Introducción a los Sistemas Operativos

Bibliografía:

- Apuntes de la asignatura, anónimos.
- Wikipedia, the free encyclopedia (https://en.wikipedia.org/).
- Páginas man de Fedora 29 de todos los comandos y ficheros que se explican en los apéndices en subsección propia, de sus respectivos autores, salvo que se indique específicamente lo contrario, así como la página signal(7).

Capítulo 1

Introducción

Un **sistema operativo** es un intermediario entre el usuario y el hardware, un programa complejo que se ejecuta cuando otro lo solicita o periódicamente para realizar ciertas tareas.

- Como **máquina extendida** o **virtual**, presenta una interfaz sencilla de la máquina al programador ocultando detalles de hardware de bajo nivel (interrupciones, relojes, control de memoria, etc.).
- Como **controlador de recursos**, proporciona una asignación ordenada y controlada de los recursos de hardware (procesadores, memoria, entrada y salida, etc.) entre los programas que compiten por ellos.

Destacamos Microsoft Windows, Apple MacOS X, Linux, y para dispositivos móviles, iOS y Android. Las características de un sistema operativo dependen del hardware en que se ejecuta; por ejemplo, este en general no puede garantizar la protección entre zonas de memoria de dos programas si el hardware no dispone de un mecanismo para ello. Por esto los sistemas operativos han evolucionado a la par con el hardware.

1.1. Historia

1.1.1. Primera generación (1945–55)

Grandes máquinas experimentales que ocupaban habitaciones enteras, formadas por miles de válvulas de vacío y cables conectados a mano, que consumían una gran cantidad de energía y eran muy poco fiables. Un solo grupo de personas diseñaba, construía, programaba, operaba y mantenía cada máquina. No existían lenguajes de programación ni sistemas operativos: la programación era en lenguaje máquina, incluyendo el código de E/S dentro del propio programa, y con frecuencia se usaban conexiones directamente para controlar funciones básicas.

La interacción con la máquina era directa: los programadores la reservaban durante un tiempo, en el que introducían y ejecutaban el programa. Si el trabajo terminaba en ese tiempo, había periodos en que la máquina no se usaba (al ser máquinas muy caras había que aprovecharlas al máximo), y si no daba tiempo a terminar, el programador tenía que volver a reservar la máquina.

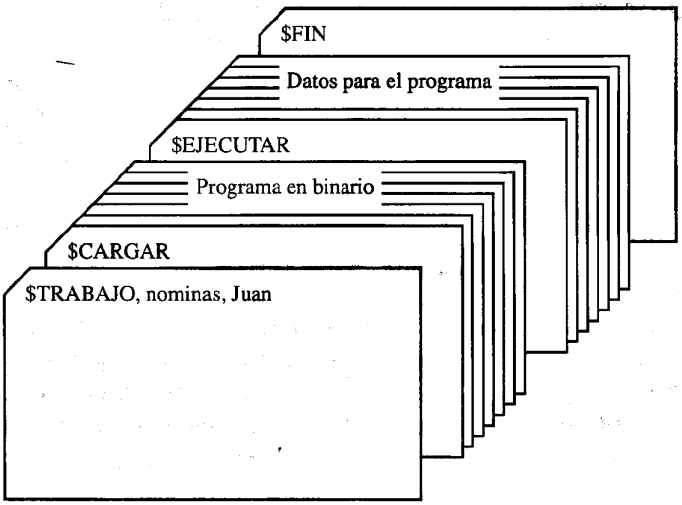

Figura 1.1: Ejemplo de trabajo por lotes.

1.1.2. Segunda generación (1955-65)

Aparecen desarrollos como los lectores de tarjetas, impresoras de líneas, cintas magnéticas y, por el lado del software, ensambladores, cargadores y enlazadores. La introducción del transistor hizo que los ordenadores se volvieran fiables, pudiendo venderse a clientes, y las bibliotecas de E/S hicieron que el programador ya no tuviera que preocuparse de este aspecto.

Si la interacción era directa, esta incluía colocar tarjetas perforadas, preparar la impresora, hacer cambios de cinta (por ejemplo, colocar la cinta con el compilador de FORTRAN para compilar un programa), etc. Si el programador no tenía experiencia en esto se podían producir grandes pérdidas de tiempo, por lo que se contrataba a un operador profesional que controlaba la máquina de forma que los programadores entregaban su trabajo y esperaban al resultado, perdiendo la interacción directa. Otra medida fue planificar los trabajos para minimizar los cambios de cinta (por ejemplo, agrupando todos los programas en FORTRAN para ser compilados tras preparar la cinta con el compilador).

Aun así, cuando se detenía un trabajo, el operador debía percatarse observando la consola, determinar si se trataba de una terminación normal o anormal, efectuar un volcado si era necesario, cargar el lector de tarjetas o la cinta con el siguiente trabajo y volver a preparar el ordenador. En este tiempo la CPU estaba inactiva, por lo que se desarrollaron sistemas de **procesamiento por lotes**, sistemas operativos rudimentarios, como el **monitor residente**, formado por un intérprete de tarjetas de control, un cargador de programas y controladores de dispositivos de E/S compartidos con los programas. Este interpretaba instrucciones de tarjetas de control insertadas entre tarjetas normales de datos, que indicaban el inicio y fin de un trabajo y tipo de trabajo a realizar.

Los dispositivos de E/S eran (y son) lentos en comparación con la CPU, que debía esperar la terminación de cada operación de E/S, por lo que se creó la **operación fuera de línea**, consistente en pasar de operaciones en línea (*on-line*) donde la CPU interactuaba directamente con la E/S lenta, a operaciones fuera de línea (*off-line*), donde la CPU interactuaba con

dispositivos más rápidos. Para esto, primero los datos se leían de las tarjetas y se guardaban en una cinta, a continuación los datos de la cinta se procesaban y se guardaban los resultados en otra cinta, y finalmente los datos de la cinta de salida se imprimen.

Esto hizo surgir la **independencia de dispositivo**, el escribir un programa para usar dispositivos de E/S lógicos y que fuera el sistema operativo el que hiciera corresponder los dispositivos lógicos con los físicos. También aumentó el tiempo que el programador debía esperar para obtener los resultados, pues las cintas se tenían que llenar, y también hizo necesarias varias máquinas en paralelo para hacer las copias de tarjetas a cintas y de cintas a la impresora. Solución:

- Hacer que los dispositivos de entrada escriban directamente en un ***buffer*** en memoria principal, mientras la CPU procesa los datos anteriores, de forma que la entrada y la CPU trabajen en paralelo. Lo mismo ocurre con la salida.
- Hacer que la entrada y la salida se den a través de ficheros en un disco, por el mecanismo de ***spooler*** (*simultaneous peripheral operation on-line*), de forma que durante el cómputo de un programa se imprime la salida de programas anteriores, leída del disco, y se lee la entrada de programas posteriores, que se imprime en el disco. Para esto el monitor necesita hacer uso de *buffers* e interrupciones.

Sigue habiendo la limitación de que un único usuario no puede mantener a la CPU y la E/S ocupados todo el tiempo.

1.1.3. Tercera generación (1965-80)

Los ordenadores se vuelven más pequeños y fiables. Se crea la **multiprogramación**: cuando un trabajo no puede continuar su ejecución porque tiene que esperar a la E/S, se pasa la CPU a otro trabajo listo para ejecutarse. Es necesaria una **planificación de trabajos**, así como evitar que estos trabajos se interfieran unos a otros en el uso de los recursos (por ejemplo, escribiendo a la vez en la impresora).

El **tiempo compartido** o **multitarea** es una variante de la multiprogramación consistente en hacer un cambio rápido entre tareas de forma que cada usuario pueda interactuar con el programa que está ejecutando como si todos se ejecutaran a la vez (**pseudoparalelismo**), recuperándose la interacción directa. Puede ocurrir que un programa no quiera liberar la CPU durante bastante tiempo, por lo que es necesario un mecanismo para obligarle a dejar la CPU.

Surgen sistemas operativos como OS/360 para el IBM 360, MULTICS y su sucesor UNIX, del que se derivan sistemas como Linux.

1.1.4. Cuarta generación (1980-95)

Los circuitos LSI (*Large Scale Integration*) y VLSI (*Very Large Scale Integration*) permiten la aparición en 1975 del primer **ordenador personal**, invento que popularizó los computadores y la informática. También aparecen las **estaciones de trabajo**, ordenadores personales muy potentes (para empresas) conectados entre sí en red.

Se populariza el sistema operativo UNIX para estaciones de trabajo y el MS-DOS, monousuario y monotarea, para ordenadores personales, si bien en estos comienzan a aparecer conceptos como la multitarea o la memoria virtual propios de UNIX. También aparecen las redes de ordenadores personales, dando lugar a sistemas operativos en red y distribuidos; los

ordenadores con varias CPUs que permiten ejecutar varios procesos a la vez (**paralelismo**), y los sistemas operativos de tiempo real.

1.1.5. Quinta generación (1995-)

Explosión del uso de Internet y aparición de dispositivos móviles, especialmente los llamados **teléfonos inteligentes** o *smartphones*, que combinan telefonía y computación, siendo el primero el Nokia N9000, lanzado a mediados de los 90, si bien estos no se popularizaron hasta el iPhone de Apple en 2007.

De Microsoft aparecen Windows 2000 y sus sucesores Server 2003, 2008, etc., destinados a servidores, y Windows XP y sus sucesores Vista, 7, etc., destinados a usuarios. También alcanzan gran popularidad Linux, MacOS X, Android e iOS.

1.2. Tipos

- **De propósito general**, que destacan por su flexibilidad y capacidad para adaptarse, mediante configuraciones concretas, tanto al hardware sobre el que se ejecutan como a los trabajos a procesar. Destacan Unix (principalmente Linux), Microsoft Windows y Apple MacOS X.
  - Unix es el más usado en **supercomputadores**, formados por un gran cantidad de procesadores, memoria, discos, etc. y capaces de procesar enormes cantidades de datos en poco tiempo, así como en los ***mainframes***, capaces de procesar a la vez muchos trabajos que requieren enormes cantidades de E/S (sobre todo, disco y red), y que suelen servir para procesamiento por lotes o *batch* (sin interacción con usuarios), transacciones (bancarias, reservas de vuelo, etc.) y trabajos multitarea (cuando los usuarios interactúan con el sistema).
  - Al pasar a sistemas menos potentes como **servidores**, computadores que sirven a varios usuarios a la vez compartiendo recursos hardware y software en red, podemos encontrar tanto Unix como MacOS X o incluso Windows. Esto también ocurre en los ordenadores personales, caracterizados por la multitarea y estar diseñados para dar buen rendimiento a un único usuario, y que hoy en día son **multiprocesadores**, pues disponen de varios **núcleos** o *cores* que actúan como CPUs separadas y comparten memoria principal.
  - En teléfonos inteligentes, tabletas y sistemas integrados, pequeños ordenadores con menos recursos que uno normal pero con un consumo de energía mucho menor, encontramos versiones de estos tres sistemas adaptados a estos dispositivos y a la interacción por pantalla táctil.
- **De red**: Permiten interactuar con otros ordenadores conectados a una misma red, si bien cada máquina es independiente. Actualmente esto es lo común, y de hecho los tres sistemas de propósito general que hemos visto también son de red.
- **Distribuidos**: También se conectan a otros mediante red, pero el conjunto de ordenadores en esta red es visto como un sistema tradicional, con lo que el usuario no es consciente de dónde se ejecutan los programas o dónde se encuentran los ficheros (**transparencia**

de localización). Las ventajas son la compartición de recursos, aceleración de los cálculos y tolerancia a fallos, pues si una máquina falla el resto puede seguir haciendo todo el trabajo. Destacan Amoeba, Plan 9 e Inferno, si bien todos han dejado de desarrollarse.

- **De tiempo real**: En los **rigurosos**, usados por ejemplo en coches y plantas nucleares, una acción se debe realizar necesariamente en cierto intervalo de tiempo o de lo contrario se podría poner el sistema en peligro. En los **no rigurosos** es aceptable no cumplir de vez en cuando un plazo, dentro de unos límites. Destacan VxWorks y QNX.

- **Para tarjetas inteligentes**, debido a las grandes limitaciones de potencia y memoria que presentan. Suelen disponer de una máquina virtual de Java (JVM) que ejecuta los *applets* que se cargan en la tarjeta, pequeñas porciones de código que no se pueden ejecutar por sí mismas sino que necesitan de un entorno proporcionado por otro programa.

1.3. Programas y procesos

Un **programa** es la representación estática y almacenada de un conjunto de instrucciones a ejecutar. Cuando es cargado en memoria y se ejecuta, lo hace como un **proceso**, la unidad de trabajo de un sistema, y de los cuales algunos pertenecen al propio sistema operativo y el resto son procesos de usuario. Un proceso es dinámico, requiriendo recursos como tiempo de CPU, memoria, ficheros y dispositivos de E/S. Varios procesos pueden ejecutar el mismo programa y un proceso (dependiendo del sistema) puede ejecutar código de varios programas.

En general los sistemas operativos se distribuyen junto con **programas de sistema**, cuya misión es ofrecer un entorno más cómodo para el desarrollo y ejecución de programas. Tipos:

- **De manipulación de ficheros**.

- **De información de estado**, que solicitan información al sistema operativo, le dan formato y la imprimen.

- **De apoyo a lenguajes de programación**, como compiladores, ensambladores, intérpretes y depuradores.

- **De comunicaciones**, para crear conexiones virtuales entre procesos, usuarios o máquinas, permitiendo el envío de mensajes, correos electrónicos o ficheros a usuarios de la misma máquina u otra y la conexión a una máquina remota.

- **De aplicación**, para efectuar otras operaciones comunes. Incluímos aquí ditores de texto, generadores de gráficos, sistemas de bases de datos, hojas de cálculo, paquetes de análisis estadístico, etc.

- **Intérprete de órdenes**, encargado de interpretar y ejecutar órdenes dadas por el usuario. Las órdenes pueden estar contenidas en el código del propio intérprete o en programas independientes.

Aunque no hay diferencia entre un programa de usuario y uno del sistema, la mayoría de usuarios suelen ver el sistema operativo desde el punto de vista de los programas del sistema.

1.4. Componentes y servicios

La interfaz entre el sistema operativo y un proceso tiene lugar mediante **llamadas al sistema**, con la siguiente estructura:

1. El programa ejecuta la instrucción de la CPU para realizar la llamada al sistema, cambiando la máquina a modo núcleo y pasando el control al sistema operativo.

2. Este lee el número de la llamada al sistema a realizar, y una vez validado, llama al procedimiento correspondiente indicado en una tabla de apuntadores.

3. La llamada termina y el control regresa al proceso de usuario, que continua su ejecución por la instrucción inmediatamente posterior.

El usuario informa al sistema operativo del número de la llamada a realizar y los parámetros necesarios (habitualmente) mediante los registros, la pila o una zona de memoria indicada en un registro. Las bibliotecas de lenguajes de programación suelen contener procedimientos que ocultan las llamadas al sistema, proporcionando una interfaz más sencilla.

A continuación vemos los distintos componentes de un sistema operativo típico junto a los servicios que ofrecen y las llamadas al sistema asociadas.

1.4.1. Administración de procesos

El sistema operativo debe ser capaz de crear y eliminar procesos, controlar el tiempo de CPU que se le da a cada uno y cargar programas en estos procesos, además de proveer medios para que los procesos se comuniquen entre sí.

Existen pues llamadas al sistema para crear y terminar procesos; finalizar el proceso actual; obtener y establecer atributos de un proceso; cargar un programa; esperar un tiempo; esperar un suceso; enviar una señal a otro proceso; crear o eliminar una conexión; enviar o recibir mensajes, o transferir información de estado.

1.4.2. Administración de memoria principal

El esquema de administración de memoria depende sobre todo del hardware, que el sistema operativo debe usar para controlar de qué procesos están usando qué zonas de memoria, asignar y recuperar memoria según se requiera y decidir qué procesos se cargarán en memoria cuando haya espacio disponible. Debe pues haber llamadas al sistema para solicitar y liberar memoria.

1.4.3. Administración de E/S

En UNIX, esta se consigue mediante una interfaz uniforme con los manejadores de dispositivo, un sistema de memoria caché y manejadores de hardware específicos para las particularidades de cada dispositivo.

Existen pues llamadas al sistema para solicitar o liberar dispositivos; leer y escribir en ellos; reposicionarnos; obtener y establecer atributos de dispositivos, y unir o separar dispositivos lógicos, sean físicos y conectados a la propia máquina o accedidos remotamente mediante red.

1.4.4. Administración de ficheros

Por comodidad, el sistema operativo organiza la memoria de los dispositivos de almacenamiento secundario en **ficheros**, unidades de almacenamiento lógico, que a su vez se organizan en directorios.

Existen pues llamadas al sistema para crear y eliminar ficheros y directorios; abrir y cerrar ficheros; leer, escribir y reposicionarnos en ficheros abiertos; obtener y establecer atributos de ficheros y directorios, y manipular el contenido de estos directorios.

1.4.5. Sistema de protección

Los procesos de un sistema operativo deben ser protegidos unos de otros, y el sistema operativo debe ser protegido de estos. Para ello existen tres mecanismos hardware:

- **Modos de ejecución** del procesador: el **modo núcleo** o **supervisor**, en el que se ejecuta el sistema operativo por ser así capaz de ejecutar cualquier instrucción, y el **modo usuario**, en el que se ejecutan los programas de usuario y que no puede ejecutar ciertas instrucciones como de E/S, configuración de memoria, etc. Los programas de usuario, para realizar E/S, deben usar una instrucción especial de **llamada al sistema** (*syscall*), tratada como una interrupción que, como todas, es tratada por el sistema operativo en modo núcleo.
- **Protección de memoria**: Impedir que un programa de usuario acceda a zonas de memoria de otros programas o del sistema operativo.
- **Interrupciones periódicas** mediante un reloj: Permiten al sistema operativo hacerse con el control de la máquina cada cierto tiempo para evitar que un proceso monopolice la CPU.

Esta protección solo funciona si los tres mecanismos funcionan y se usan de forma conjunta. El sistema operativo es el encargado de hacer funcionar estos mecanismos y usarlos para asegurarse de que los recursos puedan ser usados únicamente por los procesos que han recibido la correspondiente autorización. Además, el sistema operativo debe ser capaz de lidiar con los errores que se puedan producir a nivel de hardware o dentro de un programa de usuario, y emprender la acción adecuada en cada caso.

1.4.6. Mantenimiento de información

Si bien esto generalmente no corresponde a una única parte del sistema operativo, puede ser deseable llevar un control del uso de recursos del ordenador por parte de los usuarios, sea con fines contables para facturar a los usuarios o sencillamente para recopilar estadísticas de uso del sistema que son útiles para su administración. Además, tanto los procesos como los dispositivos o ficheros pueden poseer información adicional que se debe leer y modificar aparte, y el reloj es tratado de forma especial al ser usado en el sistema de protección.

Existen pues llamadas al sistema para obtener o establecer la fecha y la hora, datos del sistema y atributos de proceso, fichero o dispositivo.

1.5. Estructura

1.5.1. Sistemas monolíticos

Están formados por un conjunto de procedimientos que se llaman unos a otros. Ofrecen una interfaz muy clara, pero no tienen una estructura formalmente definida y cada procedimiento es visible a los demás. Están formados por el **núcleo** (el sistema operativo como tal, junto a todos los controladores de dispositivo o *drivers*) y los programas de sistema. Destacan Unix (incluyendo Linux) y Windows.

1.5.2. Sistemas por capas

El sistema operativo se divide en **capas** o **niveles**, siendo la más baja (capa 0) el hardware y la más alta la interfaz con el usuario, de forma que cada capa sólo usa funciones y servicios de capas inferiores. Esto simplifica la depuración y verificación, pues usa vez se depura una capa se asciende a la siguiente, y proporciona ocultación de información, haciendo posibles las modificaciones en una capa en cualquier momento siempre que no se modifique la interfaz al resto de capas. Sin embargo, da problemas de dependencias entre capas, pues a veces es difícil saber dónde colocar una función, por lo que se opta por tener menos capas con más funcionalidad.

Destacan THE (1968) que se definió en 6 capas (hardware, planificación de CPU, administración de memoria, consola del operador, administración de E/S y programas de usuario) y VENUS, donde las capas inferiores se implementaron directamente en hardware.

1.5.3. Sistemas cliente-servidor

Están formados por un **micronúcleo** o *microkernel* y una serie de procesos de usuario que implementan las funciones típicas de un sistema operativo. Así, los procesos **clientes** solicitan servicios de los procesos **servidores**, que realizan los trabajos solicitados por los clientes y a su vez pueden ser clientes de otros procesos, de modo que el micronúcleo se limita a controlar las comunicaciones cliente-servidor. Para la E/S, bien se incluye el servidor correspondiente en el modo núcleo o existen mensajes especiales dirigidos al núcleo para que los procese él mismo.

Estos sistemas tienen la ventaja de que un fallo en un servidor concreto no afecta a todo el sistema, y que pueden adaptarse como sistemas distribuidos dado que no importa si los procesos que se comunican están en la misma máquina o no. Destacan QNX, Minix 3 y MacOS X, cuyo núcleo está basado en la versión 2.5 del micronúcleo Mach desarrollado en los 80 en la Carnegie-Mellon University.

Capítulo 2

Gestión de procesos

Normalmente hay bastantes más procesos que CPUs, por lo que hay que repartir el uso de estas por parte de los procesos, buscando bien la eficiencia, como en sistemas antiguos, o el poder ejecutar varios programas dando la sensación de que todos se ejecutan a la vez, como ocurre actualmente. El paralelismo y el pseudoparalelismo, que se dan normalmente a la vez, permiten compartir recursos físicos y lógicos entre varios resultados y acelerar los cálculos dividiéndolos entre varias CPUs, además de aportar modularidad, al poder diseñar un sistema como un conjunto de procesos, y comodidad, permitiendo a los usuarios hacer varias cosas a la vez.

El cambio de un proceso a otro se llama **cambio de proceso** o de **contexto**. El **planificador** es el procedimiento que decide el siguiente proceso a ejecutar mediante un **algoritmo de planificación**, y modifica las estructuras necesarias para que el cambio de contexto sea correcto.

En UNIX, los procesos se organizan en un árbol de procesos y se crean con la llamada al sistema `fork()`, que crea una copia idéntica del proceso que hace la llamada y devuelve al hijo el valor 0 y al padre el PID (*process identifier*) del proceso hijo. El PID del propio proceso se obtiene con la llamada al sistema `getpid`. Las llamadas de la familia `exec`, como `execve`, hacen que un proceso sustituya su código y datos por los de otro programa, que comienza la ejecución desde el principio, si bien se conservan los ficheros abiertos (aunque se puede especificar el cierre de algunos descriptores de ficheros en `execve`), el PID y casi todas las propiedades.

En Windows la jerarquía de procesos de UNIX desaparece al poder un proceso cambiar de proceso padre, y los procesos se crean con `CreateProcess`, que equivale a ejecutar un `fork` seguido de `exec` en el proceso hijo.

La terminación voluntaria de un proceso se hace mediante la llamada al sistema `exit` en UNIX y `ExitProcess` en Windows. La terminación involuntaria es producida por el sistema operativo si se produce un error fatal (división por cero, acceso a una zona memoria incorrecta, etc.) o por otro proceso autorizado a ello con la llamada al sistema `kill` en UNIX o `TerminateProcess` en Windows.

2.1. Estados

Un proceso puede estar en uno de varios estados:

- **Nuevo**: El proceso acaba de ser creado y todavía no tiene los recursos necesarios.
- **En ejecución**: Utilizando la CPU.
- **Listo** o **listo suspendido**: Ejecutable pero detenido porque otro proceso está usando la CPU.
- **Bloqueado** o **bloqueado suspendido**: A la espera de un evento externo.
- **Saliente** (*zombie*): El proceso ha terminado su ejecución pero todavía no ha desaparecido del todo. Por ejemplo, en UNIX, cuando un proceso acaba debe esperar a que su padre recoja su estado de salida mediante la llamada al sistema `wait` o `waitpid`.

Un proceso suspendido no puede ejecutarse y suele guardarse en disco. La **suspensión** y posterior **reanudación** de un proceso, por parte de otro(s), es útil:

- Si el sistema está funcionando mal, mientras se corrige el problema.
- Si se cree que los resultados del proceso son incorrectos, para comprobar si lo son.
- Si el sistema está muy cargado, para dar servicio a procesos de más prioridad.

Cambios de estado:

- De «Nuevo» a «Listo» cuando ya dispone de los recursos o a «Listo suspendido» si no hay memoria disponible y el proceso no tiene prioridad suficiente para expulsar a otro.
- De «Listo» a «En ejecución» o viceversa, debido al planificador.
- De «En ejecución» a «Bloqueado», si el proceso hace una llamada al sistema que no se puede responder inmediatamente.
- De «Bloqueado» a «Listo» o de «Bloqueado suspendido» a «Listo suspendido», cuando ocurre el evento esperado.
- De «Bloqueado» a «Bloqueado suspendido» o de «Listo» a «Listo suspendido», o viceversa.
- De «En ejecución» a «Saliente» cuando termina.

