espacio en memoria por el nuevo código a ejecutar (por ejemplo, un mandato de Unix).

El sistema operativo Unix identifica unívocamente a cada proceso con lo que se denomina Identificador de Proceso (*PID, Process Identifier*). **No es posible que dos procesos tengan el mismo** *PID*, aunque generalmente varios procesos van a compartir otro identificador llamado *PPID* (*Parent Process Identifier*) que referencia al padre de un determinado proceso, ya que es habitual que un proceso tenga varios hijos antes de morir.

En un determinado momento, un proceso puede encontrarse en un estado, en función del punto de ejecución en que se encuentre y también de la capacidad del procesador para ejecutarlo. El diagrama de estados y transiciones (elemental) de un proceso es el mostrado en la figura 3.1. Un proceso listo sólo necesita recibir CPU para poder ejecutarse; en cambio, uno suspendido está esperando un servicio o un evento (por ejemplo, una entrada de teclado) para continuar su ejecución. A los procesos listos y en ejecución se les denomina activos. Cuando a un proceso se le expulsa o planifica tenemos un **cambio de contexto**: para esto es necesario que la imagen del proceso sea completa (de esta forma el sistema operativo sabe el punto en que ha quedado un proceso expulsado, sus ficheros, sus registros... y podrá más tarde continuar ejecutándolo desde donde lo dejó).

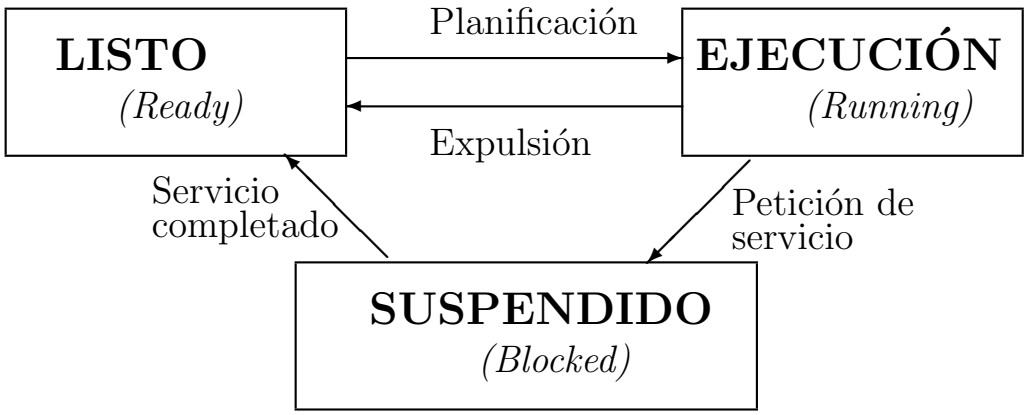


Figura 3.1: Diagrama de transiciones para los estados de un proceso

3.2 Actividades de los usuarios

Los diferentes usuarios de la máquina Unix van a lanzar procesos al sistema, los cuales se van a unir con los ya existentes (como los *daemons* o cualquier proceso de configuración), y todos juntos van a competir por conseguir *quantums* de procesador. Es posible para cualquier usuario (y por supuesto también para el `root`) examinar todos los procesos lanzados en la máquina mediante la orden `ps` (*Process Status*). Esta orden lee la información almacenada en el directorio `/proc/`, un sistema de ficheros especial que utiliza Linux (generalmente no otros Unices) para evitar otorgar privilegios especiales a `/bin/ps`, que serían necesarios para analizar la memoria del sistema viendo los procesos lanzados si trabajamos en un Unix sin `/proc/`.

Las opciones de la orden `ps` son unas de las más variables entre Unices. En la familia de Unix BSD (por ejemplo, en Linux), nosotros vamos a utilizar las siguientes :

- `-a`: Nos muestra procesos de todos los usuarios.
- `-u`: Da información sobre el propietario del proceso y la hora de lanzamiento.
- `-x`: Muestra también procesos que no están asociados a una terminal.

Por supuesto, podemos combinar todas: `ps -aux` (la orden equivalente a ésta en sistemas *System V*, como Solaris, IRIX o HP-UX, es `ps -ef`). De esta forma podremos ver todos los procesos lanzados en el sistema [1].

La orden `ps` nos va a proporcionar una serie de información acerca de los procesos que, como administradores, nos va a ser útil para controlar las actividades de los usuarios, como ya veremos en el siguiente capítulo. Los datos de un proceso que podemos considerar más importantes (aunque no todos se nos muestran con `ps -aux`) son:

USER: Obviamente, indica el propietario de un proceso, la persona que lo lanzó.
PID: El identificador de proceso que el sistema ha asignado a la tarea.
PPID: *PID* del padre del proceso.

[1] En nuevas versiones de *procps* (el `ps` que lee la información del sistema de ficheros *proc*) se han eliminado los guiones; de esta forma, `ps` con las opciones `-aux` se tecletaría `ps aux`.

%CPU: Porcentaje de tiempo de CPU consumido. Es el tiempo utilizado dividido por el tiempo que el proceso ha estado en ejecución.
%MEM: Porcentaje de memoria utilizada por el proceso.
TTY: Terminal a la que va asociado ese proceso; aunque coincide casi siempre con la *tty* desde la que se lanzó, puede no ser así (por ejemplo, en los procesos `agetty`). En caso de ser un proceso sin control de terminal (como puede ser el caso de un demonio), se imprime el símbolo *"?"*.
STAT: Estado del proceso. Los diferentes estados que Unix habitualmente reconoce son los siguientes, aunque se recomienda consultar la página del manual de *ps* para cada sistema concreto:

I: Intermedio.
O: Inexistente.
R: Ejecutándose (*Running*).
S: Durmiendo (*Sleeping*).
T: Parado o traceado (*sTopped* o *Traced*).
W: Esperando (*Waiting*).
Z: Terminado (*Zombie*).

START: Hora de lanzamiento del proceso.
TIME: Tiempo de ejecución acumulativa del proceso.
COMMAND: Nombre del proceso; por norma general, es la línea de órdenes (o parte de ella) que se tecleó en el *prompt* del sistema.

Ya sabemos que muchos de los procesos del sistema no son lanzados por los usuarios desde un *shell*. En un sistema Unix típico encontraremos entre veinte y treinta procesos considerados 'del sistema', lanzados por INIT o por el `root` de forma automática al arrancar la máquina; los más comunes son los siguientes:

- `init`. Es el **proceso del cual nacen todos los demás**, necesario para un buen funcionamiento del sistema.
- `syslogd`. Es un demonio que registra diversas actividades del sistema, como las conexiones de los usuarios, los intentos de conexión, errores generados por diversos programas...
- `klogd`. Es el demonio encargado de registrar actividades del *kernel*.
- `inetd`. Es el demonio encargado de atender peticiones de red, como `ftp` o `telnet`.
- `update`. Este programa se encarga de vaciar los *buffers* del sistema de ficheros periódicamente.
- `kerneld`. Realiza actividades propias del *kernel* en el espacio de memoria de los usuarios, como la inserción de módulos.
- `lpd`. El demonio de impresión, atiende peticiones de los usuarios y gestiona el servicio de impresión.
- `sendmail`. El encargado de gestionar correctamente el correo electrónico de los usuarios.
- `crond`. Este demonio permite la automatización de tareas, como ya veremos en capítulos posteriores.
- `httpd`. Demonio encargado de servir las conexiones WWW al sistema.
- `agetty`. El proceso `agetty` espera una conexión en una determinada terminal, tanto virtual como física.
- `gpm`. Programa que permite trabajar con el ratón desde la consola del sistema en modo texto.

Otra orden útil para analizar procesos es `top`; este mandato nos muestra en pantalla los procesos que más CPU están consumiendo en el sistema, es decir, aquellos que eventualmente pueden ocasionar problemas al ralentizar la máquina. Debemos tener cuidado con estos procesos a la hora de analizarlos y eliminarlos, ya que un proceso que consume CPU en exceso puede ser tanto una tarea útil como una que haya tenido problemas (por ejemplo, por defectos en el código). Al igual que `ps`, la orden `top` lee la información del directorio `/proc/`, por lo que si este sistema de ficheros no está montado, la instrucción no funcionará.

La información proporcionada por `top` es dinámica (por defecto se actualiza cada cinco segundos), y además nos proporciona un interfaz interactivo que nos permite manipular procesos, aceptando una serie de órdenes que a continuación citamos:

> *s*: Especifica el retardo entre actualizaciones de pantalla (podíamos indicarlo también desde la línea de órdenes con la opción *-d*).
> *h, ?*: Dibuja una pantalla de ayuda de *top*.
> *k*: Como la orden `kill`, envía una señal a un proceso. Se nos va a preguntar el *PID* del proceso al que deseemos enviar la señal (obviamente podremos enviarle cualquiera, ya que como administradores tenemos el suficiente nivel de privilegio para hacerlo).
> *i*: Ignora los procesos *idle* y *zombies* (podíamos indicar esta opción desde la línea de órdenes: `top -i`).
> *q*: Sale de `top` y vuelve al *shell*.

Un ejemplo típico del entorno ofrecido por `top` es el siguiente:

```
12:26am up 1 min, 2 users, load average:  0.71, 0.29, 0.10
30 processes: 29 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 2.6% user, 8.8% system, 0.0% nice, 88.6% idle
Mem: 14748K av, 10504K used, 4244K free, 7140K shrd, 836K buff
Swap: 16892K av, 0K used, 16892K free, 6180K cached
```

PID	USER	PRI	NI	SIZE	RES	SHRD	STAT	%CPU	%MEM	TIME	COMMAND
113	root	11	0	860	452	336	R	10.6	3.0	0:00	top
111	root	2	0	772	320	272	S	0.8	2.1	0:00	script
1	root	0	0	780	324	256	S	0.0	2.1	0:01	init
2	root	0	0	0	0	0	SW	0.0	0.0	0:00	kflushd
3	root	-12	-12	0	0	0	SW¡	0.0	0.0	0:00	kswapd
4	root	0	0	0	0	0	SW	0.0	0.0	0:00	nfsiod
5	root	0	0	0	0	0	SW	0.0	0.0	0:00	nfsiod
6	root	0	0	0	0	0	SW	0.0	0.0	0:00	nfsiod
7	root	0	0	0	0	0	SW	0.0	0.0	0:00	nfsiod
88	root	0	0	1104	628	488	S	0.0	4.2	0:00	-bash
13	root	0	0	768	296	232	S	0.0	2.0	0:00	/sbin/kerneld
95	root	0	0	976	472	372	S	0.0	3.2	0:11	sshd
12	root	0	0	756	252	200	S	0.0	1.7	0:00	/sbin/update
43	root	0	0	776	312	240	S	0.0	2.1	0:00	/usr/sbin/crond
54	root	0	0	788	384	296	S	0.0	2.6	0:00	/usr/sbin/syslogd
56	root	0	0	984	520	268	S	0.0	3.5	0:00	/usr/sbin/klogd
58	bin	0	0	796	324	260	S	0.0	2.1	0:00	/usr/sbin/rpc.port
60	root	0	0	772	324	260	S	0.0	2.1	0:00	/usr/sbin/inetd

Como vemos, la información de la tabla de procesos es bastante similar a la mostrada por `ps`. Sin embargo, en la parte superior se nos muestran muchos datos relativos al estado del procesador:

La primera línea es lo que se conoce como el *uptime* de la máquina (`uptime` es una orden de Unix, disponible para todos los usuarios, que informa sobre la carga del sistema y también sobre

el tiempo que lleva el sistema funcionando ininterrumpidamente). Las tres cargas indicadas son la media de procesos listos durante el último minuto, los últimos cinco y los últimos quince minutos. En la segunda línea se nos muestran los procesos del sistema, indicando el número de ellos en cada estado. La siguiente línea nos informa de en qué ocupa la máquina su tiempo de procesador (en este ejemplo vemos que la mayoría del tiempo está *idle*, ociosa). Para finalizar, se nos da información relativa al estado de la memoria (tanto física como *swap*). Estas líneas son equivalentes a la salida de la instrucción `free`, también disponible para todos los usuarios.

Como vemos, una de las columnas de `top` es *PRI*, que hace referencia a la prioridad (cuota de tiempo de la CPU) con que se ejecuta una tarea en el sistema. Los procesos en Unix tienen por defecto una prioridad determinada (en el caso de Linux, 0), que se hereda de padres a hijos. La prioridad se representa mediante un entero entre -20 (prioridad más alta) y 19 (prioridad más baja). Cualquier usuario puede ejecutar un proceso con una prioridad menor a la del sistema mediante la orden `nice`, que obviamente nadie utiliza. Si no se especifica un ajuste (con la opción `-n`), la prioridad se incrementa en 10 unidades. Así la orden

```
rosita:~# nice ls
```

se ejecutará con una prioridad menor a

```
rosita:~# ls
```

Sólo el administrador del sistema puede especificar prioridades negativas (altas) para procesos que necesiten tener más CPU que el resto. Si nos interesa conocer la prioridad en que trabajamos, simplemente teclearemos `nice` en la línea de órdenes; si queremos tener una prioridad por defecto diferente a la que el sistema ofrece, hemos de asignarle esa nueva prioridad a nuestro *shell*, ya que la prioridad es una característica que heredan los procesos hijos de sus padres, y recordemos que todos los procesos de una sesión son descendientes de un *shell*. No es necesario conocer de antemano la prioridad que queremos para un proceso; si una vez lanzado el proceso queremos modificar su prioridad, podemos hacerlo con la orden `renice`:

```
luisa:~# sleep 100 &
[1] 14045
luisa:~# ps -o "%U %p %n %c"
USER       PID  NI COMMAND
toni     13980   0 bash
toni     14045   0 sleep
toni     14048   0 ps
luisa:~# renice 10 14045
14045: old priority 0, new priority 10
luisa:~# ps -o "%U %p %n %c"
USER       PID  NI COMMAND
toni     13980   0 bash
toni     14045  10 sleep
toni     14050   0 ps
luisa:~#
```

3.3 Control de procesos

Con las herramientas que hemos presentado en este capítulo ya debemos ser capaces de analizar la tabla de procesos del sistema y detectar posibles tareas problemáticas. Podemos considerar una tarea problemática a un proceso que intenta afectar al sistema de cualquier forma: por ejemplo, podemos tener un proceso que cope toda la memoria de la máquina, un programa que utilice más CPU de la que en principio le corresponde, o quizás un excesivo número de procesos *zombies* (marcados como <*zombie*> o <*defunct*> al realizar un `ps`) en el sistema.

Un proceso *zombie* es aquél que ha fallecido pero su padre no ha reconocido su muerte. En este caso el sistema no va a permitir a la tarea que muera de muerte normal, sino que la coloca en este estado, en el que el proceso no está realmente muerto, sino que simplemente no está ejecutándose ni utilizando ningún recurso de la máquina. Generalmente este estado excepcional dura unos pocos segundos como máximo, hasta que el padre efectúa una operación `wait()` y recibe la señal de muerte de su hijo. Sin embargo, un proceso *zombie* que permanece en el sistema durante demasiado tiempo, o la excesiva presencia de tareas en este estado, denota algún problema. Comúnmente, este problema está relacionado con el padre del proceso, que por algún motivo está bloqueado y no puede reconocer las muertes de sus hijos. Por tanto, debemos eliminar al padre, para que INIT adopte a sus hijos *zombies* y los elimine.

Ya sabemos que para eliminar un proceso debemos utilizar la orden `kill` (en determinados Unices también existen órdenes como `killall` o `pkill`, que eliminan procesos a partir de su nombre y no de su PID). La orden `kill` envía una determinada señal a un proceso (aunque no necesariamente una señal *SIGKILL*). Una tarea puede capturar o ignorar las señales, para que no le afecten o para darles un tratamiento determinado, a excepción de la señal *SIGKILL*, que supone siempre una muerte segura. La sintaxis de `kill` es básica:

kill [-sig] PID

Si no se especifica el parámetro *sig*, por defecto `kill` va a enviar una señal de tipo *SIGTERM*. Las diferentes señales (que no son más que un software que envía interrupciones asíncronas a los procesos) de un sistema Linux (para ver una lista de señales en cualquier Unix, `kill -l`) son:

Nº de señal	Nombre	Descripción
1	SIGHUP	Desconexión de línea
2	SIGINT	Interrupción desde teclado
3	SIGQUIT	Tecla QUIT
4	SIGILL	Instrucción ilegal
5	SIGTRAP	Trace/breakpoint trap
6	SIGABRT	Abort
7	SIGUNUSED	No utilizada
8	SIGFPE	Floating point instruction expection
9	SIGKILL	Final de proceso
10	SIGUSR1	Definida por el usuario
11	SIGSEGV	Referencia a memoria no válida
12	SIGUSR2	Definida por el usuario
13	SIGPIPE	Escritura en una tubería que no tiene un proceso asociado para lectura
14	SIGALRM	Interrupción de reloj
15	SIGTERM	Señal de terminación
16	SIGTKFLT	Error en pila de coprocesador
17	SIGCHLD	Finalización de un hijo
18	SIGCONT	Continúa el proceso si está parado
19	SIGTSTOP	Detiene el proceso
20	SIGTSTP	Stop desde tty
21	SIGTTIN	Entrada desde tty, para proceso en background
22	SIGTTOU	Salida a tty, para proceso en background
23	SIGURG	Condición urgente en socket
24	SIGXCPU	Tiempo máximo de CPU excedido
25	SIGXFSZ	Excedido tamaño máximo de fichero
26	SIGVTALRM	Alarma de tiempo virtual
27	SIGPROF	Señal de profile

28	SIGWINCH	Modificación de tamaño de ventana
29	SIGIO	Error de entrada/salida
30	SIGPWR	Error de alimentación

Por último, debemos recordar que antes de eliminar un proceso de un usuario nos hemos de cerciorar que ese proceso realmente perjudica al sistema, y que además no es una tarea controlada sino que su ejecución es errónea. No podemos limitarnos a eliminar los procesos que consuman demasiada memoria o CPU, porque pueden ser importantes. Por ejemplo, en un nodo de cálculo científico existirán procesos que consuman muchísimos recursos (como el producto de matrices grandes), pero que no perjudican al sistema, sino que necesitan de él para poder obtener un resultado. El administrador ha de conocer lo que es normal en su sistema, para así detectar problemas con los procesos (generalmente estos problemas se notan por un excesivo tiempo de respuesta de la máquina o por una actividad inusual de los discos duros sin ningún motivo aparente). Es habitual, al matar a un proceso de un usuario, enviarle a éste un *mail* explicándole los motivos de la eliminación.

Capítulo 4

Sistemas de ficheros

4.1 Ficheros

En un sistema Unix habitual podemos encontrarnos diferentes clases de ficheros: planos, directorios, enlaces simbólicos (*Berkeley*), especiales orientados a carácter, especiales orientados a bloque, *pipes* (*System V*), sockets (*Berkeley*)... En función del Unix utilizado es posible que incluso encontreos tipos de ficheros propios que sólo existen en clones concretos del operativo.

Un **fichero plano** u ordinario se utiliza para almacenar información (por ejemplo un programa ejecutable, un documento de texto o registros de una base de datos). Simplemente consiste en una serie de bytes guardados en un dispositivo de almacenamiento secundario, como un CD-ROM o una unidad de disco. Los ficheros planos se guardan a su vez en ficheros de tipo **directorio**, que no son más que archivos que contienen una lista de archivos. Los directorios constituyen la base de un sistema de ficheros jerárquico; Unix fué uno de los pioneros en incorporar una jerarquía a los sistemas de ficheros, aunque hoy en día esto nos parezca habitual.

Los **ficheros especiales** no se encargan de almacenar información, sino de proveer de un interfaz uniforme entre los programas y la I/O *hardware* del ordenador. Suelen representar dispositivos físicos, y en función de estos dispositivos se dividen en dos grandes grupos: los **orientados a carácter** y los orientados a bloque:

(a) Los primeros realizan sus operaciones de entrada/salida byte a byte, ofreciendo un acceso secuencial y un tiempo de acceso no acotado. Típicos ejemplos de estos ficheros especiales son las impresoras, las terminales y los módems.

(b) Los ficheros especiales **orientados a bloque** realizan las operaciones de entrada/salida con un tamaño de bloque fijo. El acceso a los dispositivos que suelen representar es aleatorio, y el tiempo de acceso está perfectamente acotado, como en un disco duro o un disco flexible.

Los ficheros de tipo **pipe** o tubería se utilizan para enviar datos de un lugar a otro del sistema (algo parecido – aunque no idéntico – a lo que sucede por ejemplo al filtrar un mandato con '|'). El orden en que se recibe esta información en un extremo de la tubería es el mismo que tenía esta información cuando fue introducida por el otro; por este motivo también se les denomina FIFOs (*First In, First Out*: el primero que entra es el primero que sale). Generalmente estos ficheros se suelen utilizar para comunicar procesos de forma síncrona: el proceso emisor detiene su ejecución hasta que el receptor ha recogido toda la información, o el receptor detiene su ejecución hasta que el emisor envía toda la información (este modelo se denomina 'de cita' o *rendez–vous*); esta comunicación se ha de realizar dentro de nuestro sistema, no entre máquinas diferentes.

Para comunicar procesos entre varias máquinas (y también dentro de un solo sistema) tenemos

los llamados **sockets**, que establecen conexiones asíncronas entre programas que no se ejecutan al mismo tiempo pero que necesitan intercambiar datos. No vamos a entrar aquí en la forma de crear *sockets* o trabajar con ellos, ya que se trata de un campo más dirigido hacia la programación de sistemas, pero como ejemplo de la importancia de este tipo de archivos, cabe decir que la mayoría de aplicaciones de red que conocemos trabajan en base a sockets (`telnet`, FTP, *e–mail*...), aunque no todo *socket* tiene un archivo asociado.

Cada uno de estos ficheros se relaciona con un inodo (o *nodo–i*) en el sistema Unix. Como veremos en el siguiente punto, un inodo no es más que una estructura de datos que contiene información acerca del archivo.

Existen otros tipos de ficheros de los que aún no hemos hablado: los **enlaces** (*links*). Estos ficheros pueden ser de dos tipos: **enlaces duros** y enlaces simbólicos. Los primeros permiten asignar a un mismo fichero dos nombres diferentes en nuestro sistema de archivos; Unix no conoce a sus archivos por su nombre, sino por el número de inodo. Así, un enlace duro no es más que un mismo inodo referenciado con dos nombres de archivo en diferentes lugares de la estructura de directorios.

A diferencia de un *hard link*, un **enlace simbólico** es un apuntador a un archivo, pero no una duplicación de éste en el sistema. Si borramos el fichero referenciado el enlace simbólico pierde toda su utilidad; aunque no desaparezca al hacer un listado, internamente es un fichero que no existe.

Generalmente el núcleo del sistema operativo no pone ninguna condición especial para crear enlaces simbólicos; incluso es posible que un usuario sin ningún tipo de privilegios de acceso sobre un fichero lo enlace de forma simbólica (aunque evidentemente no podrá acceder a él). En el caso de los enlaces duros, sí que existen restricciones impuestas por el núcleo de Unix. En primer lugar, muchos *flavours* (Linux entre ellos) no permiten, por cuestiones de diseño, crear enlaces duros entre directorios; en los que sí se permite esta acción, se suele utilizar para cosas poco habituales y de nivel avanzado, implicando en muchos casos riesgos de pérdida de información. Si en Linux intentamos definir un enlace sobre un directorio, incluso como `root` (recordemos que estamos hablando de 'imposiciones' del *kernel*, por lo que nadie, ni el superusuario, puede saltarselas), obtendremos un mensaje de error similar al siguiente:

```
rosita:~# ln /etc/ .
ln: /etc/: hard link not allowed for directory
rosita:~#
```

La segunda gran restricción a los enlaces duros impuesta por Unix (esta sí por todos los clones) es la definición de *hard links* entre archivos de diferentes sistemas de ficheros (discos, particiones...). En este caso el motivo es obvio: estamos tratando de referenciar, a nivel de núcleo, un inodo desde un sistema de archivos distinto al aquel en el que está ubicado, lo cual no es admisible por arquitectura de Unix. Si intentamos definir estos enlaces, los errores obtenidos serán de esta forma:

```
rosita:~# ln /usr/bin/logger .
ln: cannot create hard link ‘./logger’ to ‘/usr/bin/logger’: \
Invalid cross-device link
rosita:~#
```

Antes de finalizar este punto, quizás sea interesante el indicar como crear cada uno de estos archivos. Ya debemos conocer las órdenes necesarias para crear un nuevo fichero plano, un directorio o un enlace, por lo que no vamos a insistir en ellos. Sin embargo, puede que no sepamos crear ficheros especiales, *pipes* o *sockets*. Tanto los ficheros especiales orientados a carácter como los orientados a bloque, y también los *pipes*, los vamos a crear con la orden `mknod` (*MaKe NODe*). La sintaxis básica de `mknod` es la siguiente:

mknod <fichero> [tipo] major minor

donde <*fichero*> indica obviamente el nombre que queremos asignarle al archivo. En el campo *[tipo]* especificaremos *'p'* (*pipe*), *'c'* (*character*) o *'b'* (*block*), dependiendo del tipo de fichero que deseemos generar. Por último, los campos *major* y *minor* corresponden a lo que se denomina *major number* y *minor number*, número principal y secundario de un archivo especial (estos campos no deberemos indicarlos si estamos creando un *pipe*). El número principal designa un tipo de periférico (impresoras, discos duros, terminales...), mientras que el número secundario hace referencia a un elemento de dicha clase de dispositivo (la impresora conectada a *lp0*, la terminal número 3...). El número principal no es constante entre diferentes Unices; por ejemplo, el *major* 4 puede referenciar discos duros en un Unix y terminales en otro; en el caso particular de Linux, en el fichero `/proc/devices` tenemos la relación de dispositivos reconocidos por el núcleo del sistema, con sus números mayores correspondientes (los menores se asignan sencillamente por orden).

En el caso de *sockets*, ya hemos dicho que no necesariamente un *socket* del sistema va a tener asociado un archivo de este tipo. El archivo se suele asociar al *socket* a un nivel de programación de sistema, por lo que no vamos a entrar en más detalles de cómo hacerlo.

4.2 Sistemas de ficheros

El sistema de ficheros es la parte más visible del sistema operativo; ofrece una perspectiva lógica uniforme del almacenamiento de la información en el sistema de computación, abstrayendo propiedades físicas de los dispositivos para definir una unidad lógica de almacenamiento: el archivo.

Podemos definir un sistema de ficheros en cada unidad de almacenamiento física o incluso definir varios sistemas de ficheros (y de diferente tipo) en estas unidades creando particiones con la orden `fdisk`. Esta instrucción simplemente divide una misma unidad física (como un disco) en diferentes unidades lógicas (particiones), para así poder administrar el espacio disponible con más precisión. Sin embargo, `fdisk` no crea sistemas de ficheros; para esta tarea utilizaremos la orden `mkfs`. En Linux (no así en otros Unices) existen extensiones de `mkfs` predefinidas, para poder crear sistemas de ficheros con más comodidad, como `mkfs.ext2/mke2fs` (*ext2* es el sistema que generalmente va a utilizar Linux), `mkfs.minix` o `mkfs.msdos/mkdosfs`.

Un sistema de ficheros Unix se suele dividir en cuatro partes diferenciadas entre sí, aunque dependiendo del sistema concreto podemos encontrar otras zonas no explicadas aquí:

Bloque Boot: Contiene el programa para inicializar el sistema operativo tras el arranque del ordenador.

Superbloque: El superbloque contiene información relativa al sistema de ficheros, como el espacio disponible, el espacio total, el tipo de sistema...

Lista de inodos: Encontramos una referencia de los inodos del sistema, que contienen información administrativa relativa a ficheros.

Área de datos: Parte del dispositivo lógico donde vamos a poder comenzar a almacenar información.

Hasta ahora hemos hablado de un inodo como una estructura de datos que en el sistema Unix identifica a un fichero. El inodo sirve como interfaz entre el núcleo del sistema operativo y la información almacenada en un dispositivo: recordemos que Unix reconoce a los archivos por un número de inodo, no por su nombre.

Los campos de la estructura que conforma el inodo varían entre distintos tipos de sistemas de ficheros. Así, encontraremos que la estructura del nodo-i en *UFS* puede ser distinta a la de *ext2-FS*. Sin embargo, en todos los sistemas habituales encontramos unos campos fijos:

- Dueño del fichero y grupo al que pertenece.
- Tipo del fichero.
- Protección.
- Enlaces del archivo.
- Fecha de la última modificación.

4.3 Creación e incorporación de sistemas de ficheros

Ya sabemos crear un sistema de ficheros con la orden `mkfs` y sus derivados. Ahora necesitamos incorporar ese sistema de ficheros a la estructura jerárquica de Unix, lo que habitualmente se denomina **montar** un sistema. Para ello, utilizaremos la orden `mount`. La sintaxis básica de esta orden es la siguiente:

```
mount -t tipo dispositivo directorio
```

Los tipos de sistemas de archivos se encuentran en el fichero `/proc/filesystems`. Si no especificamos el tipo con `'-t'`, `mount` intentará automáticamente buscar el modelo correcto leyendo secuencialmente este archivo (o dependiendo de la versión de Linux utilizada sólo lo intentará con la primera entrada de `/proc/filesystems`). Por ejemplo, supongamos la orden

```
rosita:~# mount /dev/hda5 /mnt
```

La instrucción `mount` leerá `/proc/filesystems`, que suele ser de la forma siguiente:

```
rosita:~# cat /proc/filesystems
        ext2
        minix
        umsdos
        msdos
        vfat
nodev   proc
nodev   nfs
        iso9660
rosita:~#
```

Tras leer el archivo, intentará utilizar un tipo *ext2* para `/dev/hda5`; si no es correcto, lo intentará con un tipo de sistema *minix*, etc, excepto para los sistemas marcados como *nodev* que son tipos especiales (los comentaremos en el último punto de este tema).

Una vez que hemos montado un sistema de ficheros y hemos trabajado con él, nos va a interesar desmontarlo; para ello usaremos la orden `umount`, que recibe como parámetro bien el directorio a desmontar o bien el dispositivo asociado a ese directorio. Por ejemplo, si en el directorio `/mnt/` tenemos montado el disco `/dev/hda3`, para desmontarlo teclearemos

```
rosita:~# umount /mnt/
```

o también

```
rosita:~# umount /dev/hda3
```

Hemos de recordar que para desmontar un sistema de ficheros ningún proceso ha de estar utilizando los recursos de este sistema; por ejemplo, si un usuario está trabajando en el directorio `/mnt/seguridad/`, no podremos desmontar el sistema.

Cuando arranca una máquina Unix ha de tener al menos un sistema de ficheros, el raíz, montado en el sistema. Los sistemas de ficheros que se montan automáticamente al arrancar se definen en el fichero `/etc/fstab`, utilizando una línea por sistema de ficheros. La sintaxis de cada una de estas líneas es la siguiente:

device point type options freq passno

donde *device* es el dispositivo que contiene al sistema de ficheros, *point* es el directorio a partir del cual se va a montar el sistema, *type* es el tipo del sistema de ficheros, *options* hace referencia a las opciones de montaje que ofrece `mount` (que para nosotros siempre van a ser las que utiliza por defecto), *freq* es la frecuencia de volcado (para nosotros un valor siempre a 1, lo que indica que se ha de volcar el sistema de ficheros con *dump*), y *passno* es el orden en el que se chequean los sistemas de archivos con `fsck` al arrancar la máquina (usaremos un 1 para el raíz y un 2 para el resto, excepto para las entradas especiales que comentaremos a continuación).

Un ejemplo típico de fichero `/etc/fstab` puede ser el siguiente:

```
rosita:~# cat /etc/fstab
/dev/hdb2        swap        swap        defaults   1   1
/dev/hdb1        /        ext2        defaults   1  1
/dev/hda3  /home   ext2       defaults   1   2
none     /proc    proc        defaults   1   1
rosita:~#
```

Podemos ver que existen dos entradas especiales, la correspondiente al área de *swap* y la del sistema de ficheros `proc`. El área de *swap* no necesita *mount point*, y el directorio `/proc/` no está representado en ningún dispositivo físico de la máquina, ya que es un tipo de sistema de ficheros especial utilizado por Linux (no por otros Unices) que actúa como interfaz entre las estructuras de datos del kernel y las aplicaciones de usuario.

La información contenida en `/etc/fstab` es estática. Es decir, los programas sólo leen este archivo pero no lo escriben; es tarea del administrador modificar convenientemente las entradas del fichero en función a las necesidades del sistema. Tenemos otro fichero, `/etc/mtab`, que se puede considerar la 'versión dinámica' del anterior. Cada vez que incorporemos un sistema de ficheros a la estructura de directorios, se almacenará una entrada en este archivo, entrada que se borrará al desmontar el sistema de ficheros. Un ejemplo de `/etc/mtab` puede ser el siguiente:

```
rosita:~# cat /etc/mtab
/dev/hdb1 / ext2 rw 1 1
/dev/hda3 /home ext2 rw 1 1
none /proc proc rw 1 1
/dev/hda5 /mnt vfat rw 0 0
rosita:~#
```

Vemos que la información almacenada en este fichero es muy similar a la de `/etc/fstab`, por lo que no vamos a entrar en más detalle con él. Además, repetimos que este fichero se actualiza automáticamente; el administrador no ha de modificarlo editándolo.

4.4 Gestión del espacio en disco

Los recursos de un sistema de computación son finitos. Por supuesto, entre estos recursos encontramos los dispositivos de almacenamiento (discos duros, disquetes, CD-ROM...). Como administradores, hemos de ser conscientes que para garantizar un correcto funcionamiento del sistema no podemos permitir que los usuarios llenen de información los discos. Quizás una opción interesante puede ser el establecimiento de un sistema de cuotas (permitir que cada usuario almacene como máximo una determinada cantidad de datos en su directorio).

Tengamos o no un sistema de cuotas establecido, hemos de estar atentos al porcentaje de utilización de los discos. Puede ser incluso necesario que si un usuario ocupa demasiado espacio le avisemos para que borre archivos, y si hace caso omiso nos ocupemos nosotros de hacerlo.