2.2. Implementación

El SO mantiene una **tabla de procesos** con una entrada, **PCB** (*Process Control Block*) o BCP para cada proceso, donde se guarda todo lo necesario para poder continuar la ejecución tras perder la CPU y recuperarla luego, junto con datos estadísticos, el estado del proceso, etc. En concreto suelen guardarse:

- Para **administración de procesos**, PID, registros (incluyendo contador del programa, palabra de estado y puntero de pila), estado del proceso, prioridad, parámetros de planificación, PID del proceso padre, grupo de procesos, señales, hora de inicio, tiempo usado de CPU.

- Para **administración de memoria**, apuntadores a los segmentos de texto, datos, memoria dinámica.
- Para **administración de ficheros**, directorio raíz, directorio actual, descriptores de fichero, identificadores de usuario y grupo.

Crear un proceso consiste en darle nombre (como el PID en UNIX), insertarlo en la tabla de procesos, determinar su prioridad inicial y asignarle recursos iniciales. En el caso de UNIX, `fork` busca una entrada libre en la entrada de procesos y copia la información del PCB del padre a esta, cambiando el PID; evita que uno sobrescriba la memoria del otro[1]; incrementa los contadores de los descriptores de ficheros del padre para reflejar que estos también están abiertos en el hijo, y marca el proceso hijo como «Listo».

Los procesos en UNIX tienen dos modos: modo usuario, en que ejecutan un programa normal, y modo núcleo, común a todos los procesos, en que ejecutan código del SO al hacer una llamada al sistema. En este último los procesos pueden bloquearse esperando a algún evento, lo que hace que se le ceda la CPU a otro proceso hasta que el evento suceda, momento en que el proceso continuará su ejecución en modo núcleo. Otro enfoque es considerar el SO como una colección de procesos de sistema distintos a los procesos de usuario, de modo que cada proceso ejecuta sólo código de usuario, lo que ocurre en los sistemas cliente-servidor.

En las llamadas al sistema, el hardware almacena en una pila el contador de programa del proceso que se interrumpe, pasa a modo núcleo y almacena un elemento del vector de interrupciones, que debe ser la dirección de inicio de alguna rutina del SO, en el contador de programa. Entonces, un procedimiento guarda el contexto del proceso activo en su PCB (pudiendo actualizarse otra información como estado, contabilidad o auditoría), elimina la información introducida en la pila por el hardware, configura una pila en el núcleo para la llamada al sistema, comprueba que esta es válida y llama a un procedimiento que procesa la llamada. Tras esta, el primer procedimiento ejecuta el planificador si lo determina necesario, teniendo en cuenta que, si el proceso se ha bloqueado en la llamada, este ya ha sido invocado y no es necesario llamarlo de nuevo, y finalmente ejecuta el **despachador**, que restaura el contexto del proceso a ejecutar, entre otras cosas cambiando a la pila de usuario e introduciendo en esta la dirección de su contador de programa.

2.3. Hilos

Un proceso tradicional es:

- Unidad de propiedad de recursos, con un espacio de direcciones, variables globales, ficheros abiertos, procesos hijos, alarmas, señales, semáforos, información contable, etc.
- Unidad de planificación y ejecución, con un contador de programa, una serie de registros, una pila y un estado.

Estas funciones son independientes, y los sistemas operativos modernos llaman **hilos** (*threads*), **procesos ligeros** (*LightWeight Processes*, LWP), **hebras** o **subprocesos** a las unidades de

[1] Según los apuntes se copian los segmentos de datos y de pila, y el de código se comparte por ser de sólo lectura. Realmente, desde hace mucho Linux implementa copia en escritura, consistente en marcar las entradas de la tabla de página de ambos de forma que el núcleo solo tenga que hacer la copia de una página cuando uno de los procesos escriba en ella.

ejecución y procesos a las de propiedad de recursos, de forma que un mismo proceso puede tener uno o varios hilos. Cuando comienza la ejecución de un programa existe un único hilo, el **hilo principal**, que puede crear otros.

En Linux y otros muchos UNIX, un hilo puede crear otro mediante `pthread_create`, que entre sus parámetros recibe un puntero a otra función que es donde comienza la ejecución el hilo hijo. Todo hilo tiene un TID (*thread identifier*), que se obtiene desde el propio hilo con la llamada al sistema `gettid`. Para facilitar la colabolación entre hilos, que pueden necesitar acceder y modificar las mismas variables globales, existen métodos de sincronización como *mutex* (exclusión mutua) y semáforos.

Ventajas de los hilos:

- Al compartir espacio de direcciones, se pueden comunicar entre sí sin intervención del núcleo, por lo que esta comunicación es más rápida.
- Los hilos se pueden bloquear mientras termina una llamada al sistema, por lo que si hay varios, la E/S puede solaparse con el cómputo.
- Es mucho más rápido crear un hilo en un proceso existente que un nuevo proceso, y el cambio de uno a otro es más rápido.
- Se puede conseguir paralelismo real dentro de un mismo proceso.

Los hilos pueden ser soportados directamente por el núcleo, como en Windows y Linux, o ser implementados mediante una biblioteca a nivel de usuario, como se hacía en Linux antes de que implementara hilos en el núcleo.

La implementación en modo usuario tiene como ventajas que el programa se puede usar en núcleos que no implementan hilos, los cambios de contexto son mucho más rápidos al no tener que pasar por el núcleo y cada proceso puede tener un algoritmo distinto de planificación. Sin embargo, en este caso una llamada al sistema bloqueante, o un fallo de página, bloquearía a todos los hilos del proceso, y no es posible obtener paralelismo real dentro de un mismo proceso.

En la implementación en modo núcleo, la creación, destrucción y sincronización entre hilos es más costosa, pero esto se puede aliviar usando una biblioteca de sincronización que sólo realice llamadas al sistema cuando sea estrictamente necesarios, y manteniendo las estructuras de un hilo después de que termine para poder usarlo posteriormente en vez de crear otro nuevo. En cualquier caso todos los hilos deben pasar a estado suspendido al mismo tiempo, pues la memoria de todo el proceso se mueve al disco.

2.4. Planificación

Metas:

- **Equidad**: Dar a cada proceso una proporción adecuada de CPU.
- Maximizar la **eficacia** de la CPU, $E := \frac{\text{Tiempo útil}}{\text{Tiempo total}} \cdot 100$, y el **rendimiento** o **productividad**, el número de tareas procesadas por unidad de tiempo.

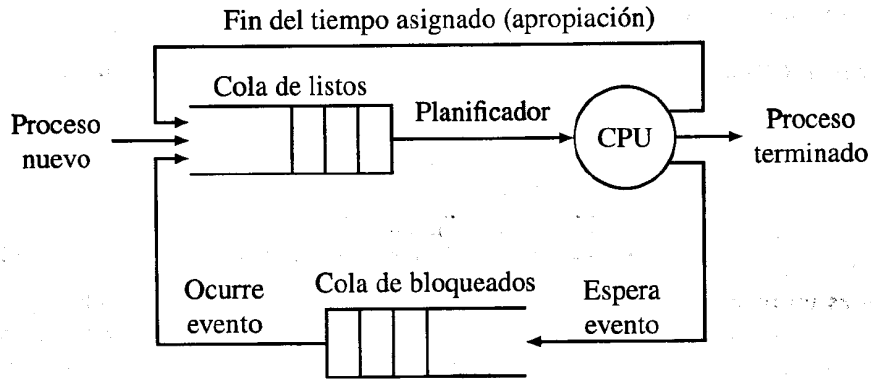

Figura 2.1: Diagrama de colas de planificación de procesos.

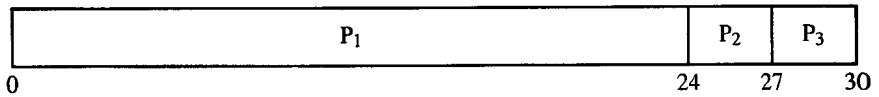

Figura 2.2: Diagrama de Gantt.

- Minimizar el **tiempo de espera** (el que pasa un proceso en estado «Listo»), el de **respuesta** (el que pasa desde que se solicita la ejecución de una acción hasta que se obtienen los primeros resultados, importante para usuarios interactivos) y el de **regreso** o **retorno** (el que pasa desde que se entrega a un trabajo hasta que termina y se obtienen sus resultados, importante para trabajos por lotes).

Algunas de estas, como el tiempo de espera y la eficacia, son contradictorias.

Una planificación es **apropiativa** si puede expulsar a procesos de la CPU para ejecutar otros sin necesidad de que se bloqueen, y **no apropiativa** o **de ejecución hasta terminar** en caso contrario. Para los procesos por lotes son convenientes tanto una planificación no apropiativa como una apropiativa con periodos de CPU largos (pues los cambios de proceso desperdician CPU), mientras que para procesos interactivos suele ser necesaria una planificación apropiativa para cambiar la CPU rápidamente de un proceso a otro.

Los procesos alternan entre ráfagas de CPU y de E/S, comenzando y terminando por una de CPU. Decimos que un proceso es **limitado por E/S** si pasa la mayor parte de su tiempo en espera de E/S, y normalmente tendrá muchas ráfagas de CPU aunque breves, y es **limitado por CPU** si usa la CPU la mayor parte del tiempo, y normalmente tendrá pocas ráfagas de CPU aunque muy largas.

Podemos representar un algoritmo de planificación mediante un **diagrama de colas**, en el que cada rectángulo es una cola, los círculos son recursos que dan servicio a las colas y las flechas indican el flujo de los procesos. Un **diagrama de Gantt** representa el proceso en ejecución en cada momento mediante rectángulos de longitud proporcional a la duración de la ráfaga de CPU.

Algoritmos no apropiativos:

- **FCFS** (*First Coming, First Served*, «primero en llegar, primero en ser servido»). Es el más simple y se puede implementar con una cola FIFO. El tiempo medio de respuesta puede ser bastante largo debido al **efecto convoy**: Un proceso limitado por CPU ocupa la CPU mucho tiempo mientras el resto terminan su E/S y entonces los dispositivos de E/S quedan inactivos hasta que el proceso en ejecución pasa a realizar su E/S; entonces los procesos limitados por E/S pasan a usar la CPU y terminan rápido por tener ráfagas de CPU breves y la CPU queda inactiva hasta que el proceso limitado por CPU continúa su ejecución, momento en que el ciclo se repite, resultando en desaprovechamiento de recursos.

- **SJF** (*Shortest Job First*, «primero el trabajo más corto»). Adecuado para procesos por lotes, donde los tiempos de ejecución aproximados se suelen conocer de antemano, y de lo contrario se puede estimar mediante **maduración**: $E_t = aE_{t-1} + (1-a)T_{t-1}$, donde E_t es la estimación de tiempo actual, E_{t-1} la estimación anterior para el mismo proceso, T_{t-1} el tiempo que tardó realmente y a un parámetro ajustable entre 0 y 1, pues un valor de a pequeño hace que se olviden rápidamente los tiempos de ejecuciones anteriores y un valor grande hace que se recuerden demasiado tiempo. Si se dispone de todos los procesos de forma simultánea, este algoritmo proporciona el mínimo tiempo medio de retorno.

Algoritmos apropiativos:

- **SRTF** (*Shortest Remaining Time First*, «primero el que tenga el menor tiempo restante»). Variante del SJF que permite quitar la CPU a un proceso para dársela a otro con tiempo total de ráfaga de CPU menor al tiempo restante de CPU de la ráfaga del proceso que se está ejecutando.

- ***Round Robin*** (**RR**) o **circular**. Es de los más antiguos, sencillos, equitativos y de mayor uso. Similar al FCFS pero con un *quantum* de tiempo q tal que, si un proceso consume un *quantum*, pasa al final de la cola de procesos listos para dar la CPU al siguiente proceso. De esta forma, si hay n procesos, ninguno tiene tiempo de espera mayor a $(n-1)q$. Si q es pequeño se desperdicia tiempo de CPU en cambios de proceso, pero si es grande, los últimos procesos tardan mucho en ser atendidos, resultando en tiempos de respuesta muy pobres en procesos interactivos.

Otras formas de planificación pueden hacerse apropiativas o no apropiativas:

- **Planificación por prioridad**. Cada proceso tiene una prioridad, normalmente un número que es mayor a mayor prioridad (en UNIX es al revés), de modo que la CPU se da al primer proceso de la cola con la mayor prioridad. La asignación de prioridad puede ser **estática**, si no cambia durante la ejecución del proceso, o **dinámica** si depende de parámetros. Un ejemplo de asignación dinámica para favorecer a los procesos limitados por E/S, y que puedan ejecutarse pronto para enviar su siguiente solicitud de E/S, es hacer que la prioridad sea inversamente proporcional a la fracción del último *quantum* usado por el proceso. Este *quantum* puede ser una referencia para medir el tiempo consumido o funcionar como en la planificación *round-robin*.
  El **bloqueo indefinido** o **inanición** ocurre cuando los procesos de baja prioridad nunca llegan a ejecutarse. Para evitarlo se puede disminuir cada cierto tiempo la prioridad del

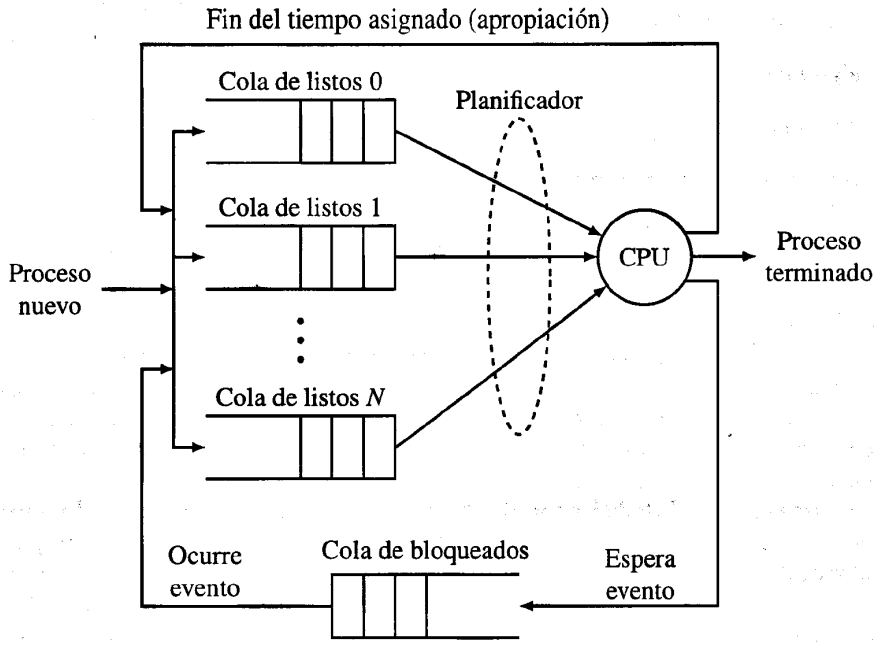

Figura 2.3: Planificación de múltiples colas.

proceso en ejecución, lo que tiene sentido si la planificación es apropiativa. Esto sigue teniendo el problema de que pueden llegar continuamente procesos de mayor cantidad que los que llevan tiempo esperando, por lo que una alternativa es aumentar cada cierto tiempo la prioridad de los procesos listos.

- **Planificación de múltiples colas con realimentación**. Es la más general, pero también la más compleja. Existen varias colas de procesos listos y un proceso va a una u otra según si es interactivo o por lotes, de sistema o de usuario, su prioridad, la cola que usó la última vez, el consumo de CPU, etc. Debe haber una planificación dentro de cada cola, y una planificación entre colas, que suele ser apropiativa por prioridad pero puede ser cualquiera, como el reparto equitativo y que cada cola se administre de una forma. Para evitar que el esquema sea inflexible, debe haber un criterio para que los procesos cambien de cola (realimentación).

Si no se dispone de suficiente memoria es necesario que algunos procesos se mantengan en disco. Para ello, los procesos en memoria los gestiona un **planificador a corto plazo** (PCP) mientras que un **planificador a medio plazo** (PMP) es llamado periódicamente y se encarga de la suspensión y reanudación de procesos, teniendo en cuenta criterios como el tiempo desde el último intercambio, el tiempo de CPU que ha consumido el proceso recientemente, el tamaño del proceso (los pequeños no causan problemas) y la prioridad de este. También es posible añadir un **planificador a largo plazo** (PLP) que decida, de entre los procesos por lotes preparados, cuál ejecutar después, eligiendo si este pasa directamente a memoria o a disco.

Capítulo 3

Seguridad y protección

Requisitos para la seguridad y amenazas asociadas:

- **Confidencialidad**: La información, incluyendo la existencia de esta, solo debe poder ser leída por usuarios autorizados.
  - **Interceptación**: Leer un elemento sin autorización. Incluye copias y lecturas no autorizadas, análisis estadístico para revelar datos ocultos y observación del tráfico de mensajes.
- **Integridad**: Los elementos del sistema solo deben poder ser modificados, incluyendo borrado y creación, por usuarios autorizados.
  - **Modificación**: Falsificar un elemento o modificarlo sin autorización, incluyendo alteración de un programa en funcionamiento.
  - **Invención**: Insertar objetos falsos en el sistema, incluyendo ficheros y mensajes.
- **Disponibilidad**: Los elementos de un sistema deben estar disponibles para usuarios autorizados.
  - **Interrupción**: Destruir un elemento o hacerlo inaccesible o inútil, incluyendo robo de equipos, consumo excesivo de recursos, eliminación de ficheros o mensajes y corte de las líneas de comunicaciones.

Para probar la seguridad de un sistema se suele contratar a un grupo de expertos (equipo tigre o de penetración) para ver si puede penetrar en él. Ataques más comunes:

- Reservar espacio de memoria o disco y leerlo, pues muchos sistemas no borran el espacio antes de asignarlo.
- Intentar llamadas inválidas, o con parámetros inválidos o no razonables.
- Conectarse un sistema y pulsar `C-c` o similares, terminando en ciertos sistemas el programa de verificación de contraseñas y permitiendo el acceso.
- Escribir un programa que aparente ser para iniciar sesión pero realmente filtren la contraseña.

- Buscar acciones que se conoce que no se deben llevar a cabo y hacerlas con todas sus variaciones.
- Convencer al administrador para que modifique el sistema de alguna forma que evite verificaciones de seguridad.
- Engañar o sobornar al personal de administración del centro de cálculo.

Ataques específicos:

- **Bombas lógicas**: «Estallan» en un instante de tiempo determinado, y muchas veces son creadas por el desarrollador del programa donde se encuentran como forma de protección.
- **Puertas traseras** (*backdoors*): Programas normales pero que además permiten el acceso al sistema desde el exterior o como administrador. Por ejemplo un programa de autenticación que con cierta combinación de usuario y contraseña proporciona acceso completo.[1]
- **Desbordamiento de *buffer***: Aprovechar fallos de programación para causar que un programa escriba (o lea) más información de la que cabe en el *buffer* que ha reservado, pudiendo llegar a ejecutar código malicioso, normalmente sobrescribiendo la dirección de retorno en la pila para que apunte a este.
- **Caballos de Troya**: Sustituyen una orden por otra con el mismo nombre.
- **Virus**: Se encuentran dentro de un programa y se replican en otros programas o ficheros, extendiéndose dentro del ordenador y copiándose a otros mediante sistemas de almacenamiento y redes.
- **Gusanos**: Como los virus, pero en vez de copiarse en otros ficheros se transmiten por red.
- ***Spyware***: Código que, sin que el usuario sea consciente de ello ni lo autorice, se ejecuta en segundo plano y recopila información del sistema y la actividad del usuario para enviarla a computadores externos.
- ***Rootkits***: Permiten acceso privilegiado continuo a un computador, normalmente mediante una puerta trasera, a la vez que ocultan activamente su presencia.

En 1975, Saltzer y Schroeder identificaron varios principios de diseño de cara a la seguridad:

1. El diseño debe ser público (muchos ojos ven más fallos que pocos).
2. El estado predefinido debe ser de «no acceso».
3. Se debe verificar la autorización en el momento de hacer la operación; no se debe verificar el permiso, determinar que está permitido y no hacer más comprobaciones.
4. Dar a los procesos el mínimo privilegio que les permita hacer su trabajo.

[1] Ver <`http://scienceblogs.com/goodmath/2007/04/15/strange-loops-dennis-ritchie-a/`>.

5. El mecanismo de protección debe ser simple, uniforme e integrado hasta las capas más bajas del sistema (la seguridad no es algo que se pueda añadir por encima).

6. El esquema debe ser psicológicamente aceptable para que los usuarios no pongan la contraseña en un *post-it* o similares.

3.1. Autenticación

La seguridad de un sistema requiere de:

- **Protección**. Distinguimos entre la **política**, que define qué se va a proteger, qué usuarios pueden acceder a qué recursos, etc., y que pueden estar determinadas en el diseño del sistema, formuladas por los administradores o definidas por los usuarios para proteger sus propios ficheros, y el **mecanismo**, que define cómo implementa esto el sistema, y es en lo que nos centraremos.

- **Entorno externo**. Por ejemplo, no exponer la consola del operador a personal no autorizado, ni permitir extraer el almacenamiento del sistema para transportarlo a otro sin protección.

La **autenticación de usuarios** es el problema de identificar a los usuarios cuando están conectados al sistema. El método más común es pedir un nombre o identificador (ID) y una contraseña para autenticar el ID. Este identificador determina los privilegios del usuario, es decir, lo que este puede o no puede hacer en el sistema. En general existe un usuario o grupo reducido con estado de **supervisor** o **superusuario**, que les permite llevar a cabo funciones protegidas especialmente. Algunos sistemas también disponen de cuentas anónimas cuyos usuarios tienen privilegios más restringidos. En muchos solo se permite acceder al sistema a los usuarios con un ID ya registrado.

3.1.1. Contraseñas en UNIX

La contraseña, junto con un **valor base** (*salt*) y el algoritmo de cifrado (función *hash*) a utilizar se pasan como parámetros a una función llamada `crypt`, que devuelve la contraseña cifrada. Esta se guarda en el fichero de contraseñas (antes `/etc/passwd` y actualmente `/etc/shadow`), con formato $*tipo*$*base*$*contraseñaCifrada*, donde *tipo* es 1 para MD5 (obsoleto), 5 para SHA-256 y 6 para SHA-512 y *base* es la base sin cifrar. El valor de la base está relacionado con el momento en que se asigna la contraseña a un usuario y sirve para impedir que contraseñas iguales se cifren igual.

3.1.2. Elección de contraseña

Las funciones *hash* usadas están diseñadas para evitar ataques por adivinación, incluso usando supercomputadores, pero algunos usuarios al elegir su contraseña eligen una fácilmente adivinable. Si se rechazan las contraseñas por longitud mínima, los usuarios pueden elegir contraseñas igualmente adivinables como su propio nombre, el de su calle, una palabra común del diccionario, etc., pero si se asignan contraseñas aleatorias, la mayoría de los usuarios no la recordarán o acabarán apuntándola en un lugar visible. Soluciones (no incompatibles):

- **Instrucción del usuario**: Explicar a los usuarios la importancia de elegir contraseñas seguras y cómo elegirlas, así como buenos hábitos como cambiarlas a menudo, no anotarlas en lugares visibles ni comunicarlas, etc. Una buena contraseña suele tener letras mayúsculas, minúsculas, números y signos de puntuación u otros caracteres alfanuméricos.
- **Inspección proactiva de contraseña**: El sistema usa algoritmos y heurísticas para determinar la seguridad de una contraseña y rechazarla si es fácilmente adivinable. Incluso el administrador del sistema puede comprobar la robustez de las contraseñas ejecutando programas de adivinación o *cracks*.

3.2. Protección

Un **derecho** es el permiso para realizar una determinada acción sobre algún objeto, y un **dominio** (**de protección**) es una función que a cada objeto le asigna un conjunto de derechos. Todo proceso se ejecuta en un dominio, y muchos sistemas proveen mecanismos que permiten que los procesos cambien de dominio. Algunas formas de representar los dominios son:

- **Matriz de protección** o **de acceso**: Con una fila por dominio, una columna por proceso y un conjunto de derechos en cada celda. Si los procesos pueden cambiar de dominio, los dominios se pueden identificar con objetos y añadir en estos el derecho de «entrar». Incluso la propia matriz se puede entender como un objeto que se puede modificar mediante una serie de operaciones. En la práctica esta matriz no se almacena.
- **Listas de control de acceso** o **ACLs** (*Access Control Lists*): A cada objeto se le asocia una lista con todos los dominios que pueden tener acceso a él y con qué derechos. Normalmente los dominios se asocian a usuarios y, si estos se pueden agrupar, las ACLs también se pueden aplicar a los grupos. Si una ACL contiene tanto una entrada para un usuario como entradas para grupos a los que este pertenece se debe establecer un criterio sobre qué derechos prevalecen.
- **Listas de posibilidades** o **de capacidades**: A cada dominio se le asocia una lista de entradas, llamadas **posibilidades** o **capacidades**, que representan un objeto (normalmente con su tipo y una referencia a este) y los derechos que se tienen sobre este (si un objeto no aparece, no se tiene ningún derecho). Normalmente los dominios se asocian a procesos, los cuales para acceder a un objeto indican la posición de la capacidad correspondiente. Como estas listas deben ser protegidas del manejo indebido, se suelen guardar dentro del sistema operativo permitiendo que los procesos las referencien por su número. En general las posibilidades tienen operaciones genéricas para copiarlas, eliminarlas, trasferirlas o generar una versión más restrictiva.

Las ACLs se corresponden directamente con las necesidades del usuario, que puede especificar los dominios que pueden acceder a un objeto cuando lo crea, pero al haber que comprobar cada acceso al objeto, son ineficientes. Las listas de posibilidades, por su parte, son mucho más eficientes, pero son más complejas al no corresponderse directamente con las necesidades de los usuarios.

3.2.1. Cancelación

La **cancelación** o **revocación** de derechos de acceso puede ser **inmediata** o **demorada**, **selectiva** o **general**, **parcial** o **total**, y **temporal** o **permanente**. Con las ACLs la cancelación es sencilla, pues basta eliminar de esta los derechos que serán cancelados.

Con las posibilidades haría falta determinar todas las existentes para un objeto y modificarlas, pero esto es ineficiente. Para cada objeto se podría llevar un registro de todas las posibilidades asociadas, pero este podría consumir mucha memoria y en algunos sistemas no es posible porque los procesos pueden pasarse posibilidades sin conocimiento del sistema.

Una técnica es hacer que cada posibilidad apunte hacia un objeto indirecto en vez de al objeto en sí, de forma que basta romper esta conexión para invalidar todas las posibilidades. Otra es asignar a cada objeto un gran número aleatorio, también presente en la posibilidad, y permitir la operación si ambos coinciden, de forma que basta cambiar el número del objeto para invalidar todas las posibilidades. Ninguna de estas dos técnicas permite la revocación selectiva.

3.2.2. Protección en UNIX

Unix combina ACLs restringidas con listas de posibilidades. Cada usuario lleva asociado un número, el **identificador de usuario** o **UID** (*User IDentifier*), que se asocia al nombre de usuario o *login* mediante el fichero `/etc/passwd`. Destaca el usuario con UID 0, normalmente llamado `root`, que tiene todos los privilegios. Es pues importante elegir una buena contraseña para esta cuenta y protegerla. Algunas distribuciones Linux no permiten iniciar sesión como `root`.

También hay grupos de usuarios, con un **identificador de grupo** o **GID** (*Group IDentifier*) asociado al nombre del grupo y la lista de usuarios pertenecientes a él a través del fichero `/etc/group`.

Pueden definirse varios usuarios o grupos con el mismo UID o GID, pero para el sistema operativo se trata del mismo usuario o grupo. Un dominio en UNIX se corresponde aproximadamente con un par (UID, GID).

Las ACLs restringidas se limitan a 3 conjuntos de usuarios: el propietario de un fichero, el grupo de usuarios al que pertenece y el resto de usuarios. Para cada uno existen 3 permisos: lectura (`r`), escritura (`w`) y ejecución (`x`). En directorios, el permiso `r` indica que se puede listar su contenido, `w` que se puede modificar (creando, borrando y renombrando ficheros y subdirectorios) y `x` que se puede hacer que este sea nuestro directorio actual y que es posible atravesarlo.

Para determinar lo que un proceso puede hacer sobre un fichero, si el usuario tiene UID 0, se considera que tiene los tres permisos; de lo contrario, si el UID del proceso coincide con el del fichero, se queda con los permisos de propietario; si esto no se cumple pero el GID del proceso coincide con el del fichero, se queda con los permisos de grupo, y en caso contrario usa los del resto de usuarios.

Estos permisos suelen mostrarse mediante una cadena de 9 letras con formato `rwxrwxrwx` para representar los permisos de los 3 conjuntos en orden, donde la letra correspondiente se sustituye por - si el conjunto de usuarios correspondiente no tiene el permiso. Esta cadena se almacena internamente mediante 9 bits, establecidos como 1 si el permiso se da o como 0 en caso contrario.

Algunos sistemas UNIX, como Linux, implementan también ACLs completas para los sistemas de ficheros que lo admiten.

Para cambiar de dominio, los ejecutable pueden poseer los bits SETUID y SETGID. Si SETUID está activo, al ejecutar el fichero, el proceso en que se ejecuta tendrá como UID efectivo (EUID) el del propietario del ejecutable, y lo mismo ocurre con SETGID para el grupo efectivo (EGID). Un programa puede conocer su usuario y grupo reales mediante las funciones `getuid` y `getgid`, así como su usuario y grupo efectivos mediante `geteuid` y `getegid`. Cuando un ejecutable posee el bit SETUID, la primera `x` en la cadena que representa los permisos cambia por una `s`.

Por ejemplo, el comando `passwd` permite cambiar la contraseña a un usuario, pero para ello necesita acceder y modificar `/etc/shadow`, que en general no puede ser leído ni escrito por nadie salvo `root`. Por tanto `passwd` tiene como propietario `root` y tiene activo el SETUID, y determina el usuario que lo ha invocado con `getuid`. Si el UID es 0, permite cambiar la contraseña de cualquier usuario; de lo contrario solo permite cambiar la del usuario que ejecutó que lo ejecutó.

Las ACLs sólo se comprueban al abrir el fichero. Si se tiene acceso a este, se crea una entrada en la **tabla de ficheros abiertos**, una lista de capacidades mantenida por el sistema operativo, y se devuelve un índice a esta llamado **descriptor de fichero**. Esto significa que si el ACL de un fichero cambia tras haber sido abierto, el proceso puede seguir accediendo a él aunque la nueva ACL no lo permitiera, incumpliendo el principio de diseño que dice que la autorización debe comprobarse en el momento de realizar la operación, pero a cambio permite un mejor rendimiento.

Capítulo 4

Sistemas de ficheros

Los sistemas de computación actuales necesitan un **almacenamiento secundario** que complemente al primario, para almacenar grandes cantidades de información que no caben en el primario; preservarla para que no desaparezca cuando termine el proceso o el sistema se apague, y permitir que la información sea fácilmente compartida y accedida. Para ello se usan discos y otros dispositivos que almacenan la información en un conjunto de bloques, o que se manejan como si fuera así, y sobre los cuales el sistema operativo crea una abstracción para facilitar su uso a usuarios y programadores.

Un **sistema de ficheros** es una parte del sistema operativo que organiza un sistema de almacenamiento secundario como una serie de ficheros o unidades de información que se agrupan en directorios, y también es el formato que adquiere el disco para ello. Desde el punto de vista del usuario es importante la forma de nombrar los ficheros, las operaciones permitidas, el tipo de protección que podemos aplicar, etc., mientras que desde el punto de vista del sistema operativo es importante la implementación.

4.1. Ficheros

En los sistemas modernos, un fichero es una secuencia de bytes con significado definido por el programa que accede al mismo. Cada fichero tiene un nombre que lo identifica, y que dependiendo del diseño del sistema de ficheros y del sistema operativo, distingue o no entre mayúsculas y minúsculas, tiene un cierto tamaño máximo, tiene o no una estructura (como «*nombre*.*extensión*»), etc.

Los ficheros suelen tener atributos, como por ejemplo:

- ACLs, contraseña, creador, propietario.
- Sólo lectura, ocultación, fichero de sistema, biblioteca, texto o binario, acceso aleatorio, temporal, bloqueo.
- Tiempos de creación, último acceso y última modificación.
- Tamaño actual y tamaño máximo.

En general hay operaciones para crear un fichero vacío, eliminar un fichero, abirlo, cerrar un fichero abierto, leer una cantidad de bytes, escribir una cantidad de bytes al final (añadir) o (si es posible) en otra posición, cambiar de posición dentro del fichero, obtener y establecer los atributos, cambiar de nombre o mover el fichero a otro directorio, y truncar el fichero a partir de una posición.

4.1.1. Ficheros regulares

Contienen información del usuario. Distinguimos entre ficheros **de texto**, codificados en UTF-8 o algún otro sistema y con líneas que terminan en CR, LF o LF/CR y que se pueden ver tal cual de forma legible, y los **binarios**, que tienen cierta estructura pero no son de texto.

Así, los ejecutables, binarios generalmente manejados por el sistema operativo, contienen una cabecera con un **número mágico** (número que identifica el formato de archivo), el tamaño de las secciones de código y datos, el punto de entrada (dirección donde debe empezar la ejecución) y otra información, seguida de las propias secciones. Por su parte, los archivos de biblioteca son ficheros binarios que almacenan código máquina de funciones (e información asociada).

Muchas veces el nombre del fichero contiene una **extensión** que identifica el tipo, que en los sistemas operativos modernos como Linux no es significativa pero sí lo es para algunos programas, como exploradores de ficheros para saber con qué programa abrirlo, y ayudan al usuario a identificar el tipo rápidamente.

Ejemplos de extensiones son `.bak` para copias de seguridad; `.hlp` para ficheros de ayuda; `.gif` o `.jpg` para imágenes GIF o JPEG; `.c` para código fuente en C; `.html` o `.tex` o `.txt` para documentos HTML, TEX o de texto plano; `.mp3` para sonido MP3; `.mpg` para vídeo MPEG; `.o` para ficheros objeto; `.pdf` o `.ps` para documentos PDF; `.zip` para archivos comprimidos ZIP; etc.

4.1.2. Ficheros especiales

No se corresponden con un fichero en el propio sistema de almacenamiento, sino con una forma de tratar otro tipo de datos como si fuera un fichero (en el almacenamiento lo que se guarda es algún tipo de referencia). En UNIX existen dos tipos:

- **De caracteres**: Representan dispositivos de E/S como terminales, impresoras y conexiones de red. No es posible cambiar de posición, escribir en una posición distinta del final o truncarlos.
- **De bloques**: Representan dispositivos que permiten acceso aleatorio, en general almacenamiento secundario.

4.1.3. Directorios

Son ficheros gestionados por el sistema operativo para registrar los ficheros y organizarlos en una **jerarquía de directorios**, de la que existen varios tipos:

- **Directorio único**, con los ficheros de todos los usuarios. Se usaban en los primeros microcomputadores. Puede haber problemas si distintos usuarios utilizan los mismos nombres de fichero.

- **Directorio por usuario**, aunque no es deseable si uno tiene muchos ficheros. Se debe decidir si un usuario puede o no acceder a los directorios de otros y cómo.
- **Árbol de directorios**. Un directorio puede contener a otros. Es lo que se usa hoy en día. Normalmente los directorios tienen una entrada especial `.` para referirse al propio directorio y `..` para referirse al directorio padre. Se accede a los ficheros (y directorios) mediante una **ruta de acceso**, que indica el nombre de los directorios por los que hay que pasar hasta llevar al fichero deseado (incluyendo el nombre del fichero), separados por un cierto caracter, que es `/` en UNIX y `\` en Windows. Dos tipos:
  - **Absoluta**: Parte desde la raíz y comienza por el caracter separador. Es única para un mismo fichero siempre que no se usen `.` y `...`
  - **Relativa**: Parte de un **directorio actual** o **de trabajo**, y no comienza por el caracter separador. Existe una cantidad numerable de rutas relativas para un mismo fichero desde un mismo directorio actual.

No podemos leer y escribir en ellos como con los otros tipos de ficheros, sino que en su lugar la apertura de un directorio sirve para recorrido y hay operaciones para obtener la siguiente entrada; ligar o enlazar un fichero al directorio, permitiendo que un mismo fichero aparezca en varios directorios, y desligar una entrada del directorio, lo que equivale a borrarla si esta es la única entrada del sistema de fichero en la que aparece.

4.2. Representación

La representación en disco debe ser tal que consiga una buena velocidad y aprovechamiento del espacio pero a la vez reduzca la fragmentación interna (partes de un bloque sin aprovechar) y externa (bloques libres que no se pueden ocupar).

4.2.1. Bloques lógicos

Por lo general los discos proporcionan una interfaz basada en una lista de **bloques físicos** o **sectores**, la unidad mínima de lectura y escritura, cuyo tamaño es constante y suele ser una potencia de 2 desde 512 B hasta 4 kB. Es común que el sistema de ficheros agrupe o incluso divida los sectores para formar **bloques lógicos** o unidades de asignación más grandes o pequeñas.

Si el bloque lógico es grande, un fichero tendrá menos bloques y el rendimiento en discos duros será mayor pero habrá más fragmentación interna, mientras que si el bloque es pequeño habrá poca fragmentación interna pero las listas de bloques ocuparán mucho.

4.2.2. Asignación de espacio

Todo fichero tiene asociado un conjunto de bloques donde guarda sus datos. La asignación de estos puede ser de varias formas:

- **Adyacente** o **contigua**. De fácil especificación, pues solo hay que recordar un número para localizar los bloques, y ofrece un excelente rendimiento en discos duros al suponer muy pocos movimientos del brazo del disco o ninguno. En general no es realizable, pues si

no se sabe el tamaño máximo de un fichero no se sabe cuánto espacio reservarle; si este es muy pequeño, cuando el fichero crezca hay que moverlo a un hueco mayor, algo costoso y no siempre posible. Sin embargo es útil en medios como CD-ROMs y DVD-ROMs, donde se sabe el tamaño de los ficheros de antemano.

- **Mediante lista ligada**. En cada bloque se guarda la dirección del siguiente. No hay fragmentación externa y la entrada de directorio correspondiente solo tiene que guardar la dirección del primer bloque, pero el acceso aleatorio es muy lento y los programas que esperan que los bloques tengan un tamaño potencia de 2 también se ralentizan.

- **Mediante lista ligada e índice**. Similar, pero la dirección del siguiente bloque no se guarda en el bloque sino en una tabla con una entrada por bloque que indica la dirección del siguiente en el fichero. Esto solo es eficiente si toda la tabla está en memoria, lo que es un problema si la tabla es grande. Se puede reducir el tamaño de la tabla usando bloques lógicos grandes, pero entonces se desperdicia espacio por fragmentación. Además, si la tabla se modifica y la modificación no se escribe en disco, una caída del sistema puede hacer que los ficheros desaparezcan o queden incompletos.
  Se puede usar un número de bloque inválido (como el 0) para indicar que un bloque está libre. El sistema FAT, usado por MS-DOS y uno de los implementados en Windows, usa este sistema.

- **Mediante nodos-i**. A cada fichero se le asigna un nodo-i (nodo índice) que contiene las direcciones en disco de los bloques del fichero, en orden. Normalmente se usan nodos-i pequeños capaces de almacenar unas pocas direcciones y se guardan varios nodos-i en un mismo bloque, de modo que una zona contigua del disco (o varias, por eficiencia), llamada **tabla de nodos-i**, se destinan a nodos-i.
  Para ficheros grandes se usa un **bloque simplemente indirecto** (BSI), que contiene direcciones de bloques de datos adicionales; un **bloque doblemente indirecto** (BDI), que contiene direcciones de más BSIs, y un **bloque triplemente indirecto** (BTI), que contiene direcciones de más BDIs, de forma que 3 de las direcciones del nodo-i se usan para estos bloques.
  Este es el esquema usado en la mayoría de sistemas de ficheros en sistemas UNIX, y permite acceder a cualquier bloque de un fichero abierto accediendo, como mucho, a 4 bloques del disco si el nodo-i está en memoria.

Además, debe haber una forma de saber que bloques de datos (así como qué números de nodos-i) están libres. Principalmente hay dos métodos:

- **Lista ligada de bloques**: Una lista con tantos números de bloques libres como pueda más un puntero al siguiente bloque. Los bloques de la lista son libres, por lo que podrán ser usados si es necesario.

- **Mapa de bits**: Contiene un bit para cada bloque, que indica si el bloque es libre u ocupado. Los bloques del mapa no son libres, pero los sistemas de ficheros actuales como Ext2, Ext3, Ext4 o NTFS lo usan porque permite buscar de forma sencilla grupos de bloques libres consecutivos.

4.2.3. Ficheros compartidos

Un fichero es compartido si puede aparecer en varios directorios con el mismo o distinto nombre, o en un mismo directorio con nombres distintos. La conexión entre un directorio y un fichero se llama **enlace**, y si existe, el sistema de ficheros pasa de ser un árbol a ser simplemente un grafo acíclico dirigido.

- **Enlace físico** o *hard link*: Los datos relativos a un fichero se guardan en una estructura de datos y las entradas de directorio correspondientes apuntan a esta. Es necesario que entre los datos del fichero exista un contador de enlaces, pues el nodo solo debe liberarse cuando se borra un fichero y no quedan más enlaces.
- **Enlace simbólico** o *soft link*: Se usa un tipo de fichero especial que contiene una ruta de acceso al fichero al que se enlaza. Cuando el fichero original se borra, el enlace simbólico queda invalidado salvo que se cree un fichero con la misma ruta.

UNIX implementa ambos. Los enlaces simbólicos tienen más coste, pero presentan menos problemas, con lo que UNIX permite enlaces simbólicos de directorios e incluso a otros sistemas de ficheros pero no permite hacer lo mismo con enlaces físicos. Un programa de copias de seguridad puede duplicar datos al copiar un fichero compartido si no usa alguna forma de detectar los enlaces para evitar esta duplicación.

4.2.4. Directorios

En MS-DOS, los directorios son ficheros salvo el directorio raíz que ocupa unos bloques fijos en disco, por lo que tiene un tamaño máximo preestablecido. Estos almacenan una lista desordenada de entradas o registros de 32 bytes, una por fichero, con los siguientes campos, en orden:

1. 8 bytes para el nombre del fichero y 3 para la extensión.
2. Un byte para atributos donde los bits se usan como banderas. Uno de ellos distingue a un directorio de un fichero normal.
3. Un campo de 10 bytes reservado para futuros usos.
4. 2 bytes para la hora de última modificación: 5 bits para la hora (0–23), 6 para el minuto (0–59) y 5 para el segundo, que debe ser par (0–58).
5. 2 bytes para la fecha de última modificación: 7 bits para el año (correspondiendo el valor 0 al 1980), 4 para el mes (1–12) y 5 para el día del mes (1–31).
6. 2 bytes para la dirección del primer bloque.
7. 4 bytes para el tamaño del fichero, en bytes.

En la mayoría de sistemas de ficheros en sistemas UNIX, los atributos se almacenan en los nodos-i y el directorio raíz no tiene tratamiento especial. En el de los primeros UNIX, las entradas de un directorio eran de 16 bytes, siendo 2 bytes para el número de nodo-i del fichero y otros 14 bytes para el nombre, que tenía pues un máximo de 14 caracteres. Posteriormente se crearon sistemas de ficheros para sistemas UNIX que soportaban nombres de hasta 255 caracteres, con entradas con los siguientes campos, en orden:

1. 4 bytes para el número de nodo-i.
2. 2 bytes para la longitud del registro.
3. 1 byte para la longitud del nombre.
4. 1 byte para el tipo de fichero (regular, de caracteres, de bloques o directorio).
5. Hasta 255 bytes para el nombre.

Una entrada no puede estar entre dos bloques, y el total de registros de un bloque debe cubrirlo enteramente; de ahí que se permita que la longitud de un registro sea mayor a la estrictamente necesaria. Siempre que se crea un directorio se añaden dos entradas, `.` y `..`. Para añadir una nueva entrada, se elige una existente, se disminuye su tamaño y se añade la nueva en el hueco, o se añade un nuevo bloque al directorio si esto no es posible. Para eliminarla se añade su espacio al de la entrada anterior, o si esto no es posible se marca el registro como libre utilizando un número de nodo-i inválido.

4.2.5. Estructura general

En general el bloque 0 del almacenamiento es el **bloque de arranque** y puede contener código de arranque del sistema operativo que se encuentra en la partición. El bloque 1 suele ser el **superbloque**, que contiene información crítica del sistema de ficheros como el total de bloques lógicos, tamaño de los mapas de bits, etc., por lo que algunos sistemas de ficheros tienen varias copias de este en otras zonas del disco. En otros, el superbloque es parte del bloque de arranque.

A partir de entonces hay mucha variación, pero en los primeros sistemas de ficheros en UNIX encontramos el mapa de bits de bloques, el de nodos-i, la tabla de nodos-i y, finalmente, el resto de bloques, para ficheros y bloques indirectos.

4.3. Cachés

Una **caché de disco**, **bloques** o ***buffers*** es una colección de bloques del almacenamiento secundario que se mantienen temporalmente en memoria principal por rendimiento. La política de sustitución podría ser LRU mediante colas, pero el LRU puro no es recomendable si se quiere mantener la consistencia del sistema de ficheros ante posibles fallos del sistema, pues por ejemplo un bloque que contiene nodos-i se usará mucho y si se modifica en caché y no se vuelve a escribir en disco el sistema de ficheros quedará inconsistente ante un fallo del sistema.

Por ello se suele usar un LRU modificado, en el que los bloques que por su tipo probablemente no se vuelvan a utilizar pronto pasan directamente al frente de la cola y los bloques esenciales para la consistencia se escriben directamente en disco sin esperar a ser expulsados. Tampoco es recomendable mantener mucho tiempo los bloques de datos modificados en caché por riesgo de perder los cambios. Así, en muchos UNIX los bloques de datos se escriben en disco como tarde a los 30 segundos de ser modificados, mientras que los de metadatos se escriben como tarde a los 5 segundos. También existe la orden `sync`, que al ejecutarse fuerza la escritura de todos los bloques modificados.

Por su parte, como las resoluciones de ruta (localizar un fichero a través de una ruta) son algo muy frecuente y una misma ruta puede resolverse muchas veces en poco tiempo, el sistema

operativo suele mantener en memoria información sobre las últimas resoluciones en otra caché, que en Linux se llama *dentry cache* o **caché de entradas de directorio**.

4.4. Particiones

Son porciones de bloques consecutivos de un disco, manejadas por el sistema operativo y tratadas de por sí como listas de bloques sobre las que se puede implementar un sistema de ficheros. Tradicionalmente la información de las particiones, como sus bloques de inicio y fin, se guarda en la **tabla de particiones** del bloque **MBR** (*Master Boot Record*, registro de arranque maestro), el primer bloque del disco, que además contiene una pequeña porción de código de arranque que la BIOS carga en memoria y ejecuta para, tras una serie de pasos, arrancar un sistema operativo. En general lo que hace este código es buscar una partición activa y cargar el código de arranque en esta.

Las particiones permiten tener distintos sistemas de ficheros en un mismo disco, o incluso sistemas operativos distintos. Por ejemplo, en Linux conviene tener al menos una partición para datos con Ext4 y una **partición de intercambio** usada para la memoria virtual. También permite dar a cada partición un uso distinto, de modo que si hay un error grave en el sistema de ficheros de una partición el resto siguen siendo usables.

Capítulo 5

Gestión de memoria

El **administrador de memoria** del sistema operativo se encarga de abstraer y administrar la jerarquía de memoria.

5.1. Sin memoria virtual

En la **multiprogramación con particiones fijas**, se asigna una zona de tamaño fijo al sistema operativo y el resto se dividen en **particiones** de igual o distinto tamaño. Podemos tener una cola por partición y colocar cada trabajo en la cola de la partición más pequeña en que quepa, pero si hay pocas particiones pequeñas sus colas serán largas porque la mayoría de procesos son pequeños y si hay muchas puede no quedar espacio para trabajos grandes. También se puede tener una cola para todas las particiones y, cuando queda una partición libre, darla a la primera tarea de la cola que quepa, lo que hace que las tareas pequeñas desperdicien espacio, o darla a la tarea más grande que quepa, lo que discrimina a las tareas pequeñas. Soluciones a este último problema son tener una partición pequeña o establecer un máximo de veces que se puede excluir una tarea.

Esto limita el **grado de multiprogramación**, el total de procesos que puede haber a la vez en memoria compartiendo CPU, además de producir fragmentación interna sobre todo en procesos pequeños y fragmentación externa si hay particiones libres con tamaño total suficiente para un proceso pero ninguna es lo bastante grande.

La **multiprogramación con particiones variables** soluciona el problema asignando a cada proceso una partición del tamaño exacto que necesita, con lo que las particiones se crean, desaparecen o intercambian según se necesita. **Políticas de asignación de huecos**:

- **Primero en ajustarse**: Se busca el primer hueco lo suficientemente grande desde el principio de la memoria, dividiendo el hueco en una parte para el proceso y un hueco libre (salvo si hay ajuste perfecto), y es rápido.
- **Siguiente en ajustarse**: Igual pero se empieza a buscar por donde se quedó la anterior búsqueda.
- **Mejor en ajustarse**: Busca el hueco más pequeño en el que quepa el proceso, es lenta y desperdicia más memoria por fragmentación externa.