Mediante el mandato `df` (*Disk Free*) vamos a obtener información del uso de los sistemas de archivos (capacidad, porcentaje utilizado...). Si vemos que uno de estos sistemas está al 100% (excepto, quizás, un CDROM) de su capacidad, podemos tener problemas si no actuamos correctamente. Hemos de localizar el proceso que está llenando de archivos un disco y eliminarlo. Normalmente se tratará de un *ftp* en *background* de algún usuario que está bajando *software* al sistema. Iremos a su directorio *$HOME* y mediante la orden `du` (*Disk Usage*) veremos en que subdirectorio tiene más disco ocupado; quizás sea necesario eliminarlo con `rm -rf`. Hemos de tener cuidado; si el proceso está llenando disco duro, pero ese usuario no tiene mucho espacio ocupado, quizás el problema venga por otro usuario que, aún sin tener ningún proceso activo (o teniéndolo) ha ocupado demasiado disco. En este caso, sería más 'ético' liberar disco del usuario que más ocupe, independientemente de si ese usuario es el que ahora está llenando disco o lo ha llenado con anterioridad.

Una práctica habitual entre administradores es montar los directorios de usuarios (los que cuelgan de `/home/`) en una unidad lógica diferente de los directorios del sistema (`/`), e incluso utilizar otra unidad para `/tmp/` (ya que recordemos que los usuarios podrán escribir también en este directorio). De esta forma, evitamos que cualquier proceso de usuario pueda perjudicar al sistema llenando todo el disco duro; como máximo perjudicará al resto de usuarios, pero no al `root` (que recordemos tiene su *$HOME* en `/root/` o en `/`).

4.5 Algunos sistemas de ficheros

En el punto 3 veíamos un ejemplo típico del fichero `/proc/filesystems`:

```
rosita:~# cat /proc/filesystems
        ext2
        minix
        umsdos
        msdos
        vfat
nodev   proc
nodev   nfs
        iso9660
rosita:~#
```

En este archivo vemos los sistemas de ficheros que son aceptables en nuestra máquina (en el caso de Linux los definimos al compilar el *kernel*).

El sistema de ficheros más utilizado en Linux es *ext2*, que reemplaza al desfasado *ext*. Soporta nombres de archivos largos (hasta 255 caracteres), y presenta unos inodos con más campos en la estructura que sistemas de ficheros anteriores.

El sistema *minix* es el utilizado por el clon de Unix del mismo nombre. Soporta nombres de archivo más cortos que el *ext2* (14 o 30 caracteres), y no se suele utilizar en la práctica, sólo para acceder a discos en los que esté instalado el sistema operativo Minix (por ejemplo, para copiar archivos desde Linux).

umsdos es un sistema de archivos que nos va a permitir instalar Linux en un PC como un directorio más del sistema MS–DOS. Como todos los sistemas de ficheros Unix soporta nombres largos, aunque tampoco se suele utilizar en la práctica por la lentitud de los accesos a fichero y por

la dependencia del sistema MS–DOS.

Los sistemas *msdos* y *vfat* son los de los sistemas operativos MS-DOS y Win95. La principal diferencia entre ambos es que el segundo soporta nombres largos mientras que el primero se restringe a los ocho caracteres para el nombre y tres para la extensión.

El sistema *iso9660* es el estándar en las unidades de CD-ROM. Siempre que montemos un CD-ROM en el sistema hemos de utilizar este tipo de sistema de ficheros. Permite nombres largos.

Tenemos dos sistemas marcados como *nodev*. Esto significa que no tenemos un dispositivo (`/dev/XXX`) asociado a ellos:

(a) El primero es el tipo *proc*, que como ya hemos comentado con anterioridad es un sistema de ficheros especial que utilizan algunos Unices para formar un interfaz entre las aplicaciones de usuario y las estructuras de datos del *kernel* (por ejemplo, en `/proc/` tenemos un subdirectorio por cada proceso activo en el sistema, de nombre el *PID* del proceso, y con información sobre el estado, memoria, etc.).

(b) El segundo es *nfs* (*Network File System*), un sistema de ficheros remoto desarrollado por Sun Microsystems e implantado en la mayoría de Unices del mercado que nos va a permitir incorporar dispositivos de almacenamiento de otros sistemas en el nuestro. De esta forma, si el directorio `/mnt/` contiene un sistema *nfs*, al cambiar a este directorio estaremos automáticamente trabajando sobre un disco que puede estar en una máquina a pocos metros de la nuestra o a miles de kilómetros (y la única diferencia que notaremos respecto a una unidad local va a ser la velocidad).

Capítulo 5

Tareas. Mantenimiento del sistema

5.1 Gestión de usuarios

Como administradores, una de nuestras funciones más habituales, y quizás la primera tarea a la que nos hemos de enfrentar, va a ser **añadir usuarios** al sistema. Obviamente, otra tarea importante y rutinaria para nosotros (aunque no tanto como la anterior) será **eliminar usuarios** que ya no necesiten recursos de cómputo de la máquina.

Para añadir usuarios tenemos disponible desde la línea de órdenes el mandato `adduser`; aunque en todos los sistemas Unix modernos existen herramientas mucho más amigables para facilitar la gestión de usuarios, no las vamos a contemplar aquí, ya que `adduser` es la orden común (aunque en algunos sistemas cambie su nombre) a todos los entornos Unix. Si ejecutamos esta orden nos va a preguntar, campo a campo, toda la estructura de datos que define a un usuario; en cualquier momento de la ejecución de la instrucción podremos interrumpir el proceso tecleando *Ctrl-C*:

```
luisa:~# adduser
Login name for new user []: avh
User ID ('UID') [ defaults to next available ]:
Initial group [ users ]:
Additional groups (comma separated) []:
Home directory [ /home/avh ]
Shell [ /bin/bash ]
Expiry date (YYYY-MM-DD) []:
New account will be created as follows:
---------------------------------------
Login name.......:  avh
UID..............:  [ Next available ]
Initial group....:  users
Additional groups:  [ None ]
Home directory...:  /home/avh
Shell............:  /bin/bash
Expiry date......:  [ Never ]

This is it... if you want to bail out, hit Control-C.  Otherwise, press
ENTER to go ahead and make the account.

Creating new account...

Changing the user information for avh
Enter the new value, or press ENTER for the default
```

```
        Full Name []: Antonio Villalon
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Changing password for avh
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
Account setup complete.
luisa:~#
```

Como podemos ver, los principales datos a introducir para cada nuevo usuario son los siguientes, coincidentes con la línea correspondiente en el fichero `/etc/passwd`:

- *Login name*: Nombre de conexión al sistema (*'login'*).
- *UID*: Identificador de usuario, el primero sin utilizar en el sistema (la herramienta lo determinará automáticamente).
- *GID*: Identificador del grupo al inicialmente que va a pertenecer el usuario.
- *Full Name*: Descripción del usuario (por lo general el nombre) para situarlo en el campo correspondiente de su entrada del archivo de contraseñas.
- *Home Directory*: Directorio al cual va a acceder ese usuario por defecto al conectar a la máquina (*$HOME*).
- *Shell*: Intérprete de órdenes a utilizar. Por defecto, bajo sistemas gratuitos (Linux, FreeBSD, etc.) este intérprete es `bash`, y bajo sistemas comerciales es `sh` o `ksh`.
- *Password*: Contraseña de acceso al sistema; por razones de seguridad es conveniente que el propio usuario teclee su clave, para así garantizar que nadie más lo conozca (esto significa que debería estar presente en el momento de darlo de alta).

Tras confirmar todos los datos, tanto `adduser` como cualquier otra herramienta que estemos utilizando - todas deben seguir un proceso común -, procederá a dar de alta al usuario introducido; para ello, en primer lugar se creará la entrada adecuada en el archivo `/etc/passwd`, donde se definen la mayor parte de los campos especificados con anterioridad, correspondiendo cada línea a un usuario diferente:

```
luisa:~# grep avh /etc/passwd
avh:x:1001:100:Antonio Villalon,,,:/home/avh:/bin/bash
luisa:~#
```

En los sistemas con *Shadow Password* (la mayoría de entornos Unix en la actualidad), se creará además una línea por cada nuevo usuario en el fichero `/etc/shadow`, en la que se almacenará la clave del usuario cifrada y diferentes parámetros de envejecimiento de contraseñas, comentados en el capítulo dedicado a la seguridad del sistema:

```
luisa:~# grep avh /etc/shadow
avh:$1$ONa/CovQ$zmOVE6Wk1UPDLeZR.K0E6/:12903:0:99999:7:::
luisa:~#
```

Una vez definidas las entradas anteriores, gracias a las cuales el nuevo usuario ya podría conectar al sistema; no obstante, cualquier herramienta de gestión de usuarios, entre las cuales por supuesto está `adduser`, se encargará de crear el directorio *$HOME* indicado para el nuevo usuario y asignarle a éste

la propiedad, de forma que en dicho directorio el usuario pueda almacenar su propia información. Además, copiará en este directorio *$HOME* los ficheros que se encuentren en `/etc/skel/`, donde como administradores podemos definir los archivos que queramos adjudicar de forma automática a cada uno de nuestros nuevos usuarios (típicamente, ficheros de configuración, como un `.profile` genérico). Una vez completado este proceso, el usuario está ya en disposición de iniciar su sesión de la forma habitual:

```
Welcome to Linux 2.4.29 (tty4)

luisa login: avh
Password:
Linux 2.4.29.
No mail.

luisa:~$
```

Para eliminar usuarios, en muchos Unices tenemos también una serie de herramientas que nos van a facilitar esta tarea, como `userdel`; en cualquier caso, utilicemos la herramienta que utilicemos, todas van a seguir un proceso similar: borrarán la línea correspondiente al usuario en `/etc/passwd` (y en `/etc/shadow`), y a continuación borrarán su directorio *$HOME* para liberar disco duro, si así lo hemos determinado; además, es posible que eliminen sus tareas programadas, y algo recomendable pero que no suelen realizar es buscar ficheros pertenecientes al usuario que estamos eliminando (en todos los *filesystems*) y borrarlos. Si el usuario es el último de un determinado grupo también puede ser aconsejable eliminar la entrada de este grupo en `/etc/group` y el directorio padre de los *$HOME* correspondientes a los usuarios del grupo (ya que será un directorio vacío si no lo compartían con otros grupos); estas últimas tareas no suele ser recomendable realizarlas mediante herramientas, sino que es preferible hacerlo de forma manual.

También es posible que no nos interese eliminar un usuario, sino únicamente bloquear su acceso de forma temporal (impedir que se conecte, por ejemplo, hasta que venga en persona a hablar con nosotros); esto lo logramos mediante la orden `'passwd -l'` en casi cualquier sistema Unix, o con `'chuser'` acompañado de la opción correspondiente en AIX; no obstante, también podemos hacerlo introduciendo un asterisco (∗) como primer carácter en la contraseña cifrada del usuario en `/etc/passwd` (en el caso de sistemas con *shadow password*, en `/etc/shadow`), bien editando el archivo con cualquier editor o, por seguridad, con la orden `vipw`:

```
toni:*Im5n0PXzz1oP.:501:100:Toni, , , :/home/toni:/bin/bash
```

Realmente, si en lugar del símbolo `'*'` utilizáramos cualquier otro carácter, en la práctica el usuario estaría también bloqueado; lo que caracteriza al asterisco es que se trata de un símbolo que **nunca** se generará en una clave cifrada, por lo que utilizándolo podemos determinar fácilmente qué usuarios están bloqueados en nuestro sistema.

Como hemos visto, cada usuario de un sistema Unix pertenece al menos a un grupo, el definido por su GID en el fichero de contraseñas del sistema; de forma adicional, como administradores podemos añadir al usuario a cuantos grupos necesitemos, cuya relación podemos encontrar en el archivo `/etc/group`. En cada línea de este fichero se define un grupo de usuarios, indicando el nombre, la contraseña del grupo (sólo utilizada en casos muy concretos), el GID asignado (número que, como sabemos, distingue entre sí a los diferentes grupos de usuarios dentro de un sistema) y, finalmente, los usuarios que pertenecen al grupo:

```
luisa:~# tail /etc/group
sshd::33:sshd
gdm::42:
shadow::43:
ftp::50:
```

```
pop::90:pop
nobody::98:nobody
nogroup::99:
users::100:
console::101:
desa::102:toni,root,avh
luisa:~#
```

Para definir nuevos grupos en el sistema, únicamente hemos de añadir la línea correspondiente al fichero anterior; para editar los archivos relacionados con la gestión de grupos, en algunos sistemas Unix disponemos de la orden `vigr`, que bloquea los ficheros en cuestión para evitar problemas. Equivalentemente, podemos utilizar cualquier entorno de gestión de usuarios disponible en los sistemas Unix más modernos o, en cualquier caso, órdenes como `groupadd`:

```
luisa:~# groupadd users2
luisa:~# tail /etc/group
gdm::42:
shadow::43:
ftp::50:
pop::90:pop
nobody::98:nobody
nogroup::99:
users::100:
console::101:
desa:x:102:toni,root,avh
users2:x:103:
luisa:~#
```

Aunque depende del sistema Unix con el que estemos trabajando, lo más habitual es que el nuevo grupo definido no posea contraseña, con lo que todos los usuarios que pertenezcan al mismo podrán acceder a él de forma automática y sin ningún conocimiento de claves especial. En cualquier caso, podemos asignar una clave a cada grupo mediante la orden `passwd`:

```
luisa:~# passwd -g users2
Changing the password for group users2
New Password:
Re-enter new password:
luisa:~#
```

Al ejecutar la orden anterior, en los sistemas Unix más antiguos se almacenará la clave cifrada en el segundo campo de `/etc/group` y, en los sistemas actuales (con mecanismos de *Shadow Password*), se añadirá al fichero `/etc/gshadow`, lugar en el cual se almacena en realidad la clave.

En cada momento, un usuario - a pesar de poder pertenecer a varios grupos -, tiene un único GID efectivo, con el que por defecto se crean archivos y directorios; el propio usuario puede modificar este parámetro en la sesión actual mediante la orden `newgrp`:

```
luisa:~$ id
uid=1000(toni) gid=100(users) groups=100(users),102(desa)
luisa:~$ touch hola
luisa:~$ ls -l hola
-rw-r--r--  1 toni users 0 2005-04-30 11:38 hola
luisa:~$ newgrp desa
Password:
luisa:~$ touch adios
luisa:~$ ls -l adios
```

```
-rw-r--r--  1 toni desa 0 2005-04-30 11:38 adios
luisa:~$ id
uid=1000(toni) gid=102(desa) groups=100(users),102(desa)
luisa:~$
```

Finalmente, para eliminar un grupo de usuarios definido dentro de un sistema Unix, podemos simplemente borrar la línea correspondiente del archivo `/etc/group` (y, si corresponde, de `/etc/gshadow`), a ser posible mediante órdenes como `vigr` para bloquear los ficheros en cuestión (frente al uso de un editor de textos convencional); de la misma forma, podemos utilizar herramientas como `groupdel`:

```
luisa:~# groupdel users2
luisa:~# tail /etc/group
sshd::33:sshd
gdm::42:
shadow::43:
ftp::50:
pop::90:pop
nobody::98:nobody
nogroup::99:
users::100:
console::101:
desa::102:toni,root,avh
luisa:~#
```

Hemos de tener presente que, igual que sucede al eliminar un usuario concreto del sistema, al borrar un grupo no se eliminan automáticamente los ficheros que pertenecen a ese grupo; podemos localizar todos estos archivos - quizás para plantearnos su borrado, o al menos su análisis para cambiar su grupo propietario - mediante una orden como `find`, teniendo en cuenta que si ya hemos eliminado el grupo en cuestión, habremos de buscar por GID y no por nombre de grupo, ya que este no existirá:

```
luisa:~# find / -group users2
find: invalid argument ‘users2’ to ‘-group’
luisa:~# find / -group 103
/tmp/prueba
/home/toni/adios
luisa:~#
```

5.2 Automatización de tareas

El programa `crond` (o simplemente `cron`, dependiendo del clon de Unix utilizado) es un demonio que va a permitir a los usuarios, y evidentemente también al administrador, ejecutar órdenes a intervalos periódicos o en un determinado momento. Este demonio se ha de invocar al arrancar la máquina una única vez: no han de existir varias copias del programa en memoria simultáneamente o el sistema puede desordenarse.
`crond` despierta una vez cada minuto y lee la información de los ficheros contenidos en `/var/spool/cron/crontabs/`. Estos ficheros han sido creados bien por los usuarios o por el administrador utilizando la orden `crontab` con las opciones adecuadas (que veremos más adelante). Sólo puede existir un fichero `crontab` por usuario, que estará almacenado en el directorio anterior, con nombre el *login* del creador.

Cada línea de un archivo `crontab` ha de contener un patrón de tiempo y una orden determinada, con un formato similar a:

c1 c2 c3 c4 c5 orden

donde *c1..c5* son los cinco campos reservados para indicar el momento de la ejecución y 'orden' es el nombre del mandato a ejecutar. Cada campo puede ser un número entero dentro del rango permitido, un asterisco (indicando que todos los valores son legítimos), o una lista de números enteros separados por comas o guión. El significado de estos campos es el siguiente:

Indicador	Campo	Rango
c1	Minutos	0-59
c2	Horas	0-23
c3	Día mes	1-31
c4	Mes	1-12
c5	Día semana	0-6 (0 corresponde al domingo)

Así, un ejemplo de archivo *crontab* puede ser el mostrado a continuación:

```
30 5 * * 1 touch /etc/passwd
* * * * * /home/toni/programacion/servercheck
```

La primera línea indica que la orden se ha de ejecutar cada lunes a las 5:30 horas, y la segunda indica que `servercheck` se ha de ejecutar cada vez que `crond` despierte (esto es, cada minuto).

Para crear un fichero `crontab` cualquier usuario puede editar un archivo con el formato anterior y luego añadirlo al directorio `/var/spool/cron/crontabs/` utilizando `crontab fichero`, o bien editarlo y añadirlo de forma automática, con la orden `crontab -e`. Para eliminar este archivo se usará `crontab -d`. Como administradores, podemos manejar los ficheros de planificación de los usuarios utilizando `-u <usuario>`. Por ejemplo, si queremos borrar la planificación del usuario *jperez*, lo haremos con la orden `crontab -d -u jperez`.

Hasta ahora hemos hablado de la facilidad ofrecida por el sistema Unix para realizar tareas a intervalos periódicos (automatización de tareas). Ahora vamos a ver las posibilidades que tenemos de planificar trabajos para que se ejecuten en un determinado momento, sin necesidad de estar presentes en la máquina a esa hora. Por ejemplo, podemos planificar copias de seguridad para que se realicen a altas horas de la noche, cuando la actividad de los usuarios es escasa. Esto lo vamos a conseguir con la orden `at` y su derivado `batch`; la diferencia entre ambas es que mientras que con `at` especificamos cuando ejecutar ciertos mandatos, con `batch` simplemente lo *sugerimos*, pero no se ejecutarán hasta que el nivel de carga de la CPU sea inferior a 1.5. Estas órdenes están disponibles para la totalidad de usuarios del sistema, pero podemos denegar su uso a ciertos usuarios individuales: en el archivo `/etc/at.deny` (por defecto vacío) especificaremos el *login* (uno por línea) de los usuarios a los que no les esté permitido utilizar las facilidades `at`. Si el número de estos usuarios es muy elevado, quizás nos interese más crear el archivo `/etc/at.allow`, con el mismo formato que el anterior, donde indicamos los usuarios a los que les está permitido el uso de esta facilidad (el resto de usuarios no tendrá acceso a ella). Si ninguno de estos dos ficheros existe, sólo el `root` puede utilizar `at`.

En el fichero `crontab` del administrador ha de existir una línea encargada de ejecutar la orden `atrun`. Este programa es el encargado de ejecutar los trabajos encolados con `at`. Al ejecutar un trabajo, el resultado (tanto si ha sido correcto como si ha existido un error) se enviará por correo al usuario que lo lanzó.

La sintaxis de `at` o de `batch` (es análoga) puede llegar a ser muy compleja; la forma más habitual de ejecutar el programa es

at/batch TIME -f archivo -q cola

donde el parámetro *archivo* no es más que un fichero de texto donde, línea a línea, hemos indicado los mandatos que queremos ejecutar. *cola* es la cola de trabajos donde queremos situar la petición,

denominada con una letra minúscula o mayúscula (*a... z, A... Z*); por defecto, es *c* para las lanzadas con `at` y *E* para las lanzadas con `batch`.

El parámetro *TIME* es el momento en que queremos ejecutar los mandatos contenidos en *archivo*. Se aceptan tiempos de la forma *HH:MM* o *HHMM*; si a continuación no se especifica *am* o *pm*, se asume un formato de 24 horas. Para especificar un día distinto al actual hemos de indicar la fecha de la forma *DD.MM.YY, DDMMYY* o *DD/MM/YY*. También se aceptan las palabras reservadas *now, noon, midnight* y *teatime* (respectivamente, ahora, mediodía, medianoche y hora del té, 16:00). Podremos indicar también un incremento de tiempos de la forma *+n unidades*, donde *unidades* puede ser *minutes, hours, days* o *weeks*.

Por último, hemos de saber que para visualizar las colas de trabajos utilizaremos la orden `atq`, y para eliminar alguno de estos trabajos usaremos *atrm*. Veamos unos ejemplos que aclaren un poco la sintaxis de la familia de órdenes `at`:

```
rosita:~# at 19:10 -f fichero2
Job 19 will be executed using /bin/sh
rosita:~# batch 19:11 -f fichero2
Job 20 will be executed using /bin/sh
rosita:~# atq
Date Owner Queue Job#
19:10:00 11/23/97 toni c 19
19:11:00 11/23/97 toni E 20
rosita:~# atrm 19
rosita:~# atq
Date Owner Queue Job#
19:11:00 11/23/97 toni E 20
rosita:~# atrm 20
rosita:~# at now +10 minutes -f fichero2
Job 24 will be executed using /bin/sh
rosita:~# at teatime -f fichero2
Job 25 will be executed using /bin/sh
rosita:~# at noon +10 minutes -f fichero2
Job 26 will be executed using /bin/sh
rosita:~# at now +15 minutes -f fichero2
Job 27 will be executed using /bin/sh
rosita:~# atq
Date Owner Queue Job#
20:10:00 11/23/97 toni c 24
16:00:00 11/24/97 toni c 25
12:10:00 11/24/97 toni c 26
20:19:00 11/23/97 toni c 27
rosita:~# atrm 24 25 26 27
rosita:~# at midnight -f fichero2
Job 30 will be executed using /bin/sh
rosita:~# at midnight +3 minutes -f fichero2
Job 33 will be executed using /bin/sh
rosita:~# batch midnight +2 days -f fichero2
Job 34 will be executed using /bin/sh
rosita:~# at now +45 week -f fichero2
Job 35 will be executed using /bin/sh
rosita:~# atq
Date Owner Queue Job#
00:00:00 11/24/97 toni c 30
00:00:00 11/26/97 toni c 31
```

```
00:03:00 11/24/97 toni c 32
00:03:00 11/24/97 toni c 33
00:00:00 11/26/97 toni E 34
20:08:00 10/04/98 toni c 35
rosita:~#
```

5.3 Actualización e instalación de paquetes software

Como administradores tenemos que mantener actualizado el software de nuestro sistema, tanto a nivel de *kernel* como de aplicaciones. No vamos a entrar en cómo reconfigurar el núcleo del sistema operativo, ya que esta tarea se realiza de forma diferente en cada clon de Unix. Aquí vamos a estudiar cómo instalar aplicaciones software para nuestro sistema. Aunque existen multitud de programas destinados a facilitar esta tarea al administrador (en el caso de Linux Slackware tenemos `installpkg` y `removepkg`), son en su mayoría propios de cada clon, por lo que estudiaremos técnicas comunes a todos los Unices.

Las aplicaciones suelen estar disponibles con un formato *.tgz* o *.tar.gz* (es análogo). *TAR* es un empaquetador (enlaza ficheros y subdirectorios en un solo archivo), mientras que *GZIP* es un compresor (reduce el tamaño del paquete *tar*). Para poder ver todos los ficheros que contiene un determinado paquete primero lo hemos de descomprimir con `gzip -d` o `gunzip`. Lo que era un fichero comprimido ahora simplemente está empaquetado, y su extensión ha cambiado a *.tar*. El siguiente paso será desempaquetar el fichero, utilizando la orden `tar` con las opciones adecuadas: `xvf` (no vamos a estudiar aquí las múltiples opciones del mandato `tar`, las veremos en el capítulo destinado a copias de seguridad). Se nos mostrarán en pantalla los ficheros y subdirectorios que crea `tar`. Todo esto lo podemos ver en un ejemplo:

```
rosita:~# ls
paquete.tgz
rosita:~# gzip -d paquete.tgz
rosita:~# ls
paquete.tar
rosita:~# tar xvf paquete.tar
Install.sh
Makefile
docs/
docs/LEEME
docs/LEEME.2
docs/programa.tex
docs/documento.ps
src/
src/archivo.c
src/main.c
src/main.h
src/prog1.c
src/prog2.c
rosita:~#
```

En determinadas versiones de `tar` (concretamente en las de GNU, las distribuidas por defecto en Linux) no es necesario descomprimir y desempaquetar por separado; podemos utilizar la opción `z` de `tar` para hacer ambas cosas a la vez:

```
rosita:~# tar xvzf paquete.tgz
```

Una vez hemos descomprimido y desempaquetado el archivo inicial del nuevo software, hemos de configurar el programa. Aunque de algunos programas sólo se distribuyen los ficheros binarios

preparados para un determinado clon, generalmente se nos ha ofrecido el código fuente y debemos adecuarlo a nuestra plataforma (HP-UX, Solaris, Linux...) y a nuestra jerarquía de directorios (por ejemplo para instalarlo en un determinado directorio que no sea el ofrecido por defecto). Para esto vamos a disponer de un fichero denominado `Makefile` o `makefile`, donde se especifican todos los parámetros de instalación: desde las opciones pasadas al compilador hasta los directorios donde se va a instalar la aplicación. Este fichero será utilizado por la orden `make`, que invocaremos (quizás con algún argumento indicado en un fichero `README` o `INSTALL`, por norma general) desde la línea de órdenes.

El formato habitual de estos ficheros `makefile` es el siguiente:

```
# Personalizacion de la instalacion (directorios, variables de entorno,
# definicion de variables de entorno...)
# Opciones de compilacion
fichero: dependencias
modo de compilacion
dependencia1: dependencias
modo de compilacion
```

En la personalización de la instalación hemos de indicar, por norma general, el directorio donde queremos instalar el paquete, e incluso la localización de ciertas herramientas (compiladores, librerías, montadores...). Hemos de procurar no instalar nada en lugares como `/bin/` o `/usr/bin/`, aunque algunos paquetes obligan a situar ficheros en estos directorios. Una buena idea sería hacerlo en `/usr/local/bin/`, creando un subdirectorio diferente por aplicación, y también otro subdirectorio desde el cual enlazaremos simbólicamente los ejecutables de las aplicaciones instaladas (por ejemplo, `/usr/local/bin/varios`), y que añadiremos al *$PATH* de los usuarios, evitando de esta forma tener un *$PATH* excesivamente largo que quizás dificultaría la localización de programas.

En el apartado reservado a opciones de compilación generalmente no vamos a realizar ninguna modificación, ya que de eso ya se ha encargado el programador de ese paquete, o incluso nosotros mismos modificando variables de entorno en el mismo fichero `makefile` (por ejemplo, variables que se suelen denominar *LIBS, CC, ARCH...*). Aquí se indica la forma de compilar un determinado programa del paquete, chequeando sus dependencias (ficheros *.h* o *.o* por norma general), y también lo que la orden `make` ha de realizar cuando reciba un determinado parámetro (como `all, config, clean`...).

Un ejemplo típico de fichero `makefile` es el siguiente:

```
# Definimos algunas variables de entorno para la correcta ejecución de
# make
SDIR = ..
BDIR = $(SDIR)/../bin
OPTIONS = -O
CFLAGS = -g
LIBS = -lm
CPROGS = anillo addmaster hello
# Comenzamos con las opciones de compilación
# Si ejecutamos make all desde el prompt:
all:  $(CPROGS)
# Si ejecutamos make clean:
clean:
      rm -f *.o $(CPROGS) $(FPROGS)
# Opciones de compilación de los programas
anillo:  $(SDIR)/anillo.c
         $(CC) $(CFLAGS) -o $@ $(SDIR)/anillo.c $(LIBS)
```

```
addmaster:  $(SDIR)/addmaster.c
            $(CC) $(CFLAGS) -o $@ $(SDIR)/addmaster.c $(LIBS)
hello:  $(SDIR)/hello.c
        $(CC) $(CFLAGS) -o $@ $(SDIR)/hello.c $(LIBS)
```

5.4 Copias de seguridad

Existen varios tipos de problemas que pueden resultar en la perdida de datos: borrado accidental de archivos, fallos del hardware, información importante almacenada en los archivos que deja de estar disponible... En estos casos, los usuarios necesitan tener la confianza de poder acceder a una copia de seguridad de los archivos 'perdidos'.

El futuro de nuestro sistema puede depender de que disponga de una copia de seguridad de esos archivos. En esos momentos, tanto el usuario como sus colegas estarán agradecidos que se haya dedicado el tiempo y el esfuerzo necesario para copiar archivos en algún tipo de medio de almacenamiento de acuerdo con un programa regular, riguroso y bien documentado. La copia de archivos no es una actividad muy atractiva, pero ningún administrador puede ignorar el proceso.

A continuación se relacionan varios temas a considerar en la realización de copias de seguridad de un sistema:

- Copias de seguridad completas o progresivas.
  Una copia de seguridad completa copia cada uno de los archivos. ¿Es necesario realizarla a diario? Una copia de seguridad completa normalmente requiere una gran cantidad de tiempo y suficientes medios de almacenamiento para guardar todos los archivos del sistema. Una copia de seguridad progresiva copia sólo los archivos que han cambiado desde la ultima copia de seguridad.

- Sistemas de archivo a salvaguardar. Como es natural, deben realizarse copias de seguridad de los sistemas de archivo activos de forma regular. Otros archivos pueden copiarse con menor frecuencia. Asegúrese de tener copias actualizadas de todos los sistemas de archivos.

- Tipos de medios para la realización de copias de seguridad. Según los dispositivos que tenga disponibles en el sistema, podrá utilizar cinta de 9 pistas, cartuchos de cinta de 1/4 de pulgada, cintas DAT de 4 u 8 mm o disquetes. Cada uno de estos sistemas ofrece ventajas con respecto a los otros en términos de espacio ocupado, capacidad de almacenamiento y coste de los dispositivos y de los medios de copia. Seleccione un medio de realización de copias que se ajuste a su presupuesto, teniendo en cuenta siempre que el medio menos caro puede ser también el que requiera una mayor inversión de tiempo.

- Efecto de las copias de seguridad en los usuarios. La realización de una operación de copia de seguridad aumenta la carga del sistema. ¿Constituirá esto una carga irrazonable para los usuarios? Y además, probablemente no se harían copias de seguridad de los archivos modificados durante el proceso de realización de la copia de seguridad, lo cual puede representar un simple inconveniente o una cuestión de importancia si está realizando una copia de seguridad de una base de datos activa. ¿Es preferible realizar las copias de seguridad cuando el sistema se encuentra inactivo?

- Órdenes que se utilizan para las copias de seguridad. Existen algunas instrucciones relativamente simples y ampliamente utilizados para crear copias de seguridad como, por ejemplo, `tar` y `cpio`. ¿Bastará con estas órdenes?

- Documentación de los archivos copiados. Es importante etiquetar todo el material copiado de forma que pueda utilizarse para recuperar archivos cuando sea necesario. Algunos procedimientos y mandatos permiten preparar una tabla de contenidos o una lista del material del que se haya realizado una copia de seguridad.

Desde el punto de vista del administrador, la copia de seguridad del sistema de archivos debe realizarse de acuerdo con algún tipo de proceso automatizado, con la mínima intervención posible por parte del operador. Debería llevarse a cabo cuando el sistema se encuentra relativamente tranquilo, a fin de que la copia de seguridad sea lo más completa posible. Esta cuestión debe sopesarse contra las de conveniencia y coste. ¿Tendría que quedarse un operador o administrador hasta la medianoche del viernes para realizar una copia de seguridad completa? Vale la pena gastarse una buena cantidad de dinero en una unidad de cinta DAT para que se pueda hacer una copia de seguridad automática de todo el sistema a las 3 de la mañana sin la intervención de ningún operador?

Considere las alternativas, determine los costes reales y tome una decisión o recomiende una acción a adoptar. Por regla general, resulta mucho más económico y siempre más fácil restaurar información bien gestionada en copias de seguridad que tener que recrearla o pasar sin ella.
En este apartado se tratarán los puntos siguientes:

- Planificación de las copias de seguridad.
- Preparación de un programa de copias de seguridad.
- Utilización de la orden `tar`.
- Cómo trabajar con cpio.

5.4.1 Consejos a considerar al realizar copias de seguridad

El propósito de la realización de copias de seguridad es poder restaurar archivos individuales o sistemas de archivos completos. Todo lo que haga en relación con este tema deberá centrarse en ese propósito.

Prepare un programa de copias de seguridad que detalle los archivos a salvaguardar, la frecuencia con que se van a realizar las copias de seguridad y la forma en que se van a restaurar los archivos. Informe a todos los usuarios sobre este programa y la forma en que pueden solicitar la restauración de archivos. Aténgase estrictamente al plan.

Verifique siempre las copias de seguridad. La comprobación puede consistir, por ejemplo, en la lectura de una tabla de contenidos del medio utilizado para realizar la copia de seguridad, después de haberlo almacenado o en la restauración de un archivo seleccionado al azar. Recuerde que cabe la posibilidad de que el medio de almacenamiento de copias de seguridad disco o cinta este defectuoso.

Haga las copias de seguridad de forma que los archivos puedan restaurarse en cualquier lugar del sistema de archivos o en otro sistema informático. Use utilidades de copia de seguridad que creen archivos que puedan utilizarse en otros sistemas informáticos Linux o Unix.

No se olvide de etiquetar todos los medios (cintas, discos, etc.) que utilice para realizar las copias de seguridad. Si un mismo proceso requiere varios medios, asegúrese de que son numerados secuencialmente y fechados. De esta forma, podrá localizar fácilmente el archivo o archivos que necesite.
Planifique pensando en lo peor. Tenga copias de los archivos del sistema de forma que este pueda restaurarse en un plazo razonable de tiempo. Almacene las cintas o discos conteniendo las copias de seguridad en un lugar diferente de donde se encuentra el sistema. Planifique la revalidación periódica de sus procedimientos de copia de seguridad a fin de asegurarse de que satisfacen sus necesidades.

5.4.2 Planificación de un programa de copias de seguridad

Es importante preparar un programa de copias de seguridad que satisfaga sus necesidades y que permita restaurar copias recientes de archivos. Una vez haya decidido su programa, procure ajus-

tarse a él.

La situación ideal sería poder restaurar cualquier archivo en cualquier momento. Desgraciadamente, eso no es posible, aunque debería poder restaurar archivos diariamente. Para ello, utilice una combinación de copias de seguridad completas y progresivas. Una copia de seguridad completa es aquella que contiene cada uno de los archivos del sistema. Una copia de seguridad progresiva es aquella que contiene los archivos que han sufrido modificaciones desde la realización de la ultima copia de seguridad.

Las copias de seguridad progresivas pueden realizarse a niveles diferentes: progresivas con respecto a la ultima copia de seguridad completa o progresivas con respecto a la ultima copia de seguridad progresiva. Es conveniente pensar que las copias de seguridad tienen lugar a niveles diferentes:

Nivel 0: Copia de seguridad completa
Nivel 1: Progresiva con respecto a la última copia de seguridad completa
Nivel 2: Progresiva con respecto a la ultima copia de seguridad progresiva

A continuación se indican algunos programas de copias de seguridad a título de ejemplo:

- Un día copia de seguridad completa, los restantes días copia de seguridad progresiva:

  Día 1 Nivel 0, copia de seguridad completa
  Día 2 Nivel 1, copia de seguridad progresiva
  Día 3 Nivel 1, copia de seguridad progresiva
  Día 4 Nivel 1, copia de seguridad progresiva
  Día 5 Nivel 1, copia de seguridad progresiva

  Si crea y guarda un índice de cada una de estas copias de seguridad, sólo necesitará la copia de seguridad de un día para restaurar un archivo individual y sólo las copias de seguridad correspondientes a dos días (la del día 1 y la de otro día) para restaurar el sistema completo.

- Copia de seguridad completa una vez al mes, semanal progresiva y diario progresiva. Este ejemplo esta preparado partiendo de un martes, pero podría aplicarse igualmente a cualquier día de la semana.

  Primer martes Nivel 0, copia de seguridad completa
  Cualquier otro martes Nivel 1, copia de seguridad progresiva
  Cualquier otro día Nivel 2, copia de seguridad progresiva

  Para restaurar un archivo individual con este programa, necesitará la copia de seguridad completa si el archivo no ha sufrido cambios durante el mes, la copia de seguridad de nivel 2 si el archivo ha sufrido cambios durante la semana anterior pero no durante esta, o la copia de seguridad de nivel 1 si el archivo ha sufrido cambios esta semana. Este programa es más complejo que el del ejemplo anterior, pero las copias de seguridad tardan menos tiempo cuando se realizan a diario.

5.4.3 Realización de copias de seguridad y restauración de archivos

Existen varias utilidades diferentes disponibles para la realización de copias de seguridad y la restauración de archivos en un sistema Linux. Algunas son sencillas y directas; otras son más complejas. Los métodos sencillos tienen sus limitaciones. Seleccione el que mejor se ajuste a sus necesidades.

Dada la importancia de la realización de copias de seguridad y la restauración de archivos, existen

bastantes sistemas de software dedicados a dicha tarea. Las secciones siguientes presentan dos de ellos:

tar

Utilidad de archivo en cinta disponible en todos los sistemas Linux o UNIX; esta versión Linux de fácil manejo puede utilizar varias cintas o discos.

Utilización de `tar`

La utilidad UNIX `tar` fue diseñada originalmente para crear un archivo de cinta (para copiar archivos o directorios a cinta y después extraer o restaurar archivos del contenedor). Puede utilizarse para copiar a cualquier dispositivo. Ofrece las siguientes ventajas:

- Es sencillo de utilizar.
- Es fiable y estable.
- Los archivos pueden leerse virtualmente en cualquier sistema Linux o UNIX.

También tiene algunas desventajas, como se describe a continuación:

- Para algunas versiones de tar, el archivo debe residir en un disco o cinta, lo que quiere decir que si falla una porción del medio (por ejemplo, a causa de un sector en malas condiciones de un disco o de un bloque defectuoso de una cinta) podría perderse toda la copia de seguridad.
- De por sí, sólo puede realizar copias de seguridad completas. Tendrá que hacer alguna programación de *shell* si desea realizar copias de seguridad progresivas.

En la siguiente tabla se relacionan algunas de las opciones de `tar` más comunes. Hay muchos parámetros adicionales del mandato que pueden utilizarse con `tar`. Consulte la página de la orden `man` para una lista completa.

- c
  Crea un contenedor
- x
  Extrae o restaura archivos desde el contenedor que se encuentra en el dispositivo predeterminado o en el dispositivo especificado en la opción `f`.
- r
  Añade archivos al final del contenedor.
- f nombre
  Especifica el contenedor o lee el contenedor desde *nombre*, donde *nombre* es un nombre de archivo o un dispositivo especificado en `/dev/`, como por ejemplo `/dev/rmt0`.
- Z
  Comprime o descomprime el contenedor `tar` usando `compress`.
- z
  Comprime o descomprime el contenedor `tar` usando `gzip`.
- M
  Crea una copia de seguridad `tar` de volumen múltiple.
- X
  Excluye los archivos y directorios especificados en el fichero nombre.

- p
  Conserva los permisos especiales de los archivos.

- t
  Crea un índice de todos los archivos almacenados en un contenedor y lista en la salida estándar, *stdout*.

- v
  Modalidad detallada. Ofrece más información sobre la operación en curso.

Veamos algunos ejemplos del uso de `tar` en la realización de copias de seguridad y restauración de archivos. La orden siguiente copia el directorio `/home/` a la unidad de disquetes `/dev/fd0`:

```
rosita:~# tar cf /dev/fd0 /home
```

En este caso, la opción `'f'` especifica que el archivo se crea en el dispositivo de unidad de disquete `/dev/fd0`. El mandato siguiente también archiva el directorio `/home/`:

```
rosita:~# tar cvMf /dev/fd0 /home | tee homeindex
```

La opción `'v'` indica la modalidad detallada, la opción `'z'` indica que debería comprimirse el archivo para ahorrar espacio, y la opción `'M'` pide a `tar` que cree una copia de seguridad de volumen múltiple. Cuando un disquete está completo, `tar` le indica que inserte otro. Se envía una lista de los archivos copiados a `homeindex`. Es una buena idea examinar ese archivo para ver que se ha copiado.

La orden `find` es útil para localizar archivos que se han modificado dentro de un cierto periodo de tiempo de forma que se puedan programar para la realización de copias de seguridad progresivas. El ejemplo siguiente utiliza el mandato `find` para crear una lista de todos los archivos modificados el día anterior:

```
rosita:~# find /home -mtime -1 -type f -print > bkuplst
rosita:~# tar cvMf /dev/fd0 `cat bkuplst` | tee homeindex
```

Para utilizar la lista como una introducción a la instrucción `tar`, coloque la orden `cat bkuplst` entre comillas inversas (comillas sencillas al revés: `` ` cat bkuplst` ``). Esto dice al *shell* que ejecute la instrucción como un *subshell* y coloque la salida de la misma en la línea de órdenes en la localización de la posición del mandato original situado entre comillas inversas.

La instrucción siguiente restaura el archivo `/home/dave/notes.txt` desde el dispositivo `/dev/fd0` (recuerde que tiene que proporcionar el nombre de archivo completo para restaurarlo):

```
rosita:~# tar xvf /dev/fd0 /home/dave/notes.txt
```

También puede utilizarse la orden `tar` para crear archivos de contenedor en el sistema de archivos Linux, en lugar de escribir a un dispositivo de copias de seguridad. De esta forma podrá guardar un grupo de archivos junto con su estructura de directorio en un mismo archivo. Para esto, deberá dar un nombre de archivo a la opción `'f'` como argumento, en lugar de un nombre de dispositivo. A continuación se proporciona un ejemplo de cómo guardar un directorio y sus subdirectorios con la instrucción `tar`:

```
rosita:~# tar cvf  /home/backup.tar /home/dave
```

Esto crea el archivo `/home/backup.tar`, que contiene una copia de seguridad del directorio `/home/dave`, y todos los archivos y subdirectorios por debajo del mismo.

Cuando utilizamos `tar` para hacer archivos de contenedor, normalmente es una buena idea intentar hacer que la introducción de nivel superior en el archivo `tar` sea un directorio. De esta forma, cuando extraiga el archivo `tar`, todos los archivos que hay en su interior se colocarán bajo un directorio central en su directorio actual de trabajo. De lo contrario, si extrae un archivo `tar`

en el lugar erróneo, puede terminar con cientos de archivos en su directorio.

Suponga que tiene un directorio por debajo de su directorio actual llamado `data/` y que el directorio `data/` contiene varios cientos de archivos. Hay dos formas básicas de crear un archivo `tar` de este directorio. Puede cambiar directorios a `data` y crear el archivo `tar` desde allí:

```
rosita:~\# pwd
/home/dave\\
rosita:~# cd data
rosita:~# pwd
/home/dave/data
rosita:~# tar cvf ../data.tar .
```

Esto crea un archivo `tar` en `/home/dave/` que contiene exactamente el contenido de `data/` sin contener una entrada para el directorio. Cuando extraiga este archivo `tar` no creará un directorio en el que colocar los archivos, simplemente obtendrá cientos de archivos en su directorio actual. Otra forma de crear un archivo `tar` es comenzar desde el directorio de datos de usuario y especificar el nombre del directorio como el objeto a archivar. Por ejemplo:

```
rosita:~# pwd
/home/dave
rosita:~# tar cvf data.tar data
```

El resultado será la creación de un contenedor del directorio `data/`, pero pondrá la entrada del directorio como el primer objeto del contenedor. De esta forma, cuando se extraiga el archivo `tar`, el primero que se creará será el directorio `data/` y todos los archivos de `data/` se colocarán en el subdirectorio `data/`.

cpio

Utilidad de uso generalizado para copiar archivos; disponible en todos los sistemas UNIX; de fácil manejo, más robusto que `tar` y puede utilizar varias cintas o discos.

Utilización de cpio

`cpio` es un mandato de uso generalizado para copiar contenedores de archivos. Puede utilizarlo bien para crear copias de seguridad utilizando la opción `-o` o bien para restaurar archivos utilizando la opción `-i`. Toma su entrada de la entrada estándar y envía su salida a la salida estándar. Las ventajas de `cpio` son las siguientes:

- Puede hacer copias de seguridad de cualquier juego de archivos.
- Almacena información de una forma mas efectiva que `tar`.
- Se salta sectores defectuosos o bloques erróneos al restaurar datos.
- Sus copias de seguridad pueden restaurarse en casi cualquier sistema Linux o Unix.

Algunas personas encuentran la sintaxis de `cpio` más confusa que la de `tar`. Además, resulta necesario realizar algo de programación de *shell* para realizar copias de seguridad progresivas.
La lista siguiente es una relación de las opciones regularmente utilizadas para `cpio`:

- -o
  Copiar fuera (*out*). Crea un archivo en salida estándar.
- -i
  Copiar dentro. Extrae archivos de entrada estándar (se asume que son el resultado de una acción de copiar fuera de `cpio`).
- -B
  Bloquea entrada o salida a 5120 bytes por registro. Útil para almacenamiento eficiente en cinta magnética.
- -m
  Conserva la fecha y la hora del fichero restaurado.
- -t
  Crea una tabla de contenidos de entrada.

Considere algunos ejemplos de `cpio` en la realización de copias de seguridad y restauración de archivos. La orden siguiente copia los archivos del directorio `/home/` en el dispositivo `/dev/fd0`:

```
rosita:~# ls /home | cpio -o >/dev/fd0
```

El ejemplo siguiente extrae los archivos del dispositivo `/dev/fd0` y crea un índice en el archivo `bkup.indx`:

```
rosita:~# cpio -it < /dev/fd0 >bkup.indx
```

El ejemplo siguiente utiliza la orden `find` para crear una lista de todos los archivos en `/home/` que han sido modificados durante el último día:

```
rosita:~# find /home -mtime -1 -type f -print | cpio -oB >/dev/fd0
```

La salida de ese mandato se conduce a `cpio`, que crea un archivo en `/dev/fd0` donde se almacenan los datos a 5120 bytes por registro.

La orden siguiente restaura el archivo `/home/dave/notes.txt` desde el dispositivo `/dev/fd0`:

```
rosita:~# echo "/home/sergio/notas.txt" | cpio -i </dev/fd0
rosita:~#
```

Deberá proporcionar el nombre de archivo completo para restaurarlo con `cpio` o especificar el conjunto de ficheros a extraer del archivo `cpio` mediante las máscaras usuales.

Consulte el manual en línea de la orden `cpio` para obtener una descripción completa de las opciones que puede utilizar con este mandato.

5.4.4 Ejemplo práctico: copia de seguridad del sistema

En esta sección vamos a ver los pasos a seguir para crear un programa en *shell* que realizará la copia de seguridad del sistema operativo (ficheros principales) y lo automatizaremos mediante el servicio `cron`.

Para este ejemplo realizaremos una copia de seguridad completa todos los lunes y progresiva el resto de días. De este modo tendremos:

> Cualquier lunes: Nivel 0, copia de seguridad completa
> Cualquier martes: Nivel 1, copia de seguridad progresiva
> Cualquier otro día: Nivel 2, copia de seguridad progresiva

El programa en *shell* será el siguiente:

```
rosita:~# cat backup
#!/bin/sh
echo -n "Creando copia de seguridad"
if [ ‘date +%a‘ = "Mon"]; then
        # Hoy es lunes
        echo "Completa."
tar -czvpX /usr/local/sbin/backup.excluir -f /dev/tape /
else
        echo "Progresiva."
        tar -czvpX /usr/local/sbin/backup.excluir -f /dev/tape ‘find /
        -mtime -1 -type f -print‘
fi
```

En el archivo *excluir* podemos especificar los archivos y directorios que no queremos que sean archivados en la copia de seguridad. Esta opción es útil cuando tenemos varios sistemas de ficheros y algunos de ellos no requieren copia de seguridad, o cuando deben seguir un programa de copias distinto (por ejemplo, nos puede interesar hacer copias de seguridad del sistema de ficheros destinado a correo o a usuarios con más frecuencia que el sistema de archivos principal, puesto que la información de los usuarios y el correo sufren modificaciones más frecuentemente). El contenido del fichero *excluir* será el siguiente:

```
rosita:~# cat excluir
/proc/*
/mnt/*
/tmp/*
/home/*
rosita:~#
```

Como vemos, no se incluirán los contenidos del directorio `/proc/`, puesto que, como vimos en un capítulo anterior, no representa ningún dispositivo físico de la máquina y no es mas que un ‘reflejo’ del estado actual del sistema (memorias, procesos, rutas de red, hardware...). Especificando `/proc/*` y no `/proc/` excluimos el contenido de los directorios, pero no el directorio en sí. De esta

manera, si extraemos el contenido de la copia de seguridad se crearan los directorios `/proc/`, `/mnt/`, `/tmp/` y `/home/`, que son esenciales para el buen funcionamiento del sistema.

Una vez que tenemos el programa que realizará la copia de seguridad, sólo nos queda automatizar el proceso, para lo que utilizaremos el fichero `crontab` del superusuario, que ya hemos visto en un punto anterior:

```
rosita:~# crontab -l
# Copia de seguridad todos los dias a la 01:00
00 01 * * * /usr/local/sbin/backup
rosita:~#
```

Recordemos que el formato de cada línea del fichero `crontab` es:

minuto hora dia mes dia_de_la_semana orden_a_ejecutar

Por lo que en la línea del `crontab` anteriormente citada se especifica que deberá ejecutarse el programa `/usr/local/sbin/backup` todos los días de la semana, todos los días de todos los meses, a la 1 de la mañana. El programa `backup` se encargará de comprobar si el día coincide con un lunes. De ser así, realizará una copia de seguridad completa de todo el sistema de ficheros principal, excluyendo los archivos y directorios especificados en el fichero `excluir`. Si no es lunes, realizará una copia de seguridad progresiva de todos los archivos y directorios del sistema (excluyendo los especificados en el fichero `excluir`) que hayan sido modificados en las últimas 24 horas.

5.4.5 Resumen

La realización de copias de seguridad regulares del sistema es un proceso critico para la protección de datos. Los datos pueden perderse a causa de fallos del hardware, errores de software, errores humanos o alguna intrusión maliciosa en su sistema. Un programa de copias de seguridad bien diseñado puede hacer que el trabajo necesario para reconstruir el sistema sea mucho más sencillo. Los mandatos `tar` y `cpio` proporcionan la capacidad necesaria para realizar copias de seguridad del sistema. Cada orden tiene sus puntos fuertes y débiles. Deberá evaluar sus propias necesidades y efectuar una selección de acuerdo con las mismas.

5.5 Instalación de la red

5.5.1 Conceptos sobre redes

El conjunto de protocolos ampliamente utilizados conocidos como *Transmission Control Protocol/Internet Protocol* (TCP/IP) es cada vez más importante ya que de él dependen importantes redes internacionales como Internet y la propuesta superautopista de la información para sus comunicaciones.

TCP/IP surgió inicialmente como un proyecto promovido por el gobierno hasta alcanzar su extenso uso actual, conectando redes de todos los tamaños. Reconocido por su capacidad para permitir comunicaciones entre diferentes equipos, se encuentra virtualmente en todas las estaciones de trabajo, miniordenadores y ordenadores de gran tamaño.

Historia de TCP/IP

A mediados de los años 70, el Departamento de Defensa (DoD) de los Estados Unidos reconoció el desarrollo de un problema de comunicaciones electrónicas dentro de su organización. La comunicación del cada vez mayor volumen de información electrónica entre el personal del DOD, laboratorios de investigación, universidades y contratistas se enfrentaba a un importante obstáculo. Las diferentes entidades tenían sistemas informáticos procedentes de diferentes fabricantes, que ejecutaban sistemas operativos distintos y utilizaban diferentes tipologías y protocolos de red. ¿Cómo podía

compartirse la información?

Se le pidió a la Agencia de Investigación de Proyectos Avanzados (ARPA) que resolviera el problema que suponía tratar con diferentes equipos y topologías de red. ARPA formó una alianza con universidades y fabricantes de sistemas informáticos para el desarrollo de estándares de comunicación. Esta alianza especificó y desarrolló una red de cuatro nodos, que es la base de la red Internet actual. Durante los años 70 esta red emigró a un nuevo diseño de protocolo central que se convirtió en la base de TCP/IP.

La mención de TCP/IP requiere una breve introducción a Internet. La red Internet conecta a cientos de miles de ordenadores. Sus nodos incluyen universidades, importantes empresas y laboratorios de investigación en los Estados Unidos y otros países. Es un repositorio para millones de programas de uso compartido, noticias sobre cualquier tema, foros públicos, intercambios de información y correo electrónico. Otra característica es la conexión remota a cualquier sistema informático en la red utilizando el protocolo *telnet*. Debido al numero de sistemas que están interconectados, pueden compartirse masivos recursos informáticos, permitiendo así la ejecución de grandes programas en sistemas remotos.

Terminología Internet

El conjunto de protocolos de Internet esta compuesto de muchos protocolos relacionados basándose en la fundación formada por TCP e IP. Para clarificar la relación de estos componentes se proporcionan algunas definiciones y notas:

- Datagrama
  Se utiliza, al igual que las palabras *paquete de datos* o *mensaje de red*, para identificar una unidad de información que se intercambia.

- DNS
  Servicio de Nombre de Dominio. Un servicio proporcionado por uno o más ordenadores de una red para ayudar a localizar una ruta a un nodo deseado. Esto ahorra a cada sistema de una red la necesidad de tener que mantener una lista de cada sistema al que desee hablar.

- GOSIP
  Perfil gubernamental de interconexión de sistemas abiertos. Una colección de protocolos OSI utilizados en redes y proyectos informáticos del gobierno de los Estados Unidos.

- Internet
  Una red informática basada en protocolos TCP/IP y asociados. Una red pública que interconecta empresas, universidades, servicios gubernamentales y centros de investigación.

- FTAM
  Gestión, acceso y transferencia de archivos. Un protocolo de transferencia y gestión de archivos de acuerdo con la especificación OSI.

- FTP
  Protocolo de trasferencia de archivos. Un protocolo que permite la transferencia de archivos entre sistemas.

- IP
  Protocolo Internet. Un protocolo responsable del transporte de datagramas dentro de la red Internet.

- NFS
  Sistema de archivos de red. Un sistema de disco virtual de red que permite a un ordenador cliente montar sistemas y directorios remotos de archivo. Originalmente desarrollado por Sun Microsystems.

- NIC
  Centro de información de red. Responsable de la administración de Internet, direcciones TCP/IP y nombres de red.

- Nodo
  Un ordenador en una red.

- OSI
  Interconexión de sistema abierto. El modelo estándar ISO para la definición de comunicación de datos.

- RFC
  Petición de comentarios. La documentación mantenida por el NIC relativa a protocolos Internet, direccionamiento, rutas, configuración y otros temas relacionados con Internet.

- RIP
  Protocolo de información sobre rutas. Se utiliza para intercambiar información entre *routers*.

- RMON
  Monitor remoto. Un monitor remoto de red que permite la recogida de información sobre tráfico de red.

- RPC
  Llamada a procedimiento remoto. Permite la ejecución de procedimientos en un servidor.

- SMTP
  Protocolo simple de transferencia de correo. Se utiliza para transferir correo electrónico entre sistemas.

- SNMP
  Protocolo simple de gestión de red. Un protocolo utilizado para gestionar dispositivos remotos de red y para recoger información de dispositivos remotos relacionada con configuración, errores y alarmas.

- TCP
  Protocolo de control de transmisiones. El protocolo entre un par de aplicaciones responsable por una transmisión de datos fiable y orientada a conexiones.

- Telnet
  El protocolo utilizado para establecer conexiones entre terminales remotos.

- UDP
  Protocolo de DATAGRAM de usuario. Un protocolo sin conexiones utilizado para transferir datos entre agentes.

- VT
  Terminal virtual. Un método para utilizar *telnet*, para conectarse a sistemas remotos en la red.

Direcciones IP

El Protocolo Internet requiere que se asigne una dirección a cada uno de los dispositivos en la red. Esta dirección es conocida como la dirección IP y está organizada como una serie de cuatro octetos. Cada uno de estos octetos define una dirección única, en la que parte de la dirección representa una red (y opcionalmente una subred) y la otra parte representa un nodo específico en la red.

Algunas direcciones tienen significados especiales en Internet, como se describe a continuación:

- Una dirección que empiece con un cero hace referencia al nodo local dentro de su red actual. Por ejemplo, 0.0.0.23 hace referencia a la estación de trabajo 23 en la red actual. La dirección 0.0.0.0 hace referencia a la estación de trabajo actual.

- La dirección de bucle interno (*loopback*) 127 es importante en procesos de resolución de problemas y diagnosis de red. La dirección de red 127.0.0.0 es el bucle interno local dentro de una estación de trabajo.

- La dirección *ALL* es representada por la activación de todos los bits, proporcionando un valor de 255. Por lo tanto, 192.18.255.255 envía un mensaje a todos los nodos de la red 192.18; de la misma forma, 255.255.255.255 envía un mensaje a cada nodo en Internet. Es importante utilizar estas direcciones para mensajes de transmisión múltiple y avisos de servicio.

Es importante no utilizar 0, 127 o 255 al asignar números de nodo a las estaciones de trabajo, ya que estos números están reservados y tienen significados especiales.

Clases de dirección IP

Las direcciones IP se asignan en rangos llamados **clases**, dependiendo de la aplicación y del tamaño de la organización. Las clases más comunes son A, B y C. Estas tres clases representan el número de bits localmente asignable disponibles para la red local. Las relaciones entre las diferentes clases de dirección, el número disponible de nodos y las configuraciones iniciales de las direcciones se muestran a continuación:

Clase	Nodos Disponibles	Bits iniciales	Direcci
A	2^{24}=1,677,772	0xxx	0-127
B	2^{16}=65,536	10xx	128-191
C	2^{8}=256	110x	192-223
D		1110	224-239
E		1111	240-255

Las direcciones de la clase A se utilizan para redes de gran tamaño o para colecciones de redes asociadas. Todas las instituciones educativas están agrupadas bajo una dirección de la clase A. Las direcciones de la clase B se utilizan para redes de gran tamaño con mas de 256 nodos (pero con menos de 65.536 nodos). Las direcciones de la clase C son las utilizadas por casi todas las organizaciones.

5.5.2 Configuración de la red

El programa `ifconfig`

`ifconfig` se utiliza para la configuración (y posterior mantenimiento) de los drivers de los interfaces de red. Normalmente se ejecuta al arrancar Unix, para poner en marcha los interfaces de red, y puede usarse más tarde para *debug* o puesta a punto del sistema.

Si no se especifican argumentos, `ifconfig` simplemente muestra el estado actual de los interfaces definidos. Si especificamos un nombre de interfaz como parámetro, se nos mostrará la información correspondiente a ese interfaz en concreto.

```
rosita:~# ifconfig eth0
eth0      Link encap:10Mbps Ethernet  HWaddr 00:20:AF:DE:06:59
          inet addr:158.42.22.41  Bcast:158.42.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4800575 errors:10921 dropped:10921 overruns:12944
          TX packets:2585877 errors:2 dropped:0 overruns:0
```

```
          Interrupt:10 Base address:0x300
rosita:~#
```

La sintaxis de este mandato es:

```
ifconfig [interfaz]
```

(para mostrar información) o

```
ifconfig interfaz opciones | dirección...
```

Las opciones que utilizaremos en este curso serán:

- interface
  Es el nombre del interfaz de red. Normalmente tendrá como nombre `eth0`, `sl3`, `nei1...`: un nombre de dispositivo seguido de un número. En los ordenadores de este curso tenemos instaladas tarjetas Ethernet de la compañia 3Com. El nombre de interfaz que las identifica es `eth0`.

- up
  Este parámetro nos permite activar el interfaz especificado. Se utiliza por defecto siempre que demos al interfaz una nueva dirección.

- down
  Al contrario de *up*, esta opción desactiva completamente el interfaz de red que especifiquemos. Si tuviésemos algún problema con la red y quisieramos aislar nuestro sistema del resto del mundo podríamos quitar el cable de red, o bien desactivar nuestra tarjeta de red mediante la orden:

  ```
  rosita:~# ifconfig eth0 down
  ```

  Es importante saber que de este modo no sólo desactivamos el interfaz de red, sino que ademas eliminamos las rutas asignadas a este dispositivo. Veremos como se maneja la tabla de rutas más adelante.
  Podremos volver a activar la tarjeta de red mediante la orden:

  ```
  rosita:~# ifconfig eth0 up
  ```

  Si queremos que todo vuelva a la normalidad deberemos de especificar de nuevo las rutas correspondientes a este dispositivo.

- netmask dirección
  Configura la máscara de red de ese interfaz. En nuestro caso la máscara de red es "255.255.0.0".

- [-]broadcast [dirección]
  Activa o desactiva el protocolo de dirección de *broadcast* para este interfaz. Si se especifica una dirección, ésta es usada como dirección de *broadcast* para este interfaz, si no, sólo se activa el *flag* de *broadcast* del dispositivo. Si ponemos un signo '-' delante, se desactiva el *flag*.

- dirección
  El nombre de *host* o dirección *ip* (si se especifica un nombre de host se intentará resolver su dirección *IP*) de ese interfaz.
  En nuestro caso, para configurar la tarjeta de red con `ifconfig`, lo haríamos con la orden:

  ```
  rosita:~# ifconfig eth0 [IP] broadcast 158.42.255.255 netmask 255.255.0.0
  ```

Donde IP es nuestra dirección númerica (por ejemplo, 158.42.10.100). Como dirección de *broadcast* utilizaríamos '158.42.255.255' y especificamos que nuestra máscara de red es '255.255.0.0'. Es importante definir correctamente nuestra dirección IP (que debe ser única en la red) para no tener conflictos con otras máquinas conectadas.

El programa `route`

Una vez configurado nuestro interfaz de red debemos asignarle rutas estáticas a otras máquinas o redes, para de este modo poder establecer comunicación con ellas.

El mandato `route` muestra o manipula las tablas de ruta del *kernel*. Si no especificamos ningun argumento, nos muestra la configuración actual:

```
rosita:~# route
Kernel routing table
Destination Gateway         Genmask      Flags MSS    Window UseIface
localnet    *               255.255.0.0  U     1500   0      16 eth0
loopback    *               255.0.0.0    U     3584   0      89 lo
default     atlas.cc.upv.es *            UG    1500   0      66 eth0
```

Si especificamos la opción `-n`, obtenemos el resultado anterior, pero se nos mostrarán las direcciones númericas de las máquinas en vez de sus nombres. Esto es útil cuando no tenemos acceso al servidor de nombres.

Otras opciones de esta orden son:

- del XXXX
  Borra la ruta asociada al destino XXXX. Por ejemplo, para borrar la ruta asociada a nuestra red local, utilizamos el mandato:

  ```
  rosita:~# route del localnet
  rosita:~# route
  Kernel routing table
  DestinationGateway      Genmask  Flags MSS   Window Use  Iface
  loopback   *            255.0.0.0U     3584  0      90   lo
  default    atlas.cc.upv.es       UG    1500  0      100  eth0
  ```

- add [-net | -host] XXXX [gw GGGG] [metric MMMM] [netmask NNNN] [dev DDDD]
  Añade una ruta a la dirección IP XXXX. La ruta especificada será una ruta de red si se especifica el modificador `-net` o si XXXX se encuentra en el fichero `/etc/networks` y no se utiliza el modificador `-host`.

  El argumento *gw GGGG* significa que cualquier paquete de red enviado a esa direccion XXXX será dirigido a través del gateway specificado. Es imprescindible que el *gateway* especificado ya sea accesible, por lo que tendremos que haber definido una ruta a ese *gateway* con anterioridad.
  El modificador *metric MMMM* no está disponible por el momento, es decir que no causa ningún efecto, pero se ha implementado para poder ser usado en un futuro próximo.

  Con *netmask NNNN* especificamos la máscara de red de la ruta a ser añadida. Esto sólo es útil cuando estamos añadiendo una ruta de red, no una ruta a una máquina concreta, y cuando XXXX es coherente con la máscara de red que estamos especificando. Si no damos ninguna máscara de red, el programa intenta adivinar cual correspondería a nuestro caso.

El parámetro *dev DDD* obliga a la ruta a ser asociada con el dispositivo especificado, de otro modo el *kernel* intentará determinar el dispositivo correcto (comprobando las rutas ya creadas, las especificaciones de los dispositivos y el destino de la ruta). En la mayoría de los casos no es imprescindible utilizar esta opción. Si la opción *dev DDDD* es la última opción de la línea de órdenes, se puede obviar la palabra `'dev'`.

Ejemplo práctico: configuración de la red en nuestro Linux

Desde el terminal en el que estamos trabajando necesitaremos definir tres rutas para cubrir todas nuestras necesidades. Una ruta interna en nuestra máquina, una ruta a nuestra red local y una ruta hacia el exterior de la red local. La más sencilla de configurar es la red interna de nuestra máquina (que recibe el nombre de *loopback*).

Como hemos visto, el primer paso es configurar el interfaz de red. En el caso de la red interna, el interfaz se denomina `lo` (*loopback*). Y se configura con la siguiente orden:

```
rosita:~# ifconfig lo 127.0.0.1
rosita:~#
```

No necesitamos especificar ni la máscara de red, ni la dirección de *broadcast* ya que las asigna por defecto. La máscara de subred será 255.0.0.0 y la dirección de broadcast 127.255.255.255.

Una vez configurado el interfaz de red, podemos asignar la ruta a la red interna de nuestra máquina. Para eso utilizamos la orden `route` de la siguiente forma:

```
rosita:~# route add -net 127.0.0.0
rosita:~#
```

Y ya tendremos configurada la red interna:

```
rosita:~# route
Kernel routing table
Destination  Gateway   Genmask   Flags MSS   Window Use  Iface
loopback     *         255.0.0.0 U     3584  0      0    lo
```

Desde este momento todos los paqueted de red que enviamos a direcciones de la forma 127.xx.xx.xx serán redigiridos a nuestra máquina. Por lo tanto sería equivalente hacer un `telnet` a la dirección 127.0.0.1 (o 127.0.0.2, o 127.0.0.3...) que hacerla a nuestra dirección numérica. Con una pequeña diferencia: al usar una dirección de la forma 127.0.0.xx todo el tráfico será interno, mientras que si especificamos la dirección numérica tendremos que pasar por el interfaz de red correspondiente (en nuestro caso será `eth0` como veremos ahora).

El siguiente paso es configurar nuestra red local. El interfaz de red que nos comunicará con el exterior sera el `eth0` (nuestra tarjeta de red Ethernet). Necesitamos saber algunos datos : la dirección numérica que corresponde a nuestra máquina, nuestra dirección de *broadcast* y la máscara de red. En los ordenadores de las aulas de la Escuela Técnica Superior de Ingenieros Industriales, las direcciones IP son de la forma 158.42.10.TT, donde TT es el número de nuestro terminal, escrito en la parte frontal de la carcasa del ordenador. Por ejemplo, el terminal 180 tendrá como dirección IP 158.42.10.180. En cuanto a la máscara de red, la red de la Universidad Politécnica de Valencia tiene asignadas las direcciones numéricas del tipo 158.42.xx.xx, y cualquier ordenador del campus puede acceder directamente por red local a estas direcciones. Por lo tanto, nuestra máscara de red será 255.255.0.0. Por ultimo, la dirección de *broadcast* es 158.42.255.255.

Conociendo todos estos datos podemos configurar el interfaz con la orden:

```
rosita:~# ifconfig eth0 158.42.10.180 broadcast 158.42.255.255 netmask 255.255.0.0
rosita:~#
```

Y de nuevo, procedemos a asignar la ruta a nuestra red local:

```
rosita:~# route add -net 158.42.0.0 netmask 255.255.0.0
rosita:~#
```

Que simplemente significa que para acceder a cualquier dirección que empiece por 158.42. utilizaremos el interfaz de red `eth0`. Ya tenemos definidas la ruta interna y la ruta a la red local:

```
rosita:~# route
Kernel routing table
Destination  Gateway   Genmask     Flags   MSS  Window  Use  Iface
loopback     *         255.0.0.0   U       3584 0       3    lo
localnet     *         255.255.0.0 U       1500 0       38   eth0
```

Lo último que nos queda por hacer es definir la ruta hacia el exterior de nuestra red local. Aquí entra en juego un elemento importante: el *gateway*. El *gateway* no es más que una máquina conectada a nuestra red que sirve de enlace entre nuestra red local y otras redes. Todos los paquetes que queramos enviar fuera de nuestra red local deberan pasar obligatoriamente por esa máquina. En la UPV la máquina encargada de este trabajo se llama *atlas.cc.upv.es* y su dirección numérica es 158.42.1.10. Especificamos la última ruta de la siguiente forma:

```
rosita:~# route add default gw 158.42.1.10 metric 1
rosita:~#
```

Esto añade una ruta por defecto, que será usada si una ruta solicitada no concuerda con ninguna de las otras rutas. Todos los paquetes de red que usen esta ruta seran encauzados mediante el *gateway* especificado (en nuestro caso *atlas.cc.upv.es*. El dispositivo que se usa para llegar a esa ruta (en nuestro caso `eth0`) dependerá de como vamos a llegar al *gateway* especificado. Como *atlas.cc.upv.es* forma parte de nuestra red local, entonces accedemos a esta ruta mediante el mismo interfaz de red de la red local, es decir, `eth0`.