- **Peor en ajustarse**: Toma el hueco libre más grande, y es lenta pero aprovecha bien la memoria.

La **compactación** consiste en mover procesos en memoria para juntar los huecos y reducir la fragmentación externa. No se suele usar porque consume mucho tiempo, proporcional a la cantidad de información a mover, si bien ordenadores como el CDC CYBERS tenían hardware especial para esto. Para la **administración de memoria libre**:

- **Mapas de bits**: Se divide la memoria en unidades de asignación del mismo tamaño, y a cada una le corresponde un bit que indica si está libre o no. Si la unidad de asignación es pequeña, el mapa de bits será grande y las búsquedas lentas, pero si es grande habrá fragmentación interna.
- **Listas ligadas**: Lista de segmentos de memoria, que pueden ser procesos o huecos. Cuando un proceso termina, debe fusionar el hueco con los adyacentes, lo que es sencillo si la lista está ordenada por direcciones. Si hay una lista para procesos y otra para huecos, las operaciones serán más rápidas por no tener que ordenar la lista de procesos. Si la de huecos se ordena por tamaño decreciente, la política de peor ajuste es $O(1)$, pero la fusión de huecos es cara. En general se accede a la información de memoria asignada a un proceso desde su bloque de control.

La asignación de hueco en la zona de intercambio del disco puede hacerse cuando el proceso deba intercambiarse o cuando se cree. Los algoritmos de asignación y registro de estos huecos son los mismos que en memoria principal, salvo que el tamaño de un hueco en disco es múltiplo del tamaño de bloque.

La **reubicación** o **relocalización** consiste en que, cuando un programa se ejecuta, puede ir a cualquier partición, por lo que el programa debe usar **código relocalizable**, que pueda ejecutarse correctamente independientemente de dónde se sitúe.

Para la protección de memoria puede haber un **registro base** con la dirección de memoria donde comienza la partición de un proceso y un **registro límite** con su tamaño, de modo que el proceso puede generar cualquier **dirección lógica** en un **espacio de direcciones lógicas** de 0 a su tamaño menos 1 que se traducen a **direcciones físicas**, y si genera una dirección mayor se produce una excepción.

5.2. Paginación

El esquema de **memoria virtual** permite ejecutar procesos con tamaño total mayor al de la memoria física, para lo que guarda en esta sólo las partes del proceso que se están usando y el resto lo guarda en disco. Cada proceso posee un espacio de direcciones virtuales, de tamaño normalmente limitado por CPU, que van a la **unidad de administración de memoria** (**MMU**) normalmente integrada en el chip de la CPU para que las traduzca a direcciones físicas.

El espacio de direcciones virtuales se divide en **páginas** y el de memoria física en **marcos de página**, del mismo tamaño, normalmente potencias de 2 entre 512 B y 8 KiB, aunque pueden ser bastante más grandes, y las transferencias entre memoria y disco son siempre en unidades de página.

Una **tabla de páginas** tiene tantas entradas como páginas y en cada una indica el marco en que se almacena junto a algunos bits:

- **Bit presente/ausente**: Indica si la página está asociada o no. Si se accede a una página no asociada, la MMU genera una excepción llamada **fallo de página** a tratar por el sistema operativo. Este selecciona un marco, escribe su contenido en disco (si estaba ocupado y se había escrito algo), lo marca como ausente en su tabla de páginas y asigna el marco a la página que provocó el fallo, que lee del disco si es necesario.
- **Bits de protección**: Tipo de acceso permitido (lectura, escritura, ejecución...). Las páginas con código suelen tener permisos de lectura y ejecución pero no de escritura, mientras que las de datos suelen tener de lectura y escritura.
- **Modificado**: Indica si se ha modificado o no el contenido del marco.
- **Referenciado**: Se establece al hacer referencia a una página, y se usa en algoritmos de reemplazo.
- **Caché**: En las máquinas con E/S mapeada a memoria, evita que el contenido de la página se almacene en caché.
- **Visible en modo núcleo**: Evita que se pueda acceder a la página en modo usuario.

Las direcciones virtuales se dividen en un **número de página**, usado como índice en la tabla de páginas, y un **ajuste** o **desplazamiento** dentro de ella.

La traducción de páginas debe ser rápida, por lo que se usa un **TLB** (*Translation Look-aside Buffer*), una caché totalmente asociativa generalmente en la MMU que contiene entradas de la tabla de páginas. El hardware verifica si el número de página está en el TLB comparando todas las entradas en paralelo (por lo que número de entradas suele ser muy pequeño). Si coincide alguna, se toma la entrada del TLB (incluyendo bits de protección, etc.). Si no está, se lee la entrada de memoria principal y, bien se elimina una entrada del TLB escribiendo sus bits de modificado y uso en la principal y se reemplaza por la nueva, o se produce un fallo de página.

Al cambiar de proceso se puede invalidar el contenido del TLB, borrando todos los bits de validez mediante una instrucción especial, o añadir un nuevo campo a cada entrada del TLB con el identificador del proceso y un registro con el del proceso activo, requiriendo hardware adicional pero ahorrando tiempo en los cambios de proceso y dando lugar a encontrar entradas en el TLB del propio proceso tras volver a tomar la CPU.

Como una tabla de páginas global puede ser muy grande, en la práctica se divide el número de página en varias partes y se usan tablas de páginas de varios niveles, de modo que la de primer nivel se indexa según la primera parte del número de página e indica la dirección de las tablas de segundo nivel, etc., y las tablas que no son necesarias no se tienen. Los Intel Pentium Pro tienen 3 niveles, los procesadores de Intel y AMD de 64 bits tienen 4, y Linux soporta hasta 5 desde la versión 4.14, permitiendo gestionar hasta 128 PiB de memoria virtual y hasta 4 PiB de RAM.

Una **tabla de páginas invertida** contiene una entrada por cada marco de página con el número de la página que almacena, el PID, una serie de bits como en una entrada de tabla de páginas normal y un puntero para encadenamiento en una tabla de dispersión abierta usada para acelerar las traducciones, que usa como clave el PID y el número de página y devuelve el número de una entrada de la tabla.

El **mapa de memoria** de un proceso es la estructura de su espacio de direcciones lógicas, en zonas contiguas de memoria virtual llamadas **regiones** como las de código, datos con valor inicial, datos sin valor inicial y pila de cada hilo, con características como:

- **Soporte**: De dónde se obtienen los datos que contiene la región; normalmente un fichero o parte de este, como un ejecutable en el caso del código. También hay regiones sin soporte como la pila, cuyas páginas, si se modifican y son expulsadas, se guardan en la zona de intercambio del disco.
- **Tipo de compartición**: Si las modificaciones que haga un proceso son visibles por otros o no.
- **Protección**: Permisos de lectura, escritura y ejecución.
- **Tamaño fijo** o **variable**: Si es variable, se indica si crece hacia direcciones mayores, como la memoria montón o *heap* usada para la memoria dinámica, o menores, como la pila. En UNIX, la memoria montón no tiene soporte, está inicialmente a 0 y crece conforme el proceso necesita memoria mediante las llamadas `brk` y `sbrk`.

El sistema operativo suele estar mapeado en todos los procesos, con el bit de visible en modo núcleo activado, por eficiencia en las transferencias de datos entre usuario y núcleo[1].

En Linux, al hacer la llamada al sistema `fork`, se usa **copia en escritura** (*copy on write*, COW), consistente en desactivar el permiso de escritura de las páginas y dar al proceso hijo una copia de la tabla de páginas del padre, de modo que si uno de los dos intenta escribir en una región en la que tendría permiso, el sistema operativo hace una copia de la página que provocó la violación, la asigna a la tabla de páginas del proceso que hace la escritura y activa el permiso de escritura en ambas páginas[2].

Algoritmos de reemplazo de páginas:

- **Algoritmo óptimo**: Se elimina la página para la que pasará más tiempo antes de ser utilizada. Modelo teórico imposible de implementar.
- **NRU** (no usada recientemente): Se limpia el bit de referenciado periódicamente en todas las páginas, y al reemplazar se prefieren páginas con este bit desactivado y, dentro de esto, con el de modificado desactivado.
- **FIFO** (primera en enterar, primera en salir): Se elimina la primera página de una lista y se añade la nueva al final. El algoritmo es de bajo coste, pero no tiene en cuenta ningún dato adicional y produce demasiados fallos.
- **Algoritmo de la segunda oportunidad**: Como el FIFO pero, si el bit de referenciado de la primera página está activo, el bit se limpia, la página pasa al final y se comprueba la siguiente.
- **Algoritmo del reloj**: Equivalente al anterior pero con una lista circular donde una «manecilla» (un puntero) apunta a la página más antigua. Si el bit de referenciado está inactivo, se expulsa y se reemplaza en la lista la página apuntada y se avanza la manecilla a la siguiente, y de lo contrario se inactiva el bit, se pasa a la siguiente página y se repite el proceso.

[1] En procesadores como Intel, esto ya no se hace debido a Meltdown.

[2] Linux usa conteo de referencias para que, si hay varios `fork`, no se active la escritura en la página original hasta que el resto de procesos tenga una copia de la página. Ver `https://stackoverflow.com/questions/13813636/how-does-copy-on-write-in-fork-handle-multiple-fork` para más detalles.

- **LRU** (usada menos recientemente): Se elimina la página no usada desde hace más tiempo. Se podría usar una lista ligada de todas las páginas de memoria, pero tendría que actualizarse en cada acceso a memoria. También se podría tener un contador hardware que se incrementa en cada referencia a memoria y un campo en las entradas de la tabla de páginas y la TLB donde copiarlo. Otra forma es usar una matriz $n \times n$, siendo n el total de páginas, inicializada a 0, con lo que al referencial el marco k, el hardware activa todos los bits de la fila k y después desactiva todos los de la columna, con lo que la fila cuyo valor en binario sea mínimo es la del marco que se usó hace más tiempo, pero esto es inviable por el tamaño de la matriz.

- **Algoritmo de maduración**: Aproximación software de LRU. En cada interrupción de reloj, se desplaza un contador en cada entrada de la tabla de páginas un bit a la derecha y se añade a la izquierda el bit de referenciado.

Se debe decidir también si la página a reemplazar se busca sólo en las del proceso (**reemplazo local**) o en todas las de memoria (**reemplazo global**), y si el número de marcos asignados a un proceso varía (**asignación dinámica**) o no (**asignación fija** o **estática**). No es posible el reemplazo global con asignación fija, pero el reemplazo local con asignación dinámica puede evitar que un proceso que empieza a producir fallos de página le quite páginas a otros siendo a la vez más flexible que una asignación fija. Llamamos **algoritmo de frecuencia de fallos de página** al consistente en asignar más marcos a los procesos que producen muchos fallos de página y quitarlos a los que producen pocos. Otros aspectos son el número mínimo de marcos por proceso, que depende entre otros del máximo de páginas que puede usar una sola instrucción[3], y el reparto de marcos entre distintos procesos (equitativo, proporcional al tamaño, etc.).

Páginas pequeñas producen menos fragmentación interna, pero páginas grandes aceleran el reemplazo porque las transferencias entre memoria y disco son normalmente de una página y hay poca diferencia entre el tiempo de transferencia de una página pequeña y el de una grande. Los tamaños más frecuentes son de 4 y 8 KiB.

La **hiperpaginación** ocurre cuando un proceso emplea más tiempo paginando (esperando a que se resuelvan sus fallos de página) que ejecutando código porque necesita muchos más marcos de los que tiene. En tal caso, aparte de añadir más memoria principal, podemos suspender temporalmente algunos procesos para liberar memoria y reanudarlos cuando la tasa de fallos de página decaiga.

Políticas de lectura y escritura de páginas:

- **Paginación por demanda**: Solo se lee la página que produce el fallo.

- **Prepaginación** o **paginación anticipada**: Se leen varias páginas más, normalmente las que se encuentran después en el espacio de direcciones virtuales.

- **Escritura por demanda**: Una página se escribe en disco cuando se expulsa, aumentando el tiempo de resolución de fallos de página.

- **Escritura anticipada**: Un hilo del núcleo, el **demonio de paginación**, cada cierto tiempo escribe en disco las páginas modificadas, aumentando el rendimiento en discos

[3] Si, por ejemplo, una instrucción debe acceder a 5 páginas incluyendo la de la propia instrucción pero sólo hay 4 marcos, uno de los accesos provocará necesariamente un fallo de página que reemplazará otra de las 4 páginas necesarias y reiniciará la instrucción, que nunca llegará a terminar.

duros al escribir varias páginas a la vez y acelerando la resolución de fallos de página. Muchas escrituras pueden ser inútiles si la páginas se modifican poco después, por lo que el demonio puede buscar páginas que, según el algoritmo de reemplazo, podrían expulsarse próximamente y escribirlas en disco si han sido modificadas. También puede liberar más páginas para tener páginas libres (**caché de páginas**) y resolver rápidamente cualquier futuro fallo de página, de forma que si una página liberada se necesita poco después su contenido siga en memoria.

5.3. Segmentación

El espacio de direcciones lógicas de un proceso se compone de **segmentos**, zonas contiguas dadas por una **base** (dirección de comienzo) y un **límite** (tamaño), y las direcciones especifican el número de segmento y el desplazamiento dentro de él, de 0 al tamaño menos 1. Esto facilita la protección y compartición de información, y se puede acelerar la traducción con un TLB, pero sufre fragmentación externa, los segmentos no pueden superar el tamaño de la memoria física y los intercambios con el disco se hacen siempre moviendo segmentos enteros.

En la **segmentación paginada** estos segmentos se paginan, por lo que no hay fragmentación externa ni hay que buscar un hueco adecuado para cada segmento. El TLB es direccionable por el número de segmento a usar y el número de la página dentro del segmento.

Capítulo 6

Gestión de E/S

Distinguimos:

- **Dispositivos de bloques**: Dispositivos de almacenamiento en general. Almacenan información en bloques de tamaño fijo que podemos leer y escribir de forma independiente de los demás.
- **Dispositivos de caracteres**: Terminales, impresoras, interfaces de red, ratones, etc. Envían o reciben un flujo de caracteres.
- Otros, como los relojes, que sólo producen interrupciones periódicas, o las tarjetas gráficas mapeadas a memoria en las que la memoria de vídeo es parte del espacio de direcciones.

6.1. El software de E/S

Objetivos:

- Crear conceptos generales que no dependan de las características específicas de cada dispositivo. Por ejemplo, podemos acceder a un dispositivo de bloques sin importar si es un CD-ROM en el que el lector tiene un motor que se debe arrancar y parar o un disco duro que está girando continuamente.
- Dar a los dispositivos nombres uniformes sin importar sus características. En UNIX esto se hace mediante ficheros especiales.
- Manejar los errores lo más cerca posible del hardware. Si el controlador descubre un error, debe intentar corregirlo. Si no puede, debe notificarlo para que lo intente el manejador de dispositivo. Si tampoco puede, debe notificarlo al software independiente de E/S.
- Convertir las transferencias asíncronas, en que la CPU inicia una transferencia y hace otras cosas hasta recibir una interrupción indicando que esta ha terminado, en síncronas, más fáciles de programar, en que el programa se bloquea hasta que los datos están disponibles[1].

[1] Muchos programas usan hilos para volverlas a convertir en asíncronas.

- Permitir la compartición de recursos para lo que esto es posible (como discos) y asignar los de uso exclusivo (como impresoras).

6.1.1. Manejadores de interrupciones

Cuando un proceso inicia una operación de E/S, se bloquea hasta que esta termine mediante una llamada al sistema específica o, en el caso de UNIX, de forma automática, cediendo la CPU a otro proceso. Cuando la operación termina, el controlador envía una interrupción que atiende el manejador de interrupciones del sistema operativo, el cual identifica la interrupción y avisa al manejador de dispositivo correspondiente para que elimine el bloqueo del proceso. En UNIX, el proceso continuará en su parte de núcleo para terminar la operación.

6.1.2. Manejadores de dispositivo o *drivers*

Contienen el código que depende del funcionamiento concreto de los dispositivos, con lo que cada uno controla solo un tipo de dispositivo o varios similares. En los sistemas monolíticos, para acceder a los registros de las controladoras, necesitan ejecutarse en modo núcleo, por lo que se les considera parte del núcleo.

Un manejador recibe solicitudes abstractas que le hace el software independiente de dispositivo y verifica la ejecución de estas solicitudes. Tras enviar la orden (u órdenes) al dispositivo, el controlador puede tener que bloquearse hasta que ocurra una interrupción, o puede que no sea necesario porque la operación no conlleve retraso. Tras terminar la operación, el manejador debe verificar los errores corrigiendo los que pueda e informar al software independiente de dispositivo, y seleccionar e iniciar alguna solicitud pendiente si tiene o, según el sistema operativo, bloquearse en espera de una solicitud.

El sistema operativo define una interfaz, una serie de funciones genéricas, a la que se deben adaptar todos los manejadores. Implementar un manejador supone conocer el funcionamiento del sistema operativo y el del dispositivo. Además muchos manejadores deben ser **reentrantes**: capaces de ser interrumpidos a mitad de atender una petición de un proceso y recibir otra petición sin haber terminado de atender la primera.

6.1.3. Software de E/S independiente de dispositivo

Sus funciones suelen ser:

- Proporcionar una interfaz uniforme al usuario, seleccionando el manejador adecuado en cada caso.
- Dar nombre a los dispositivos. En Linux cada uno tiene un **número mayor**, que identifica al manejador, y un **número menor**, que usa el manejador para identificar el dispositivo concreto.
- Evitar accesos no autorizados.
- Proporcionar un tamaño de bloque independiente del dispositivo, agrupando o dividiendo sectores, si bien esto también lo puede hacer el manejador de dispositivo.
- Proporcionar *buffers* para:

    - Dispositivos de entrada como el teclado que no son capaces de almacenar información o la envían antes de que ningún proceso la solicite, en cuyo caso el sistema operativo debe guardar la información enviada (en este caso, códigos de teclas).
    - Dispositivos de salida como impresoras que no son capaces de procesar información a la velocidad a la que un proceso puede enviarla, por lo que se debe ir enviando poco a poco.
    - Dispositivos de bloques, en los que conviene guardar temporalmente en memoria ciertos bloques para acelerar su funcionamiento y permitir a los procesos leer o escribir una cantidad arbitraria de bytes.
- Asignar y liberar dispositivos de uso exclusivo, aceptando o rechazando solicitudes según disponibilidad.
- Informar de los errores cuando el manejador de dispositivo no los puede solucionar.

Los sistemas de fichero son independientes de dispositivo y trabajan sobre dispositivos de bloques abstractos. También hay software de E/S independiente de dispositivo en el espacio de usuario:

- Bibliotecas de E/S, que llevan a cabo las llamadas al sistema.
- Sistema de *spooling* para dispositivos de uso exclusivo. Por ejemplo, el **demonio** de impresión comprueba periódicamente si hay ficheros en el **directorio de *spooling*** y, si es así, los imprime, de modo que el resto de procesos pueden imprimir un fichero situándolo (en el formato apropiado) en el directorio de *spooling*. Esto impide que un proceso mantenga abierto un dispositivo de uso exclusivo sin hacer nada con él mientras el resto de procesos esperan, y permite eliminar un trabajo antes de que sea procesado, cambiar el orden en que se deben tratar los trabajos, etc.

6.2. Discos

Los discos duros están formados por una serie de discos o platos magnéticos conectados a un eje común que gira a gran velocidad (por ejemplo, 5400 RPM) y una serie de cabezas de lectura/escritura, una por cada superficie de disco, que pueden leer o escribir sobre la porción de la superficie de los platos que está justo debajo o encima de estas, y que están todas conectadas mediante brazos a un mismo soporte que puede girar en un cierto arco haciendo que las cabezas se muevan hacia dentro o fuera sobre la superficie de los discos.

Llamamos **cilindro** al conjunto de posiciones del disco accesibles en una cierta posición de las cabezas, **pista** (**circular**) al subconjunto de estas posiciones sobre una misma superficie de disco y **sector** a la unidad mínima de lectura y escritura soportada por la controladora, con tamaños típicos de entre 512 B y 4 KiB (siempre potencias de 2), y que corresponde a una porción de pista.

Cada sector se identifica por la terna de los números de cilindro, cabeza y sector dentro de la pista, numerados desde 0, donde los cilindros más externos (más alejados del eje) tienen menor número. El **tiempo de servicio** o **de acceso** a un sector es la suma de:

1. **Tiempo de búsqueda**, para colocar las cabezas en el cilindro adecuado. El más largo.

2. Tiempo para activar la cabeza adecuada, despreciable.
3. **Tiempo de latencia**, hasta que los sectores a leer o escribir pasen junto a la cabeza.
4. **Tiempo de transmisión**, de leer o escribir los sectores en sí según van pasando por la cabeza.

Hoy en día las pistas exteriores de los discos son más largas que las interiores y por tanto contienen más sectores, dividiéndose el disco en zonas (anillos circulares) en las que cada pista contiene el mismo número de sectores. Para que el manejador de disco no tenga que conocer su estructura interna, la controladora de disco expone una interfaz a modo de lista contigua de bloques.

Debemos planificar el orden en que se atienden las solicitudes de la cola de solicitudes a disco pendientes para reducir el movimiento de las cabezas. Como el manejador no conoce la geometría del disco, en vez de la distancia entre cilindros se tiene en cuenta la distancia entre direcciones de bloque.

- **FCFS** (*First Come, First Served*): Por orden de llegada, da tiempos de servicio grandes.
- **SSF** (*Shortest Seek First*): Atiende la solicitud del bloque más cercano al actual. No es equitativo, pues atiende enseguida solicitudes de cilindros próximos al actual y no otras solicitudes que llegaron antes, y tampoco es óptimo.
- **SCAN** o **algoritmo del ascensor**: En un sentido, atiende la solicitud del bloque con el mayor número que sea menor al del actual. En el otro, la del menor número que sea mayor al del actual. Cuando no quedan peticiones en un sentido, cambia al otro. Dada cualquier colección de solicitudes, el máximo de movimientos es el doble del número de cilindros, evitando la localidad.
- **C-SCAN** o **SCAN circular**: En SCAN, cuando las cabezas llegan a un extremo e inviertan el sentido, habrá pocas solicitudes en la zona próxima, mientras que en la otra habrá muchas y llevarán más tiempo. El C-SCAN modifica este algoritmo de modo que, si el sentido es de mayor a menor bloque y no hay ninguna solicitud de bloque de número menor, se toma el bloque de mayor número y se sigue en el mismo sentido, y lo mismo ocurre al revés si el sentido es el otro.

Si la tecnología cambia, pueden ser necesarios nuevos algoritmos. Los discos duros actuales suelen tener una pequeña caché en la controladora para leer por adelantado pistas enteras o partes de ellas, por lo que se prefiere atender una solicitud que acaba de llegar pero está próxima a la última servida antes de otra que lleva más esperando, aunque si lleva mucho tiempo esperando se atiende de inmediato.

La implementación del sistema de ficheros puede tener gran influencia sobre las solicitudes, pues no es lo mismo si estos están en un área contigua en disco (requiriendo pocos movimientos de las cabezas) o dispersos (muchos movimientos), en cuyo caso se podría defragmentar el disco colocando los bloques de cada fichero contiguos o próximos. También podemos colocar los bloques de datos de directorios próximos a los bloques de nodos-i de sus ficheros, como hacen ext2/3/4, XFS o JFS.

Un **disco RAM** es un dispositivo de bloques en que los bloques se almacenan en memoria principal, permitiendo acceso instantáneo, con lo que cuando el manejador recibe un mensaje para lectura o escritura de un bloque, basta que calcule el lugar de la memoria del disco RAM donde se encuentra.

6.3. Relojes

Los relojes más sencillos están sujetos a la línea de corriente alterna y provocan una interrupción por cada ciclo de voltaje (a 50 o 60 Hz). Los **relojes programables** constan de un oscilador de cristal de cuarzo que genera una señal periódica de 5 a 100 MHz o más de muy alta precisión; un contador que a partir de esta señal cuenta de forma decreciente hasta 0 y, cuando llega, provoca una interrupción, y un registro de carga para actualizar el contador.

En **modo de disparo único**, el reloj copia al iniciarse el valor del registro de carga en el contador y, cuando este llega a 0, se detiene hasta ser iniciado de nuevo por el software, mientras que en **modo de onda cuadrada**, cuando llega a 0, el registro de carga se copia automáticamente al contador, generando interrupciones periódicas llamadas **marcas** o **tics de reloj**. Los chips suelen contener 2 o 3 relojes programables independientes, así como otras opciones como contar en forma ascendente o desactivar las interrupciones.