Finalmente podemos verificar el estado de la tabla de rutas:

```
rosita:~# route
Kernel routing table
Destination Gateway         Genmask      Flags  MSS   Window  Use   Iface
loopback    *               255.0.0.0    U      3584  0       3     lo
localnet    *               255.255.0.0  U      1500  0       38    eth0
default     atlas.cc.upv.es *            UG     1500  0       6365  eth0
```

Veamos algunos ejemplos de utilización de estas rutas:

Si queremos acceder a nuestra máquina local, podemos utilizar el interfaz de red *loopback*. Nuestra dirección local sería 127.0.0.1. Como esta dirección concuerda con la ruta a *loopback* (que incluye todas las direcciones que empiecen por 127.) entonces se va a utilizar el interfaz `lo`.

Si intentamos conectar con una direccion del tipo 158.42., la ruta implicada será la red local (*localnet*) y nuestros paquetes de red se enviarían mediante el interfaz `eth0` Esto nos incluye a nosotros mismos. Es decir, si nuestra dirección IP es 158.42.10.180 y la utilizamos para acceder a nosotros mismos, vamos a estar utilizando la red local y por lo tanto el interfaz `eth0` (y no el interfaz `loopback`).

Por último si queremos acceder a cualquier dirección IP que no corresponde a ninguna de las rutas definidas (por ejemplo 194.179.32.1), los paquetes de red se enviarán al *gateway* definido, y este *gateway* se encargaría de enviarlos a su vez al exterior. Es importante ver que al estar el *gateway* en nuestra red local, debemos obligatoriamente tener definida correctamente una ruta que abarque la red local, de no ser así no podríamos acceder al *gateway*, y por lo tanto no podríamos acceder

al exterior. En realidad, cuando enviamos un paquete de red a una dirección 'externa' (como en el ejemplo antes citado) primero pasa por nuestra red local hasta el *gateway*, y de ahí al exterior.

Configuración del servidor de nombres

Una vez configurados los interfaces de red y la identidad de nuestra máquina, debemos proporcionar al sistema unos datos básicos sobre la red en la que nos encontramos para poder trabajar eficazmente. Algunos de los ficheros de configuración a modificar en función de nuestro entorno de red son:

El fichero `/etc/hosts`

Este fichero contiene una lista de máquinas a las que se accede normalmente. Debe contener al menos las siguientes líneas:

```
127.0.0.1           localhost
aaa.bbb.ccc.ddd     mi_maquina
```

donde *aaa.bbb.ccc.ddd* es la dirección IP de nuestra máquina, y *mi_maquina* su nombre asignado, por ejemplo:

```
158.42.10.180 term180.aiind.upv.es  term180
```

El formato de cada línea del fichero `/etc/hosts` se compone por lo tanto de una ip numérica y el nombre asignado a esa ip numérica, que puede estar seguido por un *alias* de esa máquina. En el ejemplo anterior sería equivalente hacer un `telnet term180.aiind.upv.es` y `telnet term180`.

Normalmente se incluyen en este fichero las máquinas a las que más accedemos, o las más importantes de nuestra red local, acompañadas de su *alias* correspondiente para poder acceder más rapidamente. De esta manera, no tenemos que recordar una IP confusa, como 158.42.10.180, sino una palabra, *term180*.

Así, cuando accedemos a una máquina mediante su nombre (*term180*), se busca su dirección numérica en `/etc/hosts`, y si se encuentra, se conecta a la direccion IP en cuestión.

El fichero `/etc/resolv.conf`

Desgraciadamente, ni podemos recordar miles de direcciones numéricas, ni podemos tener un fichero `/etc/hosts` con miles y miles de direcciones numéricas especificadas una a una. Por un lado porque el fichero sería enorme y costaría mucho encontrar la dirección deseada. Por otro lado, ¿cómo tener la información de ese fichero correctamente actualizada?

Para suplir este problema existen los servidores de nombres (DNS, *Domain Name Server*). Cada vez que intentamos acceder a una máquina por su nombre, y este nombre no se encuentra en `/etc/hosts`, se intenta resolver mediante una tercera máquina, que actúa como servidor de nombres. Esta máquina nos devolverá la dirección numérica de la máquina que pedimos, siempre y cuando exista. En el fichero `/etc/resolv.conf` especificamos los servidores de nombres que queremos usar. Es recomendable poner más de uno, por si el principal fallase. Un ejemplo de este archivo sería:

```
rosita:~# cat /etc/resolv.conf
# /etc/resolv.conf
#
domain upv.es
nameserver 158.42.3.1
nameserver 158.42.4.1
nameserver 158.42.1.5
```

```
nameserver 158.42.2.1
nameserver 194.73.30.2
rosita:~#
```

Con `domain` especificamos el dominio de nuestra red local. A continuación se especifican los servidores de nombres que se desean, precedidos por la palabra `nameserver`. Los servidores de nombres se prueban en el orden listado. En primer lugar se suelen colocar los servidores de nombres más fiables y más cercanos, de forma que las consultas usuales sean rápidas (del mismo modo, si los primeros servidores de nombres estuvieran desconectados, se tardaría menos en saltar al siguiente de la lista).

El fichero `/etc/host.conf`

Este archivo indica que servicios usar, y en que orden, para resolver una dirección IP. Se pueden usar varias opciones en este archivo:

`order`: indica en que orden se prueban los distintos mecanismos de resolución de nombres. Los servicios de resolución se prueban en el orden listado. Se admiten los siguientes mecanismos de resolución de nombres:

- hosts: Se intenta solucionar el nombre mirando en el archivo `/etc/hosts` local.
- bind: Se consulta un servidor de nombres DNS para solucionar el nombre.
- nis: Se utiliza el protocolo de Servicio de Informacion de Red (NIS, *Network Information Service*) para intentar solucionar el nombre del sistema.

`multi`: Toma *on* u *off* como argumento. Se usa para determinar si un sistema esta autorizado a tener más de una dirección IP indicada en `/etc/hosts`. `multi` sólo se usa juntamente con las consultas `hosts`. Esta opción no tiene ningun efecto sobre las consultas NIS o DNS.
Un ejemplo de fichero `/etc/host.conf` podría ser:

```
rosita:~# cat /etc/host.conf
order hosts, bind
multi on
rosita:~#
```

Esto indica que para resolver un nombre primero se consulta el fichero `/etc/hosts` y si no se puede resolver mediante este archivo, se hace la consulta via DNS (utilizando los servidores de nombres listados en el archivo `/etc/resolv.conf`).

5.5.3 Configuración de servicios en Internet

El servidor `inetd`

El programa `inetd` es el demonio que se ocupa de gestionar los servicios que ofrecemos en internet. `inetd` se ejecuta en el arranque del sistema y espera conexiones en ciertos puertos de la máquina. Cuando se efectúa una conexión a alguno de esos puertos, el demonio decide que servicio se está pidiendo (comprobando que servicio corresponde a ese puerto) e invoca el programa destinado a responder a esa petición. Una vez que el programa ha finalizado, el demonio vuelve a esperar otra petición en ese puerto (excepto en algunos casos que veremos más adelante). En resumen, `inetd` permite tener un solo demonio ejecutándose, que se encargará de invocar a otros cuando sean requeridos, lo que disminuye la carga del sistema, ya que evita tener todos los servicios en marcha constantemente.

Cuando se ejecuta en el arranque, `inetd` lee su fichero de configuración que, por defecto, es `/etc/inetd.conf`:

El fichero `/etc/inetd.conf`

El fichero de configuración de `inetd` contiene entradas con varios campos. Cada uno de ellos tiene que estar definido en una línea, y se separan entre sí mediante tabuladores o espacios. Los comentarios se marcan con un # al principio de la línea. Los campos son:

<servicio> <tipo de socket> <protocolo> <wait/nowait> <usuario> <programa> <opciones>

El servicio especifica el nombre válido de uno de los servicios especificados en el fichero `/etc/services`, por ejemplo, `telnet`. El tipo de socket debe estar definido con una de las palabras clave `stream`, `dgram`, `raw`, `rdm` o `seqpacket`, dependiendo del tipo de socket que va a necesitar el servicio. El protocolo debe ser uno de los protocolos válidos especificados en el fichero `/etc/protocols`. Por ejemplo, *tcp* o *udp*.

La opción *wait/nowait* se aplica solo a sockets de tipo *datagram* (*dgram*). Para otros tipos de sockets debe especificarse obligatoriamente *nowait*. Si el servicio que es requerido no bloquea el uso del puerto que tiene asignado, y permite recibir más peticiones al mismo puerto, entonces se especifica con la palabra *nowait*. De lo contrario se especifica con *wait*.

La entrada *usuario* debería contener el nombre de usuario con el que queremos que se ejecute el programa que da el servicio. Esto permite lanzar programas servidores con menos privilegios que los del `root`. Si un servidor no necesita ejecutarse con permisos de root, es aconsejable ejecutarlo con un usuario con menos privilegios, por motivos de seguridad. El campo *programa* especifica que programa debe ejecutar `inetd` cuando recibe una peticion en el socket asignado a ese servicio. Si se accede a un servicio 'interno' de `inetd` hay que especificar `'internal'`. Por ejemplo, si `inetd` recibe una petición en el puerto de `ftp` (puerto 21) ejecutará el servidor de `ftp` (normalmente `/usr/sbin/wu.ftpd`, `/usr/sbin/ftpd` o `/usr/sbin/in.ftpd`).

Con el campo *opciones* podemos llamar al programa que responde al servicio pasándole argumentos. Si el servicio es interno de `inetd`, debemos especificar *internal*.
Un ejemplo de línea de `inetd.conf` sería:

```
ftp     stream  tcp     nowait  root    /usr/sbin/wu.ftpd -a
```

Esto especifica como debe comportarse `inetd` ante una petición del servicio de `ftp`. El nombre del servicio es `ftp`; podemos deducir, mirando el fichero `/etc/services`, que el puerto asignado a este servicio es el 21 y que el protocolo usado es `tcp`:

```
rosita:~# grep ftp /etc/services
ftp             21/tcp
rosita:~#
```

El tipo de socket usado en una conexión *ftp* es *stream*. Al no tratarse de un socket *datagram*, se especifica *nowait*. El usuario con el que se va a ejecutar el programa servidor desde `inetd` es `root`. Finalmente, el programa a ejecutar será `/usr/sbin/wu.ftpd`, con el argumento `'-a'`. Por lo tanto, cuando `inetd` detecte una conexión al puerto 21 de nuestra máquina, ejecutará el mandato `/usr/sbin/wu.ftpd -a` como *root*, que atenderá la petición.

En realidad, en nuestro caso el servicio `ftp` está especificado en el fichero `/etc/inetd.conf` de la siguiente forma:

```
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  wu.ftpd -a
```

Con lo cual `inetd` no ejecuta el programa servidor `wu.ftpd` con la opción `'-a'`, sino que ejecuta el programa `/usr/sbin/tcpd` con los argumentos `wu.ftpd -a`. El programa `/usr/sbin/tcpd` se denomina *TCP Wrapper* y lo veremos en el tema de seguridad.

Ejemplo práctico : Instalación de un nuevo servicio en Internet.

El primer paso a realizar es determinar que tipo de servicio queremos ofrecer, y que programa va a ocuparse de dar este servicio. En este caso vamos a instalar un programa muy simple, que imprima en pantalla un mensaje cuando se conecte al puerto 12345 de nuestra máquina. Para ello utilizaremos la utilidad `bban` instalada en el sistema, que genera un *banner* a partir de un texto que recibe como argumento:

```
rosita:~# bban Hola\!\!
                                        !!!      !!!
  H    H    OOOO    L          AA      !!!      !!!
  H    H   O    O   L         A  A     !!!      !!!
  HHHHHH   O    O   L        A    A     !        !
  H    H   O    O   L        AAAAAA
  H    H   O    O   L        A    A    !!!      !!!
  H    H    OOOO    LLLLLL   A    A    !!!      !!!
rosita:~#
```

Por lo tanto crearemos un *shellscript* que funcione como programa servidor.

En el fichero `/usr/sbin/hola` escribimos lo siguiente:

```
rosita:~# cat /usr/sbin/hola
#!/bin/sh
#
# Imprime el banner "Hola!!" en pantalla
bban Hola\!\!
rosita:~#
```

Y hacemos que sea ejecutable mediante la orden:

```
rosita:~# chmod 755 /usr/sbin/hola
rosita:~#
```

El siguiente paso es crear un nuevo servicio en el fichero `/etc/services` que definirá que nombre, puerto y protocolo va a utilizar nuestro servicio.

Editamos el fichero `/etc/services` y añadimos la siguiente línea al final del archivo:

```
hola 12345/tcp
```

Por último habilitaremos el nuevo servicio en el fichero `/etc/inetd.conf`, añadiendo la siguiente línea al final del archivo :

```
hola    stream  tcp     nowait  root    /usr/sbin/hola
```

Una vez hecho esto, hay que indicarle al proceso `inetd` activo que debe leer de nuevo el archivo de configuración ya que ha sido modificado. Para ello mandamos la señal *SIGHUP* al proceso en memoria:

```
rosita:~# ps -aux|grep inetd
root        51  0.0  0.3   828   188  ?  S   Nov 24   0:02 /usr/sbin/inetd
rosita:~# kill -HUP 51
rosita:~#
```

Desde este momento cualquier persona puede conectar al puerto 12345 de nuestra máquina, desde cualquier sitio. Para comprobarlo haremos lo siguiente:

```
rosita:~# telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

                                        !!!      !!!
  H    H    OOOO    L          AA      !!!      !!!
  H    H   O    O   L         A  A     !!!      !!!
  HHHHHH   O    O   L        A    A     !        !
  H    H   O    O   L        AAAAAA
  H    H   O    O   L        A    A    !!!      !!!
  H    H    OOOO    LLLLLL   A    A    !!!      !!!

Connection closed by foreign host.
rosita:~#
```

Cuando definimos un nuevo servicio en el fichero `/etc/services`, también creamos un *alias* para el puerto numérico, de manera que `telnet localhost 12345` y `telnet localhost hola` (recordemos que `'hola'` es el nombre del servicio) son equivalentes.

A partir de aquí podemos experimentar, conectando al puerto 12345 de los otros alumnos del curso, modificando el fichero `/usr/sbin/hola`, cambiando el puerto asignado a este servicio, etc.

5.5.4 Utilidades de red

Disponemos de varias utilidades básicas para determinar el estado de la red:

`ping`

Nos permite mandar paquetes a un *host* determinado para comprobar si está funcionando y conectado a la red:

```
rosita:~# ping www.linuxhq.com
PING linuxhq.com (204.209.212.113): 56 data bytes
64 bytes from 204.209.212.113: icmp_seq=7 ttl=50 time=295.0 ms
64 bytes from 204.209.212.113: icmp_seq=8 ttl=50 time=309.0 ms
64 bytes from 204.209.212.113: icmp_seq=11 ttl=50 time=342.6 ms
64 bytes from 204.209.212.113: icmp_seq=13 ttl=50 time=359.5 ms
64 bytes from 204.209.212.113: icmp_seq=16 ttl=50 time=377.8 ms
64 bytes from 204.209.212.113: icmp_seq=19 ttl=50 time=328.8 ms
--- linuxhq.com ping statistics ---
52 packets transmitted, 29 packets received, 44% packet loss
round-trip min/avg/max = 276.1/322.7/414.2 ms
```

Como vemos, también nos devuelve información sobre la velocidad a la que viajan los paquetes de datos, y estadísticas sobre la conexión (paquetes enviados, paquetes recibidos, porcentaje de pérdidas y tiempo medio de retardo).

`traceroute`

El programa `traceroute` nos permite ver que ruta siguen nuestros paquetes de red hasta un destino concreto. Una salida típica del programa puede ser:

```
rosita:~# traceroute www.linuxhq.org
traceroute to www.linuxhq.org (207.179.23.253), 30 hops max, 40 byte
packets
```

```
1 atlas.cc.upv.es (158.42.1.10) 1.256 ms 1.165 ms 1.136 ms
2 A1-0-3.EB-Valencia1.red.rediris.es (130.206.211.185) 2.526 ms 2.674 ms
3.153 ms
3 A1-0-2.EB-Madrid1.red.rediris.es (130.206.224.5) 11.614 ms 12.915 ms
12.344 ms
4 A1-0-1.EB-Madrid0.red.rediris.es (130.206.224.69) 13.591 ms 9.28 ms
18.692 ms
5 * * * [...]
```

Por lo tanto, para conectar con la máquina *www.linuxhq.org* tenemos que pasar antes por varios *routers*, el local (*atlas.cc.upv.es*), un *router* en Valencia, dos *routers* consecutivos en Madrid... Podemos comprobar lo que vimos anteriormente sobre nuestra red local, haciendo un `traceroute` a cualquier máquina de nuestro dominio:

```
rosita:~# traceroute luisvive.upv.es
traceroute to luisvive.euiti.upv.es (158.42.44.29), 30 hops max, 40 byte packets
 1  luisvive.euiti.upv.es (158.42.44.29)  0.85 ms  0.776 ms  0.742 ms
rosita:~#
```

Como explicamos en el apartado de la configuración de rutas, accedemos a cualquier máquina de nuestra red local directamente, sin tener que pasar por ningun *gateway*.

5.6 El servicio de impresión

Como en todo sistema de computación, en nuestras máquinas Unix vamos a tener la posibilidad de configurar una serie de impresoras (locales y/o remotas). En Linux, es necesario disponer en primer lugar de soporte para impresoras en el núcleo; los dispositivos locales están representados en el sistema como archivos especiales orientados a carácter llamados generalmente `lpX` (*Line Printer*) en el directorio `/dev/`:

```
rosita:~# ls -al /dev/lp*
crw-r-----   1 root     daemon     6,   0 Apr 28  1995 lp0
crw-r-----   1 root     daemon     6,   1 Apr 28  1995 lp1
crw-r-----   1 root     daemon     6,   2 Apr 28  1995 lp2
rosita:~#
```

Vemos que el propietario de las impresoras es el superusuario, y el grupo al que pertenecen es `daemon`. El *major number* es 6, lo que nos indica que se trata de impresoras (en Linux, recordemos que en otros Unices no tiene porqué ser así). El *minor number* nos sirve, como ya sabemos, para distinguir entre diferentes dispositivos. Lo que desde MS-DOS se denomina *LPT1* (puerto paralelo) vamos a encontrarlo en Linux sobre PC como `/dev/lp1` en un AT (la mayoría de máquinas) con Linux 2.0, o bien como `/dev/lp0` si usamos Linux 2.2 o un XT (máquinas más antiguas).

Trabajando con Linux (y por supuesto con cualquier Unix) el tipo de fichero más elemental a imprimir es el de texto ASCII; simplemente redirigiendo la salida de un `cat` al dispositivo adecuado lo conseguiremos:

```
rosita:~# cat archivo >/dev/lp1
rosita:~#
```

Para realizar esta operación necesitamos ser superusuarios; para que cualquier usuario del sistema pueda hacerlo, hemos de establecer los permisos adecuados en el archivo que referencia a nuestra impresora. Esto puede representar un problema de seguridad para la máquina, por lo que no suele ser una práctica recomendable. Además, de esta forma no aprovechamos la multitarea del sistema, ya que hemos de completar la ejecución del mandato (que puede ser largo en impresoras lentas) hasta volver al *shell* y poder seguir introduciendo órdenes (aunque lo podríamos haber lanzado

en *background*). La forma correcta de trabajar con impresoras es utilizando el mandato `lpr`. Esta orden se encarga de encolar un trabajo, depositándolo en lo que se denomina el directorio de *spool*.

Una vez allí, el demonio de impresión, `lpd`, es el encargado de gestionar e imprimir esta petición enviándola a un dispositivo físico. El demonio `lpd` utiliza la información almacenada en el archivo `/etc/printcap`, un fichero de configuración que presenta una entrada por impresora. En cada una de estas entradas se especifica el nombre del dispositivo físico y como ha de tratar esa impresora la información que recibe. Por ejemplo, se podría indicar que dispositivo se utiliza, el directorio de *spool* de la impresora y que filtro utilizar con la información a imprimir:

```
lp:lp=/dev/lp1:sd=/usr/spool/lp1:sh
```

Si además de texto ASCII queremos imprimir *PostScript*, necesitaremos un filtro adecuado a nuestra impresora, y que este filtro se invoque desde `/etc/printcap`. Por ejemplo, para una impresora HP-510 tenemos disponible (en distribuciones de Linux y también en INet) el siguiente filtro:

```
#!/bin/sh
PRINTER=djet500
nenscript -ZB -p- | gs -q -sDEVICE=$PRINTER \
-sPAPERSIZE=letter -dNOPAUSE \
-dSAFER -sOutputFile=- -
```

que se invoca desde `/etc/printcap` como:

```
lp:lp=/dev/lp1:sd=/var/spool/lpd/lp:mx%0:if=/etc/filter:sh
```

Como superusuarios, vamos a disponer de la orden `lpc` para realizar tareas administrativas con las impresoras, como detenerlas, analizar su estado, manipular trabajos en la cola de impresión, etc. Esta orden incluye su propio *prompt*, desde el cual vamos a realizar estas tareas. Podemos ver un ejemplo del funcionamiento de `lpc`:

```
rosita:~# lpc
lpc> status lp
lp:
queuing is enabled
printing enabled
2 entries in spool area
daemon started
lpc> disable lp
lp:
queuing disabled
lpc> stop lp
lp:
printing disabled
lpc> start lp
lp:
printing enabled
lpc> enable lp
lp:
queuing enabled
lpc> quit
rosita:~#
```

Otro par de utilidades interesantes que nos pueden resultar útiles para gestionar la cola de impresión son `lpq` y `lprm`. La primera orden nos informará del estado de la cola de impresión, esto es, del contenido del área o directorio de *spool* para un dispositivo determinado. De toda la información que proporciona `lpq`, la más interesante va a ser el identificador de trabajo (*Job ID*), que nos

permitirá cancelar trabajos de la cola (los no activos) mediante `lprm` A este mandato generalmente le pasaremos como parámetro el identificador, pero si lo que queremos es eliminar todos los trabajos de la cola, en lugar de hacerlo uno a uno utilizaremos la opción '`-`':

```
rosita:~# lpq
lp is ready and printing
Rank    Owner     Job  Files               Total Size
active  toni       67  impresion            4517 bytes
rosita:~# lprm -
rosita:~# lpq
no entries
rosita:~#
```

5.7 Configuración del correo electrónico

5.7.1 Conceptos

Uno de los servicios más usados y más interesantes de una red extensa es el correo electrónico. Este servicio ofrece la posibilidad de enviar mensajes personales, informes, datos, documentos, archivos... de un lugar a otro en cuestión de segundos o minutos.

Cuando se envía correo electrónico, el sistema informático se encarga de realizar la entrega; esto significa que se puede poner el mensaje en una red para que sea entregado en alguna otra ubicación. En este momento, puede considerarse que se ha enviado el mensaje. Poco después, éste llega al sistema del destinatario.

Si el remitente y el destinatario están en el mismo sistema informático, todo este proceso tiene lugar en el propio sistema. El sistema de correo de la máquina de destino verifica que exista el destinatario y el mensaje se agrega a un archivo que guarda todos los *e–mails* de ese usuario (si no se está conectado a una red, el sistema informático local verifica el destinatario). El archivo de almacenamiento de correo recibe el nombre de buzón del sistema del usuario, y tiene la misma denominación que el del usuario que recibe el correo. Por ejemplo, para un usuario *juan*, su buzón del sistema es el archivo *juan* en el directorio `/var/spool/mail/`.

El correo electrónico guarda ciertas similitudes con el correo convencional. De la misma manera que una carta tiene que recorrer varias oficinas postales hasta llegar a su destino, un mensaje electrónico debe recorrer varias máquinas donde se le dirigirá hacia la máquina de destino. En el caso de un sistema Unix también tenemos una especie de 'cartero', que se encarga de determinar si el mensaje es enviado a nuestro sistema, si la persona a la que se envía ese mensaje realmente existe y por último, de depositarlo en el buzón del sistema que corresponda.

Existen varios programas que pueden gestionar el correo electrónico en un sistema Unix, pero el más conocido y utilizado es el programa `sendmail`.

5.7.2 El gestor de correo `sendmail`

`sendmail` es un agente de transporte de correo electrónico. Su función es recolectar un mensaje y, dependiendo de su cabecera y de la configuración que hayamos creado para el programa, distribuir el correo ya sea local o remotamente.

Los directorios y ficheros relacionados con el programa `sendmail` son:

`/etc/aliases`: fichero con los alias de correo
`/etc/aliases.db`: base de datos de los alias (en formato especial)
`/etc/sendmail.cf`: fichero de configuración de *sendmail*

`/var/spool/mqueue/*`: cola de correo y ficheros temporales
`/var/spool/mail/*`: buzones de correo del sistema

Antes de enviarse cualquier mensaje, ya sea internamente o a sistemas remotos, éste debe pasar por el directorio de la cola de correo.

De por sí, `sendmail` puede funcionar como programa de correo, pero no es recomendable usarlo directamente. Otros programas actúan como interfaz de usuario a la hora de enviar correo y son más fáciles y agradables de usar (como `pine` o `elm`).

Opciones *sendmail*

- -bd
  Ejecutar `sendmail` como demonio. El programa se queda en segundo plano y espera conexiones al puerto 25. Este es el modo de uso que más se utiliza, pero no por ello es el mejor ni el más recomendable.
- -bD
  Igual que la opción `-bd` pero el programa no pasa a segundo plano.
- -bi
  Inicializar la base de datos que contiene los alias de correo. Debemos ejecutar esta orden (o `newaliases`) siempre que modifiquemos el fichero `/etc/aliases`, ya que la base de datos NO se regenera automáticamente.
- -bp
  Muestra una lista de los mensajes en la cola.
- -bt
  Modo de test. Con esta opción podemos comprobar que haría `sendmail` paso a paso para mandar un mensaje a la dirección de correo especificada. Es muy útil para detectar posibles problemas en la configuración de `sendmail`.
- -q[tiempo]
  Con esta opción especificamos en que intervalos de tiempo debe chequearse la cola de correo. Si no especificamos un intervalo de tiempo, se procesa la cola una vez. Algunos ejemplos de como se especificaría el tiempo serían:

  -q10m cada 10 minutos -q30m cada 30 minutos -q1h30m cada hora y media

  Para especificar las unidades de tiempo utilizaremos m para los minutos, h para las horas, d para los días y w para las semanas.
- -v
  Mostrar mucha información. Si se especifica esta opción, `sendmail` ofrece más información sobre las tareas que realiza.

Por ejemplo, si quisieramos ejecutar `sendmail` en modo demonio y de forma que procese la cola de correo cada 15 minutos podemos hacerlo con la orden

```
rosita:~# /usr/sbin/sendmail -bd -q 15m
```

Normalmente no tenemos que arrancar nosotros el programa `sendmail` ya que se ejecuta automáticamente al arrancar el sistema.

El fichero `/etc/sendmail.cf`

El fichero `/etc/sendmail.cf` guarda la configuración utilizada por `sendmail`. Su modificación avanzada es muy compleja ya que debemos tener unos grandes conocimientos de como funciona el

correo electrónico y del lenguaje utilizado para crear las diferentes reglas por las cuales se rige el gestor de correo.

Con estas reglas se define cómo debemos comportarnos según el mensaje que nos llegue. Hay multitud de reglas distintas, y el número de reglas que deberemos tener en cuenta dependen de si somos un simple receptor de correo, de si gestionamos el correo de nuestro propio dominio, o de si gestionamos el correo de toda una subred. Normalmente sólo debemos conocer dos datos sobre nuestro sistema, y modificar el fichero `/etc/sendmail.cf` acorde con nuestra situación.

Por una parte debemos conocer cual es la máquina de correo de nuestra red local, es decir, quien gestiona todo el correo de nuestra red. Esa máquina es la que nos entrega los mensajes. Se podría considerar la oficina de correos central, todo envío, ya sea hacia dentro o hacia fuera de nuestra red, debe pasar por ahí. En nuestro caso, la máquina que se encarga del correo es *vega.cc.upv.es*, y por cuestiones de comodidad tiene un sobrenombre llamado *mailhost*, que ayuda a identificar su función dentro de la red. Por lo tanto, para especificar que ésta es nuestra máquina central de correo, debemos tener la siguiente declaración en el fichero `/etc/sendmail.cf`:

```
# "Smart" relay host (may be null)
DSmailhost
```

Que sería equivalente a:

```
# "Smart" relay host (may be null)
DSvega.cc.upv.es
```

El otro dato que debemos conocer es con qué identidad recibimos el correo. Cuando `sendmail` recibe un mensaje, debe de poder reconocerlo como nuestro. La configuración de esta parte depende en gran medida de cómo se gestionen los dominios de correo en nuestra red local y de cómo esté configurado el servidor de correo central.

Supongamos que nuestra máquina es *term180.aiind.upv.es*, y que recibimos todo el correo que llegue al dominio *aiind.upv.es*, así como a nuestra máquina local (*term180.aiind.upv.es*, *term180.upv.es*, *term180* y *localhost*). Debemos especificar todas las maneras como se nos identifica de la siguiente manera en el fichero de configuración:

```
Cwlocalhost term180 term180.aiind.upv.es term180.upv.es aiind.upv.es
```

Obviamente, esto no sirve de nada si el servidor de correo de nuestra red local no nos tiene definidos como destinatarios. Veamos varios ejemplos.

Supongamos que llega un mensaje para *root@aiind.upv.es* al servidor de correo central, *vega.cc.upv.es*. Este servidor tiene su propio fichero `sendmail.cf`, con todas las reglas necesarias para encauzar correctamente el mensaje. Entre esas reglas debería de existir una que especifique que 'si llega un mensaje destinado al dominio *aiind.upv.es*, debe ser enviado a *term180.aiind.upv.es* ya que éste se ocupa del correo en ese dominio'. Por lo tanto, *vega.cc.upv.es* se conectará a nuestra máquina (a nuestro puerto 25, es decir, a nuestro `sendmail`) y nos entregará el mensaje. Como tenemos especificado que nuestro `sendmail` acepte mensajes destinados a *aiind.upv.es*, `sendmail` considerará que debe ser entregado localmente, y, después de verificar que el usuario de destino existe, lo volcara al buzón del sistema de ese usuario.

Sigamos el camino contrario. Enviemos un mensaje a una dirección externa a nuestra red, por ejemplo *linus@funet.fi*. Nuestro gestor de correo local recibe el mensaje. Como no se especifica sólo el nombre de usuario y el dominio *funet.fi* no se encuentra entre los nombres con los cuales nos identificamos para el correo, no tenemos forma de ocuparnos de ese mensaje nosotros mismos, así que se lo enviamos a la 'oficina central' (es decir, *vega.cc.upv.es*) para que lo tramite correctamente. A su vez, *vega.cc.upv.es* recibe el correo, verifica si alguno de los dominios a los cuales entrega correo

coincide con *funet.fi*. Como no es así, mandará el mensaje a otro servidor de correo superior, y se repetirá la operación entre otras máquinas hasta que el mensaje llegue a su destino.

El fichero `/etc/aliases`

El fichero `/etc/aliases` describe *aliases* de usuario usados por `sendmail`. El formato de cada línea es el siguiente:

nombre: nombre1, nombre2, nombre3...

nombre es el nombre al que queremos asignar un *alias* y *nombre1, nombre2,..., nombreN* son los *aliases* para ese nombre. Si empezamos una línea con un espacio, esa línea se considera continuación de la anterior. Las líneas que empiecen con # se consideran comentarios y son ignoradas al procesar el archivo.

Este fichero es útil cuando queremos que un mismo mensaje llegue a varias personas, o cuando queremos redireccionarlo a otro buzón de correo, ya sea local o en otro sistema. Un fichero `/etc/aliases` de ejemplo podría ser:

```
##
# Sendmail Alias File
# @(#)$Header:  aliases,v 1.6.109.1 91/11/21 12:06:14 kcs Exp $
#
# Mail to an alias in this file will be sent to the users, programs, or
# files designated following the colon.
# Aliases defined in this file will NOT be expanded in headers from
# mailx(1), but WILL be visible over networks and in headers from
# /bin/rmail(1).
# # >>>>>>>>>>> The program "/usr/bin/newaliases"must be run after
# >> NOTE >> this file is updated, or else any changes will not be
# >>>>>>>>>>> visible to sendmail.
##

  # Alias for mailer daemon
MAILER-DAEMON : root

  # RFC 822 requires that every host have a mail address "postmaster"
postmaster :  root

  # Aliases to handle mail to msgs and news

  # System Admistration aliases
operator :  root
uucp :  root
daemon :  root

  # trap decode to catch security attacks
decode :  root
```

En el ejemplo anterior vemos que si llega un mensaje al usuario *postmaster* éste se redirecciona al buzón de correo del *root*. De este modo el administrador del sistema no tiene que estar pendiente de varios buzones, y recibe todo el correo administrativo en el mismo buzón.
Algunos ejemplos de *aliases* que nos pueden ser útiles son:

```
root : administrador, root
```

Todo mensaje que se reciba para *root* se envía también al usuario *administrador*.

```
alumnos : paco, jose, toni, alberto
```

Si se recibe un mensaje para *alumnos* se manda a los usuarios *paco*, *jose*, *toni* y *alberto*. Esto es útil cuando queremos mandar un mismo mensaje a un grupo de usuarios. Del mismo modo podríamos crear un *alias* para todos los usuarios del sistema (pero si se borra o añade algún usuario habrá que modificar el *alias* a mano).

```
pepe :  jose@maquina.remota
```

En este caso si recibimos un mensaje para *pepe* lo reenviaremos a la dirección de correo especificada (*jose@maquina.remota*).

Capítulo 6

Seguridad

6.1 Seguridad física de la máquina

Con el término genérico *seguridad física* solemos referirnos a todos los riesgos que un equipo informático, ya sea Unix o no, soporta dentro del entorno en que está ubicado; más claramente, nos estamos refiriendo a la integridad de nuestro *hardware*. Este es un tema que en muchas ocasiones se suele olvidar, o que incluso provoca risa al oir hablar de ciertos riesgos potenciales. Sin embargo, si un sistema no posee un mínimo de seguridad física, no importa el nivel de seguridad lógica que tenga: sin duda se trata de un sistema muy vulnerable, en el que nunca debemos confiar para guardar nuestra información.

Un ejemplo de amenaza física a nuestro sistema puede ser un incendio o una inundación. En definitiva, cualquier catástrofe natural que pueda dañar el *hardware* del equipo. Debemos considerar la instalación de sistemas detectores de humo y extintores, así como la resistencia a un posible terremoto tanto del *hardware* como de la sala o edificio donde esté ubicada la máquina.

Sin entrar en estas catástrofes, algo tan simple como el polvo también pueden causar daños a los equipos; es un elemento abrasivo que acorta la vida útil de medios magnéticos y ópticos, como discos o cintas de backup. Por eso debemos aislar estos elementos de cualquier suciedad guardándolos en lugares apropiados. Además de esto, es común que la suciedad se acumule en los sistemas de ventilación, llegando en ocasiones a bloquear el flujo de aire: las máquinas se pueden calentar mucho, y al ser el polvo un buen conductor eléctrico, puede causar cortocircuitos.

Los ordenadores son también muy sensibles a los picos de la corriente eléctrica. Es conveniente que todos los equipos informáticos dedicados a actividades serias (servidores, sistemas de cálculo, sistemas con bases de datos...) estén conectados a un equipo de supresión de picos de corriente, para de esta manera reducir la posibilidad de daños; del mismo modo es recomendable el uso de un SAI (Servicio de Alimentación Ininterrumpida). Así, siempre nos será posible seguir suministrando energía a nuestro equipo ante un corte de la electricidad, al menos durante el tiempo necesario para detener nuestro sistema correctamente: Unix es capaz de comunicarse con el SAI y detectar un corte del suministro, así como una restauración de la corriente (en el caso de Linux, mediante el demonio `powerd`).

Otro aspecto de la seguridad informática física radica en la prevención de acceso por parte de personas no autorizadas a la ubicación física del ordenador. Si cualquiera puede entrar en la sala de ordenadores, sentarse delante de una consola y manipularla sin que nadie lo controle, realmente tendremos un grave problema de seguridad. El control de acceso a los ordenadores hace que sea más difícil que alguien robe o dañe los datos, o incluso el propio equipo. Es conveniente establecer políticas de acceso a las instalaciones informáticas, en función del grado de seguridad requerido: siempre hemos de tener presente que abrir una CPU y extraer un disco duro es cuestión de segundos

para un intruso.

Finalmente, hemos de hablar de accidentes, atentados, y riesgos catastróficos o poco probables. Como ejemplo de accidente, podemos imaginarnos a nosotros mismos derramando una taza de café en la máquina, que puede llegar a dañar seriamente el *hardware* (esto es mucho más común de lo que en principio podríamos imaginar); hemos de tener cuidado con este tipo de descuidos, y recordar que decir *"no lo hice a propósito"* no va a recuperar al sistema.

El caso de los atentados depende en gran medida de la ubicación y uso de la máquina; obviamente si trabajamos en un sistema de seguridad de un ministerio o de un partido político corremos más riesgo de atentados que si lo hacemos en una universidad. Sin embargo, en este último caso podemos considerar como *atentado* un ataque de locura de un estudiante enfadado con el sistema donde hace prácticas que comienza a pegar patadas al equipo (esto tampoco es tan extraño, hasta el punto que en Internet podemos encontrar incluso manuales - más o menos en clave de humor - de cómo golpear a un ordenador para dañarlo o destruirlo más eficazmente...).

Por último, tenemos las catástrofes o los riesgos poco probables, contra los que, por definición, no se suelen tomar medidas: si en la ubicación física del sistema cae una bomba atómica, o un país enemigo lanza un misil contra el edificio, poco podemos hacer; esto se considera como catástrofe o riesgo poco probable en un sistema de investigación, pero no así en un sistema militar (al menos teóricamente, nadie sabe con certeza lo que sucede en los sistemas informáticos militares, excepto los propios militares). Otro buen ejemplo de catástrofe sería que los directivos de nuestra organización decidan sustituir todos los sistemas Unix por otro sistema operativo para PC muy conocido (ejemplo de catástrofe extraído de [Gar91]). Finalmente, una amenaza poco probable en cualquier situación puede ser la abducción del equipo por parte de fuerzas extraterrestres; esta situación, que provoca risa, es el típico riesgo contra el que nadie toma, o nadie puede tomar, medidas.

6.2 Shadow Password

Uno de los puntos más débiles de la seguridad de sistemas Unix reside en las contaseñas de los usuarios y en el archivo `/etc/passwd` que las contiene. En principio, las contraseñas cifradas se almacenan en este fichero, que ha de tener de forma obligada un permiso de lectura para todos los usuarios por cuestiones en las que no vamos a entrar en profundidad, ya que desde un simple `ls` hasta el programa `login`, que permite el acceso al sistema, utilizan la información del archivo de claves en la mayoría de Unices.

Al mostrarse las contraseñas cifradas de todos los usuarios, y aunque el criptosistema utilizado por Unix para encriptar las claves es irreversible (una variante del criptosistema de clave privada DES, *Data Encryption Standard*, el más utilizado en todo el mundo a pesar de su seguridad, puesta muchas veces en entredicho), una forma de ataque propia de muchos piratas novatos (no les llamemos *hackers*) consiste en conseguir de una u otra forma este archivo y ejecutar sobre él un programa denominado *adivinador* (*Crack, Jack* o *John The Ripper* son los más utilizados). Estos programas realizan un ataque llamado de fuerza bruta sobre el *password* cifrado: cifran combinaciones de letras y las comparan con la contraseña encriptada; si coinciden, se ha adivinado un *password*. Generalmente las combinaciones no son aleatorias, sino que se usan palabras de diccionario o combinaciones de éstas (ahora sabemos por qué *patata, internet*, o *pepe64*, NO son *passwords* aceptables).

Por todo esto, se han diseñado alternativas a este sistema que no permitan a los usuarios leer directamente el archivo de *passwords*. La más conocida de ellas es la llamada **shadow password**.

En sistemas con *shadow*, el archivo `/etc/passwd` no contiene ninguna clave, sino un símbolo (normalmente $*$ o $+$) en el campo correspondiente. Las claves reales, cifradas, se guardan en `/etc/shadow`, un archivo que sólo el superusuario puede leer. De esta forma, aunque un intruso

a	Cambio de clave obligado por el sistema.
b	Periodo sin posibilidad de cambiar la contraseña.
c	Periodo de cambio voluntario.
d	Periodo de cambio obligatorio.
e	Si al finalizar el periodo anterior el usuario no ha cambiado su clave, se le deniega el acceso al sistema (por norma general).

Figura 6.1: Envejecimiento de contraseñas

consiga el archivo `/etc/passwd` no podrá intentar un ataque contra las contraseñas encriptadas para romper la seguridad de la máquina.

Una entrada típica en el archivo `/etc/passwd` de un sistema *shadow* puede ser

```
toni:*:1001:101:Toni Villalon:/home/toni:/bin/bash
```

Vemos que en el segundo campo, el reservado a la clave encriptada, no aparece nada que el intruso pueda violar. La clave cifrada está guardada en `/etc/shadow`, un archivo con entradas de la forma

```
toni:LEgPN8jqSCMCg:10322:0:99999:7:::
```

En el primer campo se guarda el *login* y en el segundo la contraseña cifrada del usuario. El resto de campos se utilizan para lo que se denomina *Aging Password* o envejecimiento de contraseñas, técnica que da a las claves un período de vida (en el caso más extremo una contraseña servirá simplemente para una única sesión, *One time passwords*, habiendo de cambiarla en la siguiente). Gracias a esto, se consigue que un potencial intruso que haya capturado una clave no pueda acceder de forma indeterminada al sistema.

Cada cierto tiempo, el sistema va a ordenar a cada usuario cambiar su contraseña. Después de cambiarla, existirá un periodo en el que no la podrá volver a cambiar, para evitar que vuelva a poner la vieja inmediatamente después de cambiarla. Después de este periodo, se permite un cambio de forma voluntaria durante otro intervalo de tiempo, al finalizar el cual se obliga al usuario de nuevo a realizar un cambio. Gráficamente, lo podemos ver como se muestra en la figura 6.2.

El significado concreto de cada uno de los campos de `/etc/shadow` lo podemos ver en la página de manual de *shadow*, en la sección 5. Como hemos visto, no todos los campos son obligatorios.

6.3 Ataques externos al sistema

6.3.1 Introducción

Desde el primer momento en el que tenemos un equipo conectado a una red informática (bien sea de área local o más extensa) cualquier otra persona conectada a dicha red puede acceder a nuestro sistema. Dicho acceso puede tener diferentes efectos:

- **Acceso de lectura**
  Lectura o copia de información de nuestra máquina o nuestra red local.

- **Acceso de escritura**
  Escritura o destrucción de datos de nuestra máquina o red local (incluyendo la posibilidad de introducción de programas troyanos, canales cubiertos, virus, y puertas traseras).

- **Negación de servicio**
  Impedir el uso normal de los recursos de nuestro sistema; por ejemplo inutilizar la red consumiendo todo el ancho de banda, abortar procesos consumiendo la memoria, etc.

6.3.2 Negación de servicio (Denial of Service)

Como ya sabemos, nuestra máquina ofrece una serie de servicios a los que se puede acceder de forma remota. Una negación de un servicio ocurre cuando se sobrecarga accediendo a él de forma continua y excesiva. Por ejemplo, conectar a un servicio determinado, y antes de finalizar dicha conexión realizar una nueva al mismo, así repetidamente y de forma contínua. Algunos servicios que típicamente pueden verse afectados son el servidor de correo (sendmail), servidor de ftp, servidor de acceso remoto (demonio de telnet), servidor de web...

Un ejemplo de cómo realizar un DoS puede ser:

```
while :  ; do
telnet victima.del.ataque 25 &
done
```

Con esto hacemos que se ejecuten múltiples programas *sendmail* en la máquina remota, con el consiguiente uso de memoria y CPU.
Otro ejemplo de DoS comúnmente usado es usando el mandato `finger`: antiguas versiones del demonio *finger* nos permitían realizar un *finger* a una máquina A desde otra máquina B, de este modo:

```
finger @maquina_a@maquina_b
```

En este caso *maquina_b* ejecuta un *finger* local (usando sus recursos del sistema) a *@maquina_a*. Por lo tanto con la instrucción

```
finger @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@victima.del.ataque
```

Conseguimos que la máquina *victima.del.ataque* realice *fingers* a sí misma, tantas veces como símbolos '@' hayamos especificado. Cuantos más especifiquemos, más recursos de la máquina *victima.del.ataque* se usarán.

La forma de evitar estos ataques es controlando el acceso a nuestros servicios, ya sea limitando el número de peticiones que puedan recibir (servidor de ftp, servidor de web), eliminando del sistema servicios que no se usan (*echo, daytime, chargen, finger* externo...) o bien restringiendo el acceso a ese servicio a máquinas "fiables" (servidor de correo, servidor de acceso remoto...). Otra de las soluciones que podemos usar es instalar versiones de los programas que estén preparadas para combatir estos ataques (por ejemplo, hay versiones recientes del demonio de *finger* que no permiten redireccionar una consulta).

6.3.3 Acceso a servicios

Como ya vimos en el tema de instalación de la red, cuando el sistema recibe una petición de un servicio, este responde ejecutando un determinado programa, con unos determinados privilegios (*root* en la mayoría de los casos: *ftp, sendmail, pop3...*). Por lo tanto, cuando una persona accede de forma externa a uno de los servicios de nuestra máquina, lo que realmente esta haciendo es ejecutar el programa correspondiente, con sus correspondientes privilegios. Desgraciadamente la Informática no es una ciencia exacta y cualquier programa puede tener fallos en la programación (también llamados *bugs*). Estos fallos pueden hacer que el programa se comporte de manera

anómala permitiendo que se obtengan resultados no deseados. Dichos *bugs* están constantemente en evolución, por ello es interesante estar suscrito a listas de correo, las cuales nos informan de los fallos que se van descubriendo, con lo que podremos actuar rápidamente en consecuencia. También es recomendable usar versiones actualizadas de los programas 'importantes' del sistema, los cuales suelen tener corregidos muchos de los *bugs* mencionados anteriormente con sus actualizaciones.

Es sencillo detectar qué servicios ofrece una máquina cualquiera, con la ayuda de escaneadores de puertos (programas que recorren uno a uno todos los puertos de una máquina determinando si están activos o no). Uno de los más conocidos es el programa `strobe`:

```
maquina:~# strobe localhost
strobe 1.03 (c) 1995 Julian Assange (proff@suburbia.net).
localhost                       echo                 7/tcp
localhost                       discard              9/tcp sink null
localhost                       systat              11/tcp users
localhost                       daytime             13/tcp
localhost                       netstat             15/tcp
localhost                       chargen             19/tcp ttytst source
localhost                       ftp                 21/tcp
localhost                       telnet              23/tcp
localhost                       smtp                25/tcp mail
localhost                       finger              79/tcp
localhost                       www                 80/tcp http  #
WorldWideWeb HTTP
localhost                       auth               113/tcp tap ident
authentication
localhost                       nntp               119/tcp readnews untp #
USENET News Transfer Protocol
localhost                       login              513/tcp
localhost                       shell              514/tcp cmd           # no
passwords used
localhost                       printer            515/tcp spooler
# line printer spooler
localhost                       unknown           6000/tcp unassigned
```

De nuevo la única forma de evitar el abuso de los servicios que ofrecemos es limitando el acceso a estos, ya sea restringiendo el conjunto de máquinas que puedan acceder a dichos servicios o eliminando del sistema servicios que no se usan o que no sean imprescindibles. Por ejemplo, limitando el acceso a nuestro servicio `sendmail` a la máquina que gestiona todo el correo de nuestra subred (normalmente el resto de máquinas no necesitan acceder a nuestro sistema y mandan sus mensajes a través del servidor de correo central), o recortando el acceso a nuestro servidor de impresión (demonio `lpd`) únicamente a las máquinas que lo necesiten.

6.3.4 Información sobre el sistema

A través de conexiones remotas se puede llegar a obtener información acerca de nuestro sistema, como datos de los programas instalados y sus versiones, información acerca de los usuarios,etc:

```
andercheran:~# telnet victim.com
Trying 122.69.15.1...
Connected to victim.com.
Escape character is '^]'.

Linux 2.0.27 (victim.com) (ttyp4)
^^^^^^^^^^^^
victim login:
```

De esta conexión deducimos que la máquina funciona bajo Linux, y que está usando la versión 2.0.27 del kernel.

```
andercheran:~# ftp victim.com
Connected to victim.com.
220 victim FTP server (Version wu-2.4(1) Tue Dec 5 20:51:15 CST 1995) ready.
                      ^^^^^^^^^^^^^^
Name (victim.com:root):
```

De esta conexión deducimos que servidor *ftp* se usa (`wu-ftpd`) y su versión.

```
andercheran:~# telnet victim.com 25
Trying 122.69.15.1...
Connected to victim.com.
Escape character is '^]'.
220 victim.com ESMTP Sendmail 8.8.4/8.8.4; Fri, 15 May 1998 03:15:27 +0200
                     ^^^^^^^^^^^^^^^^^^^^
vrfy root
250 root <root@victim.com>
vrfy admin
550 admin... User unknown
vrfy lp
250 lp <lp@victim.com>
vrfy juan
250 Juan Fabregat <juan@victim.com>
```

Con esta conexión obtenemos la versión del servidor de correo que usa esta máquina (`sendmail 8.8.4`) y también información sobre los usuarios (existen las cuentas *root* y *lp*, pero no *admin*, y la cuenta *juan* existe, a nombre de 'Juan Fabregat').

```
andercheran:~# finger @victim.com
[victim.com]

Login    Name             Tty  Idle  Login Time   Office     Office  Phone
juan     Juan Fabregat    p1  6:40  Feb  2 15:06 (victim.com)
root     root             1         Feb  2 15:06
root     root             p0  6:36  Feb  2 15:06 (:0.0)
root     root             p2 11:08  Feb  2 15:07 (:0.0)
root     root             p3  6:50  May 14 18:20 (:0.0)

andercheran:~# finger juan@victim.com
[victim.com]

Login: juan                             Name: Juan Fabregat
Office: D1, 963865541                   Home Phone: 929613321
            ^^^^^^^^^                               ^^^^^^^^^
Directory: /home/juan                   Shell: /bin/csh
On since Tue Feb  2 15:06 (MET DST) on ttyp1 from victim.com
   6 hours 39 minutes idle
Mail last read Fri May 15 02:21 1998 (MET DST)
No Plan.
```

Gracias a `finger` podemos saber quién está conectado actualmente a la máquina, e incluso los datos personales de algunos usuarios (incluyendo el numero de teléfono particular, en algunos casos). Otros servicios 'interesantes' a la hora de conocer información acerca de una máquina son *systat* (puerto 11), que lista todos los procesos de la máquina, o *netstat* (puerto 15) que muestra todas las conexiones activas en el sistema.

Otra vez, recordamos que la mejor manera de evitar ofrecer demasiados datos es limitar el acceso a los servicios ofrecidos en nuestra máquina, o incluso recompilar nuestros programas, evitando que den información innecesaria (la versión del programa, fecha de compilación, etc...). Podemos llegar al extremo de eliminar servicios que consideremos superfluos o que ofrecen demasiada información acerca de nuestro sistema.

6.4 Ataques internos al sistema

6.4.1 Introducción

Tanto si acceden a nuestro sistema de una forma remota como si tenemos usuarios legítimos en nuestra máquina, debemos tener la seguridad de que no podrán acceder a información que no les concierne, obtener privilegios especiales en la máquina o simplemente acabar con los recursos de sistema haciendo que este se vuelva inutilizable.

6.4.2 Demonios y SUID

Un archivo setuidado o setgidado no es más que un archivo con el bit *setuid* o *setgid* activo, lo cual significa que quien lo ejecute tendrá durante un cierto tiempo privilegios del dueño del archivo o del grupo del dueño, dependiendo de si es setuidado o setgidado.

Este tipo de archivos es una gran fuente de problemas para la integridad del sistema, pero son completamente necesarios. Por ejemplo, imaginando que la orden `/bin/passwd` no tuviera el bit *setuid activo*, no permitiría cambiar la clave de acceso al no poder escribir en el fichero de *passwords*, por lo que cada vez que un usuario quisiese cambiar su contraseña debería recurrir al administrador. Esto sería una gran carga para este en un sistema Unix medio, con aproximadamente 100 usuarios. Casi todos los medios de acceso no autorizado a una máquina se basan en algún fallo de la programación de estos archivos. No es que los programadores de Unix sean incompetentes, sino que simplemente es imposible no tener algún pequeño error en las miles de lineas de código que componen un programa.

Como ejemplo de la compartición de privilegios y del riesgo que ello conlleva, si alguien tuviera un *shell* setuidado con el UID de otro usuario, habría conseguido un intérprete de órdenes con todos los privilegios de ese usuario, lo cual le permitiría borrar sus archivos, leer su correo, etc...

Para localizar los archivos setuidados/setgidados podemos utilizar la orden `find` de la siguiente forma:

```
rosita:~# find / -type f -perm +6000 -exec ls -al {} \;
```

Por otra parte tenemos los programas que pueden ser usados por los usuarios y que se ejecutan con privilegios especiales, típicamente los de planificación de tareas (`at, crontab`). Si estos programas están mal diseñados se podría abusar de ellos, haciendo que ejecuten ordenes arbitrarias con los privilegios del usuario que los arranca (usuario *bin* en el caso de `atrun`, usuario *root* en el caso de `crond`).

La mejor manera de prevenir el mal uso de los programas setuidados/setgidados es reduciendo su numero en el sistema, quitando el bit de *setuid* (con la orden `chmod -s`) en todos aquellos que consideremos innecesarios o que no nos parezcan fiables. En un sistema Linux recién instalado podemos encontrar un gran número de ejecutables setuidados, llegando a ser más de cincuenta los ficheros de ese tipo. Una vez revisados y eliminados aquellos que no son imprescindibles podemos llegar a tener menos de cinco. En cuanto a los demonios de planificación de tareas, hay que asegurarse de usar las últimas versiones revisadas y libres de *bugs*, o incluso limitar el uso de los mandatos `crontab` y `at` a ciertos usuarios (o a todos), como ya vimos en el tema de automatización de tareas.

6.4.3 Acceso a información

En un sistema Unix existen gran número de archivos que contienen información **delicada** acerca del propio sistema, dicha información relevante debería ser únicamente accesible al administrador. También es importante denotar que cada usuario es poseedor de su propia información, la cual no debería estar al alcance de los demás usuarios, y es tarea del administrador del sistema el conservar la privacidad de ambos tipos de datos.

Un ejemplo de tipos de archivos con información **delicada** son todos los relacionados con información del sistema, los cuales encontraremos en el directorio `/var/adm` y serán tratados más adelante (`syslog, messages, sulog`...). También los archivos relacionados con la codificación de las claves de acceso de los usuarios existentes en nuestro sistema (`/etc/shadow`) se consideran información privilegiada.

Ejemplos de ficheros de usuarios que no deberían estar al alcance de todos los demás son, primordialmente, los mensajes de correo almacenados en `/var/spool/mail/`.

La única forma de asegurarnos de que nuestros usuarios tienen acceso única y exclusivamente al tipo de información que les corresponde es haciendo una verificación de que los permisos de los archivos importantes son los adecuados, y utilizando apropiadamente la orden `chmod`.

6.4.4 DoS

Ya hemos tratado con anterioridad el concepto de Negación de Servicio (*Denial of service*). De la misma manera que existen negaciones de servicio externas, también existen métodos para impedir el uso normal de nuestros recursos locales, consumiendo CPU o memoria, pero estando conectados dentro del sistema.
Un ejemplo de negación de servicio interno podría ser el siguiente:

```
while :  ; do
mkdir .xxx
cd .xxx
done
```

Este ejemplo nos muestra como (si el usuario no tiene impuesta una cuota de disco) se pueden crear directorios anidados de nombre `.xxx` de forma continua y bajo un bucle infinito, consumiendo CPU y espacio en disco.

Otro ejemplo de como consumir recursos del sistema seria un programa simple en C:

```
main(){
fork();
main();
}
```

Con este programa, y si el usuario no tiene cuota de procesos, crearíamos multitud de procesos, consumiendo CPU y llegando al punto de bloquear el sistema. Para evitar este tipo de ataques podemos limitar (ya sea a nivel de núcleo o en el entorno de cada usuario) el espacio en disco, el número de procesos y la cantidad de memoria que puede usar un usuario, de manera que no pueda consumir todos los recursos del sistema.

6.5 Registro de actividades

6.5.1 syslog/syslog.conf

El demonio `syslogd` (*Syslog Daemon*) se encarga de guardar informes sobre el funcionamiento de la máquina. Recibe mensajes de las diferentes partes del sistema (núcleo, programas,...) y los envía y/o almacena en diferentes localizaciones (locales o remotas) siguiendo un criterio definido en el fichero de configuración (`/etc/syslog.conf`). Este programa se lanza automáticamente al arrancar la máquina.

El programa `syslogd` carga por defecto la configuración guardada en el archivo `/etc/syslog.conf`, donde se especifican las reglas a seguir para gestionar el almacenamiento de mensajes del sistema. Las líneas de este archivo que comienzan por # son comentarios, con lo cual son ignoradas, así como las líneas en blanco. Si ocurriera un error al interpretar una de las líneas se ignoraría la línea entera.
Cada regla tiene dos campos: un campo de selección y un campo de acción, separados por espacios o tabuladores:

- El campo de selección está formado, a su vez, de dos partes: una del servicio que envía el mensaje y otra de su prioridad, separadas por un punto ('.'). Ambas partes son indiferentes a mayúsculas y minúsculas. La parte del servicio contiene una de las siguientes palabras clave: *auth, auth-priv, cron, daemon, kern, lpr, mail, mark, news, security* (equivalente a *auth*), *syslog, user, uucp* y *local0* hasta *local7*. Esta parte especifica el 'subsistema' que ha generado ese mensaje (todos los programas relacionados con el correo generarán mensajes ligados al servicio *mail*).

  La prioridad está compuesta de uno de los siguientes términos, en orden ascendente: *debug, info, notice, warning, warn* (equivalente a *warning*), *err, error*(equivalente a *err*), *crit, alert, emerg*, y *panic* (equivalente a *emerg*). La prioridad define la gravedad o importancia del mensaje almacenado. Todos los mensajes de la prioridad especificada y superiores son almacenados de acuerdo con la acción requerida.

  Además de los anteriormente mencionados términos, el demonio *syslogd* emplea los siguientes caracteres especiales:

  – '∗' (asterisco)
  Empleado como 'comodín' para todas las prioridades y servicios anteriores, dependiendo de dónde son usados (si antes o después del carácter de separación '.'):

  ```
  # Guardar todos los mensajes del servicio mail en /var/adm/mail
  #
  mail.* /var/adm/mail
  ```

  – ' ' (blanco, espacio, nulo)
  Indica que no hay prioridad definida para el servicio de la línea almacenada.

  – ',' (coma)
  Con este carácter es posible especificar múltiples servicios ,con el mismo patrón de prioridad, en una misma línea. Es posible enumerar cuantos servicios se quieran:

  ```
  # Guardar todos los mensajes mail.info y news.info en
  # /var/adm/info
  mail,news.=info /var/adm/info
  ```

  – ';' (punto y coma)
  Es posible dirigir los mensajes de varios servicios y prioridades a un mismo destino, separándolos por este carácter:

  ```
  *.=info;*.=notice /var/log/messages
  ```

  – '=' (igual)
  De este modo solo se almacenan los mensajes con la prioridad exacta especificada y no incluyendo las superiores:

  ```
  # Guardar todos los mensajes críticos en /var/adm/critical
  #
  *.=crit /var/adm/critical
  ```

- '!' (exclamación)
  Preceder el campo de prioridad con un signo de exclamación sirve para ignorar todas las prioridades, teniendo la posibilidad de escoger entre: la especificada (*!=prioridad*) y la especificada mas todas las superiores (*!prioridad*). Cuando se usan conjuntamente los caracteres = y *!*, el signo de exclamación *!* debe preceder obligatoriamente al signo igual =, de esta forma: *!=*.

```
# Guardar mensajes del kernel de prioridad info, pero no de
# prioridad err y superiores
# Guardar mensajes de mail excepto los de prioridad info
kern.info;kern.!err /var/adm/kernel-info
mail.*;mail.!=info /var/adm/mail
```

- El campo de acción describe el destino de los mensajes, que puede ser :
  - Un fichero plano
    Normalmente los mensajes del sistema son almacenados en ficheros planos. Dichos ficheros han de estar especificados con la ruta de acceso completa (empezando con `/`).

    Se puede preceder cada entrada con el signo menos, - para omitir la sincronización del archivo (vaciado del *buffer* de memoria a disco). Puede ocurrir que se pierda información si el sistema cae justo después de un intento de escritura en el archivo. A pesar de este problema se puede conseguir una mejora en la velocidad, especialmente si se están ejecutando programas que mandan muchos mensajes al demonio `syslog`.

```
# Guardamos todos los mensajes de prioridad critica en "critical"
#
*.=crit /var/adm/critical
```

  - Un terminal (o la consola)
    También tenemos la posibilidad de enviar los mensajes a terminales. De este modo podemos tener uno de los terminales 'dedicado' a listar los mensajes del sistema, y pueden ser consultados con solo cambiar a ese terminal:

```
# Enviamos todos los mensajes al tty12 (ALT+F12 en Linux) y todos
# los mensajes críticos del núcleo a consola
# *.* /dev/tty12
kern.crit /dev/console
```

  - Una máquina remota
    Se pueden enviar los mensajes del sistema a otra máquina, de manera a que sean almacenados remotamente. Esto es útil si tenemos una máquina 'segura' conectada a la red, en la que podamos confiar, de esa manera se guardaría allí una copia de los mensajes de nuestro sistema y no podrían ser modificados en caso de que alguien entrase en nuestra máquina. Esto es especialmente útil para detectar usuarios "ocultos" en nuestro sistema (un usuario malicioso que ha conseguido los suficientes privilegios para ocultar sus procesos o su conexión):

```
# Enviamos los mensajes de prioridad warning y superiores al
# fichero 'syslog' y todos los mensajes (incluidos los
# anteriores) a la máquina 'secure.org'
#
*.warn /usr/adm/syslog
*.* @secure.org
```

  - Unos usuarios del sistema (si están conectados)
    Se especifica la lista de usuarios que deben recibir un tipo de mensajes simplemente escribiendo su *login*, separados por comas:

```
# Enviamos los mensajes con la prioridad 'alert' a root y toni
#
*.alert root, toni
```

– Todos los usuarios que estén conectados
Los errores con una prioridad de 'emergencia' se suelen enviar a todos los usuarios que estén conectados al sistema, de manera que se den cuenta de que algo va mal:

```
# Emergency messages will be displayed using wall
#
*.=emerg *
```

6.5.2 Informes del sistema

Como hemos visto, el demonio `syslog` guarda información de vital importancia para el administrador de la máquina. Gracias a esta información se pueden descubrir fallos en el sistema (ya sean de tipo humano o de la propia configuración de la máquina, arranque de ésta, etc) o posibles intrusiones externas. Por ello es recomendable revisar periódicamente (a intervalos bastante cortos) los informes. Normalmente estos archivos se encuentran en los directorios `/var/adm` y `/var/log` (este último suele ser un enlace a `/var/adm`). Los más comunes son (aunque dependiendo de cada sistema pueden aparecer otros múltiples):

- `debug`
Archivo donde los programas guardan la información llamada comúnmente de "depuración". Cualquier programa que tengamos instalado puede mandar líneas de información a este archivo:

```
Dec 24 00:02:17 localhost kernel:  VFS: Disk change
detected on device 02:00
Apr 26 00:25:03 andercheran sshd[16885]:  debug:  sshd version 1.2.22
[i586-unknown-linux]
Apr 26 00:25:03 andercheran sshd[16885]:  debug:  Initializing random
number
generator; seed file /usr/local/etc/ssh_random_seed
```

- `lastlog`
Fichero con formato especial. Base de datos donde se guardan la información de la última conexión de cada usuario del sistema (fecha, hora, línea y origen de la conexión efectuada). El tamaño de este fichero depende de la cantidad de usuarios de la máquina, a mayor número, mayor espacio ocupa.

  Esta información se visualiza al acceder al sistema o al consultar la información de un usuario no conectado mediante la orden `finger`:

```
andercheran login:  alumno
Password:
Linux 2.0.33.
Last login: Wed May 13 13:26:51 on ttyq5 from algol.cc.upv.es.
You have mail.

andercheran: $ finger toni
Login:  toni Name:  Antonio Villalon
Directory:  /home/toni Shell:  /bin/bash
Last login Wed May 13 01:53 (MET DST) on ttyp1 from
pleione.cc.upv.es
New mail received Wed May 13 14:04 1998 (MET DST)
Unread since Wed May 13 01:53 1998 (MET DST)
No Plan.
```

- `faillog`
  Equivalente al archivo anterior (`lastlog`), pero guarda la información de la última conexión fallida de cada usuario. Una conexión se considera fallida cuando la clave de acceso o *password* del usuario no es introducida correctamente).