El **manejador del reloj** se encarga de:

- Controlar la hora del día, incrementando un contador en cada marca del reloj registrando el tiempo transcurrido desde, por ejemplo, el inicio del año 1970 como se hace en UNIX, momento conocido como *The Epoch* (La Época[2]). El contador se puede implementar, por ejemplo, como:
  - Contador de marcas. Si es de 64 bits a 60 Hz, puede registrar más de 9800 m.a..
  - Contador de segundos y otro de marcas hasta llegar al segundo. Si son de 32 bits desde *The Epoch*, el desbordamiento ocurre en 2106 si se guarda como entero sin signo o 2038 si se guarda con signo.
  - Instante de arranque en segundos y contador de marcas desde el arranque. Si el contador es de 32 bits a 60 Hz, habrá que reiniciar tras algo más de 2 años y 3 meses.

  Para obtener la fecha al arrancar, el sistema operativo puede pedirla al usuario o tomarla del **reloj de tiempo real** de la placa base, un pequeño circuito integrado con una pequeña batería que mantiene la hora incluso con la máquina apagada.
- Controlar el tiempo de ejecución de los procesos, con un contador con el valor del *quantum* del proceso en marcas del reloj para llamar al planificador cuando el contador llegue a 0.
- Contabilizar el uso de CPU. Una forma es iniciar un segundo reloj cada vez que se asigne la CPU a un proceso y leer el valor del contador cuando el proceso deje la CPU para añadir el tiempo a un campo en la entrada de la tabla de procesos. El reloj debe parar cuando la CPU atienda una interrupción. Otra forma más sencilla es incrementar un campo en la tabla de procesos por cada marca de reloj, pero esta es menos exacta porque, por ejemplo, un proceso puede pasar a usar la CPU justo tras producirse una marca y dejarla justo antes de la siguiente y esto no contaría.
- **Alarmas**. Un proceso en UNIX puede ejecutar la llamada `alarm` para pedir al sistema operativo que le avise tras un cierto tiempo enviándole una señal `SIGALRM`. Si el manejador de reloj tiene relojes físicos suficientes, se puede usar uno para cada solicitud, pero como esto es improbable se simular relojes virtuales.

[2] ¿A que al traducirlo pierde toda la gracia?

Una forma es tener una tabla con el tiempo de señalización de cada alarma y una variable que indica el tiempo en marcas hasta la próxima señal, de modo que al actualizar la hora del día se decrementa esta variable y, si llega a 0, se envía la señal al proceso, se busca la siguiente alarma y se actualiza la variable. Otra forma es tener una lista ligada de alarmas según tiempo de ocurrencias en que cada elemento indica el número de marcas de reloj a esperar tras la siguiente señal.

- **Cronómetros guardianes**. Son alarmas establecidas por el propio sistema operativo. Por ejemplo, para acceder a un DVD hay que activar el motor del lector y esperar a la velocidad adecuada, por lo que es buena idea dejar el motor encendido por un tiempo por si hay más operaciones de E/S y detenerlo si pasa un tiempo sin que haya. Funcionan como alarmas pero, en vez de provocar una señal, llaman a un procedimiento proporcionado por quien hizo la llamada dado que en el núcleo no hay señales.

Apéndice A

El entorno de comandos de GNU

En UNIX/Linux existen tres grandes familias de intérpretes de órdenes o *shells*: `sh`, `csh` y `ksh`. Nos centraremos en `bash`, el Bourne-Again Shell, un *shell* libre de la familia de `sh`. Podemos acceder a este:

- Directamente, si el sistema arranca en modo texto.
- El sistema suele tener 12 terminales virtuales. El modo gráfico suele usar las dos primeras, por lo que podemos acceder a modo texto pulsando `C-M-F`i con $i = 3, \ldots, 12$ y volver al modo gráfico con `C-M-F1` o `C-M-F2` dependiendo del sistema.
- También, desde el modo gráfico, podemos usar un programa de simulación del terminal, que permite ejecutar órdenes desde el entorno de ventanas. Destacan `konsole` de KDE y `gnome-terminal` de GNOME.

`bash` es programable a partir de ficheros de texto con listas de órdenes, llamados **guiones *shell*** (*shell scripts*) o ***scripts***.

En general, las órdenes que aceptan varias opciones a la vez de una sola letra permiten escribir, por ejemplo, `-abcd` en vez de `-a -b -c -d`, y esto también vale si la última orden requiere un argumento. Todos los procesos, cuando finalizan, devuelven un **código de salida** (*exit code*) o **estado de salida** (*exit status*) al proceso padre, que generalmente es 0 cuando la ejecución es correcta y un valor entre 1 y 255 cuando hay un error.

Ejecutar *`comando`* `--help` en general muestra una breve ayuda sobre una orden externa. Otras fuentes son la ayuda de `bash`, la de `man`, la documentación de las distribuciones, sitios como FAQs, foros, etc., y proyectos de documentación libre como TLDP.[1]

Las órdenes se ejecutan siempre desde el **directorio actual** en el que nos encontremos. Cada usuario tiene un **directorio personal**, que suele ser `/home/`*`usuario`*, siendo *`usuario`* el nombre del usuario.

Los ficheros y directorios tienen una serie de permisos para el propietario (o usuario), el grupo y el resto de usuarios. Para cada uno de estos se definen los siguientes permisos:

[1] De aquí en adelante buscad en todos estos sitios antes de preguntarme a mí.

Permiso	**Fichero**	**Directorio**
Lectura (`r`)	Leer el contenido.	Obtener la lista de entradas. No es necesario para acceder a ellas si conocemos su nombre.
Escritura (`w`)	Modificar el contenido.	Crear, borrar o modificar entradas.
Ejecución (`x`)	Ejecutar el fichero.	Acceder al nodo-i de las entradas para acceder a subdirectorios o ficheros u obtener los metadatos de estos.

Además de los permisos especiales:

- `t` (*sticky bit*): En un directorio, un usuario con permiso de lectura puede crear ficheros, pero sólo puede borrar los que le pertenecen.
- `u+s` (suid): En un ejecutable, el UID cambia al del propietario al ejecutarlo.
- `g+s` (sgid): En un ejecutable, el GID cambia al del grupo del fichero al ejecutarlo. En un directorio, los ficheros creados dentro pertenecen al grupo del directorio, y no (necesariamente) al grupo activo de quien lo crea.

Los permisos se representan en notación estándar como la cadena `rwxrwxrwx`, con 3 grupos para usuario, grupo y otros, respectivamente, donde se sustituyen los caracteres de los permisos que no están por `-`. La `x` de usuario o grupo, respectivamente, cambia a una `s` si está activo el suid o el sgid. La `x` de otros cambia a una `t` si está activo el *sticky bit*. También se puede representar en octal, considerando en esta cadena que un permiso que se da es un 1 y uno que no es un 0, y convirtiendo el número binario resultante a octal.

A continuación vemos versiones reducidas y traducidas de las páginas `man` (del manual) de los principales comandos, que definen un subconjunto de la funcionalidad de estos en la implementación de GNU.

A.1. `bash`

Si bien veremos que el *shell* permite crear variables, estas son variables locales, que son heredadas por los hijos del *shell* (si bien al modificarlas estos no se actualizan en el proceso padre ni al revés) pero desaparecen en cuanto estos llaman a `exec`. No obstante, cada proceso en UNIX incluye un **entorno** con una serie de **variables de entorno**, que se heredan de padres a hijos en `fork`, no se borran al usar `exec` y los procesos pueden crear, modificar o borrar. Por ejemplo, `LOGNAME` contiene el nombre del usuario que ha iniciado sesión.

Los procesos disponen por defecto de tres descriptores de fichero:

- **Entrada estándar** (0): Para recibir datos de entrada. Normalmente es el teclado.
- **Salida estándar** (1): Para mostrar resultados. Normalmente la pantalla.
- **Salida estándar de error** (2): Para mostrar mensajes de error. Normalmente la pantalla.

Estos se pueden redireccionar para almacenar datos de salida o recibir datos de entrada de un fichero, o comunicar unos procesos con otros

`bash` — ***GNU Bourne-Again SHell***

SINOPSIS

```
bash [opciones] [comando | archivo]
```

OPCIONES

Todas las opciones de un caracter documentadas en la descripción del comando interno `set` pueden usarse como opciones al invocar el *shell*.

ARGUMENTOS

Si quedan argumentos tras las ***opciones***, el primero de estos se entiende como el nombre de un fichero con comandos *shell*, que guarda en el parámetro `$0`, y el resto de argumentos los guarda en los parámetros posicionales, en orden creciente. Entonces `bash` lee y ejecuta los comandos de este archivo y termina su ejecución. El estado de salida de `bash` es el del último comando ejecutado en el *script*.

INVOCACIÓN

Cuando `bash` se invoca como *shell* de inicio de sesión interactivo, ejecuta los comandos de `~/.bash_profile`, si existe. Cuando un *shell* de inicio de sesión interactivo termina, ejecuta los comandos de `~/.bash_logout`. Cuando se inicia un *shell* interactivo que no es de inicio de sesión, `bash` ejecuta los comandos de `~/.bashrc`, si existe.

DEFINICIONES

Blanco Espacio o tabulador.

Palabra Secuencia de caracteres interpretada como una unidad por el *shell*.

Nombre Palabra con solo caracteres alfanuméricos y barras bajas que no comienza por un número.

Metacaracter Caracter que, si no está entre comillas, separa palabras. Uno de `|`, `&`, `;`, `(`, `)`, `<`, `>`, espacio, tabulador y salto de línea.

Operador de control Palabra que realiza una función de control. Una de `||`, `&`, `&&`, `;`, `;;`, `;&`, `;;&`, `(`, `)`, `|`, `|&` y salto de línea.

GRAMÁTICA DEL SHELL

Comandos simples Secuencia opcional de asignaciones de variables seguida de una lista de palabras separadas por blancos, seguida de redirecciones, y terminada por un operador de control. La primera palabra indica el comando a ejecutar, pasado como argumento 0, y el resto son los argumentos a dicho comando.

Tuberías, *pipelines* o *pipes* Formadas por uno o varios comandos simples separados por `|`, para conectar la salida estándar del de la izquierda con la entrada estándar de la siguiente, o por `|&`, abreviación para `2>&1|`, y que se ejecutan en paralelo en sus respectivos procesos.

Listas Formadas por una o más tuberías separadas por `;`, `&`, `&&` o `||`, y terminadas opcionalmente por `;`, `&` o un salto de línea. `&&` y `||` tienen igual precedencia, seguidos por `;` y `&`, de igual precedencia. Una secuencia de uno o más saltos de línea puede aparecer en una lista en lugar de un `;` para delimitar comandos.

Si un comando termina con `&`, se ejecuta en segundo plano en una *subshell*. Comandos separados por `;` se ejecutan secuencialmente.

Las listas Y (AND) y O (OR) son secuencias de una o más tuberías separadas por `&&` o `||`, respectivamente. Una lista AND tiene forma *comando1* `&&` *comando2*, y en esta *comando2* se ejecuta si y sólo si *comando1* devuelve 0. Una lista OR tiene forma *comando1* `||` *comando2*, y en esta *comando2* se ejecuta si y sólo si *comando1* devuelve un valor distinto de 0. El estado de salida de las listas Y y O es el del último comando ejecutado en la lista.

Comandos compuestos

En la mayoría de los siguientes casos los `;` pueden sustituirse por uno o más saltos de línea.

`(`*lista*`)` La *lista* se ejecuta en un *subshell*.

`{` *lista*`; }` La *lista* se ejecuta en el *shell* actual.

`if` *lista*`; then` *lista*`; [ elif` *lista*`; then` *lista*`; ] ... [ else` *lista*`; ] fi`

Ejecuta la *lista* en `if` y, si devuelve cero, ejecuta la *lista* en `then`. De lo contrario evalúa la *lista* de cada `elif` y, si devuelve cero, ejecuta la *lista* en el `then` correspondiente y termina el comando, y si esto no ocurre, se ejecuta la *lista* en `else`.

`case` *palabra* `in [` *patrón* `[ | ` *patrón* `] ... ) ` *lista* `;; ] ... esac` Busca el primer *patrón* que coincida (según «Expansión de nombres de archivo» pero sin considerar `/` de manera especial) con el valor de la *palabra* una vez expandida y ejecuta la *lista* correspondiente.

`while` *lista1*`; do` *lista2*`; done` Ejecuta *lista1* y, si devuelve 0, ejecuta *lista2* y repite esta acción.

`for` *nombre* `in` *palabra ...* `; do` *lista* `; done` Una vez expandida la lista de *palabra*s, para cada una, ejecuta *lista* con la variable *nombre* establecida a dicha palabra, en orden.

Definiciones de funciones Una función *shell* es un objeto que se llama como un comando simple y ejecuta un comando compuesto con un nuevo conjunto de parámetros posicionales. Se declaran como:

`function` *nombre comando_compuesto*

PARÁMETROS

Un parámetro es una entidad que almacena valores, y puede ser un nombre, un número o uno de los caracteres en «Parámetros especiales». Si es un nombre se le llama **variable**, y puede ser asignado mediante la sentencia `nombre=[`*valor*`]`. No puede haber espacios entre el *nombre*, el signo `=` y el *valor*.

Parámetros posicionales Se denotan por un número positivo distinto de 0 en decimal.

Parámetros especiales

`@` Todos los parámetros posicionales, en orden creciente.

`#` Número de parámetros posicionales, en decimal.

`?` Estado de salida de la tubería ejecutada más recientemente en primer plano.

`$` PID del *shell* actual. Si se ejecuta en un *subshell* con `()`, no devuelve el PID del *subshell*.

`!` PID del último trabajo puesto a ejecutar en segundo plano.

`0` Nombre con el que se ha invocado al *script*.

Variables de shell

`PWD` El directorio actual, variable establecida por el *shell*.

`HOME` Directorio personal del usuario actual.

`PATH` Lista de directorios separados por `:` en los que el *shell* busca comandos.

EXPANSIÓN

Una barra invertida (\) anula el significado especial del caracter siguiente. Encerrar caracteres entre comillas simples (`'`) preserva el valor literal de todos, incluso de \, anulando la expansión y la separación de palabras. Encerrarlos entre comillas dobles (`"`) anula toda expansión salvo la expansión de parámetros, la sustitución de comandos y el significado de \ cuando el siguiente caracter es `$`, `‘`, `"` o un salto de línea.

Expansión de llaves Una palabra que contenga `{`*arg1*`,`*arg2*`,...}` es sustituida (recursivamente) por varias palabras resultado de sustituir dicha expresión por cada una de las palabras entre llaves.

Expansión de tilde En una palabra no entrecomillada, si empieza por `~` seguido de `/`, o si este es el único caracter de la palabra, este es sustituido por el valor del parámetro `HOME`.

Expansión de parámetros `${`*parámetro*`}` es sustituido por el valor del *parámetro*. Las llaves son opcionales y sirven para delimitar el parámetro (de lo contrario se considera que es el nombre más largo formado por caracteres consecutivos después del `$` o, si no hay ninguno, por el primer caracter). Si el primer caracter del *parámetro* es `!`, el valor de la variable formada por el resto del *parámetro* se usa como nombre del parámetro y es expandido en lugar del propio parámetro.

Sustitución de comandos `$(`*comando*`)` o `‘`*comando*`‘` ejecuta el *comando* en una *subshell* y es sustituido por la salida del comando. Sin comillas, el intérprete entiende los saltos de línea dentro de la salida como separadores de palabras.

Expansión aritmética

`$((`*expresión*`))` se sustituye por el resultado de evaluar la *expresión* aritmética (ver «Evaluación aritmética»).

División de palabras Se preservan los argumentos nulos `""` y `''`, pero no los que resultan de una expansión de un parámetro sin valor.

Expansión de rutas de archivo Cuando una palabra contiene `*`, `?` o `[`, el intérprete lo sustituye por una lista de nombres de archivo que cumplan el patrón indicado. El caracter `.` al principio de un nombre o inmediatamente detrás de una `/` debe reconocerse explícitamente en el patrón. En este, la mayoria de caracteres se reconocen a sí mismo, `*` reconoce cero o más caracteres cualesquiera, `?` un caracter cualquiera, y `[...]` indica cualquiera de los caracteres entre corchetes, salvo porque si el primer caracter es `!`, el significado pasa a ser cualquier caracter no indicado tras el `!`, pudiendo en ambos casos indicar rangos como *a*`-`*z*,

Eliminación de comillas Una vez realizadas todas las expansiones se eliminan todas las ocurrencias no entrecomilladas de los caracteres `\`, `'` y `"` que no fueran generadas por dichas expansiones.

REDIRECCIÓN

Estos operadores pueden aparecer detrás de un comando para redirigir su entrada y salida.

Redirección de entrada Con `[`*n*`]<`*fichero* para redirigir la entrada del descriptor *n* (o 0) desde *fichero*.

Redirección de salida Con `[`*n*`]>`*fichero* para redirigir la salida del descriptor *n* (o 1) al *fichero*, truncándolo si ya existía.

Anexado de salida redirigida Con `[`*n*`]>>`*fichero*, similar a lo anterior pero si el *fichero* existe añade los nuevos datos al final.

Duplicado de descriptores de archivo Con `[`*n*`]>&`*m* para duplicar la salida del descriptor *n* (o 1) al descriptor *m*.

ALIAS

Permiten que una cadena sea sustituida por una palabra cuando se usa como la primera de un comando simple. Si esta primera palabra no está entre comillas, se comprueba para ver si es un alias, en cuyo caso se reemplaza por el texto del alias.

EVALUACIÓN ARITMÉTICA

Las expresiones entre paréntesis (que no tienen que ir precedidos por \) se evalúan primero. De mayor a menor prioridad:

`- +` Menos y más unarios.

`**` Exponencial.

`* / %` Multiplicación, división y resto.

`+ -` Suma y resta.

`=` Asignación (a la izquierda está el nombre de una variable).

La expansión de parámetros se realiza antes de evaluar la expresión, y los valores de parámetros se fuerzan a un entero. Las constantes que empiezan por `0` se interpretan en octal, mientras que las que empiezan por `0x` o `0X` se interpretan como hexadecimal. No debe haber espacio entre operandos y operadores.

EJECUCIÓN DE COMANDOS

Si el nombre del comando no contiene `/`, si es el nombre de una función la ejecuta, o si es el nombre de un comando interno, y de lo contrario busca en los directorios en `PATH` uno que contenga un ejecutable con el nombre del comando. Si contiene `/`, lo interpreta como una ruta a un ejecutable. La ejecución de un fichero ejecutable se realiza mediante un `fork` tras el cual el proceso hijo hace un `exec` y el padre, salvo que el proceso se deba ejecutar en segundo plano, espera a que termine el hijo con `wait`.

PROMPTS

El intérprete suele mostrar un prompt como `[`*`usuario`*`@`*`nombremáquina directorio`*`] $`, donde *`usuario`* es el nombre del usuario conectado al sistema, *`nombremáquina`* es el nombre de la máquina y *`directorio`* es el nombre del directorio actual, o `~` para el directorio personal del usuario.

COMANDOS INTERNOS DEL SHELL

Muchos de estos también son externos pero han sido incorporados al código de `bash` para acelerar su ejecución.

`alias` *`nombre=valor`* Establece un alias con el *`nombre`* y *`valor`* dados.

`break` Sale de un bucle `for` o `while`.

`cd [`*`dir`*`]` Cambia el directorio actual al último en que habíamos estado si *`dir`* es `-`, al directorio *`dir`* en caso contrario, o al directorio personal si *`dir`* no se especifica.

`continue` Continúa un bucle `for` o `while` por la siguiente iteración.

`echo [`*opción*`]... [`*arg*`]...` Imprime los *arg*umentos por la salida estándar, separados por espacio y seguidos de un salto de línea. Con `-n`, no se imprime el salto de línea al final. Con `-e`, interpreta las secuencias:

`\b`	Retrocede una posición.
`\f`	Alimentación de página.
`\n`	Salto de línea.
`\t`	Tabulador horizontal.

`exit` *n* Provoca que el *shell* termine su ejecución con un estado de salida de *n*.

`export [`*nombre*`[=`*palabra*`]]` El *nombre* dado se convierte en variable de entorno. Si se proporciona una *palabra*, se asigna como valor a la variable.

`help` *comando* Muestra ayuda detallada de un *comando* interno.

`let` *expr*`...` Evalúa una lista de *expr*esiones aritméticas.

`printf` *formato* `[`*arg*`]...` Imprime los *arg*umentos por la salida estándar según un *formato*, al estilo de C.

`pwd` Imprime la ruta absoluta del directorio actual.

`read [`*nombre*`]` Lee una línea de la entrada estándar y la asigna como valor de *nombre*.

`set [`*opción*`]...` Sin opciones, imprime el nombre y el valor de todas las variables del *shell* (incluyendo las de entorno). Opciones:

`-u`	Trata las variables no asignadas como errores en la expansión, mostrando un mensaje de error.
`-v`	Muestra las líneas de entrada de órdenes conforme se leen.
`-x`	Tras expandir cada comando simple, `for` o `case`, muestra el comando y sus argumentos expandidos.

`shift` Los parámetros posicionales desde el 2 cambian su número a uno menos, y el representado por el número `$#` se desestablece.

`test` *expr* Funciona como la orden externa `test`.

`[` *expr* `]` Ídem.

`env`

Imprime el nombre y el valor de las variables de entorno.

A.2. Ficheros y directorios

`ls` [*OPCIÓN*]... [*FICHERO*]...

Lista información sobre los *FICHERO*s (por defecto, el directorio actual).

`-l` Lista en formato largo. Cada línea se corresponde con una entrada. El primer caracter es `d` para un directorio, `-` para un fichero regular, `l` para un enlace simbólico, `b` para un fichero especial de bloques, etc. A continuación, los permisos en notación estándar, el número de enlaces físicos que tiene la entrada, el usuario y el grupo al que pertenece, el tamaño del fichero en bytes, su fecha y hora de última modificación y finalmente su nombre. Si no se especifica se usa el formato corto, que solo muestra el nombre.

`-R` Lista también los subdirectorios de forma recursiva.

`-a` No omite los ficheros y directorios «ocultos» (los que empiezan por `.`).

`-S` Ordena por tamaño, empezando por el fichero más grande.

`-t` Ordena por tiempo de modificación, empezando por la entrada modificada más recientemente.

`-r` Invierte el orden del listado.

`-d` Lista los directorios que se pasan como parámetros y no su contenido.

Si hay dos opciones contradictorias, predomina la última.

`touch` [*OPCIÓN*]... *FICHERO*...

Cambia el tiempo de modificación de cada *FICHERO* al actual. Un *FICHERO* que no existe es creado vacío.

`cp`

`cp` [*OPCIÓN*]... [*-T*] *ORIGEN DESTINO*
`cp` [*OPCIÓN*]... *ORIGEN*... *DIRECTORIO*
`cp` [*OPCIÓN*]... *-t DIRECTORIO ORIGEN*...
Copia *ORIGEN* a *DESTINO*, o múltiples *ORÍGEN*es a un *DIRECTORIO* existente.

`-r` Copia directorios recursivamente. Usado para copiar directorios.

`-i` Pide confirmación antes de sobrescribir.

`-a` Equivale a `-dR --preserve=all`, que conserva todos los atributos posibles de los *ORÍGEN*es. Sin esto, los elementos copiados tienen como usuario y grupo a quien hace la copia, como tiempos asociados los de cuando la copia termina, los permisos dependen de la configuración del sistema y usuario, etc.

`mv`

`mv [`*`OPCIÓN`*`]... [-`*`T`*`] `*`ORIGEN DESTINO`*
`mv [`*`OPCIÓN`*`]... `*`ORIGEN ... DIRECTORIO`*
`mv [`*`OPCIÓN`*`]... -`*`t DIRECTORIO ORIGEN ...`*

Cambia el nombre de *ORIGEN* a *DESTINO*, o mueve *ORÍGEN*es a *DIRECTORIO*.

`-i` Pide confirmación antes de sobrescribir.

`rm [`*`OPCIÓN`*`]... [`*`FICHERO`*`]...`

Borra (desenlaza) los *FICHERO*s.

`-i` Pide confirmación antes de borrar cada entrada.

`-r` Elimina directorios y sus contenidos recursivamente.

`-f` Nunca pide confirmación.

`mkdir [`*`OPCIÓN`*`]... `*`DIRECTORIO ...`*

Crea los *DIRECTORIO*s, si no existen ya.

`rmdir [`*`OPCIÓN`*`]... `*`DIRECTORIO ...`*

Elimina los *DIRECTORIO*s, si están vacíos.

`cat [`*`OPCIÓN`*`]... [`*`FICHERO`*`]...`

Concatena *FICHERO*s a la salida estándar.

`less [`*`OPCIÓN`*`]... [`*`FICHERO`*`]...`

Muestra el contenido de los *FICHERO*s (o de la entrada estándar) página a página, pudiendo avanzar y retroceder.

COMANDOS

/patrón Busca hacia delante una línea que contenga el *patrón*.

`chown [`*`OPCIÓN`*`]... [`*`PROPIETARIO`*`][:[`*`GRUPO`*`]] `*`FICHERO ...`*

Cambia el usuario y/o grupo de cada fichero dado.

`-R` Opera en ficheros y directorios recursivamente.