  Esta información se visualiza al acceder al sistema si ha habido algún intento fallido de conexión:

  ```
  andercheran login:  alumno
  Password:
  Linux 2.0.33. 1 failure since last login. Last was 14:39:41 on ttyp9.
  Last login:  Wed May 13 14:37:46 on ttyp9 from antares.cc.upv.es.

  andercheran: $
  ```

- `messages`
  Fichero en formato texto en el cual se almacenan datos 'informativos' de los programas. Suelen ser mensajes de baja prioridad, destinados más a informar que a avisar de sucesos importantes:

  ```
  Mar 24 16:38:52 andercheran wu.eftpd[15619]:  connect from
  proxy.rediris.es
  Mar 24 16:38:53 andercheran ftpd[15619]:  USER anonymous
  Mar 24 16:38:53 andercheran ftpd[15619]:  PASS squid@
  Mar 24 16:38:53 andercheran ftpd[15619]:  TYPE Image
  Mar 24 16:39:26 andercheran wu.eftpd[15640]:  connect from
  Piacenza4-20.tin.it
  Mar 24 16:40:03 andercheran in.telnetd[15669]:  connect from
  term00.etsii.upv.es
  Apr 4 09:56:34 andercheran login[8227]:  ROOT LOGIN on 'tty3'
  Apr 4 09:58:48 andercheran passwd[7580]:  password for 'cunix00'
  changed by user 'cunix00'
  ```

- `sulog`
  Archivo en formato texto que guarda la información sobre la utilización de la instrucción `su` (orden que cambia el usuario actual por otro a especificar) argumentando fecha, hora, resultado de la operación ('+' y '-') línea de conexión desde la que se efectuó y operación realizada):

  ```
  SU 12/11 04:37 + ttyce toni-operador
  SU 01/15 13:52 + ttyce monica-root
  SU 03/19 20:42 + ttyce root-usuario
  SU 05/12 19:10 + ttyq4 root-pepe
  SU 05/12 21:12 - ttyq9 usuario-operador
  ```

- `syslog`
  Archivo en formato texto, similar al fichero `messages`, pero que contiene información de mayor relevancia para la seguridad del sistema. Como por ejemplo todos los errores de conexión, autenticación, acceso... a la máquina (aunque estos mensajes pueden ser redirigidos a otro archivo por medio del fichero de configuración `/etc/syslog.conf`):

  ```
  Mar 24 16:40:12 andercheran login[15670]:  invalid password for
  'JOMONRA' on 'ttyp6' from 'term00.etsii.upv.es'
  Mar 25 09:27:35 andercheran login[3068]:  invalid password for
  ```

```
‘vbarbera’ on ‘ttyq3’ from ‘daind03.etsii.upv.es’
Mar 25 16:04:50 andercheran in.telnetd[4008]:  refused connect from
pc-433.tlc.upv.es
Mar 25 16:05:08 andercheran in.telnetd[4027]:  refused connect from
pc-433.tlc.upv.es
Mar 25 17:03:07 andercheran su[7050]:  Authentication failed for root
Mar 27 06:44:58 andercheran sshd[6487]:  refused connect from
oeiras25.dial.telenet.pt
Mar 27 06:44:58 andercheran in.telnetd[6488]:  refused connect from
oeiras25.dial.telenet.pt
Mar 27 06:44:58 andercheran sendmail[6489]:  refused connect from
oeiras25.dial.telenet.pt
Mar 27 06:44:58 andercheran in.pop3d[6490]:  refused connect from
oeiras25.dial.telenet.pt
Mar 27 06:45:04 andercheran in.rlogind[6497]:  refused connect from
oeiras25.dial.telenet.pt
Mar 30 18:14:05 andercheran in.rlogind[16097]:  refused connect from
johermo@158.42.54.13
May 11 16:58:28 andercheran kernel:  end_request:  I/O
error, dev 02:00, sector 0
```

- `wtmp`
  Fichero con formato especial donde se guarda la lista de conexiones efectuadas a la máquina. El contenido de este archivo se visualiza con la orden `last`:

```
andercheran:~$ last
alumno    ftp          pc0708.dsic.upv. Wed May 13 16:15 - 16:15  (00:00)
ftp       ftp          proxy.rediris.es Wed May 13 16:14 - 16:39  (00:25)
toni      tty3                          Wed May 13 16:10 - 16:23  (00:12)
operador  ttypb        158.42.112.2     Wed May 13 15:55 - 15:59  (00:04)
toni      tty3                          Wed May 13 15:35 - 15:49  (00:14)
ircd      ttyq0        venus02.egb.uv.e Wed May 13 15:24 - 16:20  (00:56)
vbarbera  ttype        egaja3.etsii.upv Wed May 13 15:24 - 15:27  (00:02)
vbarbera  ttype        egaja3.etsii.upv Wed May 13 15:20 - 15:21  (00:01)
mbenet    ttyp6        pc0708.dsic.upv. Wed May 13 14:49   still logged in
mbenet    ttyp6        pc0717.dsic.upv. Wed May 13 14:42 - 14:48  (00:06)
```

- `utmp -> /var/run/utmp`
  Archivo con el mismo formato que el anteriormente mencionado, pero contiene las conexiones que se están efectuando en este momento a la máquina. La informacion almacenada en este fichero se utiliza por varios programas, entre ellos los mandatos `w` y `finger`.

```
andercheran:~$ w
  4:57pm  up 19:22,  4 users,  load average: 0.40, 0.59, 0.60
USER     TTY      FROM              LOGIN@  IDLE   JCPU   PCPU  WHAT
toni     tty1                      2:42pm 39.00s  0.95s  0.10s  talk sergio
toni     tty2                      2:42pm  0.00s  1.11s  0.37s  pine
mbenet   ttyp6    pc0708.dsic.upv. 2:49pm  9.00s  0.23s  0.05s  screen -r
sergio   ttyp9    brubaker.etsii.u 4:49pm  0.00s  0.29s  0.04s  screen -r

andercheran:~$ finger mbenet
Login: mbenet                           Name: Manuel Benet
Directory: /home/mbenet                 Shell: /bin/bash
On since Wed May 13 14:49 (MET DST) on ttyp6 from pc0708.dsic.upv.
   25 seconds idle
```

```
Mail last read Wed May 13 16:06 1998 (MET DST)
No Plan.
```

- `btmp`
  Fichero equivalente a `wtmp`, pero donde se almacenan las conexiones fallidas al sistema. Se visualiza con la orden `last -f btmp`:

```
andercheran:/var/adm# last -f btmp |less
abarra    ttyq4        daindmast.etsii. Wed May 13 13:03   still logged in
vbarbera  ttyq5        term00.etsii.upv Wed May 13 12:30   still logged in
jomonra   ttyp7        daind03.etsii.up Wed May 13 09:03   still logged in
mpascu    ttyp0        algol.cc.upv.es  Tue May 12 21:44   still logged in
toni      tty1                          Tue May 12 20:27   still logged in
fvalerov  ttypa        pc-128.tlc.upv.e Tue May 12 19:25   still logged in
joomonra  ttyqb        term00.etsii.upv Tue May 12 18:13   still logged in
ircd      ttyp4        venus01.egb.uv.e Tue May 12 16:16   still logged in
vbarbera  ttyp4        daind03.etsii.up Tue May 12 15:00 - 16:16  (01:15)
joo       ttyqa        term00.etsii.upv Mon May 11 13:52 - 14:00  (00:08)
convidat  ttyq5        term119.aiind.up Sat May  9 09:36 - 13:36  (04:00)
invitado  ttyq5        term119.aiind.up Sat May  9 09:36 - 09:36  (00:00)
anonymou  ttyq5        term119.aiind.up Sat May  9 09:36 - 09:36  (00:00)
root      ttyp8        term118.aiind.up Sat May  9 09:35 - 12:15 (3+02:39)
anonymou  ttyp8        term118.aiind.up Sat May  9 09:35 - 09:35  (00:00)
guest     ttyp8        term118.aiind.up Sat May  9 09:35 - 09:35  (00:00)
ftp       ttyp8        term124.aiind.up Sat May  9 09:31 - 09:35  (00:03)
exit      ttyp8        term124.aiind.up Sat May  9 09:31 - 09:31  (00:00)
cursos    ttyp8        term124.aiind.up Sat May  9 09:31 - 09:31  (00:00)
```

6.5.3 Ejemplos del fichero `/etc/syslog.conf`

- Configuración 'sencilla' (Linux Slackware 3.4):

```
# /etc/syslog.conf
# For info about the format of this file, see "man syslog.conf" (the BSD man
# page), and /usr/doc/sysklogd/README.linux.
#
# NOTE: YOU HAVE TO USE TABS HERE - NOT SPACES.
# I don't know why.
#

*.=info;*.=notice                                       /usr/adm/messages
*.=debug                                                /usr/adm/debug
*.err                                                   /usr/adm/syslog
*.*                                                     /dev/tty12

#
# This might work instead to log on a remote host:
# *                     @hostname
```

- Configuración 'compleja' [1] (Linux Debian):

[1] Léase *¿dónde se ha metido ese mensaje?*

```
#  /etc/syslog.conf     Configuration file for syslogd.
#
#                      For more information see syslog.conf(5)
#                      manpage.

#
# First some standard logfiles.  Log by facility.
#

auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
lpr.*                           -/var/log/lpr.log
mail.*                          /var/log/mail.log
user.*                          -/var/log/user.log
uucp.*                          -/var/log/uucp.log

#
# Logging for the mail system. Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                       -/var/log/mail.info
mail.warn                       -/var/log/mail.warn
mail.err                        /var/log/mail.err

# Logging for INN news system
#
news.crit                       /var/log/news/news.crit
news.err                        /var/log/news/news.err
news.notice                     -/var/log/news/news.notice

#
# Some Catch-all' logfiles.
#
*.=debug;\
        auth,authpriv.none;\
        news.none;mail.none     -/var/log/debug
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages

#
# Emergencies are sent to everybody logged in.
#
*.emerg                         *

#
# I like to have messages displayed on the console, but only on a virtual
# console I usually leave idle.
#
#daemon,mail.*;\
```

```
#       news.=crit;news.=err;news.=notice;\
#       *.=debug;*.=info;\
#       *.=notice;*.=warn       /dev/tty8

# The named pipe /dev/xconsole is for the nsole' utility.  To use it,
# you must invoke nsole' with the -file' option:
#
#    $ xconsole -file /dev/xconsole [...]
#
# NOTE: adjust the list below, or you'll go crazy if you have a reasonably
#      busy site..
#
daemon,mail.*;\
        news.crit;news.err;news.notice;\
        *.=debug;*.=info;\
        *.=notice;*.=warn       |/dev/xconsole

local2.*                -/var/log/ppp.log
```

6.6 Software de Seguridad para máquinas Unix

6.6.1 Tcp Wrappers

ftp://ftp.win.tue.nl/pub/security/

Muchos de los demonios que vienen preinstalados en el sistema no guardan suficiente información en los archivos de log... ¡algunos incluso no guardan ninguna información! En estos casos se usa un programa denominado *TCP Wrappers*, que no es más que un filtro entre la petición de un servicio y la ejecución del programa que da ese servicio. Antes de que una petición de servicio se responda, puede ser procesada por *TCP Wrappers*, que realiza varias comprobaciones, como por ejemplo, si la máquina que nos solicita ese servicio tiene permitido su uso. De esta manera podemos restringir las máquinas que pueden acceder a nuestro sistema.

Los archivos que especifican como regular las conexiones que se realizan al sistema son dos : `/etc/hosts.allow` y `/etc/hosts.deny`. En el fichero `hosts.deny` especificaremos que servicios y a que máquinas están prohibidos, mientras que en `hosts.allow` especificaremos los que están permitidos. Lo habitual es indicar `ALL:ALL` en `hosts.deny` y habilitar servicios sólamente a máquinas que realmente los vayan a utilizar, especificando ambos en `hosts.allow`, tal y como se ve a continuación.

Cuando se realiza una conexión a nuestro sistema, *TCP Wrappers* consultará en primer lugar el fichero `/etc/hosts.allow`, comprobando, línea a línea, si la máquina que conecta tiene permitido el acceso. Si se encuentra una condición que concuerde con la petición de servicio, se permitirá el acceso. Si no se encuentra, se comprobará el fichero `/etc/hosts.deny` y el acceso será denegado si la conexión cumple alguna de las condiciones especificadas en este fichero; si no, será permitida.

La sintaxis a utilizar es la misma en los dos ficheros, `/etc/hosts.deny` y `/etc/hosts.allow`. Estos ficheros pueden estar vacios o consistir de varias líneas, que se procesarán ordenadamente. Estas líneas son de la forma:

```
lista_demonios :  lista_maquinas [ :  commando_shell ]
```

`lista_demonios` es una lista de uno o más demonios, o máscaras. Los demonios se especifican con el nombre de su programa (por ejemplo `in.telnetd`, `in.fingerd`...)
`lista_maquinas` es una lista de uno o más nombres de máquinas, direcciones, patrones o máscaras

que serán comparadas con el nombre o la dirección numérica de la máquina que conecta a nuestro sistema.

Los elementos de las listas pueden estar separados por espacios o por comas. Los patrones que pueden utilizarse en la lista de máquinas son de la forma:

- Una cadena que empiece por el carácter '.' . Un nombre de máquina concordará si su parte final concuerda con el patrón especificado. Por ejemplo, el patrón *.dominio.org* concuerda con el nombre de máquina *maquina1.dominio.org* (así como con *maquina.subdominio.dominio.org*).
- Una cadena que acabe con el carácter '.' . Una dirección numérica de una máquina concordará si sus primeros campos numéricos concuerdan con la cadena especificada. Por ejemplo, el patrón *158.42.* concuerda con todas las máquinas que tengan una dirección numérica del tipo *158.42.x.x.*

También pueden utilizarse unas máscaras especiales:

ALL	Es la maáscara universal, siempre concuerda.
LOCAL	Concuerda con cualquier máquina cuyo nombre no contenga el carácter '.'.
UNKNOWN	Concuerda con cualquier máquina cuyo nombre **o** dirección numérica no se conozca.
KNOWN	Concuerda con cualquier máquina cuyo nombre **y** dirección numerica se conozca.
PARANOID	Concuerda con cualquier máquina cuyo nombre no corresponda a su dirección numérica.

Por último, se permite el uso de 'operadores' en la especificación de demonios o máquinas:

EXCEPT	Se utiliza del siguiente modo: *lista_1 EXCEPT lista_2.* Esto concuerda con cualquier elemento que esté en *lista_1* **pero** que no se encuentre en *lista_2.*

Veamos algunos ejemplos de configuración del `TCP Wrapper`:

∗ Si queremos que nuestro sistema sea 'cerrado', es decir, que el acceso esté prohibido por defecto, y que sólo se permita a máquinas a las que se autorize explícitamente el acceso:

La política por defecto (denegar el acceso), se implementa con una regla simple en el fichero `/etc/hosts.deny`:

```
/etc/hosts.deny:
ALL:ALL
```

Esto deniega cualquier servicio, a cualquier máquina, a menos de que estén especificados en el fichero `/etc/hosts.allow`:

```
/etc/hosts.allow:
ALL: .upv.es
ALL: .rediris.es EXCEPT ftp.rediris.es
```

La primera regla permite el acceso a cualquiera de nuestros servicios a toda máquina del dominio *upv.es.* La segunda regla permite el acceso a cualquiera de nuestros servicios a toda máquina del dominio *rediris.es* a excepción de la máquina *ftp.rediris.es.*

∗ Si queremos el comportamiento contrario, es decir, que el acceso esté permitido por defecto, y que sólo se prohíban ciertos servicios a las máquinas que especifiquemos, lo haremos de la siguiente forma:

La política por defecto (se permite el acceso a todas las máquinas) hace que el archivo `/etc/hosts.allow` sea redundante, por lo tanto puede ser omitido (no contener ninguna regla).

Las máquinas a las que queremos que se prohiba el acceso deben ser listadas en el fichero `/etc/hosts.deny`:

```
/etc/hosts.deny:
ALL: www.uv.es, .upc.es
ALL EXCEPT in.fingerd:  pleione.cc.upv.es, .bar.upv.es
```

La primera regla prohibe el acceso a cualquiera de nuestros servicios a la máquina *www.uv.es*, así como a toda máquina del dominio *.upc.es*. La segunda regla también prohibe todos los servicios a la máquina *pleione.cc.upv.es* y al dominio *.bar.upv.es* pero les permite seguir accediendo al servicio de *finger*. *TCPWrappers* es, con diferencia, el programa de seguridad más utilizado en sistemas Unix. De ahí que haya merecido más atención en esta parte del temario de la que vamos a prestar al resto de programas presentados aquí.

6.6.2 Crack

ftp://ftp.ox.ac.uk/pub/comp/security/software/crackers/

Con este programa se puede comprobar la seguridad de las claves que usan los usuarios. Como ya sabemos, no se puede descifrar una clave que se encuentre en el fichero `/etc/passwd` o en el fichero `/etc/shadow`, pero si se pueden cifrar palabras y compararlas con la clave cifrada. Por lo tanto, usando diccionarios y reglas de sintaxis podemos comprobar un gran número de claves que pueden considerarse como *débiles*. Obviamente no vamos a poder comprobar **todas** las claves posibles, ya que una clave puede ser de hasta ocho caracteres y hay 62 caracteres distintos que se pueden usar. El número total de claves posibles es por tanto 62^8=218340105584896. Suponiendo que pueden comprobarse 100000 claves por segundo, necesitariamos 2183401056 segundos para comprobar la clave, o lo que es lo mismo, 69 años. Lo que se hace entonces es comprobar las palabras de uso mas común (mediante diccionarios) y tambien combinaciones de los datos del campo *GECOS* del fichero `/etc/passwd`.

6.6.3 Tripwire

ftp://coast.cs.purdue.edu/pub/COAST/Tripwire
ftp://ftp.rediris.es/mirror/coast/COAST/Tripwire/

Supongamos que alguien ha conseguido acceder al sistema, y ha podido utilizar privilegios de administrador. No podemos saber qué ficheros han podido ser modificados, o si se ha remplazado algún binario importante con otro 'modificado' (*troyano*). Si utilizamos el programa *Tripwire*, configurado adecuadamente, podemos saber qué ficheros han sido modificados. *Tripwire* chequea el sistema y comprueba el tamaño y las fechas de los ficheros, avisándonos de los cambios que se hayan podido producir desde la última vez que ejecutamos el programa. Es conveniente tener un fichero *resultado* de este programa actualizado, y conservarlo en un sistema de ficheros no modificable.

6.6.4 COPS

ftp://info.cert.org/pub/cops/1.04/
ftp://info.cert.org/pub/cops/1.04/cops_104.tar.Z

Aunque COPS 1.04 es un programa bastante antiguo, contiene varios programas y *scripts* para chequear posibles fallos en el sistema. Comprueba *passwords*, permisos de ficheros, grupos, estructura del fichero `/etc/passwd`, directorios de usuarios...

6.6.5 Secure Shell

http://www.cs.hut.fi/ssh/
http://www.ssh.fi

SSH, *Secure Shell*, se utiliza para establecer conexiones cifradas entre dos máquinas; de esta forma evitamos ataques por parte de potenciales intrusos utilizando *sniffers*, programas o circuitos dedicados a analizar los paquetes que circulan por una determinada subred. Los servicios habituales (`telnet, ftp, rlogin...`) trabajan con texto plano, es decir, todos los datos transmitidos pasan sin ninguna protección por muchos sistemas hasta llegar al destino. Utilizando SSH acabamos con este tráfico de texto plano (sobre todo evitamos el tráfico de *passwords*).

Para poder utilizar SSH, la máquina servidora (en este caso nuestro Linux) ha de estar ejecutando `sshd`, el demonio que atiende las peticiones cifradas, y el cliente que quiera conectar ha de utilizar un programa que pueda cifrar datos para enviarlos al demonio, y que este lo entienda (SSH, el cliente, está disponible para Unix, Windows 95/NT, MacOS...

6.6.6 LSOF, ls Open Files

ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/

`lsof` va a listar los ficheros abiertos en nuestro sistema Unix. Esto nos puede resultar muy útil para evitar ataques DoS causados por el exceso de ficheros abiertos en nuestro sistema, y también para localizar posibles procesos de usuarios malintencionados que pueden estar intentando *crackear* el fichero de contraseñas, o incluso que han realizado un acceso como *root* y están corriendo un analizador de red en nuestra máquina (estos programas necesitan escribir en un log, y mantener este fichero abierto durante su ejecución).

Apéndice A

El sistema X Window

Cualquier sistema operativo que hoy en día quiera ser competitivo para los puestos de trabajo de usuario debe tener una interfaz gráfica fácil de utilizar. Los sistemas más conocidos son Windows, Macintosh y OS/2. Aunque esos entornos han tenido gran aceptación, no permiten ejecutar aplicaciones gráficas a través de una red heterogénea (pero en un futuro próximo será posible).
Linux permite ejecutar aplicaciones en ventana a través de una red heterogénea debido a la incorporación de la implementación *XFree86* del estándar *Xll* de *X Window*, creado en el MIT. Este sistema es mucho más que una interfaz gráfica para ejecutar aplicaciones. Es un sistema cliente/servidor muy potente que permite que las aplicaciones se ejecuten y se compartan a través de la red. Aunque *XFree86* está diseñado para ejecutarse en un entorno de red, también puede ejecutarse en máquinas individuales. Para ejecutar aplicaciones *XFree86* o *X Window* no se necesita una red.
En este capítulo aprenderemos:

- Qué son X Window y XFree86
- Utilización de X Window
- Gestores de ventanas para X Window

A.1 ¿Qué es *X Window*?

El sistema *X Window* es un potente entorno operativo gráfico que da soporte a muchas aplicaciones en la red. Ha sido desarrollado por el MIT y es de libre distribución. La versión de *X Window* que utilizaremos en este curso será la *X11R6.3*.
El sistema *X Window* surgió de un esfuerzo cooperativo entre dos secciones del MIT: la responsable de un programa de red denominado *Athena Project* y una sección llamada Laboratorio de Ciencia Informática. Ambas utilizaban grandes cantidades de terminales UNIX y pronto se dieron cuenta de que estaban, cada uno, volviendo a inventar la rueda constantemente, cuando se trataba de programar interfaces gráficas de usuario (GUI) para estaciones de trabajo UNIX. Para disminuir la cantidad de código que ambos grupos estaban escribiendo, se pusieron de acuerdo para crear un sistema de ventanas sólido y ampliable: *X Window*.
En 1987, varios distribuidores, confiando crear un sistema único de ventanas para las estaciones de trabajo UNIX, formaron una organización, que se llamó *X Consortium*, para promocionar y estandarizar *X Window*. Gracias a este esfuerzo la informática abierta es ahora una realidad. El *X Consortium* está formado por entidades como, por ejemplo, IBM, DEC y MIT. Este grupo de grandes organizaciones supervisa la creación y lanzamiento de nuevas versiones de *X11*.
XFree86 es una marca registrada de *XFree86 Project, Inc*. Los mismos programadores que llevaron el *X Window* a la plataforma 80386 decidieron comenzar el proyecto para hacerse miembros del *X Consortium*. Al hacerse miembros del *X Consortium* el proyecto *XFree86* puede obtener acceso al trabajo en desarrollo y así pueden llevar las nuevas funciones a *XFree86* al mismo tiempo que se implementan en *X Window*, en lugar de tener que esperar a que salga la versión oficial.

X Window es una serie de piezas que trabajan conjuntamente para presentar al usuario una GUI. El sistema base de ventana es un programa que proporciona servicio al sistema *X Window*. La pieza siguiente es un protocolo para comunicación en la red: el protocolo *X Network*. Por encima del programa de implementación del protocolo *X Network* está una interfaz de bajo nivel, que se encuentra entre el sistema base de red y los programas de más alto nivel. Esta interfaz de bajo nivel se llama *Xlib*. Los programas de aplicación normalmente utilizan funciones de *Xlib* en vez de otras funciones de bajo nivel. Un administrador de pantallas es el que une esas piezas en un conjunto. El administrador de pantallas es una aplicación *X Window* cuyo propósito es controlar como se presentan las pantallas a los usuarios.
El sistema base de ventanas no proporciona los objetos de interfaz de usuario, como barras de desplazamiento, botones o menús. En esto se diferencia de la mayoría de los demás sistemas de ventanas. Los elementos de interfaz de usuario se dejan para los componentes de capas más altas y para el administrador de pantallas.
Las aplicaciones *X Window* incluyen no sólo administradores de ventanas, sino también juegos, utilidades de gráficos, herramientas de programación y muchos otros extras. Casi cualquier aplicación que se necesite ha sido escrita o trasladada a *X Window*.
X Window implementa un administrador de ventanas para realizar la tarea de crear y controlar la interfaz que compone la porción visual de sistema. No se debe confundir éste con el *OS/2 Presentation Manager* o el *Administrador de Programas de Microsoft Windows*. Aunque el administrador de ventanas de *X Window* no controla el comportamiento y posición de las ventanas, no existe un icono definido por el sistema o panel de control para mantener los valores del sistema Linux.
Por lo tanto *¿qué es X Window? X Window* es un sistema cliente/servidor controlado por dos piezas de software, una ejecutándose en el cliente y otra en el servidor. Las piezas cliente y servidor de este rompecabezas pueden estar en sistemas distintos o, como es el caso en la mayoría de las computadoras personales, ambos pueden residir en la misma máquina. Las secciones siguientes explican que significa cliente y servidor en el entorno *XFree86*.

A.2 ¿Qué es un sistema Cliente/Servidor?

Una de las palabras de moda mas usadas hoy en día en el sector informático es cliente/servidor. Éste, como la mayor parte de los conceptos básicos del sector, se ha utilizado en demasía hasta el punto de crear confusión a los usuarios. En el sentido tradicional un servidor es una máquina que simplemente proporciona recursos, espacio en disco, impresoras, módems, etc. a otras computadoras de la red. Un cliente es un consumidor de esos servicios; en otras palabras, un cliente utiliza espacio en disco, impresora o módems proporcionados por el servidor.
En el mundo de *X Window* esta relación es opuesta a la que existe en el actual mundo del PC, donde la noción de servidor más comúnmente aceptada, es que proporciona servicios a un cliente que los utiliza. En la forma más básica el cliente muestra la aplicación que se esta ejecutando en el servidor.
X Window emplea un punto de vista distinto de la relación cliente/servidor. En el mundo de *X Window* el servidor muestra la aplicación que se está ejecutando en el cliente. Esto puede parecer algo confuso al principio, pero tendrá sentido al familiarizarse más con el sistema *X Window*.
Por tanto *¿qué es un cliente?* Un cliente es un recurso que proporciona los programas y recursos necesarios para ejecutar una aplicación, lo que en el sentido tradicional se llamaría un servidor. Los recursos residen en el sistema cliente (recuerde que los sistemas cliente y servidor pueden estar en la misma máquina), mientras que la aplicación se muestra e interactúa en el sistema servidor.
La posibilidad de una aplicación *X Window*, que es el cliente, de ejecutarse bajo un servidor ubicado en el mismo computador o en otro distinto, se llama transparencia de red. Así, las aplicaciones X se ejecutan indistintamente en una máquina local o remota. Esta posibilidad puede utilizarse para ejecutar en otro servidor tareas de larga duración, permitiendo que el cliente local esté libre para realizar otras distintas.

A.2.1 Posibilidades de salida

El sistema básico de ventanas proporciona a *X Window* una gran cantidad de operaciones gráficas de mapas de bits. *X Window* y las aplicaciones de *X Window* utilizan esas operaciones para presentar a los usuarios información en forma gráfica. *XFree86* ofrece solapamiento de ventanas, dibujado inmediato de gráficos, imágenes y gráficos de alta resolución de mapas de bits y texto de alta calidad. Mientras que los sistemas iniciales de *X Window* eran en su mayoría monocromos, ahora *X Window* y *XFree86* admiten una amplia gama de sistemas en color.
X Window también admite las posibilidades de multiproceso de UNIX; de este modo, *XFree86* admite las posibilidades de multiproceso de Linux. Cada ventana que se muestra bajo *X Window* puede ser una tarea distinta que se ejecuta bajo Linux.

A.2.2 Posibilidades de la interfaz de usuario

El *X Consortium* dejó a un lado las reglas estándar para las interfaces de usuario. Ahora nos parece una falta de previsión, sin embargo, en aquel momento se había investigado muy poco acerca de la tecnología de la interfaz de usuario, por lo que no había una interfaz clara que se considerase la mejor. Incluso hoy, considerar unilateralmente una interfaz como la mejor puede ofender a muchas personas. La interfaz de usuario que desea cada uno es una decisión muy personal. El *X Consortium* quería hacer de *X Window* un estándar en los puestos de trabajo UNIX, y ésta es una de las razones por las que *X Window* se distribuye libremente en Internet. Esta distribución gratuita de *X Window* promueve la interoperabilidad, que es la piedra angular de los sistemas abiertos. Si el *X Consortium* hubiera dictaminado una interfaz de usuario, *X Window* quizás no hubiese obtenido el actual nivel de aceptación.

A.2.3 Posibilidades de entrada

Los sistemas que ejecutan *X Window* normalmente suelen tener algún dispositivo para señalar, generalmente un ratón. *XFree86* necesita un ratón u otro dispositivo, como un *trackball* , que emule a un ratón. Si no tiene ese dispositivo, no puede utilizar *XFree86* con Linux. *X Window* convierte en eventos las señales que recibe tanto desde el dispositivo para señalar cómo desde el teclado y a continuación responde a esos eventos realizando las acciones apropiadas.

A.3 Trabajando con *X Window*

Para iniciar el entorno gráfico en la máquina en la que nos encontramos trabajando podríamos simplemente ejecutar:

```
rosita:~# X
```

En unos segundos aparecerá una pantalla de color gris. Si pinchamos con el ratón en cualquier parte de esta pantalla, observaremos que no ocurre nada. Aún así, estamos realmente utilizando un servidor X.
Podemos volver a cualquiera de las terminales virtuales en modo texto mediante la combinación de teclas Ctrl-Alt-F[1-6]. Desde uno de estos terminales ejecutamos la siguiente orden:

```
rosita:~# xpaint &
```

Y volvemos a la pantalla del servidor X, que se ejecuta en el séptimo terminal virtual asignado por Linux, con la combinación de teclas Alt-F7.
Como podemos observar, se ha creado una nueva ventana correspondiente al programa *xpaint*. Por lo tanto, pese a no disponer de menús interactivos, ni haber ejecutado ningún gestor de ventanas, las aplicaciones cliente pueden conectarse a nuestro servidor *X Window*. Finalizamos el servidor X con la combinación de teclas Ctrl-Alt-Borrar, lo que nos devuelve al intérprete de órdenes en modo texto. Desde ahí volvemos a lanzar el servidor X mediante la orden:

```
rosita:~# startx
```

Si ya esta familiarizado con otros GUI (*Graphic User Interface*) como, por ejemplo, *Microsoft Windows* o la interfaz de usuario de *Macintosh*, verá que no hay una gran diferencia entre estos y *X Window*. Éste presenta al usuario varias ventanas, cada una de las cuales muestra la salida de una aplicación *X Window*, llamada un cliente. Éste puede estar ejecutándose o en el PC del usuario, o en otra estación de trabajo de la red.
La forma de moverse por *X Window* depende en gran medida de los gestores de ventanas, que se tratarán mas adelante en este capítulo. La mayoría de las ventanas utilizan un elemento apuntador en pantalla llamado cursor que indica dónde esta trabajando. Éste puede adoptar muchas formas, dependiendo de lo que esté haciendo y de qué gestor de ventanas esté ejecutando.
El gestor de ventanas que se ejecuta por defecto en los sistemas Linux de este curso es *fvwm95*, que "emula" el interfaz gráfico usado por *Windows 95*. Podemos observar que el diseño de los bordes de las ventanas y la barra de tareas corresponden con los del popular producto de la casa Microsoft.

A.3.1 Navegación por *X Window*

X Window, al igual que casi todos los GUI, permite la realización de entradas procedentes del teclado y del dispositivo apuntador, normalmente un ratón. Por lo general, para que una ventana acepte una entrada, deberá ser una ventana activa, que normalmente presenta un aspecto diferente (por ejemplo, un borde destacado) al de las ventanas inactivas. La activación de una ventana depende del gestor de ventanas. Algunos permiten activar una ventana al mover el cursor dentro de dicha ventana. Otros requieren situar el cursor en la ventana y pulsar un botón del ratón, al igual que ocurre en *Microsoft Windows*. Hoy en día, muchos GUI instalados en PC proporcionan menús descendentes y ascendentes, que, también, dependen del gestor de ventanas, incluyendo los tipos de selecciones de menú proporcionados. La mayoría de los gestores de ventanas de *X Window* no tienen una barra de menú principal a lo ancho de la parte superior del sistema; en su lugar utilizan un menú flotante, que normalmente se activa pulsando un botón del ratón en una zona vacía del escritorio. Con el botón del ratón pulsado, se arrastra el cursor por las diferentes selecciones de menú hasta encontrar la selección de menú deseada, entonces se suelta el botón y la selección se activa; es muy parecido a la forma en que se navega por los menús de un sistema *Macintosh* y muy diferente a la forma en que se navega por los menús bajo *Microsoft Windows*.

A.3.2 Xterm

Para poder acceder a la línea de órdenes usando *X Window* sin tener que volver a las terminales virtuales, disponemos del programa *xterm*. El programa *xterm* es un emulador de terminal para el sistema *X Window*. Proporciona compatibilidad con diferentes terminales, como ANSI, vt100 o vt102. Cuando ponga en marcha una sesión *xterm* podrá ejecutar cualquier programa de línea de órdenes o ejecutar cualquier instrucción Linux como lo haría en cualquiera de las terminales virtuales suministradas por Linux.
Si abrimos una ventana *xterm* podremos ejecutar programas en nuestro servidor X especificándolos en la línea de órdenes. Por ejemplo, si queremos ejecutar el programa *xeyes*, teclearemos en la ventana de *xterm*:

```
rosita:~# xeyes &
```

Con lo que aparecerá en pocos segundos en nuestra pantalla de X. Es recomendable ejecutar los programas de X en segundo plano, para así poder seguir usando la misma ventana *xterm* para ejecutar más aplicaciones.

A.3.3 Conexiones remotas

Como hemos dicho anteriormente, es posible ejecutar aplicaciones de X en sistemas remotos, pero visualizando la ejecución gráfica en nuestro propio terminal local. De esta manera podemos trabajar con aplicaciones preparadas para otros sistemas desde nuestra máquina. Obviamente existe una

manera de regular las aplicaciones que podrán conectarse a nuestro servidor de X. De no ser así cualquier persona desde cualquier máquina podría conectarse a nuestro servidor y, en el mejor de los casos, ocasionar una simple molestia visual. El mandato que se utiliza para habilitar o deshabilitar el acceso a máquinas remotas es *xhost*:

`+ [nombre]` Añade a *nombre* a la lista de conexiones permitidas a nuestro servidor de X. nombre puede ser una máquina o un nombre de usuario. Si no se especifica un nombre, se permite la conexión a cualquier máquina, aunque no se encuentre en la lista de acceso (esto inhabilita el control de acceso a nuestro servidor X).

`- [nombre]` Borra a *nombre* de la lista de conexiones permitidas a nuestro servidor de X. *nombre* puede ser una máquina o un nombre de usuario. Las conexiones en funcionamiento no se paran, pero las siguientes conexiones serán denegadas. Si no se especifica un nombre, se restringe la conexión a las máquinas especificadas en la lista de acceso (esto habilita el control de acceso a nuestro servidor X).

Si no se especifica ningún parámetro, *xhost* devuelve un mensaje indicando si el control de acceso está activado o desactivado, seguido de la lista de máquinas o nombres de usuario que tienen permitido el acceso.

De esta manera, si queremos ejecutar aplicaciones en nuestro servidor de X desde otras máquinas, deberemos antes permitir el acceso a estas máquinas mediante la instrucción `xhost`. Por ejemplo, supongamos que nos encontramos trabajando en *máquina1* y queremos ejecutar el navegador de páginas web *Netscape* en *máquina2*, visualizando el resultado y manejando dicho programa desde nuestro ordenador. El primer paso a seguir es permitir a *máquina2* que conecte a nuestro servidor de X, por lo que escribimos en la sesión *xterm*:

```
máquina1~# xhost + máquina2
máquina2 being added to access control list
```

Podemos listar las máquinas que tienen permitida la conexión a nuestro servidor de X:

```
máquina1~# xhost
access control enabled, only authorized clients can connect
INET:máquina2
INET:localhost
LOCAL:
```

El siguiente paso es conectar al sistema *máquina2*, ya sea mediante *telnet*, *rlogin*o cualquier otro protocolo de acceso remoto:

```
maquina1~# rlogin máquina2
Password:
Last login:  Sun Nov 2 14:34:27 from localhost
maquina2~$
```

Una vez dentro del sistema remoto, debemos asegurarnos que tenemos definida la variable de entorno *DISPLAY* y que ésta contiene la dirección de nuestro servidor X:

```
maquina2~$ echo $DISPLAY


maquina2~$ export DISPLAY="maquina1:0.0"
maquina2~$ echo $DISPLAY
maquina1:0.0
maquina2~$
```

Ya estamos listos para conectar aplicaciones X desde *máquina2* a nuestro servidor de X (*máquina1*). Podemos entonces ejecutar el programa *Netscape*:

```
maquina2~$ netscape &
[1] 33541
```

Si todo ha ido bien, debería aparecer en nuestro servidor X la ventana del programa *Netscape*. De lo contrario podemos obtener el siguiente error:

```
maquina2~$ netscape &
[1] 36012
Error:  Can't open display:  maquina1:0.0
maquina2~$
```

Este error puede ser debido o bien a que no hemos especificado correctamente *máquina2* en la lista de *hosts* autorizados a conectar con nosotros (mediante *xhost*) o a que no hemos exportado correctamente la variable *DISPLAY*.
Como hemos visto es posible trabajar desde Unix con aplicaciones que se ejecutan en otras máquinas. Nosotros vemos el resultado del programa, e interactuamos con él, pero utilizando la CPU y la memoria de la máquina remota. Esto es tremendamente útil cuando no disponemos de un ordenador local suficientemente potente, o queremos trabajar con aplicaciones que no han sido desarrolladas para nuestro sistema operativo. Por ejemplo, desde nuestro Linux podríamos utilizar una aplicación de diseño gráfico en tres dimensiones en un servidor *Silicon Graphics* o un complejo programa de cálculo en una *Convex*. La velocidad de nuestro ordenador no influirá en la de la aplicación que estemos usando ya que ésta utiliza los recursos del sistema remoto.
Otra de las características de la conexión remota entre servidores X es poder conectarse directamente a cualquier máquina remota como si estuviésemos enfrente de su pantalla, siempre y cuando la máquina a la que conectemos tenga ese servicio disponible. La manera de conectarnos a este servicio desde Linux es mediante la orden:

```
rosita:~# X -query máquina.servidora
```

Normalmente nos aparecerá una pantalla de entrada donde introduciremos nuestro *login* y *password*, después de lo que estaremos trabajando con el sistema remoto exactamente igual que si hubiésemos accedido a él desde la consola principal.