`chgrp [`*`OPCIÓN`*`]... `*`GRUPO FICHERO ...`*

Cambia el grupo de cada *FICHERO* a *GRUPO*.

`chmod [`*`OPCIÓN`*`]... `*`MODO FICHERO`*`...`

Establece los permisos de los *`FICHERO`*s a los indicados en *`MODO`*, que puede ser una representación simbólica de los cambios o un número en octal.

La representación simbólica tiene formato `[ugo...][[-+][`*`perms`*`...]...]`, donde *`perms`* es 0 o más letras de `rwxXst` o una sola letra del conjunto `ugo`. La combinación de las letras `ugo` controla qué permisos cambian: los de usuario (`u`), grupo (`g`) u otros (`o`). El operador `+` causa que los bits seleccionados se añadan a los existentes, y `-` causa que se eliminen. Las letras `rwxXst` seleccionan los bits de modo.

`find [`*`punto-de-inicio`*`]... [expresión]`

Busca en los árboles de directorios a partir de cada `punto-de-inicio` dado (por defecto `.`) evaluando la `expresión`, de izquierda a derecha, de acuerdo a las reglas de precedencia, hasta que se conoce el resultado.

EXPRESIÓN

Está compuesta por una secuencia de:

Tests Devuelven verdadero o falso normalmente según alguna propiedad del fichero.

Acciones Tienen efectos secundarios y devuelven verdadero o falso normalmente según si tienen éxito o no.

Opciones globales Afectan a la operación de los tests y acciones en cualquier parte de la línea de comandos. Siempre devuelven verdadero.

Operadores Unen los otros elementos dentro de la expresión.

OPCIONES GLOBALES

`-depth` Procesa el contenido de cada directorio antes del directorio en sí.

`-maxdepth` *`N`* Desciende un máximo de *`N`* (entero no negativo) niveles. Así, si *`N`* es 0, solo se procesan los `puntos-de-inicio`.

TESTS

Los argumentos numéricos (indicados como *`n`*) pueden ser especificados como *`+n`* para mayor que *`n`*, `-`*`n`* para menor que *`n`* o *`n`* para exactamente *`n`*.

`-empty` El fichero o directorio es vacío.

`-name` *`patrón`* El nombre coincide con el patrón, el cual reconoce `*`, `?` y `[` igual que `bash` pero no excluye del reconocimiento un `.` al comienzo del nombre de archivo.

`-iname` *`patrón`* Como `-name`, pero sin distinguir mayúsculas de minúsculas.

`-size` *`n`*`[cwbkMG]` El tamaño del fichero es, redondeando al alza a la unidad especificada, *`n`*. La unidad puede ser: `c` para bytes, `w` para palabras de 2 bytes, `b` para bloques de 512 bytes (por defecto), `k` para kiBs, `M` para MiBs y `G` para GiBs.

`-mtime` *n* El archivo fue modificado por última vez hace *n* días, redondeando a la baja.

`-atime` *n* Igual pero para el último acceso.

`-ctime` *n* Igual pero para el último cambio de estado (ubicación y metadatos).

`-newer` *fichero* El fichero ha sido modificado más recientemente que el fichero dado.

`-type` *c* El fichero es de tipo *c*, que es `d` para directorios, `f` para ficheros ordinarios, `l` para enlaces simbólicos, etc.

`-perm [-|/]`*mode* El fichero tiene los permisos indicados (si no hay un prefijo), al menos estos permisos (`-`) y alguno de estos (`/`).

ACCIONES

`-print` Imprime la ruta de archivo completa seguida de un salto de línea y devuelve verdadero.

`-exec` *comando* Ejecuta un comando, formado por todos los argumentos que siguen a `-exec` hasta encontrar un argumento «`;`» (exclusive), sustituyendo en este la cadena `{}` por el nombre del archivo. Devuelve verdadero si y sólo si el comando devuelve 0.

`-ok` *comando* Como `-exec` pero pregunta primero al usuario si desea ejecutar el comando, y si no acepta, devuelve falso. La entrada de dicho comando se redirige desde `/dev/null`.

`-printf` *formato* Imprime *formato* interpretando secuencias de escape con `\` como en C y directivas `%`: `%p` indica el nombre del fichero, `%s` su tamaño, `%u` el nombre de su propietario, etc.

OPERADORES

De mayor a menor prioridad:

`(` *expr* `)` Fuerza prioridad.

`!` *expr* Cierto si *expr* es falso.

expr1 expr2 Conjunción. *expr2* no es evaluada si *expr1* devuelve falso.

expr1 `-a` *expr2* Igual que *expr1 expr2*.

expr1 `-o` *expr2* Disyunción. *expr2* no es evaluada si *expr1* devuelve verdadero.

`mktemp`

Crea un fichero temporal de forma segura e imprime su nombre.

`dd [OPERANDO]...`

Copia un fichero, convirtiendo y formateando según los operandos.

`bs=BYTES` Lee y escribe hasta *BYTES* bytes a la vez.

`count=N` Copia sólo *N* bloques de la entrada.

`if=FICHERO` Lee del *FICHERO*.

`of=FICHERO` Escribe al *FICHERO*.

BYTES y *N* pueden ir seguidos de sufijos multiplicativos: M = 1024 · 1024.

`/dev/zero`

Las lecturas desde este fichero especial siempre devuelven bytes que contienen 0.

A.3. Ayuda

`apropos [OPCIÓN]... palabra-clave...`

Busca la *palabra-clave* en las descripciones de las páginas del manual.

`whatis [OPCIÓN]... nombre...`

Busca entre los nombres de las páginas del manual aquellos que se asemejen al *nombre* dado.

`man`

```
man [-a] [-S lista] [[sección] página... ]...
man -k palabra-clave...
man -f página...
```

DESCRIPCIÓN

Muestra la *página* indicada en la *sección* indicada (si no se especifica, va buscándola en las secciones en un cierto orden) dentro del manual del sistema. Las *página*s están ordenadas en *seccion*es para programación, administración del sistema, funciones de biblioteca de C, ficheros de configuración, etc. El número de *sección* aparece, junto con el nombre de la *página*, en la primera línea de esta.

OPCIONES

`-a` No termina la búsqueda en la primera página, sino que muestra todas las coincidentes.

`-S lista` *lista* es una lista de números de sección separadas por : que sobrescribe el orden en que se buscan las páginas (puede especificarse una sola sección).

`-k` Equivale a `apropos`.

`-f` Equivale a `whatis`.

FICHEROS

`/etc/man.db.conf` Configuración de `man`.

A.4. Filtros

Son programas que reciben información por la entrada estándar, la procesan y muestran el resultado por su salida estándar.

`grep [`*`OPCIÓN`*`]... PATRÓN [`*`FICHERO`*`]...`

Busca un *`PATRÓN`* en cada *`FICHERO`* (o en la entrada estándar).

Control de reconocimiento

`-i` Ignora diferencias entre mayúsculas y minúsculas

`-v` Invierte el sentido del reconocimiento, seleccionando sólo las líneas que no coinciden.

`-w` Selecciona solo las líneas que contienen el patrón como palabra completa, esto es, al principio de la línea o precedido por un caracter no alfabético, y al final de la línea o seguido de un caracter no alfabético.

`-x` Selecciona sólo las líneas que coinciden completamente con el patrón.

Control de salida general

`-c` No muestra las líneas coincidentes sino su número.

`-l` No muestras las líneas coincidentes sino los archivos en los que hay alguna.

`-o` Muestra solo las partes coincidentes no vacías de cada línea, con cada parte en una línea separada.

`-q` No imprime nada a la salida estándar.

Control de los prefijos de línea en la salida

`-h` Suprime el nombre de archivo como prefijo de cada línea. Esto es por defecto cuando sólo hay un *`FICHERO`* (o sólo la entrada estándar).

`-n` Prefija cada línea con su número dentro del fichero, siendo la primera la 1.

EXPRESIONES REGULARES

Una expresión regular es un patrón que describe un conjunto de cadenas de caracteres. La mayoría de caracteres son expresiones que se reconocen a sí mismas. Un caracter con significado especial debe ser precedido por `\` para que tenga este comportamiento. El `.` reconoce un único caracter cualquiera.

Clases de caracteres y expresiones entre corchetes La sintaxis `[...]` indica un caracter cualquiera entre los corchetes, salvo si el primero es `^`, que entonces es un caracter cualquiera que no este entre dicho `^` y el `]`, exclusive. En ambos casos se pueden usar rangos como *`a-z`*, que indican todos los caracteres entre *`a`* y *`z`*, inclusive.

Posicionamiento `^` y `$` se refieren respectivamente a una cadena vacía al principio o al final de una línea.

Repetición Una expresión puede ir seguida por `*` para indicar que esta se puede repetir cero o más veces, por `\+` indicando que se puede repetir una o más veces o por `\{`*`n`*`,`*`m`*`}` para indicar que se puede repetir entre *`n`* y *`m`* veces, inclusive.

Concatenación

Dos expresiones regulares se pueden concatenar para formar una expresión que reconoce a cualquier cadena formada por la concatenación de dos subcadenas reconocidas respectivamente por estas subexpresiones.

ESTADO DE SALIDA

0 si alguna línea coincide, 1 si no coincide, y 2 si hay un error y no se ha especificado `-q`.

`sort [`*`OPCIÓN`*`]... [`*`FICHERO`*`]...`

Concatena los *`FICHERO`*s (o lee de la entrada estándar) e imprime por la salida estándar el resultado de ordenarlos.

`-n` Ordena numéricamente (por defecto lo hace alfabéticamente), incluyendo números negativos y con decimales, considerando que un número termina cuando aparece el primer caracter no numérico, incluyendo espacios. El separador decimal es `.` en inglés y `,` en español.

`-r` Invierte el resultado de las comparaciones.

`-k` *`KEYDEF`* Ordena según una clave dada por *`KEYDEF`*. Se puede especificar varias veces para indicar una lista de claves de las que cada una se comprueba en caso de empate en la anterior.

`-t` *`SEP`* Usa el caracter *`SEP`* como separador de campos.

KEYDEF tiene formato `F[.C][OPTS][,F[.C][OPTS]]` para las posiciones de inicio y final, donde `F` es un número de campo y `C` una posición de caracter en el campo, ambos empezando por 1, y la posición de final por defecto es el final de línea. Si no se ha especificado `-t`, los caracteres de un campo se cuentan desde el principio del espacio en blanco precedente, y `OPTS` es una o más *OPCION*es de una letra que sobrescriben las opciones de orden global para dicha clave.

`tr` [*OPCIÓN*]... *CONJUNTO1* [*CONJUNTO2*]

Sustituye, elimina repeticiones o elimina caracteres de la entrada estándar, escribiendo a la salida estándar.

`-c` Usa el complemento de *CONJUNTO1*.

`-d` Elimina caracteres de *CONJUNTO1*, no sustituye.

`-s` Reemplaza secuencias de caracteres que aparecen en el último *CONJUNTO* dado por una sola ocurrencia.

Los *CONJUNTO*s se especifican como secuencias de caracteres. La mayoría se representan a sí mismos. Secuencias interpretadas:

`\n` Salto de línea.

`\t` Tabulador.

a`-`*z* Todos los caracteres desde *a* hasta *z* en orden ascendente.

`[`*c*`*`*n*`]` *n* copias de *c*.

`[:alnum:]` Todas las letras y dígitos.

`[:blank:]` Todos los caracteres de espaciado horizontal.

`[:digit:]` Todos los dígitos.

`cut` *OPCIÓN*... [*FICHERO*]...

Imprime las partes seleccionadas de líneas de cada *FICHERO*, o de la entrada estándar, a la salida estándar.

`-c` *LISTA* Selecciona solo estos caracteres.

`-d` *DELIM* Usa *DELIM* en vez de el tabulador como delimitador de campos.

`-f` *LISTA* Selecciona sólo estos campos; imprimiendo también por defecto cualquier línea que no contenga delimitador.

`--complement` Imprime el complemento de los caracteres o campos seleccionados.

`-s` No imprime líneas que no contengan delimitadores.

Usar `-c` o `-f`, pero no ambos. Cada *LISTA* contiene un rango, o varios separados por comas. Cada rango puede ser:

n Caracter o campo *n*-ésimo, contando desde 1.

n-*m* Del caracter o campo *n*-ésimo al *m*-ésimo, inclusive.

Nótese que los campos vacíos también cuentan, por lo que a veces es necesario usar `tr` para eliminar repeticiones de *DELIM*.

`uniq [`*OPCIÓN*`]... [`*ENTRADA* `[`*SALIDA*`]]`

Filtra lineas adyacentes coincidentes de *ENTRADA* (o la entrada estándar), escribiendo a *SALIDA* (o la salida estándar). Sin opciones, las líneas coincidentes (iguales) se eliminan a partir de la segunda.

`-c` Añade antes de cada línea el número de ocurrencias.

`-d` Sólo imprime líneas duplicadas, una por grupo.

`-D` Imprime todas las líneas duplicadas.

`-i` No distingue mayúsculas de minúsculas al comparar.

`-s` *n* No compara los *n* primeros caracteres.

`-w` *n* Sólo compara los *n* primeros caracteres.

`head [`*OPCIÓN*`]... [`*FICHERO*`]...`

Imprime las primeras 10 líneas del *FICHERO* (o la entrada estándar) a la salida estándar.

`-`*n* Imprime las primeras *n* líneas en vez de las primeras 10.

`tail [`*OPCIÓN*`]... [`*FICHERO*`]...`

Imprime las últimas 10 líneas del *FICHERO* (o la entrada estándar) a la salida estándar.

`-n+`*n* Imprime empezando por la línea *n*.

`column [`*OPCIÓN*`]... [`*FICHERO*`]...`

Formatea su entrada en múltiples columnas. Modos:

Tabla Determina el número de columnas de la entrada y crea una tabla. Este modo se activa con la opción `-t`.

`wc [`*OPCIÓN*`]... [`*FICHERO*`]...`

Imprime el número de saltos de línea, palabras y bytes del *FICHERO* (o la salida estándar).

`-l` Imprime el número de saltos de línea.

`tee [`*OPCIÓN*`]... [`*FICHERO*`]...`

Copia la entrada estándar a cada *FICHERO*, y también a la salida estándar.

A.5. *Scripting*

`seq` *PRIMERO INCREMENTO ÚLTIMO*

Imprime los números de *PRIMERO* a *ÚLTIMO*, inclusive, en pasos de *INCREMENTO*.

`echo [`*OPCIÓN*`]... [`*CADENA*`]...`

Muestra las *CADENA*s de caracteres por la salida estándar.

`test`

`test` *EXPRESIÓN*
`[` *EXPRESIÓN* `]`

Termina con el estado determinado por la *EXPRESIÓN*, devolviendo 0 si esta es verdadera y 1 si es falsa.

`(` *EXPR* `)` Fuerza precedencia. Devuelve lo que devuelva la expresión.

`!` *EXPR* «No» lógico.

EXPR1 `-a` *EXPR2* «Y» lógico.

EXPR1 `-o` *EXPR2* «O» lógico.

INT1 `-eq` *INT2* Los enteros son iguales.

INT1 `-ne` *INT2* Los enteros son distintos.

INT1 `-lt` *INT2* *INT1* $<$ *INT2*.

INT1 `-le` *INT2* *INT1* $\leq$ *INT2*.

INT1 `-gt` *INT2* *INT1* $>$ *INT2*.

INT1 `-ge` *INT2* *INT1* $\geq$ *INT2*.

`-z` *STR* La cadena es vacía.

`-n` *STR* La cadena no es vacía.

STR1 `=` *STR2* Las cadenas son iguales.

STR1 `!=` *STR2* Las cadenas son distintas.

STR1 `<` *STR2* *STR1* $<$ *STR2*, en orden lexicográfico.

STR1 `>` *STR2* *STR1* $>$ *STR2*.

`-e` *`FICHERO`* El *`FICHERO`* existe.

`-r` *`FICHERO`* Existe y se tiene permiso de lectura.

`-w` *`FICHERO`* Existe y se tiene permiso de escritura.

`-x` *`FICHERO`* Existe y se tiene permiso de ejecución.

`-f` *`FICHERO`* Existe y es regular.

`-d` *`FICHERO`* Existe y es un directorio.

`-s` *`FICHERO`* Existe y no es vacío.

`true`

Devuelve 0.

`false`

Devuelve 1.

Apéndice B

Gestión de software

B.1. Administración de paquetes

Muchos programas Linux se distribuyen como código fuente, que el usuario construye, junto con las páginas de manual, ficheros de configuración, etc., si bien la mayoría del software se distribuye en **paquetes** precompilados y listos para instalar.

El sistema de paquetes de Fedora es RPM (*Red Hat Package Manager*), en el que los paquetes se llaman como *nombre*`-`*versión*`-`*revisión*`.[`*arquitectura*`|src|noarch].rpm`. La arquitectura es `i386` para Intel x86 (IA-32) y compatibles, `i586` para Pentium y posteriores, `i686` para Pentium II y posteriores, `x86_64` para IA-64 (Intel/AMD de 64 bits), etc. Los paquetes `src` son los fuentes, a compilar por `rpmbuild`, y los `noarch` son independientes de la arquitectura. El sistema gestiona una base de datos de paquetes instalados con información de su estado y ficheros. Podemos referirnos a paquetes instalados simplemente por su *nombre*. Puede haber dependencias entre paquetes, de modo que para instalar uno sea necesario tener otro, que no se puede desinstalar sin desinstalar el primero (ni causar errores).

Podemos usar `rpm`, que da error al instalar o eliminar si las dependencias no se satisfacen, o `dnf`, que resuelve automáticamente las dependencias y descarga los paquetes de repositorios.

`rpm`

OPCIONES DE INSTALACIÓN Y ACTUALIZACIÓN

`rpm -i` *FICHERO* instala un paquete.

`rpm -U` *FICHERO* actualiza a una nueva versión (los ficheros de configuración modificados de la versión anterior se guardan como *nombre_fichero*`.rpmsave`, y si el fichero ha cambiado de formato y no puede ser adaptado a partir de las modificaciones del anterior, se deja el fichero antiguo y se crea uno nuevo *nombre_fichero*`.rpmnew`).

OPCIONES DE ELIMINACIÓN

`rpm -e` *NOMBRE* elimina un paquete.

OPCIONES DE CONSULTA

`rpm -q {[`*`OPCIÓN_SELECCIÓN`*`]...|[`*`OPCIÓN_CONSULTA`*`]... `*`PAQUETE`*`...}`

Opciones de selección

`-f` *`FICHERO`* Paquete al que pertenece el *`FICHERO`*.

Opciones de consulta

`-i` Información del paquete.

`-l` Listado de ficheros en el paquete.

OPCIONES DE VERIFICACIÓN

`rpm -V` *`PAQUETE`*

`dnf [`*`OPCIÓN`*`]... [`*`COMANDO`*`] [`*`ARGUMENTO`*`]...`

`-h` Muestra la ayuda.

`dnf check-update` Comprueba si hay actualizaciones.

`dnf download` *`PAQUETE`*`...` Descarga la última versión de los paquetes indicados, sin llegar a instalarlos.

`dnf erase` *`PAQUETE`*`...` Alias obsoleto para `remove`.

`dnf install` *`PAQUETE`*`...` Se asegura de que los paquetes dados y sus dependencias están instalados. (Para instalar el repositorio `rpmfusion`, parte libre pero con más paquetes que la instalación por defecto, usar `dnf install http://download1.rpmfusion.org/free/fedora/rpmfusion-free-release-$(rpm -E%fedora).noarch.rpm`. También se pueden instalar paquetes de los repositorios y paquetes previamente descargados.)

`dnf remove` *`PAQUETE`*`...` Elimina los paquetes indicados junto con cualquiera que dependa de estos.

`dnf update` Alias obsoleto para `upgrade`.

`dnf upgrade` Actualiza cada paquete a la última versión disponible y resoluble.

`dnf upgrade` *`PAQUETE`*`...` Actualiza cada paquete indicado a la última versión. Actualiza dependencias según sea necesario.

FICHEROS

Configuración principal `/etc/dnf/dnf.conf`

Repositorio `/etc/yum.repos.d/` (Contiene ficheros `.repo`, que se dividen en tres secciones: una para paquetes normales, otra para paquetes de depuración y otra para paquetes fuentes, y que informa a `dnf` de dónde encontrar la lista de espejos para cada sección, de donde descargar los paquetes. `fedora.repo` contiene información para la instalación base, y `fedora-updates.repo` contiene la información para localizar los paquetes a actualizar.)

B.2. Módulos del núcleo

Aunque Linux tiene una arquitectura monolítica, su diseño es modular, y hay porciones (**módulos**) que se pueden cargar o descargar en caliente, que proporcionan soporte para distintos sistemas de ficheros, periféricos y protocolos de red, y que se suelen guardar con extensión `.ko.xz` en `/lib/modules/`*`versión`*, según la *`versión`* de Linux, con los siguientes subdirectorios entre otros:

`drivers` Gestión de dispositivos hardware.

`fs` Sistemas de ficheros.

`net` Protocolos de red.

`sound` Tarjetas de sonido.

Para que un módulo se pueda usar, el núcleo debe tener exportadas todas las funciones que este usa. Algunas están en otros módulos, por lo que existen dependencias entre módulos descritas en `/lib/modules/`*`versión`*`/modules.dep`. Los módulos se pueden cargar en cualquier momento, aunque suelen cargarlos el *script* `init` del disco RAM (para acceder al sistema de ficheros raíz), los servicios de `systemd` al iniciarse o `udev` al detectar un nuevo dispositivo.

`lsmod`

Formatea el contenido de `/proc/modules`, mostrando los módulos del núcleo cargados (nombre, tamaño, contador de usos y módulos que lo usan).

`modinfo` *`NOMBRE_MÓDULO`* `...`

Extrae información sobre los módulos dados (fichero, autor, descripción, licencia, dependencias, parámetros, etc.)

`insmod` *`FICHERO_MÓDULO`*

Inserta un módulo en el núcleo.

`rmmod` *`NOMBRE_MÓDULO`*

Elimina un módulo del núcleo (si no se está usando).

`modprobe` *NOMBRE_MÓDULO*

Añade o elimina un módulo del núcleo (junto a los módulos de los que depende).

`-r` Elimina en vez de insertar. Si los módulos de los que este depende tampoco se están usando, `modprobe` intentará eliminarlos también.

`--show-depends` Lista las dependencias de un módulo.

`/etc/modprobe.d/`

Configuración para `modprobe`. Los ficheros bajo el directorio con extensión `.conf` especifican opciones.

COMANDOS

`alias` *alias nombre_módulo* Da un nombre alternativo a un módulo.

`blacklist` *nombre-módulo* Los módulos pueden contener sus propios alias. `blacklist` indica que todos los alias internos del módulo deben ser ignorados.

`install` *nombre_módulo comando* `...` Ejecuta un comando en vez de insertar el módulo.

`options` *nombre_módulo opción* `...` Añade opciones al módulo cada vez que se inserta en el núcleo.

`remove` *nombre_módulo comando* `...` Como `install` pero se invoca al ejecutar `modprobe -r`.

`depmod`

Crea una lista de dependencias entre módulos bajo `/lib/modules/`*versión* (la versión del núcleo actual). Esta lista es escrita a `modules.dep`. Finalmente, genera un fichero `modules.`*dispositivo* si los módulos proporcionan nombres de dispositivo que deben añadirse a `/dev` en el arranque (`modules.usbmap`, `modules.pcimap`, etc., que sirven para indicar los módulos a cargar cuando se conecta un dispositivo, pero que no se usan actualmente).

Apéndice C

Usuarios y grupos

Internamente los usuarios se identifican por un número, el UID, que es 0 para el superusuario, y que en general está en el rango 1–999 para las cuentas del sistema y es mayor o igual a 1000 para usuarios normales. Una cuenta del sistema ejecuta programas o posee ficheros para ciertos servicios, pero no está asociada a una persona. Por ejemplo, la cuenta `apache` ejecuta servicios para un servidor Web, el demonio `httpd` de Apache.

Los usuarios también tienen un nombre de usuario, contraseña, un directorio personal y una *shell*, programa que se ejecuta al iniciar sesión. Se puede indicar que esta sea un ejecutable específico, no necesariamente una *shell*, convirtiendo la cuenta en una **cuenta restringida**, pues el usuario solo puede ejecutar este fichero y al finalizar su ejecución sale del sistema.

Cuando un usuario crea un fichero, su grupo propietario es el grupo activo del usuario (generalmente su grupo primario), pero al determinar los permisos que tiene un usuario sobre un fichero, se usan todos los grupos a los que pertenece.

`/etc/passwd`

Fichero de texto que describe las cuentas de usuario del sistema. Debe ser legible por todos los usuarios pero escribible sólo para el superusuario.

Tradicionalmente no había problema con este permiso general de lectura. Hoy en día se usa el caracter `x` en la celda de contraseña para indicar que las contraseñas están en `/etc/shadow`, que sólo puede ser leída por el superusuario.

Cada línea del fichero describe un sólo usuario, y contiene los siguientes campos:

```
nombre:contraseña:UID:GID:GECOS:directorio:shell
```

`nombre` Nombre de usuario.

`contraseña` Contraseña encriptada del usuario, `*` o `x`.

`UID` El UID.

`GID` ID numérico del grupo primario del usuario.

`GECOS` Opcional, con propósito informativo. Normalmente contiene el nombre completo del usuario.

directorio Directorio personal del usuario, donde se le posiciona tras iniciar sesión.

shell Programa a ejecutar al iniciar sesión.

/etc/shells

Fichero de texto que contiene las rutas absolutas de los *shells* de inicio de sesión válidos.

/etc/shadow

Fichero que contiene la información de las contraseñas de los usuarios e información opcional de caducidad. Debe no ser legible para usuarios normales por seguridad. Hay una entrada por línea, con formato:

```
nombre:contraseña:último_cambio:vida_mínima:vida_máxima:periodo_alerta:
periodo_inactividad:fecha_expiración:reservado
```

nombre Nombre de usuario.

contraseña Contraseña encriptada. Si contiene algo que no es una contraseña encriptada válida, como `!!` o `*`, el usuario no podrá usar una contraseña Unix para iniciar sesión.

último_cambio Fecha del último cambio de contraseña.

vida_mínima Número de días que el usuario debe esperar antes de poder cambiar nuevamente su contraseña.

vida_máxima Número de días tras el cual el usuario deberá cambiar su contraseña. Tras esto la contraseña podría seguir siendo válida, pero se pedirá al usuario una nueva contraseña la próxima vez que inicie sesión.

periodo_alerta Número de días antes de que la contraseña expire durante los cuales se avisará al usuario.

periodo_inactividad Número de días después de que la contraseña expire durante los que esta seguirá siendo válida. Tras este tiempo el usuario no podrá iniciar sesión.

fecha_expiración Fecha en la que la cuenta expira, tras la cual el usuario no podrá iniciar sesión.

/etc/group

Fichero de texto que define los grupos del sistema. Hay una entrada por línea, con formato:

```
nombre:contraseña:GID:lista_de_usuarios
```

/etc/gshadow

Información de seguridad de los grupos. Debe no ser legible por usuarios normales por seguridad. Hay una entrada por línea, con formato:

```
nombre:contraseña:administradores:miembros
```

`/etc/issue`

Fichero de texto con un mensaje a imprimir antes de pedir el nombre de usuario en un inicio de sesión por terminal. Puede contener secuencias `@`*`caracter`* o `\`*`caracter`* si la terminal lo soporta (por ejemplo, `\l` se sustituye po el nombre de la terminal: `tty1`, `tty2`, etc.).

`/etc/motd`

Fichero de texto a mostrar tras un inicio de sesión por terminal exitoso y antes de ejecutar el *shell*.

`nologin`

Muestra un mensaje de que una cuenta no está disponible y termina. Usado como campo *`shell`* en `/etc/passwd` para denegar el inicio de sesión a una cuenta.

`id` *`USUARIO`*

Muestra la información de usuario y grupos para el *`USUARIO`* indicado.

`groups` *`USUARIO`*`...`

Muestra los grupos a los que pertenece cada *`USUARIO`*.

`newgrp` *`GRUPO`*

Inicia sesión en un nuevo grupo. Si el usuario no es `root`, se le pedirá una contraseña si este no tiene o no es miembro del grupo y el grupo tiene contraseña. Al usuario se le denegará el acceso si la contraseña de grupo es vacía y el usuario no es miembro.

`who`

Muestra información sobre los usuarios con sesiones abiertas.

`su` [*`OPCIÓN`*]`...` [*`USUARIO`*]

Ejecuta un nuevo *shell* con los ID de usuario y grupo indicados (por defecto `root`). Cambia las variables de entorno `HOME` y `SHELL` (más `USER` y `LOGNAME` si el *`USUARIO`* no es `root`).

`-`, `-l` Inicia el *shell* como uno de inicio de sesión con un entorno similar a un inicio de sesión real.

`whoami`

Imprime el nombre de usuario asociado con el ID efectivo de usuario (EUID) actual.

`useradd [`*`OPCIÓN`*`]... `*`USUARIO`*

Crea una nueva cuenta de usuario usando los valores especificados más los valores por defecto del sistema. Por defecto la contraseña de este está desactivada.

`-d `*`DIRECTORIO`* Indica el *`DIRECTORIO`* personal del usuario. Por defecto, aunque depende de la configuración, se crea un directorio con ruta `/home/`*`USUARIO`* y se copia dentro de este el contenido de `/etc/skel`.

`-g `*`GRUPO`* El nombre o número del grupo primario del usuario, que debe existir. Si no se especifica, dependiendo de la configuración, se creará un nuevo grupo con el mismo nombre que el usuario o se le asignará un grupo por defecto.

`-G `*`GRUPO1`*`[,`*`GRUPO2`*`,... [,`*`GRUPON`*`]]]` Lista de grupos adicionales de los que el usuario también es miembro.

`-m` Crea el directorio personal del usuario si no existe ya. Por defecto, la creación de este depende de la configuración.

FICHEROS

`/etc/default/useradd` Valores por defecto para la nueva cuenta.

`/etc/login.defs` Configuración adicional.

`usermod [`*`OPCIÓN`*`]... `*`USUARIO`*

Modifica los ficheros de la cuenta del *`USUARIO`* para reflejar los cambios especificados.

`-c `*`COMENTARIO`* El nuevo valor del campo de comentario del fichero de contraseña para el usuario.

`-g `*`GRUPO`* Nombre o número del nuevo grupo primario del usuario, que debe existir.

`-G `*`GRUPO1`*`[,`*`GRUPO2`*`,... [,`*`GRUPON`*`]]]` Lista de grupos adicionales del que el usuario también es miembro. Si el usuario es miembro del algún grupo que no está en la lista, será eliminado del grupo, salvo si se añade la opción `-a` que añade al usuario esta lista en vez de sustituir.

`-s `*`SHELL`* El nombre de la nueva *shell* inicial del usuario.

`userdel `*`USUARIO`*

Borra del sistema todas las entradas de usuario que hacen referencia al *`USUARIO`* dado, que debe existir. Por defecto no borra el directorio personal.

`groupadd `*`GRUPO`*

Crea un nuevo grupo.

`groupdel` *GRUPO*

Borra del sistema todas las entradas de grupo que hacen referencia al *GRUPO* dado, que debe existir.

`gpasswd` [*OPCIÓN*]... *GRUPO*

Administra `/etc/group` y `/etc/gshadow`. Cada grupo puede tener administradores, miembros y una contraseña.

Salvo por las opciones `-A` y `-M`, las opciones no pueden combinarse.

`-a` *USUARIO* Añade el *USUARIO* al *GRUPO*.

`-d` *USUARIO* Elimina al *USUARIO* del *GRUPO*.

`-A` *USUARIO*`,...` Establece la lista de administradores.

`-M` *USUARIO*`,...` Establece la lista de miembros.

`passwd` [*OPCIÓN*]... [*USUARIO*]

Actualiza la información de autenticación del *USUARIO*. Por defecto cambia su contraseña. Sólo `root` puede especificar el *USUARIO*; si no se especifica, se cambia la contraseña propia.

`-l` Bloquea la contraseña del usuario. Sólo para `root`.

`--stdin` La contraseña se lee de la entrada estándar, que puede ser una tubería.

`-u` Desbloquea la contraseña del usuario. Sólo para `root`.

`-e` El usuario será forzado a cambiar la contraseñá en el siguiente intento de inicio de sesión. Sólo para `root`.

Este comando fuerza algunos requisitos de seguridad en la nueva contraseña (longitud mínima, etc.).

`chage` [*OPCIÓN*]... *USUARIO*

Cambia el número de días entre cambios de contraseña y la fecha de último cambio para el *USUARIO*.

`-E` *FECHA* Establece la fecha de expiración de la cuenta, con formato `YYYY-MM-DD`.

`-I` *DÍAS* Establece el número de días de inactividad tras la expiración de una contraseña durante los cuales el usuario todavía puede iniciar sesión.

`-M` *DÍAS* Establece el máximo de días durante los que una contraseña es válida.

`-W` *DÍAS* Establece el número de días antes de la expiración de una contraseña en los que el usuario es avisado de ello.

NOTA

Este programa requiere de un fichero `/etc/shadow`, y sólo puede usarlo `root`s.

Apéndice D

Administración de discos

D.1. Particionamiento estándar

Existen dos formatos principales para el particionamiento de discos:

- **Particionamiento DOS** o **MBR**: El formato original de MS-DOS. Permite hasta 4 **particiones principales** definidas en el primer sector del disco, el MBR (*Master Boot Record*), junto con el código de arranque. Una de estas puede ser una **partición extendida**, dentro de la cual podemos crear **particiones lógicas** o **secundarias**. En Linux, las particiones primarias tienen números del 1 al 4 y las lógicas comienzan por el 5.
  Los números de los sectores en el disco, llamados **LBA** (*Logical Block Address*), se almacenan en 32 bytes, por lo que si los sectores son de 512 B (lo normal), el tamaño máximo particionable es de 2 TiB.
  Las particiones tienen un tipo que indica qué va a contener o para qué se va a usar, con propósito únicamente informativo. Algunos tipos son `0x82` (Linux swap), `0x83` (Linux) o `0x8e` (Linux LVM).

- **Particionamiento GPT** (*GUID Partition Table*): Usa LBAs de 64 bits, luego el tamaño máximo con bloques de 512 B es de 8 ZiB. Permite un número prácticamente ilimitado de particiones, aunque los sistemas operativos suelen limitarlo a 128 para mantener la tabla de particiones de un tamaño razonable (16 KiB). Esta tabla abarca el sector 1 del disco y sucesivos y tiene una suma de verificación CRC32 para verificar su integridad y una copia de respaldo en los últimos sectores del disco. Por compatibilidad, el sector 0 almacena una tabla de particiones DOS con una única partición de tipo `0xEE` (GPT) llamada **partición MBR de protección**, tan grande como sea posible para intentar cubrir todo el disco.
  Cada partición disco y cada partición tiene un **GUID** (*Globally Unique Identifier*), un número de 128 bits con formato `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX` (cada `X` es una cifra hexadecimal) que lo identifica unívocamente. Desaparece la distinción entre particiones primarias, lógicas y extendidas.

Normalmente la primera partición comienza en el sector 2048, usando el hueco que queda para el gestor de arranque.

`fdisk`

`fdisk [`*`OPCIÓN`*`]... `*`DISPOSITIVO`*
`fdisk -l [`*`DISPOSITIVO`*`]...`

Programa interactivo para la creación y manipulación de tablas de particiones que entiende, entre otros, los formatos MBR y GPT.

`-l` Lista las tablas de particiones para los *`DISPOSITIVO`*s especificados y termina.

ÓRDENES

`p` Muestra la tabla de particiones.

`n` Crea una nueva partición.

`d` Elimina una partición.

`w` Guarda en disco la tabla de particiones resultante.

`q` Sale sin guardar los cambios.

`m` Muestra una ayuda con todas las órdenes.

TAMAÑOS

El diálogo de último sector acepta un tamaño de partición según el número de sectores o la notación `+<TAMAÑO>{K,B,M,G,...}`. Si el tamaño lleva delante `+` se interpreta como relativo al primer sector de la partición. En este caso el tamaño se espera en bytes y puede ir seguido de sufijos multiplicativos `K` $= 1024$, `M` $= 1024^2$, etc.

`lsblk [`*`OPCIÓN`*`]...`

Muestra información de todos los dispositivos de bloques disponibles en forma de árbol por defecto. La salida está sujeta a cambios, pero actualmente muestra las columnas `NAME` (nombre), `SIZE` (tamaño) y `TYPE` (tipo: disco, partición o volumen lógico), entre otras.

`-f` Muestra información sobre los sistemas de ficheros: columnas `NAME`, `FSTYPE` (tipo de sistema de ficheros), `LABEL` (etiqueta), `UUID` (identificador) y `MOUNTPOINT` (punto de montaje).

`-p` Muestra rutas de fichero (absolutas) completas.

D.2. LVM

Un **gestor de volúmenes lógicos** (LVM, *Logical Volume Manager*) es un sistema que «virtualiza» el almacenamiento para crear discos lógicos y proporcionar métodos de asignación de espacio más flexibles que las particiones.

Nos centraremos en el LVM de Linux, llamado LVM. Este parte de **volúmenes físicos** (PVs, *physical volumes*), que pueden ser cualquier dispositivo de bloques y se dividen en **extensiones físicas** (PEs, *physical extents*) de un tamaño fijo, que en Linux por omisión es de 4 MiB. Estos se

agrupan en **grupos de volúmenes** (VGs, *volume groups*), que agrupan las extensiones físicas para crear encima **volúmenes lógicos** (LVs, *logical volumes*), que se dividen en **extensiones lógicas** (LEs, *logical extents*) del mismo tamaño que las extensiones físicas y actúan como cualquier otro dispositivo de bloques. Algunos tipos de volúmenes lógicos:

- **Lineales**: A cada extensión lógica le corresponde una única extensión física, en orden lineal.
- **Repartidos**: El volumen se divide en porciones fijas con un **tamaño de porción** (*stripe size*) generalmente menor que una extensión lógica (por defecto 64 KiB), con lo que porciones consecutivas se almacenan en volúmenes físicos de discos diferentes de forma cíclica. Esto hace que el rendimiento de E/S sea mayor que el de un único disco al poder operar en paralelo pero disminuyendo su fiabilidad, pues la probabilidad de que falle un volumen repartido es mayor que la de que falle un disco individual.
- **Reflejados**: A cada extensión lógica le corresponde una extensión física en cada uno de dos o más volúmenes físicos determinados, consiguiendo tolerancia a fallos en caso de que un volumen físico falle. En general, las escrituras tardan lo mismo que en un disco físico individual pero se mejora el rendimiento de lectura. Esto se implementa mediante volúmenes lógicos internos lineales.
- **RAID 5**: Funcionan como un volumen lógico repartido al que se le añade un volumen físico adicional, de modo que si tomamos una porción de un volumen físico y la correspondiente del resto, una de ellas lo que contiene es el resultado de aplicar una función de paridad sobre el resto. Esto permite que no se pierdan datos si un volumen físico deja de funcionar, y al mismo tiempo proporciona un mejor rendimiento.

Los volúmenes lógicos pueden aumentar de tamaño aun con sistemas de ficheros montados, en cuyo caso, si el sistema de ficheros lo permite, se puede aumentar el tamaño de este. También pueden reducir su tamaño (salvo los reflejados o RAID 5 en la implementación actual), pero en este caso los sistemas de ficheros no pueden reducirse estando montados.

También se pueden añadir volúmenes físicos a un grupo de volúmenes, o eliminarlos si no se están usando, en vivo, así como mover extensiones físicas de un volumen a otro, de modo que si el hardware permite cambiar discos en caliente, podemos actualizar o reemplazar dispositivos de almacenamiento sin parar el sistema.

`pvcreate` *`VOLUMEN`*`...`

Inicializa un volumen físico para su uso por LVM.

`pvdisplay [`*`VOLUMEN`*`]...`

Muestra las propiedades de los volúmenes físicos indicados, o de todos si no se indica ninguno.

`vgcreate` *`NOMBRE_GRUPO VOLUMEN`*`...`

Crea un nuevo grupo de volúmenes sobre dispositivos de bloques. Si los dispositivos no habían sido inicializados por `pvcreate`, `vgcreate` los inicializa.

`vgdisplay` *NOMBRE_GRUPO* `...`

Muestra las propiedades de los grupos de volúmenes indicados.

`-v, --verbose` Incrementa la cantidad de salida por cada aparición. Con una ya muestra los volúmenes físicos que forman parte de cada grupo.

`lvcreate [`*OPCIÓN*`]...` *GRUPO*

Crea un nuevo volumen lógico en un grupo de volúmenes. Por defecto crea un LV lineal (`linear`).

`--type raid5` Crea un volumen RAID 5. Se debe indicar también la opción `-i`.

`-i` *N* Crea un volumen, por defecto repartido (`striped`), entre *N* volúmenes físicos. Si se crea un volumen RAID 5, este número no incluye el volumen extra requerido para la paridad.

`-m, --mirrors` *N* Crea un volumen físico reflejado (`raid1`|`mirror`) entre *N* volúmenes físicos.

`-L` *tamaño*`[`*UNIDAD*`]` Obligatorio; tamaño de la nueva unidad.

`-I` *tamaño*`[`*UNIDAD*`]` Tamaño de porción en un volumen lógico repartido o RAID 5.

`-n` *Nombre* Nombre del nuevo volumen.

`lvdisplay [`*OPCIÓN*`]...` *VOLUMEN* `...`

Muestra las propiedades de los volúmenes lógicos indicados.

`-m, --maps` Muestra la correspondencia de extensiones lógicas a volúmenes y extensiones físicas.

`lvextend [`*OPCIÓN*`]...` *VOLUMEN*

Extiende un volumen lógico al tamaño especificado. Debe indicarse una, y sólo una, de entre `-l` y `-L`.

`-l [+]`*Número*`[`*PORCENTAJE*`]` Nuevo tamaño del volumen en extensiones lógicas. El total de extensiones lógicas será mayor cuando se necesiten datos redundantes. El tamaño también se puede indicar como porcentaje. El sufijo `%VG` denota el tamaño total del grupo de volúmenes, y `%FREE` el espacio libre en este. Si se usa el prefijo `+`, el valor no es absoluto sino relativo, y se añade al tamaño actual.

`-L [+]`*Tamaño*`[`*UNIDAD*`]` Nuevo tamaño del volumen. Si se usa el prefijo `+`, el valor no es absoluto sino relativo, y se añade al tamaño actual.

`-r, --resizefs` Cambia el tamaño del sistema de ficheros subyacente junto al del volumen.

`lvreduce [`*OPCIÓN*`]... `*VOLUMEN*

Reduce el tamaño de un volumen lógico.

`-L [-]`*Tamaño*`[`*UNIDAD*`]` Nuevo tamaño del volumen. Si se usa el prefijo `-`, el valor no es absoluto sino relativo, y se reduce del tamaño actual.

`-r, --resizefs` Cambia el tamaño del sistema de ficheros subyacente junto al del volumen.

`vgextend`

Añade uno o más volúmenes físicos a un grupo de volúmenes.

`vgreduce`

Elimina uno o más volúmenes físicos sin usar de un grupo de volúmenes.

`pvmove`

Mueve extensiones físicas de un volumen físico fuente a uno o más volúmenes físicos destino.

`lvremove `*VOLUMEN*`...`

Elimina uno o más volúmenes lógicos.

`vgremove `*GRUPO*`...`

Elimina uno o más grupos de volúmenes.

`pvremove `*VOLUMEN*`...`

Borra la etiqueta de un dispositivo para que LVM deje de reconocerlo como volumen físico.

D.3. Dispositivos de bucle

En Linux solo podemos montar dispositivos de bloques, pero existen dispositivos de bloques llamados ***loop***, normalmente llamados `/dev/loop0–7` que representan ficheros regulares, que a su vez suelen crearse como un fichero grande inicialmente lleno de ceros.

`losetup`

Obtener información `losetup [`*DISPOSITIVO*`]`

Desasociar un dispositivo *loop* `losetup -d `*DISPOSITIVO*`...`

Desasociar todos los dispositivos *loop* `losetup -D`

Asociar un dispositivo *loop* `losetup [`*OPCIÓN*`]... -f|`*DISPOSITIVO FICHERO*

`-a` Muestra el estado de todos los dispositivos *loop*. No toda la información está disponible si no se es superusuario.

`-f` Encuentra el primer dispositivo *loop* sin usar.

`-P` Fuerza al núcleo a escanear la tabla de particiones del dispositivo *loop* recién creado.

D.4. Sistemas de ficheros

El sistema de ficheros ext2 fue el principal en Linux hasta que apareció ext3 que, al contrario que ext2, es **transaccional**: añade un registro o *journal* que permite recuperar rápidamente la consistencia tras una caída del sistema, algo que en ext2 llevaba mucho tiempo al tener que analizar el sistema de ficheros por completo. Actualmente se usa ext4, muy similar a ext3 pero con mejoras que permiten un menor uso de CPU y mayor rapidez de lectura y escritura.

FAT es una familia de sistemas de ficheros introducida a finales de los 70 por Microsoft. La versión inicial era muy básica y dio paso a FAT12, ideada para su uso en disquetes y reemplazada posteriormente por FAT16 para soportar discos duros de mayor tamaño. Con Windows 95 surge VFAT, una extensión opcional que permite almacenar nombres de fichero de hasta 255 caracteres en vez de los 8 para nombre y 3 de extensión que se permiten desde FAT12. Una revisión de Windows 95 introduce FAT32, con mayor soporte para discos duros más grandes, por lo que actualmente este sistema es una de las formas más sencillas (y limitadas) de compartir ficheros entre Linux y Windows. El manejador de FAT para Linux más adecuado (en general) es `vfat`, que soporta FAT32 con VFAT. Los ficheros en FAT no tienen propietario, grupo ni permisos asociados.

XFS fue creado por Silicon Graphics para IRIX, su implementación de UNIX, liberado en el año 2000 bajo licencia GPL y posteriormente soportado por Linux. NTFS apareció en Windows NT y es el sistema de ficheros por defecto de esta familia de sistemas operativos.

En Linux hay una única jerarquía de directorios, y cada sistema de ficheros se **monta** en un directorio llamado **punto de montaje**, de forma que, hasta que este se **desmonte**, podemos acceder a los ficheros que contiene a partir de este directorio. El sistema de ficheros raíz es el que se monta en `/`, el primero en montarse durante el arranque, y no se puede desmontar. Normalmente solo el superusuario puede montar un sistema de ficheros.

En ocasiones también puede montarlo un usuario normal, pero sin indicar opciones extra. Tras montar un sistema de ficheros, el punto de montaje toma los permisos, propietario y grupo del directorio raíz del sistema de ficheros que se monta.

`/etc/fstab`

Contiene información sobre sistemas de ficheros a montar. Cada sistema de ficheros se describe en una línea separada. Los campos en cada línea se separan con tabulaciones o espacios.

`dispositivo` Dispositivo de bloques a montar. Se puede usar `LABEL=<label>` o `UUID=<uuid>` en vez de un nombre de fichero.

`punto_de_montaje` Punto de montaje para el sistema de ficheros.

tipo Tipo de sistema de ficheros. Linux soporta muchos tipos: `ext3`, `ext4`, `vfat`, `iso9660`, `swap` y muchos más.

opciones Opciones de montaje asociadas al sistema de ficheros. Lista separada por comas. Contiene al menos el tipo de montaje (`ro` o `rw`, que puede estar dentro de `defaults`).

frecuencia_copia Usada por `dump` para determinar los sistemas de ficheros a los que hacer copia de seguridad. Por defecto es 0 (no hacer copia).

comprobación Usada por `fsck` para determinar el orden de comprobación de los sistemas de ficheros al arrancar (se comprueban primero los que tengan valor 1, después 2, etc.). El sistema raíz debería indicarse con un valor de 1. Otros sistemas deberían tener un valor de 2. Por defecto es 0 (no comprobar).

mkfs [*OPCIÓN*]... *DISPOSITIVO*

Crea un sistema de ficheros Linux en un dispositivo, normalmente una partición de disco duro. Actualmente `mkfs` es sólo una fachada sobre los creadores de sistemas de ficheros (`mkfs.`*tipo*) disponibles en Linux. El creador específico al sistema de ficheros se busca solo mediante la variable de entorno `PATH`.

`-t` *tipo* Especifica el *tipo* del sistema de ficheros a crear.

mke2fs [*OPCIÓN*]... *DISPOSITIVO*

Crea un sistema de ficheros ext2, ext3 o ext4. Si se ejecuta como `mkfs.XXX`, se entiende que el tipo concreto es *XXX*.

e2label *DISPOSITIVO* [*ETIQUETA*]

Cambia la etiqueta de un sistema de ficheros ext2, ext3 o ext4.

fatlabel

Establece u obtiene la etiqueta de un sistema de ficheros MS-DOS.

blkid

Determina el tipo de contenido de un dispositivo de bloques, así como los atributos a partir de los metadatos de contenido (como celdas `LABEL` o `UUID`).

mount

`mount` [*OPCIÓN*]... *DISPOSITIVO*|*DIRECTORIO*
`mount` [`-t` *tipo*] [*OPCIÓN*]... *DISPOSITIVO DIRECTORIO*

Monta un sistema de ficheros. Si solo se indica el directorio o el dispositivo, `mount` busca un punto de montaje (y si no lo encuentra, un dispositivo) en el fichero `/etc/fstab`.

Listado de montajes El siguiente comando lista todos los sistemas de ficheros montados (o de tipo *tipo*):

`mount [-l] [-t` *tipo*`]`

OPCIONES DE LÍNEA DE COMANDOS

`-o` *opciones* Opciones de montaje, lista separada por comas.