A.4 Gestores de ventanas para *X Window*

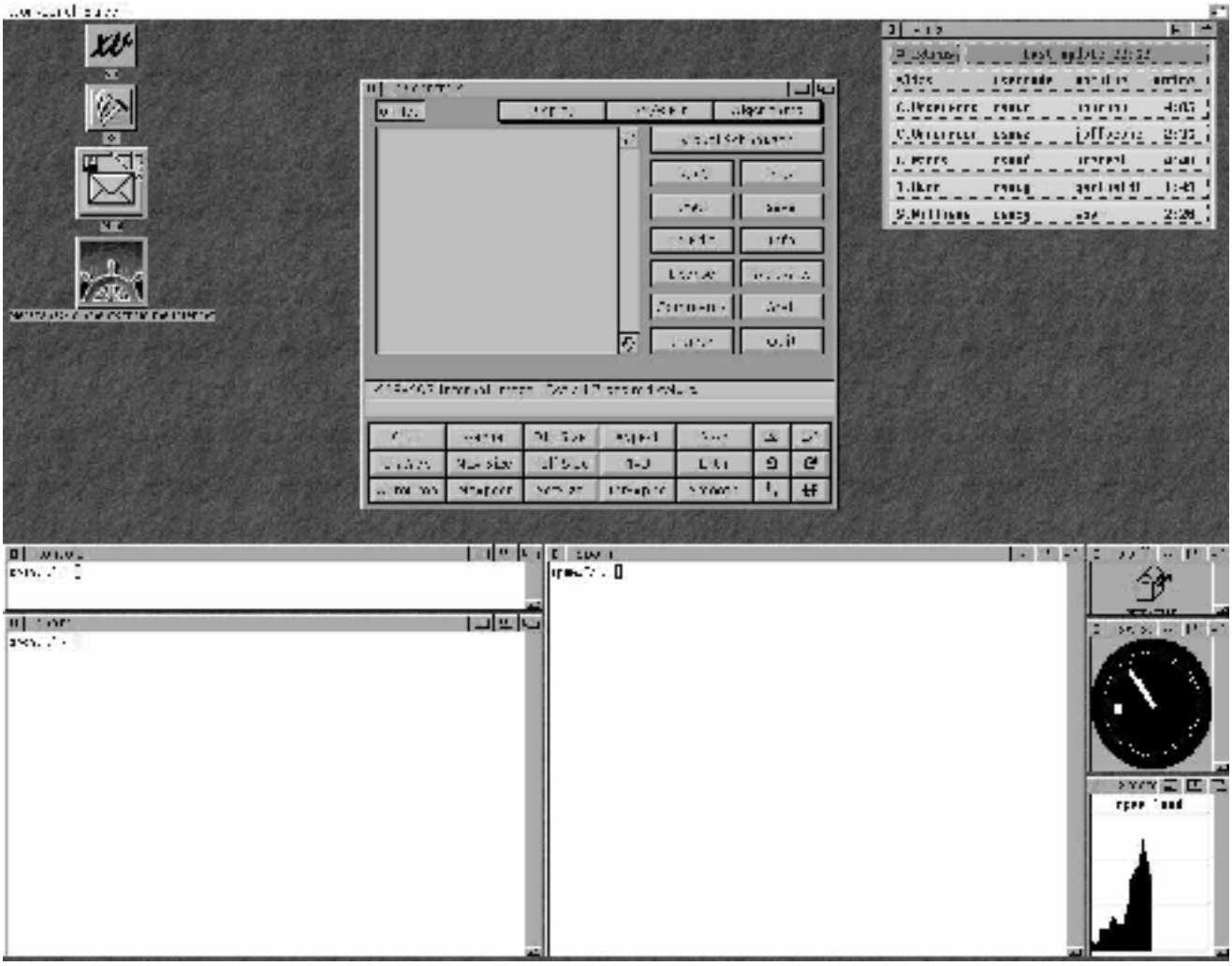
Como ya hemos dicho anteriormente, *X Window* no especifica un gestor de ventanas. La apariencia y percepción de *X Window* pasa a ser la prerrogativa del usuario. De acuerdo con esta filosofía, Linux no proporciona sólo un gestor de ventanas para *X Window*, aunque la instalación predeterminada instala el gestor de ventanas *fvwm95* como gestor por defecto. Nosotros mismos tenemos la posibilidad de cambiar el gestor de ventanas, o añadir uno nuevo, según nuestros gustos. A continuación se muestran algunos de los gestores de ventanas más conocidos para *X Window*:

afterstep

Emula el aspecto y diseño del interfaz de usuario de *NEXTSTEP*, y le añade algunas mejoras. Se basa en el gestor de ventanas *fvwm*. *Afterstep* es uno de los gestores de ventanas más populares (si no el más popular).

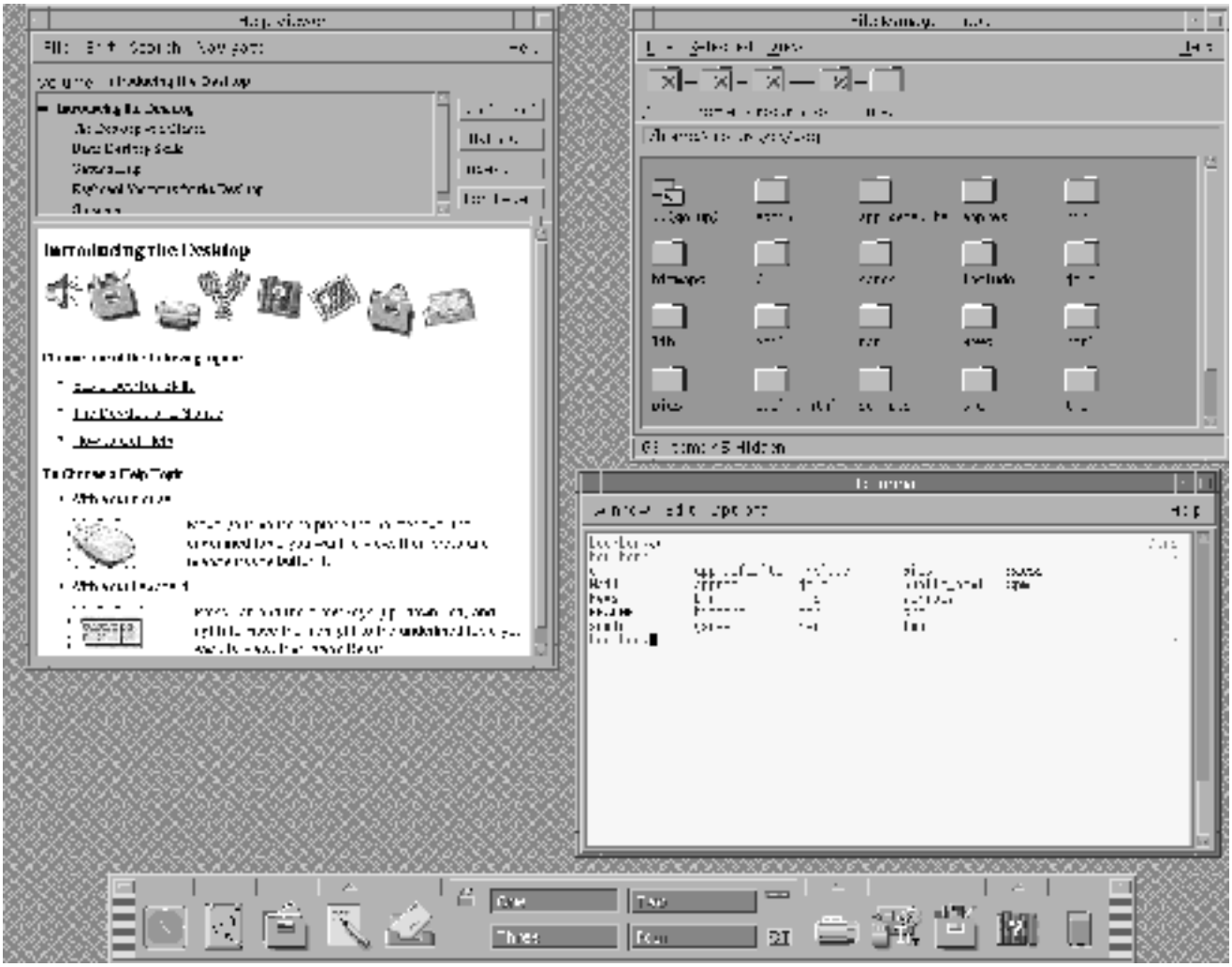
amiwm

AmiWm es un gestor de ventanas para fanáticos del Commodore Amiga. Emula el aspecto del entorno gráfico *Workbench* de este ordenador. Soporta múltiples áreas de trabajo, que pueden ser llevadas arriba y abajo, como en el Amiga, y fondos distintos para cada área.



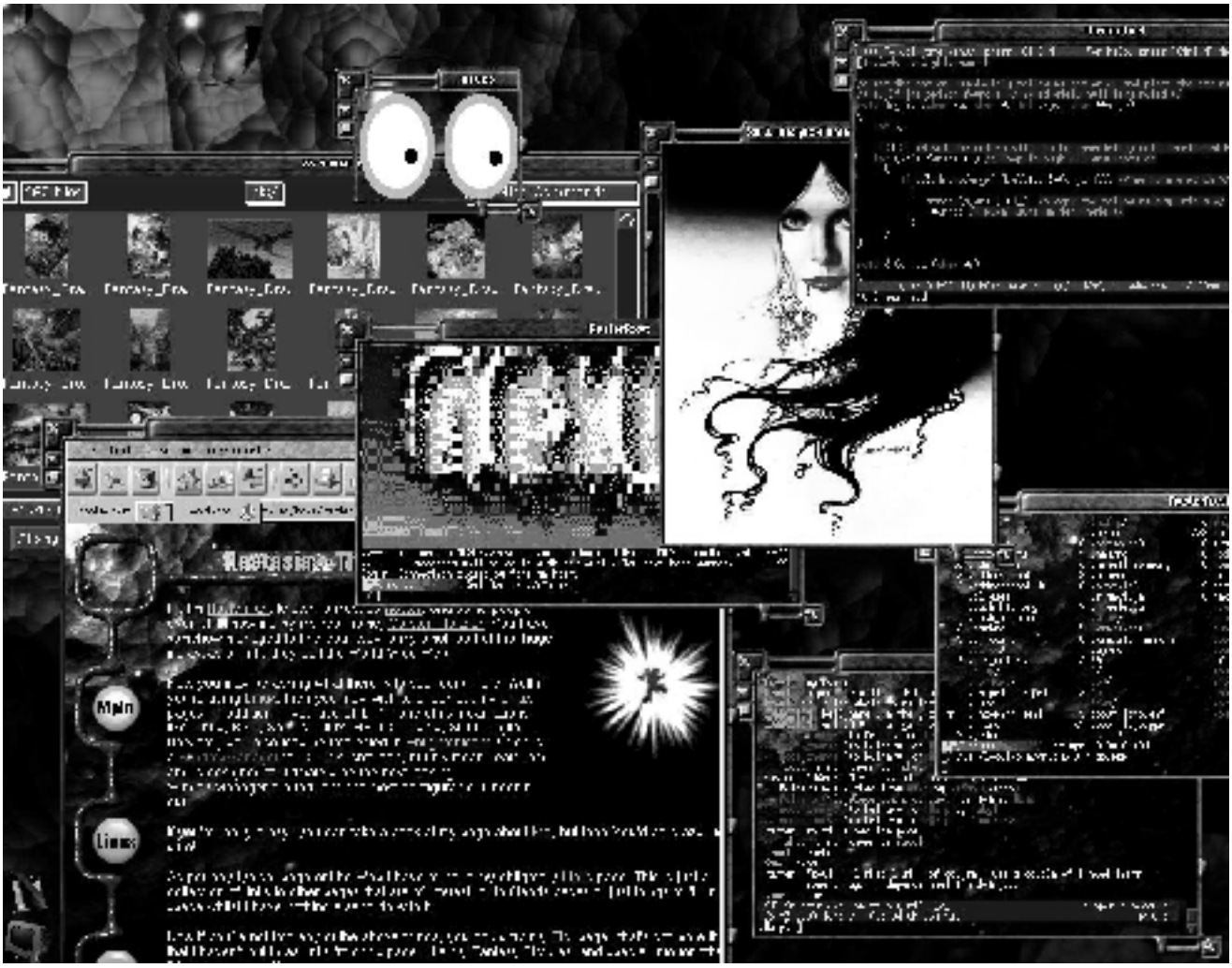
dtvwm

El gestor de ventanas CDE (*Common Desktop Environment*) es un interfaz de usuario gráfico comercial para varias plataformas UNIX (*AIX*, *Digital UNIX*, *HP/UX*, *Solaris*, *UnixWare*, etc.). El escritorio ha sido desarrollado en común por *Hewlett-Packard*, *IBM*, *Novell* y *Sun Microsystems*. Está siendo adoptado como el entorno operativo estándar por esas compañías y muchas más en el mercado de las estaciones de trabajo UNIX.



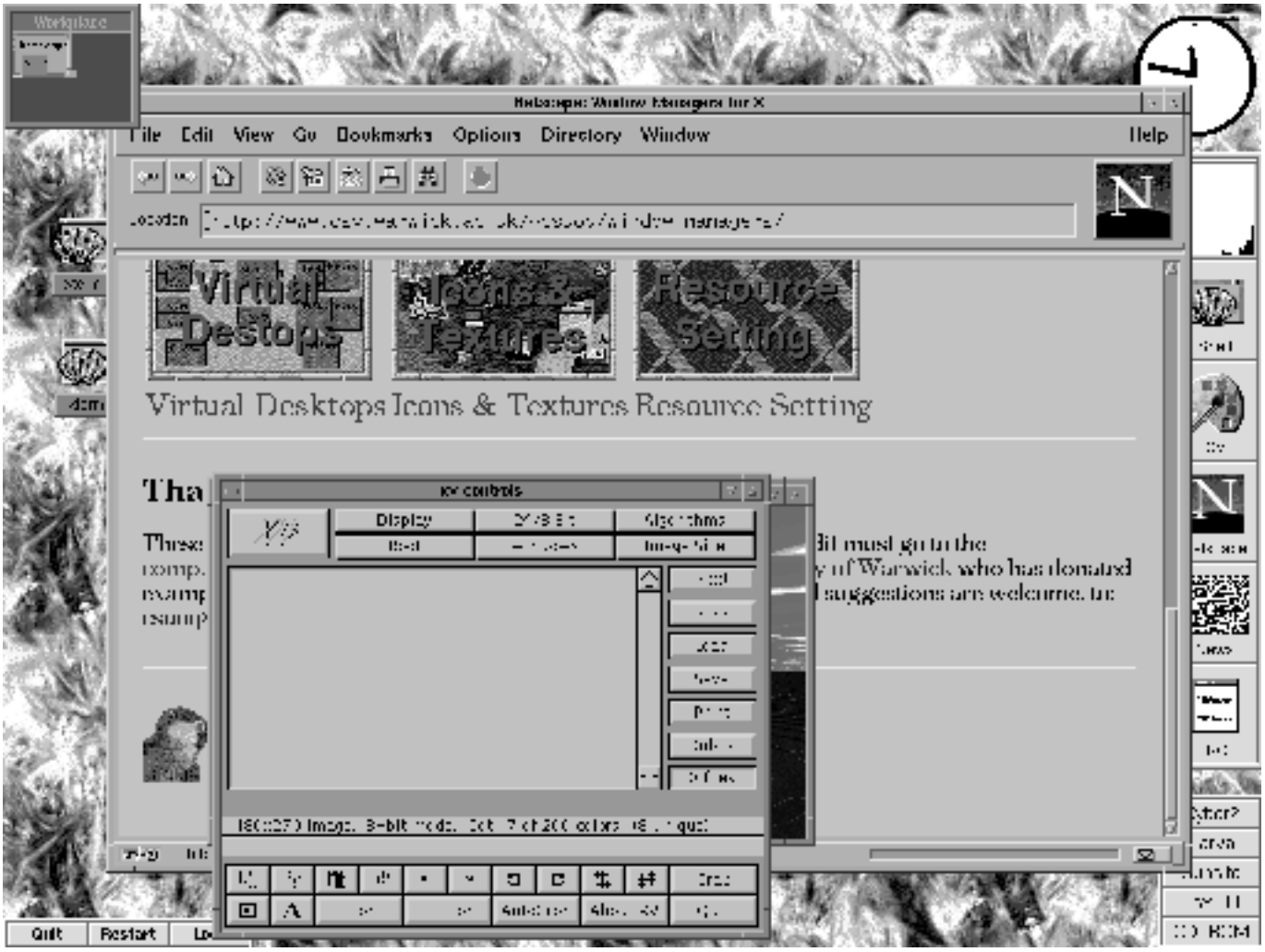
enlightenment

Enlightenment (también conocido como *E*) se basa originalmente en *fvwm2*, pero ha sido reescrito completamente desde entonces, y aparentemente no comparte código con ningún otro gestor de ventanas. Eso hace que sea preciso, rápido y eficiente para la tarea que desempeña. Se ha compilado con éxito en plataformas Linux, Solaris y Digital.



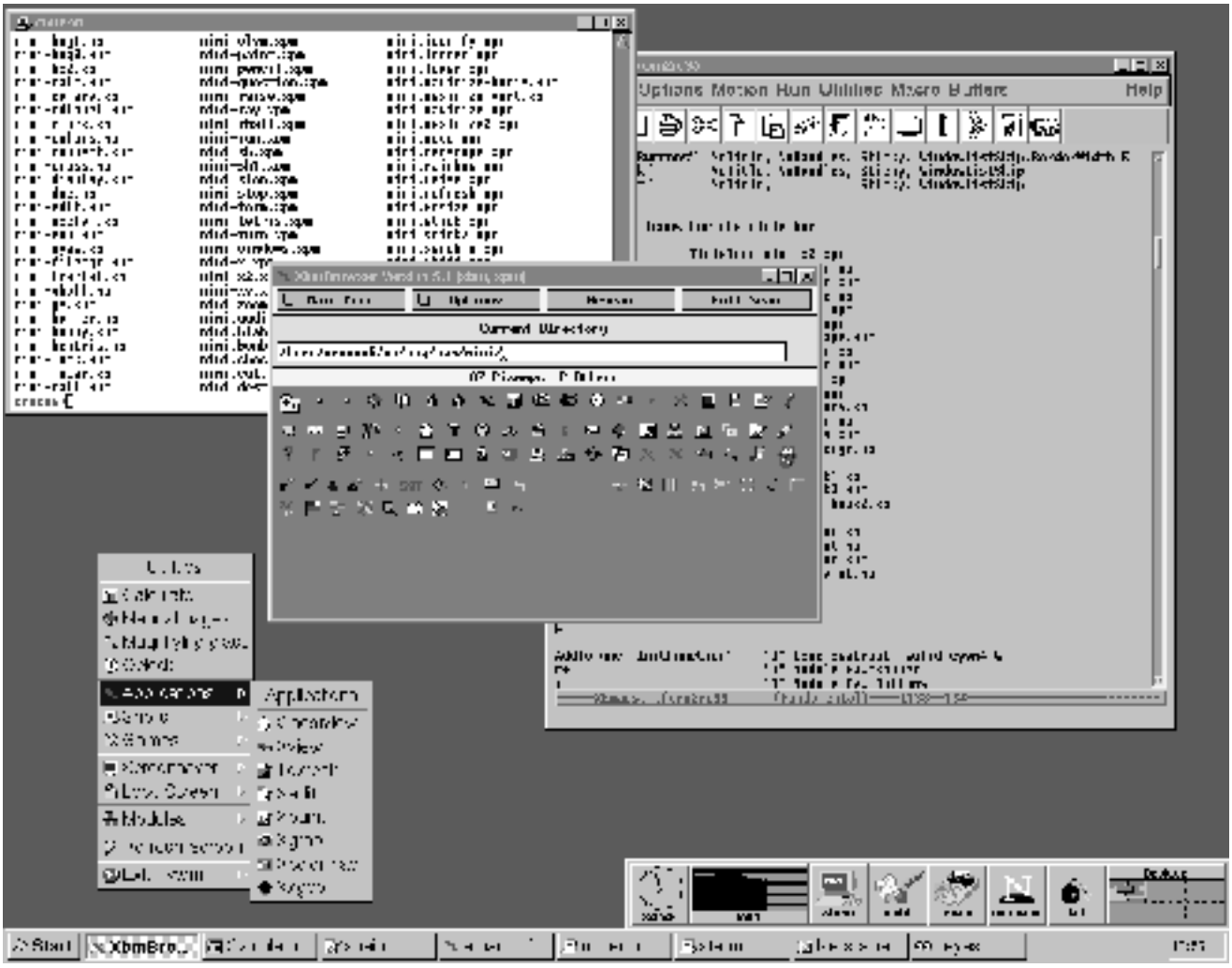
fvwm

FvWm, desarrollado por Robert Nation, parece ser el más conocido de los gestores de ventanas virtuales que se usan en la actualidad (sobre todo en la comunidad Linux). Se basa en buena parte en el gestor de ventanas *twm*. *Fvwm* es un derivado del *twm*, rediseñado para minimizar el consumo de memoria, proporcionar un aspecto tridimensional a las ventanas y un escritorio virtual simple. Su funcionalidad puede ser mejorada con el uso de varios módulos.



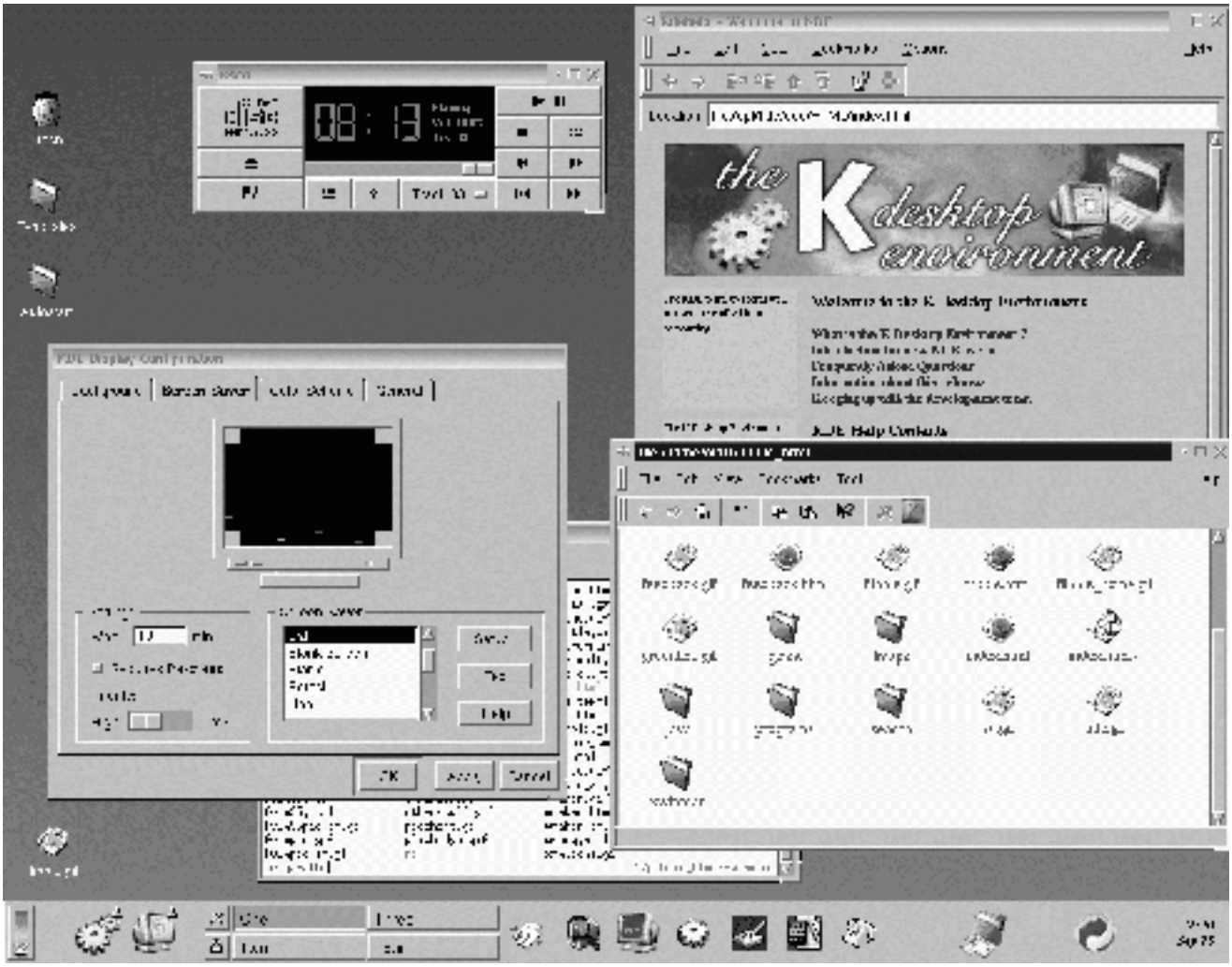
fvwm95

*Fvwm95*es una versión especial basada en el *fvwm2.x*. Intenta emular algunos aspectos de Windows 95, pero sin desaprovechar la funcionalidad de *fvwm2*. Proporciona un aspecto diferente al escritorio y una barra de tareas adicional.



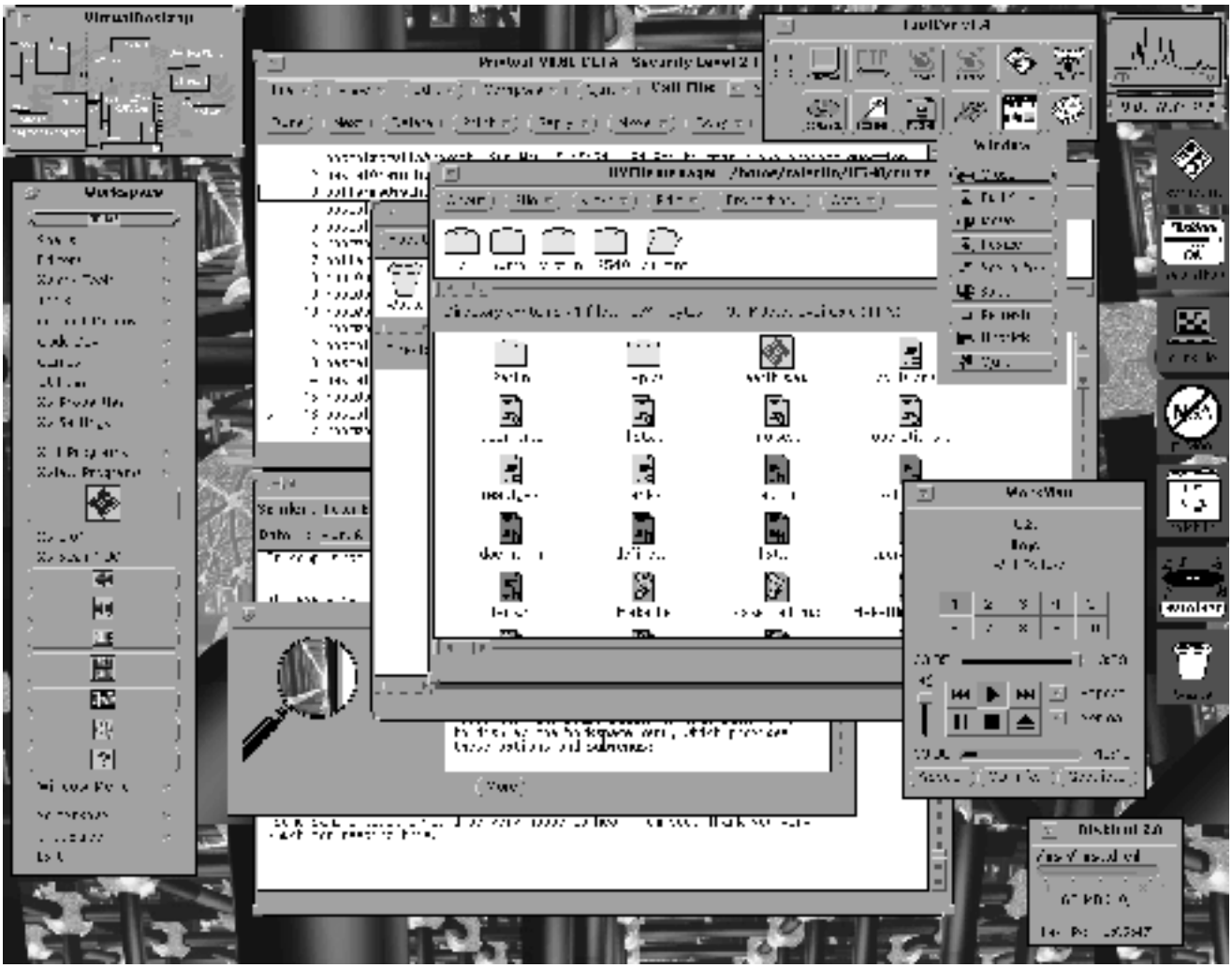
kde

KDE intenta proporcionar un interfaz robusto para las aplicaciones de X, en aspecto y en funcionamiento. *KDE* tendrá un conjunto de aplicaciones básicas como un gestor de ventanas (llamado *kwm*), un administrador de archivos, un emulador de terminal, un sistema de ayuda en línea y configuración de pantalla. Este proyecto está en fase *beta* actualmente, pero ya se ve muy estable y utilizable.



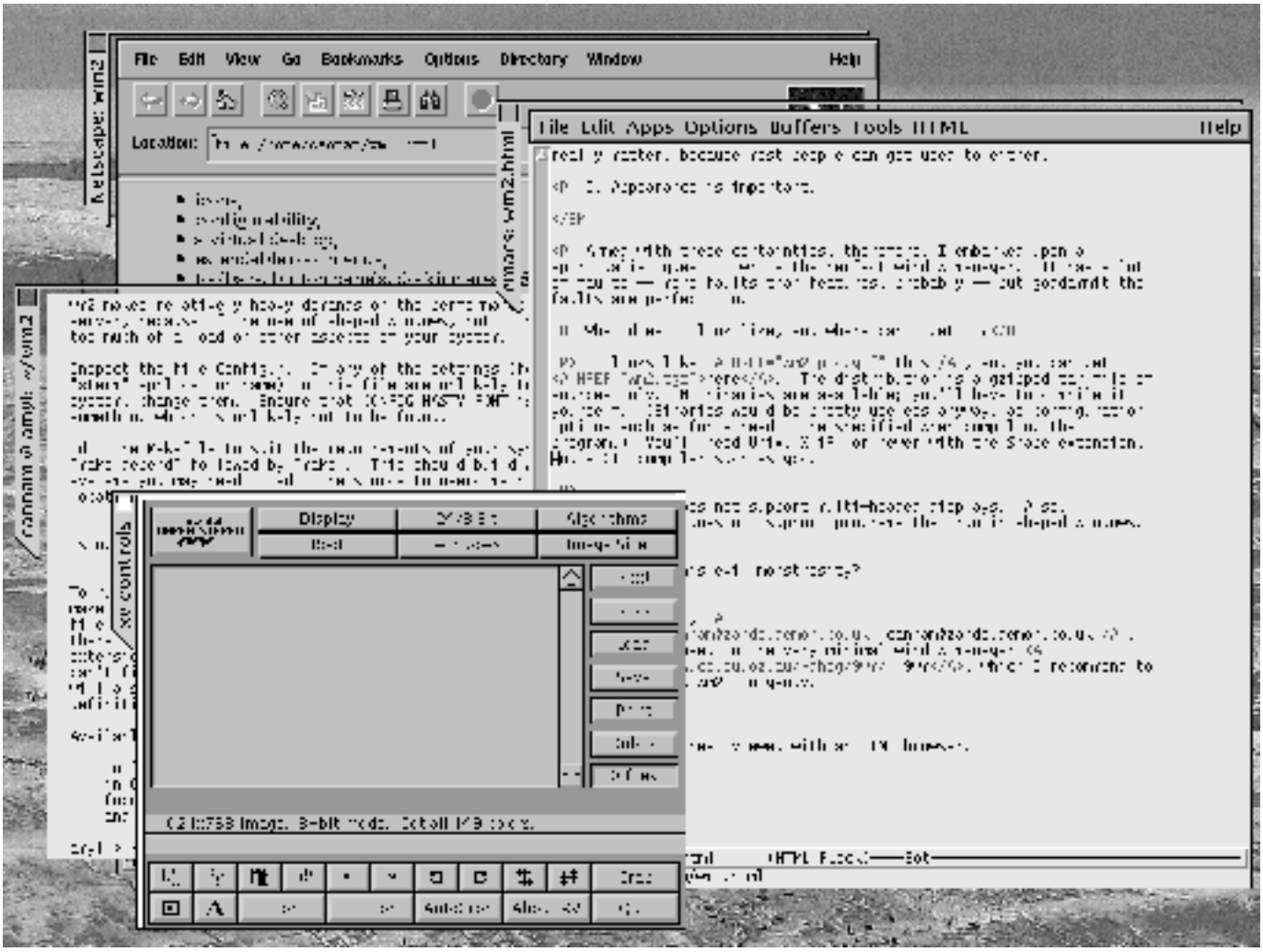
olvwm

El gestor de ventanas *OpenLook*. Es el gestor de ventanas estándar del *OpenWindows* de *Sun Microsystems*.



wm2

Este gestor de ventanas adopta la solución minimalista. No proporciona ninguna facilidad de configuración, iconos, un escritorio virtual, ni paneles de control. Sólo añade un marco a cada ventana e intenta adoptar un aspecto agradable. De esta manera es sencillo, rápido y pequeño.



Apéndice B

Direcciones de interés en INet

B.1 Varios

- http://sunsite.unc.edu/LDP/index.html
  Linux Documentation Project
- http://www.chaoscomp.com/linux/
  Linux resources
- http://sunsite.unc.edu/mdw/#howtos
  Linux HOWTOs Documents
- http://www2.shore.net/~jblaine/vault/
  Jeff´s Unix Vault
- http://www.xnet.com/~blatura/linux.shtml
  Linux and UNIX Resources
- http://www.linuxhq.com
  Linux HQ v2
- http://www.eklektix.com/lwn
  Linux Weekly News
- ftp://luna.gui.uva.es/pub/linux.new/software
  Software para Linux
- http://www.kernel.org
  Kernel Home Page
- http://www.cs.vu.nl/~tmgil/vi/vi.html#intro
  The VI lover´s page
- http://www.geocities.com/SiliconValley/Vista/8009/index1.html
  Unix Humor
- http://www.freshmeat.net
  Fresh Meat: news, forum, and more
- http://SAL.KachinaTech.COM/index.shtml
  SAL: Scientific Applications on Linux
- ftp://sunsite.unc.edu/pub/Linux/
  Sunsite Linux FTP

B.2 Seguridad

- http://www.cs.purdue.edu/coast/
  COAST
- http://www.ecst.csuchico.edu/~jtmurphy
  Linux Security Home Page
- http://www.deter.com/unix/
  MLD´s Unix Security Page
- http://csrc.nist.gov/tools/tools.htm
  Unix Security Tools
- http://www.ja.net/CERT/
  Artículos, software... (CERT)
- http://iris.acs.neu.edu/goedicke/security.html
  Unix Security Information
- http://www.alw.nih.gov/Security/security.html
  NIH page on Unix Security
- http://andercheran.aiind.upv.es/toni/unix/
  Seguridad en Unix

B.3 Revistas On-Line

- http://www.ssc.com/lj/index.html
  Linux Journal
- http://www.wcmh.com/uworld/
  UnixWorld Online Magazine
- http://mercury.chem.pitt.edu/~angel/LinuxFocus/
  LinuxFocus
- ftp://ftp.rediris.es/software/linux/lg
  Linux Gazette
- http://www.unixreview.com/
  UNIX Review
- http://www.netline.com/sunex/
  SunExpert
- http://www.sunworld.com/
  SunWorld
- http://www.polaris.net/ugu/
  Unix Guru Universe
- http://www.linuxworld.com/
  LinuxWorld
- http://www.netline.com/rs/
  RS/Magazine
- http://www.scoworld.com/
  SCO World Magazine

- http://www.samag.com/
  Sys Admin

B.4 X-Window

- http://www.plig.org/xwinman/
  Gestores de ventanas para el sistema X-Window
- http://www.camb.opengroup.org/tech/desktop/x/
  Página oficial de Opengroup y X-Window
- http://www.x11.org/
  Página de X11, con todo tipo de aplicaciones para X-Window
- http://www.xfree86.org/
  XFree86
- http://www.gtk.org/
  GTK (GIMP Toolkit)
- ftp://ftp.x.org/contrib/applications/
  Software for X

B.5 Programación

- http://www.cs.buffalo.edu/~milun/unix.programming.html
  Davin´s collection of Unix programming
- http://www.lsi.us.es/cursos/
  Cursos de programación en Unix: *X Window* y *sockets*
- http://www.nowao.edu/~rstevens/
  Código fuente de los libros de R. Stevens (ver bibliografía)
- http://www.linuxprogramming.com/
  Linux Programming
- http://andercheran.aiind.upv.es/toni/prog/
  Programación de sistemas Unix
- http://www.linuxos.org/Lprogram.html
  LinuxOS: Linux Programming
- http://www.ecst.csuchico.edu/~chafey/prog/sockets/sinfo1.html
  Begginners guide to sockets
- http://www.kernel-panic.com/links/devel.html
  Linux Development Software
- http://www.ee.mu.oz.au/linux/programming/
  Linux Programming BouncePoint
- http://sci173x.mrs.umn.edu/~bentlema/unix/unix.html
  Unix Programming
- http://www.erlenstar.demon.co.uk/unix/
  The Unix Programming Resources Page

B.6 Compañías

- http://www.sun.com
  Sun Microsystems
- http://www.hp.com
  Hewlett Packard
- http://www.redhat.com
  Red Hat Software, Inc.
- http://www.qnx.com
  QNX Software Systems Ltd.
- http://www.suse.com
  S.u.S.E.
- http://www.calderasystems.com
  Caldera Systems, Inc.
- http://www.digital.com
  Digital Equipment Corporation
- http://www.bsd.com
  Berkeley Software Design, Inc.
- http://www.systemv.com
  System V
- http://www.sco.com
  Santa Cruz Operation

Apéndice C

El núcleo de Linux

C.1 Introducción

Como ya sabemos, el núcleo o *kernel* de cualquier sistema es tan importante que se puede considerar como el sistema operativo en sí mismo; incluso si no lo consideramos así, y contemplamos al sistema operativo como el conjunto formado por el núcleo y una serie de herramientas (editor, compilador, enlazador,*shell*...), es innegable que el *kernel* es la parte del sistema más importante, y con diferencia: mientras que las aplicaciones operan en el espacio de usuario, el núcleo siempre trabaja en el modo privilegiado del procesador (RING 0). Esto implica que no se le impone ninguna restricción a la hora de ejecutarse: utiliza todas las instrucciones del procesador, direcciona toda la memoria, accede directamente al *hardware* (más concretamente, a los manejadores de dispositivos), etc. De esta forma, un error en la programación, o incluso en la configuración del núcleo puede ser fatal para nuestro sistema.

Visto esto, está claro que conviene dedicar un poco de tiempo a familiarizarnos con el *kernel* de Linux, su localización, sus funciones, su configuración... para así evitar muchos problemas. En este capítulo intentaremos aprender algo más sobre él. Para un conocimiento más profundo, se pueden consultar las excelentes obras [Car97] y [Bec97] o, en el caso del núcleo de cualquier sistema Unix en general, [Bac87].

C.2 La organización del núcleo

Generalmente el código fuente del núcleo de un sistema Linux está en el directorio `/usr/src/linux/`. Este directorio suele ser un enlace simbólico a otro directorio `/usr/src/linux-version-del-nucleo/`, como `/usr/src/linux-2.0.36/` (esto se hace habitualmente por cuestiones de compatibilidad). Si accedemos a este directorio (o a su enlace, recordemos que es equivalente) nos encontraremos con el código de la versión de Linux correspondiente, una estructura de ficheros y directorios más o menos parecida a la siguiente:

```
rosita:~# ls -l /usr/src/linux/
total 1513
-rw-r--r--   1 root     root        18458 Jul 13  1998 COPYING
-rw-r--r--   1 root     root        39338 Jul 13  1998 CREDITS
drwxr-xr-x   6 root     root         1024 Jul 13  1998 Documentation
-rw-r--r--   1 root     root        10221 Jul 13  1998 MAINTAINERS
-rw-r--r--   1 root     root        10021 Jul 13  1998 Makefile
-rw-r--r--   1 root     root        12056 Jun 26  1996 README
-rw-r--r--   1 root     root         4526 Sep 20  1996 Rules.make
-rw-r--r--   1 root     root       127300 Mar 16 20:59 System.map
drwxr-xr-x   8 root     root         1024 Apr 22  1996 arch
```

```
drwxr-xr-x  11 root     root         1024 Dec 12 17:21 drivers
drwxr-xr-x  20 root     root         3072 Mar 16 20:57 fs
drwxr-xr-x  12 root     root         1024 Mar 16 20:47 include
drwxr-xr-x   2 root     root         1024 Mar 16 20:52 init
drwxr-xr-x   2 root     root         1024 Mar 16 20:59 ipc
drwxr-xr-x   2 root     root         1024 Mar 16 20:53 kernel
drwxr-xr-x   2 root     root         1024 Mar 16 20:59 lib
drwxr-xr-x   2 root     root         1024 Mar 16 20:55 mm
drwxr-xr-x   2 root     root         2048 Mar 16 20:52 modules
drwxr-xr-x  15 root     root         1024 Mar 16 20:58 net
drwxr-xr-x   3 root     root         1024 Mar 16 20:52 scripts
-rwxr-xr-x   1 root     root      1300925 Mar 16 20:59 vmlinux
rosita:~#
```

Esto que acabamos de ver es el código del núcleo de Linux; no debemos confundirlo con el propio núcleo, que es un fichero ejecutable del que hablaremos más adelante. Es posible que en nuestro sistema no tengamos el código (lo podemos haber borrado para ahorrar espacio, aunque esto no es conveniente), pero es **imposible** que no tengamos el núcleo en sí [1]. La organización de todos estos directorios es la mostrada en la figura C.1.

¿Qué es lo que podemos encontrar en cada uno de estos directorios hijos de `/usr/src/linux/`? Vamos a verlo sin entrar en muchos detalles:

- `Documentation/`
  En el directorio `Documentation` tenemos documentación de todo tipo relativa al núcleo. Podemos encontrar desde recomendaciones para el estilo de programación de los desarrolladores hasta ficheros que nos explican el funcionamiento de los módulos cargables dinámicamente o la nomenclatura de los dispositivos *hardware* en Linux, pasando por el fichero de ayuda para la configuración del sistema o una imagen del logo de Linux (el pinguino Tux, obra de Larry Ewing).

- `arch/`
  En este directorio encontramos ficheros dependientes de plataforma, generalmente en lenguaje ensamblador. Como ya sabemos, Linux puede funcionar en diferentes tipos de arquitecturas (i386, PowerPC, MIPS, ALPHA...); cada una de estas arquitecturas tiene un lenguaje ensamblador diferente, por lo que la pequeña parte del núcleo escrita en este lenguaje (habitualmente tareas a muy bajo nivel necesarias para el arranque) difiere de un sistema a otro. En el directorio `arch` (de *architecture*) tenemos diferentes subdirectorios, cada uno para un tipo de máquina diferente, con sus correspondientes ficheros en ensamblador (archivos `.s` o `.S`).

- `drivers/`
  Casi la mitad del código del núcleo de Linux está localizado en el directorio `drivers`. Como su nombre indica, lo que encontramos aquí son manejadores de los diferentes dispositivos que podemos conectar al computador: CD-ROMs, tarjetas de red de todo tipo, dispositivos SCSI... Cada vez que un nuevo dispositivo aparece en el mercado, o al menos casi todas las veces, hay alguien que se encarga de programar un *device driver* o manejador para ese dispositivo; esto hace que Linux sea uno de los Unices (si no el que más) *hardware* soporta "de serie", es decir, sin tener que instalar software adicional: simplemente reconfigurando el núcleo (como ya veremos) e indicándole que tenemos un nuevo dispositivo conectado al sistema.
  En el directorio `drivers` tenemos a su vez varios subdirectorios, cada uno para una familia de dispositivos concreta: de sonido, genérica de bloques, genérica de caracteres, dispositivos

[1] Quizás no lo tenemos en el sistema de ficheros, por ejemplo si hemos arrancado el sistema con el núcleo de un diskette o trabajamos en una estación *diskless*, sin disco duro, pero **sí** en memoria

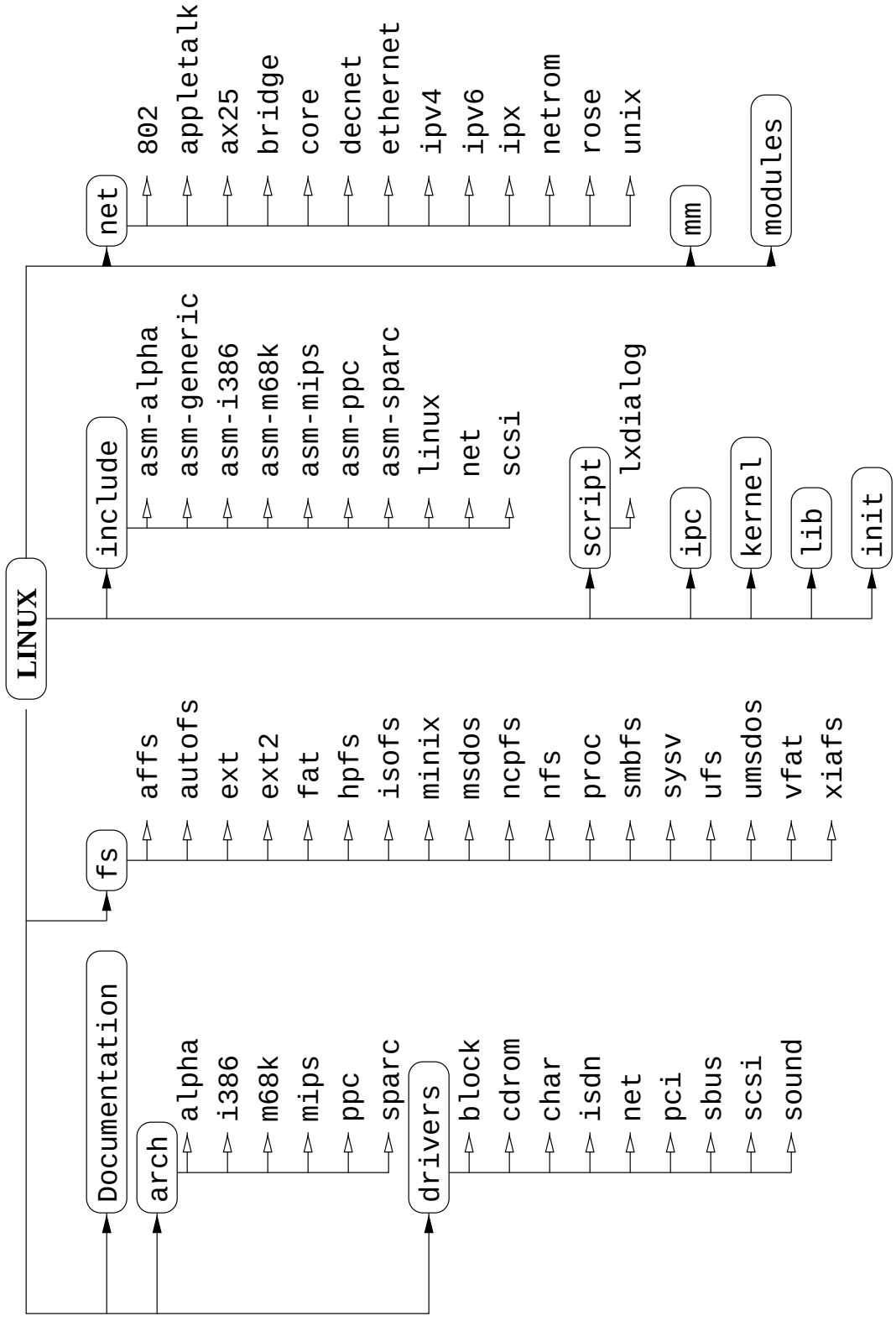


Figura C.1: Jerarquía de directorios del *kernel* de Linux

PCI, SCSI... Dentro de estos subdirectorios tenemos los manejadores de dispositivo, ficheros en C que van a permitir que Linux reconozca cada pieza de *hardware* que tengamos en el sistema.

- `fs/`
  Este directorio contiene la implementación del sistema de ficheros de Linux; dentro de él tenemos ficheros genéricos del diseño del *Virtual File System*, como el manejo de FIFOs o los sistemas de inodos, y también directorios correspondientes a la implementación de los diferentes sistemas de ficheros que Linux soporta en su núcleo: `ext2`, `vfat`, `ufs`, `minix`...

- `include/`
  En el directorio `include` tenemos ficheros de cabecera (`.h`) necesarios para compilar el núcleo del sistema; dentro de él tenemos diferentes subdirectorios que contienen y clasifican estos ficheros en varias categorías (genéricos, del subsistema de red, de dispositivos SCSI...
  También tenemos subdirectorios con ficheros de cabecera dependientes de plataforma (`asm-i386` para la familia Intel, `asm-m68k` para los Motorola 68000...), y un enlace a uno de ellos denominado simplemente `asm`. Este enlace ha de apuntar al subdirectorio que referencia la arquitectura sobre la que está trabajando el sistema (en la mayoría de los casos, al trabajar en un PC, apuntará a `asm-i386`). Con ello conseguimos que los ficheros que hagan referencia a una cabecera determinada no tengan que diferenciar en que arquitectura están siendo compilados, ya que referenciarán por ejemplo a `asm/io.h`, sin necesidad de saber si se trata realmente de `asm-i386/io.h` o `asm-alpha/io.h`. Esta es otra de las utilidades de los enlaces simbólicos en Unix.

- `init/`
  Aquí simplemente tenemos un par de ficheros con las funciones necesarias para la inicialización del núcleo. El núcleo de Linux no es más que un ejecutable algo especial, en C (con algunas partes en ensamblador, como ya hemos visto), y en los ficheros del directorio `init` tendremos lo que sería el equivalente (salvando las distancias) al fichero principal de un programa normal en C, donde estaría la función `main()` (aunque en el núcleo por supuesto no tenemos esta función como tal).

- `ipc/`
  En el directorio `ipc` encontraremos el código correspondiente a la implementación del IPC (*InterProcess Communication*) de Unix System V en el núcleo de Linux: semáforos, sistemas de paso de mensajes, memoria compartida...

- `kernel/`
  En este directorio tenemos la parte más importante del sistema, el núcleo del núcleo, las secciones principales de éste. Aquí encontramos la implementación de las principales llamadas al sistema, archivos como `sched.c` (el planificador de procesos), `signal.c` (gestor de señales), etc.

- `lib/`
  En el directorio `lib` tenemos codigo auxiliar para el arranque del sistema,en el que se apoyan otras partes del núcleo; por ejemplo, tenemos la rutina que nos permite utilizar una imagen del *kernel* comprimida (`/vmlinuz`) en lugar de una imagen sin comprimir, que seria `/vmlinux`. También encontramos en `lib` algunas funciones de librería estándares de C, para poder utilizar convenciones de la programación en este lenguaje para programar el núcleo.

- `mm/`
  Este es el directorio correspondiente a la gestión de memoria (*Memory Management*) de Linux. Tenemos el código encargado de la reserva de memoria para procesos, el de la paginación, el del mecanismo de *swap*, etc.

- `modules/`
  Cuando recompilamos un núcleo y construimos sus módulos, éstos se depositan en este directorio, para ser instalados después en `/lib/modules/XX.X.XX`, siendo `xx.x.xx` la versión

del *kernel* utilizada. Si no hemos recompilado el núcleo ninguna vez, este directorio estará probablemente vacío.

- `net/`
  Todo el subsistema de red de Linux está incluido en este directorio. Tenemos subdirectorios con los ficheros correspondientes al soporte de diferentes tipos de redes (Appletalk, Ethernet, Decnet...), así como a diferentes protocolos, como IPv4 e IPv6.

- `script/`
  En `script` tenemos diferentes `scripts` (pequeños programas interpretados, en lenguajes como `awk`, `sed` o `sh`) que nos van a ayudar en el proceso de configuración o compilación del núcleo. Por ejemplo, tenemos los ficheros en Tk para la configuración gráfica, el fichero *shellscript* de funciones para configuración en terminal con `ncurses`, o el fichero en `awk` para generar dependencias entre elementos del núcleo.

C.3 Configurando y compilando el núcleo

Lo primero que deberíamos hacer tras instalar un sistema Linux debería ser reconfigurar y recompilar el núcleo. O bien utilizamos el núcleo que acompaña a la distribución (poco recomendable, a no ser que sea muy nueva, porque seguramente será un *kernel* desfasado) o conseguimos la última versión estable. Seguramente este nuevo núcleo sea un fichero llamado linux-XX.YY.ZZ.tar.gz, por ejemplo `linux-2.0.36.tar.gz`; hemos de descomprimir y desempaquetar este fichero en el directorio `/usr/src/`. Al hacerlo generará un directorio denominado `linux-XX.YY.ZZ`, o simplemente `linux` (en este segundo caso hemos de borrar previamente el enlace simbólico `linux`, que apuntaría al directorio del código del antiguo núcleo) donde está el nuevo *kernel*.

Cuando hayamos descomprimido el fichero linux-XX.YY.ZZ tendremos en `/usr/src/` algo parecido a:

```
lrwxrwxrwx   1 root     root           12 Feb  6 10:08 linux -> linux-2.0.35/
drwxr-xr-x  15 root     root         1024 Feb  6 13:42 linux-2.0.35/
drwxr-xr-x  15 root     root         1024 Mar 21 16:58 linux-2.0.36/
```

En este caso hemos instalado el código de Linux 2.0.36 para sustituir al de Linux 2.0.35. Nos hemos de fijar en el enlace simbólico `linux`: está apuntando al directorio donde está el *kernel* antiguo. Por cuestiones de compatibilidad, lo borraremos (si no lo hemos hecho antes), y lo volveremos a crear apuntando al directorio del nuevo núcleo:

```
rosita:/usr/src# rm linux
rosita:/usr/src# ln -s linux-2.0.36 linux
rosita:/usr/src# ls -l
lrwxrwxrwx   1 root     root           12 Mar 21 17:08 linux -> linux-2.0.36/
drwxr-xr-x  15 root     root         1024 Feb  6 13:42 linux-2.0.35/
drwxr-xr-x  15 root     root         1024 Mar 21 16:58 linux-2.0.36/
rosita:/usr/src#
```

Ya tenemos todo listo para generar un nuevo núcleo, a partir de su código fuente. Cuando se habla de *compilar el kernel*, habitualmente nos referimos a tres pasos: en primer lugar hemos de **configurarlo** (indicarle los dispositivos de nuestra máquina, los tipos de sistema de ficheros que queremos utilizar, la red que utilizaremos...), después hemos de **generar las dependencias** para los ficheros *Makefile* con la configuración que hemos creado, y finalmente pasaremos al proceso de **compilación** en sí, que consiste en generar un fichero imagen del nuevo *kernel*, denominado habitualmente `vmlinux`. En la arquitectura PC podemos utilizar una imagen comprimida del núcleo para arrancar el sistema, que se llamará `vmlinuz`.

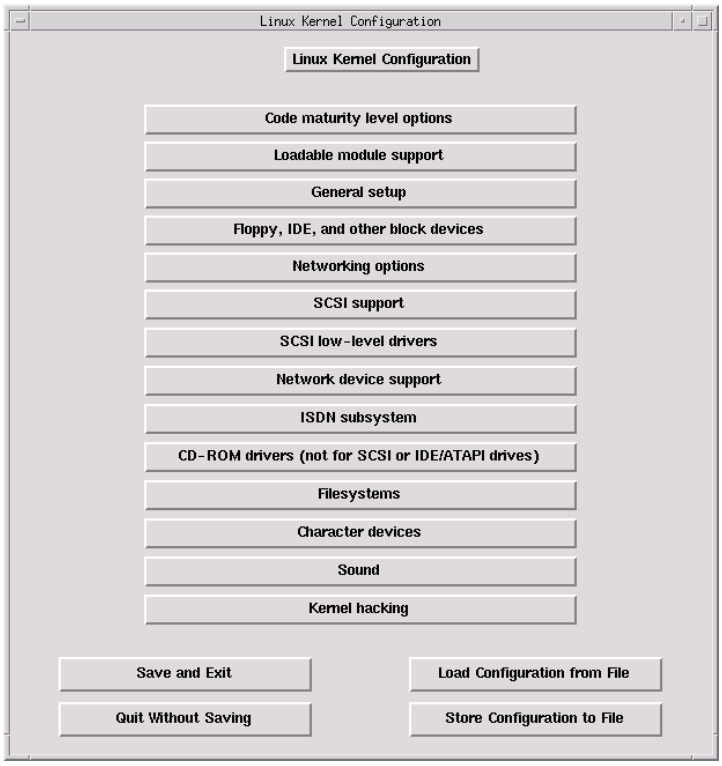


Figura C.2: Aspecto del interfaz Tk para la configuración del núcleo

El primer paso, la configuración, lo podemos realizar mediante la orden `make config` en el directorio `/usr/src/linux/` (en lo sucesivo, si no se indica lo contrario, asumiremos que trabajamos en este directorio). Si disponemos de un sistema con `ncurses`, una forma más comoda de configurar el núcleo es con `make menuconfig`, y si disponemos de entorno de ventanas la forma más cómoda de configuración es mediante `make xconfig`; este último mecanismo utiliza un interfaz Tk para seleccionar mediante el ratón todas las opciones de configuración que nos interesen. En la figura C.2 se muestra el aspecto de este interfaz. Aquí no vamos a entrar en detalles de la configuración o problemas de compatibilidad de *hardware* con Linux; aparte de la gran intuitividad que ofrece el interfaz de configuración y la ayuda *on-line* que se ofrece al configurar, hay muchísimos documentos en Internet que tratan los problemas de la configuración y explican el significado concreto de cada sección, como el *Kernel HOWTO*, traducido al castellano.

Una vez hemos configurado el núcleo (y grabado esta configuración) hemos de generar las dependencias del código; esto lo conseguimos simplemente tecleando `make config` desde el directorio `/usr/src/linux/`.

Finalmente, el tercer paso es la compilación del código propiamente dicha; dependiendo de la potencia de nuestra máquina, este es un proceso que puede durar desde unos minutos en un Pentium hasta varias horas en un 386. El resultado de la compilación será un fichero imagen, que va a ser el que nos permitirá arrancar el sistema con el nuevo núcleo.
Para generar este fichero imagen se ofrecen varios métodos. Los más elementales (y en la arquitectura PC los menos comunes) consisten simplemente en teclear `make`, que generará la imagen `vmlinux`, o `make boot` que generará esta misma imagen arrancable (en un PC, una imagen comprimida denominada `arch/i386/boot/zImage`). Con este fichero imagen ya podemos arrancar el sistema (por ejemplo, volcándolo a un diskette con la orden `dd if=zImage of=/dev/fd0`); sin embargo, tenemos otras opciones para generar la imagen, bastante más utilizadas en el entorno PC. Por ejemplo,

podemos utilizar `make zdisk` para que se escriba la imagen directamente en el diskette después de ser generada; si arrancamos mediante LILO (*LInux LOader*) podemos teclear `make zlilo` para que se genere la imagen, se copie en `/vmlinuz` (el fichero imagen `/vmlinuz` anterior, si existía, se guardará como `/vmlinuz.old`), y automáticamente se ejecute la orden `/sbin/lilo` para actualizar el arranque. Si lo que queremos es simplemente generar la imagen en `arch/i386/boot/zImage` también podemos teclear `make zImage`; si la imagen es demasiado grande para ser arrancada (ocupa más de 509 KB), ejecutando `make bzImage` aumentaremos el grado de compresión de ésta para hacerla más pequeña. Como vemos, tenemos una multitud de opciones para pasarle a `make`; las comentadas aquí son las más habituales, pero no las únicas (en la documentación relativa al núcleo se puede consultar la utilidad de todas).

A modo de resumen de lo visto aquí, imaginemos que queremos compilar un nuevo núcleo, disponemos de sistemas de ventanas, y la imagen queremos guardarla en un diskette (esto nos puede ser muy útil: si hay un problema en el arranque, simplemente sacamos el diskette y arrancamos con el núcleo antiguo). Teclearíamos lo siguiente:

```
rosita:/usr/src/linux# make xconfig
rosita:/usr/src/linux# make dep
rosita:/usr/src/linux# make zdisk
```

Puede darse el caso que hayamos reconfigurado un *kernel* que ya habíamos compilado, y queramos volver a generar una imagen. En este caso, es conveniente ejecutar `make clean` para eliminar los ficheros objeto anteriores, generalmente después de crear las dependencias (esto es, después de `make dep`, y antes de compilar el núcleo). Una *limpieza* más exhaustiva se consigue con `make mrproper`, que borrará incluso los ficheros generados al configurar el *kernel*: deja el código tal y como si lo acabáramos de descomprimir[2]

C.4 Módulos

Seguramente nos hemos fijado al compilar el núcleo que muchos elementos ofrecían tres opciones para su configuración: Y (sí), N (no) y M. Esta M indica que deseamos compilar el elemento correspondiente como un **módulo**. ¿Qué es esto? Un módulo es un fichero objeto (generalmente `.o`) que se puede insertar o eliminar de forma dinámica en el núcleo del sistema; es decir, estamos modificando el *kernel* en tiempo de ejecución, sin necesidad de reiniciar la máquina y arrancar con el nuevo núcleo.

Imaginemos esta situación: en nuestro sistema eventualmente deseamos ejecutar ficheros binarios de Java (ficheros `.class`, *bytestreams*); esto podemos hacerlo a través del programa `java` (`java fichero.class`), o, si tenemos soporte para este tipo de ficheros en nuestro núcleo, directamente dándole permiso de ejecución al archivo `.class` e invocándolo (`chmod +x fichero.class; fichero.class`). Sin embargo, esto sólo lo hacemos en ocasiones contadas; ¿debemos mantener el soporte para ficheros binarios de Java en el núcleo de forma permanente? NO. Podemos compilar el soporte como un módulo, y cuando deseemos ejecutar ficheros `.class` de esta forma, *añadir* ese módulo al núcleo que se está ejecutando, sin necesidad de parar la máquina y arrancar con otro *kernel*. Utilizando módulos para insertar elementos del núcleo que no utilicemos permanentemente conseguimos un núcleo más pequeño, y por tanto más rápido.

¿Cómo hacemos esto? Lo primero que debemos haber hecho es utilizar una M para los elementos deseados a la hora de configurar el núcleo; así estos elementos no se integran en la imagen del núcleo, `vmlinuz`, sino que se mantienen de forma independiente. Tras compilar el núcleo debemos hacer lo mismo con los módulos: compilarlos. Esto lo conseguimos con la orden `make modules`, como siempre desde el directorio `/usr/src/linux`. Tras compilar los módulos (ahora los tenemos en el directorio `modules`) hemos de instalarlos en un directorio determinado, donde Linux los buscará cuando le

[2]Este paso es recomendable después de aplicar un *patch* al núcleo, algo que no veremos aquí.

digamos que queremos insertar uno de ellos; este directorio es `/lib/modules/version-kernel/` (por ejemplo, `/lib/modules/2.0.36/`), y la orden para instalarlos de forma adecuada es `make modules_install`. Tras ejecutarla, en `/lib/modules/version-kernel` tendremos una serie de directorios (`scsi`, `net`...) con los módulos que hayamos generado.

Ahora llega la hora de insertar un módulo; para ello, disponemos de la orden `insmod`. Con `insmod modulo` insertamos el módulo deseado en el núcleo, y a partir de ese momento el sistema se comportará como si ese código estuviera realmente incluido en la imagen del *kernel*; con la excepción de que, igual que hemos añadido el módulo podemos eliminarlo, utilizando la orden `rmmod`: con `rmmod modulo` eliminamos el módulo del kernel que se está ejecutando (NO borramos el fichero objeto de ese módulo). Si en cualquier momento queremos ver los módulos insertados en el *kernel* que se está ejecutando en nuestro sistema, podemos utilizar la orden `lsmod`.

Vamos a ver todo esto con un ejemplo; imaginemos que, como hemos planteado antes, hemos incluido el soporte para ficheros binarios Java como un módulo en nuestro núcleo. Vamos a insertarlo, ver que realmente funciona, y eliminarlo, para ver también que una vez eliminado del *kernel* ha perdido su funcionalidad:

```
rosita:~# ls  /lib/modules/2.0.35/fs/binfmt_java.o
/lib/modules/2.0.35/fs/binfmt_java.o
rosita:~# insmod binfmt_java
rosita:~# lsmod
Module:        #pages:  Used by:
binfmt_java        1 0
rosita:~# chmod +x HelloWorld.class
rosita:~# HelloWorld.class
Hello World!
rosita:~# rmmod binfmt_java
rosita:~# lsmod
Module:        #pages:  Used by:
rosita:~# HelloWorld.class
bash: ./HelloWorld.class: cannot execute binary file
rosita:~#
```

Bibliografía

[Bac87] Bach, Maurice *The Design of the Unix Operating System.*
Prentice Hall, 1987

[Bec97] M. Beck et al *Linux Kernel Internals.*
Addison Wesley, 1997

[Car97] Card, Dumas, Mével *Programación Linux 2.0: API de sistema y funcionamiento del núcleo.*
Gestión 2000, 1997

[Che94] Cheswick & Bellovin *Firewalls and Internet Security: Repelling the wily hacker.*
Addison Wesley, 1994

[Chr94] Christian & Richter *The Unix Operating System.*
John Wiley and Sons, 1994

[Cur92] Curry *Unix System Security: A Guide for Users and Systems Administrators.*
Addison-Wesley, 1992

[Fri95] Frisch, Æleen *Essential System Administration.*
O´Reilly and Associates, 1995

[Gar91] Garfinkel & Spafford *Practical Unix Security.*
O´Reilly and Associates, 1991

[Hun91] Hunter & Hunter *Unix System - Advanced Administration and Management Handbook.*
MacMillan, 1991

[Ker84] Kernighan & Pike *The Unix Programming Environment.*
Prentice Hall, 1984

[Lam84] Lamport, Leslie *LaTeX, A document preparation system.*
Addison Wesley, 1984

[Man91] Manger, Jason *Unix: The complete book.*
Sigma Press, 1991

[Nem89] Nemeth et al *Unix System Administration Handbook.*
Prentice Hall, 1989

[Rib96] Ribagorda, Calvo, Gallardo *Seguridad en Unix: Internet y Sistemas Abiertos.*
Paraninfo, 1996

[Rei93] Reiss & Radin *Aplique X-Window.*
MacGraw-Hill, 1993

[Roc85] Rochkind *Advanced Unix programming.*
Prentice Hall, 1985

[Ste90] Stevens, Richard *Unix Network Programming.*
Prentice Hall, 1990

[Tan91] Tanenbaum, Andrew *Operating Systems: Design and Implementation.*
Prentice Hall, 1991

[Tan97] Tanenbaum, Andrew *Redes de Computadoras.*
Prentice Hall, 1997 (3 Edición)