`-r` Sinónimo de `-o ro`.

`-t` *tipo* Tipo del sistema de ficheros.

OPCIONES DE MONTAJE INDEPENDIENTES DEL SISTEMA DE FICHEROS

`auto` Puede ser montado con la opción `-a`.

`noauto` Sólo puede ser montado de forma explícita.

`defaults` Usa las opciones por defecto: `rw`, `suid`, `dev`, `exec`, `auto`, `nouser` y `async`.

`nodev` No interpreta los dispositivos especiales de caracteres y bloques en el sistema de ficheros.

`exec` Permite la ejecución de ficheros.

`noexec` No la permite.

`remount` Intenta volver a montar un sistema de ficheros ya montado. Se usa normalmente para cambiar las opciones de montaje.

`ro` Monta el sistema de ficheros para solo lectura.

`rw` Monta el sistema de ficheros para lectura y escritura.

`users` Permite a cualquier usuario montar y desmontar el sistema de ficheros. Implica `noexec`, `nosuid` y `nodev` (salvo que lo sobrescriban opciones posteriores).

`suid` Permitir el efecto de los bits SUID o SGID.

`nosuid` No permitirlo.

OPCIONES DE MONTAJE ESPECÍFICAS AL SISTEMA DE FICHEROS

`grpquota|usrquota` Aceptadas pero ignoradas. (Las utilidades de cuotas podrían reaccionar a estas cadenas en */etc/fstab*. Concretamente, activan las cuotas de grupo y usuario, respectivamente, si el sistema de ficheros soporta un mecanismo de cuotas.)

Opciones de montaje para FAT

`uid=`*`valor`* y `gid=`*`valor`* Establecen el propietario y grupo de todos los ficheros.

`umask=`*`valor`* Establece la máscara-u (máscara de permisos que **no** están presentes). El valor se da en octal.

`dmask=`*`valor`* Establece una máscara-u solo en directorios.

`fmask=`*`valor`* Establece una máscara-u solo en ficheros.

`umount {`*`DIRECTORIO`*`|`*`DISPOSITIVO`*`}`

Desmonta un sistema de ficheros. No se puede desmontar cuando está «ocupado» (*busy*).

`findmnt [`*`PUNTO_MONTAJE`*`]`

Lista todos los sistemas de ficheros o busca uno. Por defecto imprime todos los sistemas de ficheros en formato similar a un árbol.

`fuser [`*`OPCIÓN`*`]... `*`NOMBRE`*`...`

Muestra los PIDs de procesos que usan los ficheros o sistemas de ficheros especificados. En el modo por defecto, a cada nombre de fichero le sigue una letra indicando cada tipo de acceso:

`c` Directorio actual.

`e` Ejecutable siendo ejecutado.

`f` Fichero abierto.

`-m` *`NOMBRE`* *`NOMBRE`* especifica un fichero montado en un sistema de ficheros o dispositivo de bloques montado.

`-v` Muestra una salida detallada. Los campos `PID`, `USER` y `COMMAND` son similares a los de `ps`. `ACCESS` muestra cómo el proceso accede al fichero. También se muestra cuándo un fichero particular es accedido como punto de montaje.

`df [`*`OPCIÓN`*`]...`

Muestra el espacio disponible en todos los sistemas de ficheros actualmente montados.

`-h` Muestra los tamaños en potencias de 1024 (p. ej., `1023M`).

`-i` Muestra información de uso sobre los nodos-i en vez de sobre los bloques.

`-T` Imprime el tipo del sistema de ficheros.

`du [OPCIÓN]... [FICHERO]...`

Muestra un resumen de uso de disco en los *FICHERO*s, recursivamente en caso de directorios.

`-a` Muestra el uso de espacio para todos los ficheros, no solo directorios.

`-h` Mostrar los tamaños en formato legible para humanos (p. ej., `1K`, `234M`, `2G`).

`-s` Mostrar sólo el total para cada argumento.

Los valores mostrados están en unidades de 1024 bytes (o 512 si se establece `POSIXLY_CORRECT`).

D.5. Cuotas de disco

Algunos sistemas de ficheros soportan **cuotas**, permitiendo establecer límites sobre la cantidad de espacio que puede usar cada usuario. En el caso de ext2, ext3 y ext4, cada usuario tiene límites en el número de bloques y el de nodos-i, de dos tipos:

- **Límite duro** (*hard limit*): No se puede sobrepasar.
- **Límite blando** (*soft limit*): Inferior al duro, se puede sobrepasar durante cierto **periodo de gracia**, en el que se informa al usuario.

En Linux, las cuotas se deben activar explícitamente.

`quotacheck -nm SISTEMA_DE_FICHEROS...`

Examina cada sistema de ficheros (indicado por el punto de montaje), crea una tabla con el uso de disco actual y la compara con la contenida en el fichero de cuotas de disco del sistema de ficheros. Si se encuentran inconsistencias, tanto el fichero como la copia en el sistema de las cuotas incorrectas se actualizan. `quotacheck` espera que cada sistema de ficheros a comprobar tenga ficheros de cuota llamados `[a]quota.user` y `[a]quota.group` en la raíz. Si un fichero no está presente, `quotacheck` lo crea.

`quotaon SISTEMA_FICHEROS...`

Anuncia al sistema que las cuotas de disco deben ser activadas en uno o más sistemas de ficheros (indicados por el punto de montaje). Los ficheros de cuota deben estar presentes en el directorio raíz del sistema de ficheros.

`setquota`

`setquota -u NOMBRE LÍMITE_BLANDO_BLOQUES LÍMITE_DURO_BLOQUES LÍMITE_BLANDO_NODOS_I LÍMITE_DURO_NODOS_I SISTEMA_DE_FICHEROS`
`setquota -t PERIODO_GRACIA_BLOQUES PERIODO_GRACIA_NODOS_I SISTEMA_DE_FICHEROS`

Editor de cuotas.

`-u` Establece cuotas para el usuario nombrado.

`repquota` *SISTEMA_DE_FICHEROS*

Imprime un resumen de los usos de disco y las cuotas para los sistemas de ficheros indicados.

`quota`

Imprime el uso de disco y los límites del usuario.

D.6. Copias de seguridad

Es responsabilidad del administrador que las copias de seguridad de los sistemas de ficheros se hagan en el momento oportuno y estén en un lugar seguro, para evitar la posible pérdida de datos. Tipos de copia:

- **Completa**: Tardan mucho en realizarse y la recuperación de un único fichero puede no ser inmediata, por lo que sólo están justificadas si los ficheros cambian mucho y son vitales para el trabajo de mucha gente o ante grandes cambios como nuevo software, sistema operativo, etc.
- **Diferencial**: Sólo se copian los ficheros creados o modificados después de la última copia completa.
- **Incremental**: Sólo se copian los ficheros creados o modificados desde la última copia de seguridad, por lo que ocupan menos que las diferenciales. Muchas veces se hace una copia completa con poca frecuencia seguida de copias incrementales con mucha frecuencia, incluso a diario.

`tar` *OPCIÓN*`...` [*ARGUMENTO*]`...`

Almacena múltiples ficheros en uno solo (un **archivo**), y manipula dichos archivos.

En **estilo tradicional**, el primer argumento es un clúster de opciones y el resto son argumentos a aquellas opciones que lo requieran. En **estilo UNIX**, a cada opción le precede un guión. Si una opción toma un argumento, este lo sigue, bien como palabra separada en la línea del comando o inmediatamente detrás de la opción. Cualquier número de parámetros que no tome argumentos puede juntarse tras un solo guión. En **estilo GNU**, cada opción comienza con dos guiones y tiene un nombre descriptivo. Los tres estilos pueden mezclarse.

OPCIONES

Modo de operación

`-c` Crea un nuevo archivo. Los argumentos indican los nombres de los ficheros a archivar. Los directorios se archivan recursivamente.

`-r` Añade fichero al final de un archivo. Los argumentos tienen el mismo significado que para `-c`.

`-t` Lista el contenido de un archivo. Los argumentos son opcionales. Cuando aparecen, son los nombres de los miembros a listar.

`-x` Extrae ficheros de un archivo. Los argumentos son opcionales. Cuando aparecen, son los nombres de los miembros del archivo a extraer.

Tratamiento de atributos de fichero

`-p` Extrae información sobre permisos de fichero.

Selección y cambio de dispositivo

`-f` *`ARCHIVO`* Usa el fichero o dispositivo de archivo indicado.

Opciones de compresión

`-z` Filtra el archivo a través de `gzip`.

Selección de ficheros local

`-h` Sigue enlaces simbólicos; archiva los ficheros a los que apuntan.

`-N, --after-date=`*`FECHA`* Sólo almacena ficheros posteriores a *`FECHA`*. Si *`FECHA`* empieza por `/` o `.` se toma como un nombre de fichero; la fecha de creación de este fichero se usa como fecha.

`-P` No elimina las barras al principio de nombres de fichero al crear archivos (los guarda con ruta absoluta).

Salida informativa

`-v` Lista los ficheros procesados.

`cpio`

En modo copia hacia fuera, lee una lista de nombres de fichero, uno por línea, de la entrada estándar, y escribe el archivo en la salida estándar. En modo copia hacia dentro, lee un archivo de la entrada estándar.

Modo principal de operación

`-i` Extrae ficheros de un archivo (copia hacia dentro).

`-o` Crea un archivo (copia hacia fuera).

`-t` Imprime una tabla de contenidos de la entrada.

Modificadores de operación para ambos modos

`-v` Lista los ficheros procesados, o con `-t`, proporciona un listado estilo `ls -l`.

`-F` *`ARCHIVO`* Usa este *`ARCHIVO`* en vez de la entrada o salida estándar.

`--no-absolute-filenames` Crea todos los ficheros relativos al directorio actual.

Modificadores de operación para copia hacia fuera

`-a` Resetea los tiempos de acceso de los ficheros tras leerlos.

Modificadores de operación para copia hacia dentro

`-d` Crea directorios cuando se necesite.

`-m` Presenta las fechas de modificación previas al crear ficheros.

`rsync [`*`OPCIÓN`*`]... `*`FUENTE DESTINO`*

Puede copiar localmente o hacia/desde otro ordenador sobre una *shell* remota (demonio `sshd`). Usa un algoritmo que reduce la cantidad de datos enviados por red enviando solo la diferencia entre los archivos fuente y los existentes en el destino.

`-v` Un solo `-v` te dará información sobre qué ficheros están siendo transferidos.

`-a` Recursión y preservar casi todo.

`--backup` Los ficheros de destino preexistentes cambian de nombre.

`--backup-dir=`*`DIR`* Combinado con `--backup`, guarda las copias de seguridad en el directorio especificado del destino.

`-n` Ejecución de prueba que no realiza ningún cambio (y produce en general la misma salida que una ejecución real).

`--delete` Borrar ficheros del lado receptor que no están en la fuente.

`-z` Comprime los datos de los ficheros conforme se envían al destino, reduciendo la cantidad de datos transmitidos.

D.7. `proc` y `sysfs`

`proc` (montado en `/proc`) y `sysfs` (montado en `/sys`) son sistemas de ficheros virtuales (no se almacenan en ningún disco) que nos permiten acceder a la información del núcleo sobre dispositivos hardware disponibles, buses, módulos, etc. En `/proc`, tenemos:

`devices` Dispositivos de bloques y caracteres actualmente configurados (cuyos módulos están cargados).

`interrupts` Interrupciones tratadas y dispositivos asociados.

`iomem` Rangos de memoria para E/S mapeada a memoria.

`ioports` Puertos registrados para E/S no mapeada a memoria.

`modules` Lista de módulos del núcleo cargados.

La información más completa se encuentra en `/sys`.

`lshw`

Extrae información detallada sobre la configuración hardware. Configuración de memoria, versión de CPU, configuración de caché, buses, etc.

`lsusb`

Muestra información sobre los buses USB y dispositivos conectados.

`-v` Muestra información detallada sobre los dispositivos.

`-d` *`FABRICANTE`*`:`*`PRODUCTO`* Sólo muestra el dispositivo con el ID de fabricante y de producto indicados en hexadecimal.

`lspci`

Muestra información sobre los buses PCI y dispositivos conectados.

`-v` Muestra información detallada.

D.8. Planificación de E/S en discos

La escritura a disco pasa por una **caché de disco** en RAM, y los bloques modificados son realmente escritos por demonios del núcleo. Para bloques que no están en la caché, existen 3 planificadores:

- **CFQ** (*Completely Fair Queuing*): Va asignando el disco a cada proceso durante cierto tiempo, en el cual solo se sirven peticiones de este proceso, siguiendo básicamente un algoritmo SCAN.
- ***Deadline***: Asigna plazos de tiempo a cada petición, de forma que si hay peticiones con plazos incumplidos, estas son atendidas mediante FCFS y, de lo contrario, aplica un algoritmo SCAN.
- **NOOP** (*no-operation*): FCFS.

Para discos duros, CFQ proporciona mejor rendimiento en general, por lo que es el **planificador por omisión**; *deadline* funciona mejor para bases de datos, y NOOP es el peor. Para SSDs y ciertos dispositivos inteligentes como sistemas RAID por hardware, el mejor es NOOP, pues el resto introducen pequeños retrasos que sólo tienen sentido para discos duros.

Podemos obtener el planificador para un disco leyendo de `/sys/block/`*`nombre_disco`*`/queue/scheduler` (siendo *`nombre_disco`*, por ejemplo, `sda`), que nos muestra todos los planificadores con el actual entre corchetes, y podemos cambiarlo «sobreescribiéndolo» con el nombre del planificador que queremos (`noop`, `deadline` o `cfq`). El parámetro del núcleo `elevator=`*`planificador`* establece el planificador por omisión.

En CFQ, los procesos tienen prioridades de E/S.

`ionice [`*`OPCIÓN`*`]... [-p` *`PID`*`|`*`COMANDO`*`...]`

Establece u obtiene la clase de E/S y prioridad para un proceso. Sin parámetros o sólo con `-p`, obtiene la clase de planificación y prioridad de E/S para el proceso. Si se da un *`COMANDO`*, ejecuta el comando con los parámetros dados. Si no se especifica una *`clase`*, se ejecuta con clase de prioridad «mejor esfuerzo». El nivel de prioridad por defecto es 4.

Un proceso puede estar en tres clases de planificación E/S:

Desocupado (*idle*) Sólo obtiene el disco cuando ningún otro programa lo ha pedido durante un cierto periodo de gracia.

Mejor esfuerzo (*best-effort*) Toma un argumento de prioridad de 0–7, con menor número para mayor prioridad. Los procesos con igual prioridad se sirven en *round-robin*. Un proceso que no ha pedido una prioridad de E/S hereda su clase de planificación de CPU. Su prioridad se deriva del nivel *nice* de CPU ($\left\lfloor \frac{\text{valor nice}+20}{5} \right\rfloor$).

Tiempo real (*realtime*) Se le da primer acceso al disco, independientemente del resto del sistema. Se definen 8 niveles de prioridad (0–7) según lo grande que es la porción de tiempo que recibirá un proceso. No permitida para usuarios no `root`.

`-c, --class` *`CLASE`* Número de la clase; `0` para ninguna, `1` para tiempo real, `2` para mejor esfuerzo, `3` para desocupado.

`-n, --classdata` *`NIVEL`* Para tiempo real y mejor esfuerzo, nivel de prioridad.

`-p, --pid` *`PID`* ID del proceso del que obtener o establecer los parámetros.

`iotop`

Muestra una tabla de uso de E/S actual de los procesos o hilos. (Las columnas son: identificador de hilo `TID`, clase/nivel de prioridad `PRIO`, propietario `USER`, tasa de transferencia de lectura `DISK READ`, de escritura `DISK WRITE`, porcentaje de tiempo de espera en fallos de página `SWAPIN`, porcentaje de tiempo de espera a peticiones de E/S a disco `IO` y línea de órdenes `COMMAND`.)

Use las flechas izquierda y derecha para cambiar la ordenación, `r` para invertir el orden (se indica `<` o `>` en la columna en la que se ordena para orden ascendente o descendente, respectivamente).

`-P` Solo muestra procesos.

`sync`

Sincroniza las escrituras a caché a almacenamiento permanente.

Apéndice E

Procesos y memoria

E.1. Gestión de procesos

Linux realiza planificación por prioridades dinámicas, con una **prioridad base** o valor *nice* que va de −20 (máxima) a 20 (mínima), donde 19 o 20 indica que al proceso solo se le da la CPU cuando nadie más la quiera. Por defecto cada proceso hereda la prioridad base de su padre, y sólo `root` puede aumentar la prioridad e un proceso. La prioridad dinámica se calcula según la prioridad base dependiendo del consumo de CPU realizado, ejecución de código en el núcleo, etc.

Un proceso pueden recibir señales como STOP (19) que lo detiene, CONT (18) que lo reanuda, KILL (9) que lo elimina (salvo si es zombie o bloqueado por E/S) o TERM (15) que le notifica de que debe terminar. Los procesos pueden **capturar** algunas señales, como TERM pero no KILL, cambiando su comportamiento por defecto al recibirlas.

`ps`

Muestra información sobre procesos activos. Acepta varios tipos de opciones:

1. Opciones UNIX, que pueden ser agrupadas y deben ser precedidas por un guion.
2. Opciones BSD, que pueden ser agrupadas y no deben ser usadas con guion.
3. Opciones largas GNU, precedidas por dos guiones.

Pueden mezclarse opciones de distintos tipos, pero puede haber conflictos. Por defecto, `ps` selecciona los procesos con igual EUID que el usuario actual y asociados a terminal.

SELECCIÓN SIMPLE DE PROCESOS

`a` Quita la restricción «sólo tus procesos».

`x` Quita la restricción «debe tener una terminal».

CONTROL DE FORMATO DE SALIDA

`l` Formato largo de BSD (con campos como `ppid`, `uid`, `pri`, `ni`).

`o` *formato* Idéntico a `-o`.

`-o` *formato* Formato definido por el usuario en forma de lista separada por comas, una forma de especificar columnas de salida (ver `ESPECIFICADORES DE FORMATO ESTÁNDARES`).

`u` Formato orientado a usuario (`user,pid,%cpu,%mem,vsz,rss,stat,tty`).

CÓDIGOS DE ESTADO DE PROCESOS

`D` Bloqueo ininterrumpible (normalmente E/S).

`R` En cola de ejecución.

`S` Bloqueo interrumpible (esperando un evento).

`T` Parado.

`Z` Zombie, terminado pero no recogido por su padre.

`<` Alta prioridad (*not nice*, < 0).

`N` Baja prioridad (*nice*, > 0).

`s` Líder de sesión. (Una sesión está asociada a un grupo de procesos y el primero de ellos es el líder. Por ejemplo, si se crean desde una terminal, el líder es el de la terminal en sí misma.)

`+` En primer plano.

ESPECIFICADORES DE FORMATO ESTÁNDARES

`%cpu` Tiempo de CPU usado entre tiempo de ejecución del proceso como porcentaje.

`%mem` Tamaño del conjunto residente del proceso entre memoria física, como porcentaje.

`euid` ID efectivo de usuario (alias `uid`).

`euser` Nombre de usuario efectivo.

`ni` Valor *nice*.

`pid` ID del proceso.

`ppid` ID del proceso padre.

`pri` Prioridad del proceso.

`rss` Tamaño del conjunto residente, la memoria física usada (en KiB).

`stat` Estado del proceso. Ver `CÓDIGOS DE ESTADO DE PROCESOS`.

`time` Tiempo acumulativo de CPU en segundos.

`tty` Terminal.

`uid` Ver `euid`.

`user` Ver `euser`.

`vsz` Tamaño de memoria virtual en kiB.

`pstree`

Muestra los procesos en ejecución como un árbol. La raíz del árbol es `init`. `pstree` une ramas idénticas poniéndolas entre corchetes y añadiendo un prefijo con el número de repeticiones. Los hilos hijos de un proceso se encuentran bajo el proceso padre con el nombre del proceso entre llaves.

`top`[1]

Muestra la actividad del procesador y procesos con una interfaz interactiva que se actualiza cada cierto tiempo (normalmente cada 3 segundos). Las 5 primeras líneas contienen:

1. `top - hh:mm:ss up DD days, HH:MM, N users, load average: l1 l5 l15`
   Hora actual, tiempo en marcha del sistema, número de terminales de usuario abiertas y carga en los 1, 5 y 15 últimos minutos en tanto por 1, con máximo igual al número de núcleos de CPU.

2. `Tasks: K total, R running, S sleeping, P stopped, Z zombie`
   Número de procesos total, en ejecución, durmiendo, parados y zombis.

3. `%Cpu(s): U us, S sy, C ni, I id, W wa, H hi, E si, T st`
   Porcentaje de CPU para modo usuario, modo núcleo (no incluye tratamiento de interrupciones), usuario *not nice* (prioridad > 0), ociosa (*idle*), ociosa esperando E/S, tratando interrupciones hardware, tratando interrupciones software y espera involuntaria (*stolen*; en una máquina virtual, el tiempo usado por la máquina real u otras virtuales que no dependen de esta; en una máquina real debería ser 0).

4. `KiB Mem : t total, f free, u used, b buff/cache`
   Memoria total disponible, usada, totalmente libre y usada en *buffers*/cachés.

5. `KiB Swap: T total, F free, U used. A avail Mem`
   Memoria de intercambio total disponible, usada, libre y estimación de la que se podría usar (incluyendo la libre, que no se usa en absoluto).

A continuación una lista de procesos, con las siguientes columnas:

`PID` ID del proceso.

`USER` Usuario.

[1] Debido a la complejidad de la página `man`, se ha optado por no basarse en ella.

`PR` Prioridad dinámica.

`NI` Prioridad base (*nice*).

`VIRT` `VSZ` de `ps`.

`RES` `RSS` de `ps`.

`SHR` Memoria que podría ser compartida con otros procesos.

`S` Estado del proceso.

`%CPU` Porcentaje de CPU usado en el intervalo de actualización.

`%MEM` Porcentaje de RAM usado.

`TIME+` `TIME` de `ps`.

Comandos interactivos:

`k` Envía una señal a un proceso.

`n` Cambia el número de procesos mostrados.

`q` Sale.

`r` Cambia la prioridad de un proceso.

`M` Ordena según `%MEM`.

`N` Según `PID`.

`P` Según `%CPU` (por defecto).

`T` Según tiempo.

`nice [`*`OPCIÓN`*`]... `*`COMANDO ...`

Ejecuta el *`COMANDO`* con el valor *nice* ajustado.

`-n` *`N`* Añade *`N`* al valor *nice*.

`renice` *`PRIORIDAD PID`*

Altera la prioridad de un proceso.

`kill [-`*`SEÑAL`*`] `*`PID ...`*

Envía la *`SEÑAL`* indicada a los procesos. Si no se especifica la señal, se envía TERM.

`killall [-`*`SEÑAL`*`] `*`NOMBRE`*

Envía una señal a todos los procesos ejecutando el comando especificado. Si no se especifica la señal, se envía TERM.

E.2. Programación de tareas

`at [`*OPCIÓN*`]... `*TIEMPO*

`at` ejecuta comandos en una fecha especificada (de esto se encarga `atd.service`).

`atq` lista los trabajos pendientes.

`atrm` elimina trabajos.

Acepta tiempos de la forma `HH:MM` para ejecutar un trabajo a un tiempo específico del día. Los comandos se leen de la entrada estándar o del fichero especificado.

`cron`

Demonio para ejecutar comandos programados (`crond.service`).

FICHEROS

`/etc/cron.d/` Directorio con ficheros `cronjob` para distintos usuarios. (El fichero `crontab` `/etc/cron.d/0hourly` está preparado para ejecutar cada hora las tareas en `/etc/cron.hourly/`, de las que ejecuta `anacron`.)

`crontab(5)`

Un fichero `crontab` contiene instrucciones para el demonio `cron`. Cada línea tiene cinco celdas de fecha y hora seguidas de un comando. Los comandos se ejecutan cuando las celdas «minuto», «hora» y «mes» coinciden con la fecha actual **y** al menos «día del mes» o «día de la semana» coinciden con la fecha actual. Las celdas son:

Minuto	0–59
Hora	0–23
Día del mes	1–31
Mes	1–12 (o nombres)
Día de la semana	0–7 (0 o 7 es domingo, o nombres)

Una celda puede contener un asterisco (`*`), que siempre significa *primero*`-`*último*. Se permiten rangos de números. Los rangos son dos números separados por un guion. El rango es inclusive.

Se permiten conjuntos de números (o rangos) separados por comas. Añadir «`/`*número*» detrás de un rango especifica saltos del valor del número.

`crontab [`*OPCIÓN*`]...`

Mantiene ficheros `crontab`.

`-l` Muestra el `crontab` actual.

`-r` Elimina el `crontab` actual.

`-e` Edita el `crontab` actual usando el editor.

`anacron`

Ejecuta comandos periódicamente. No asume que la máquina se ejecute continuamente. (Por defecto está configurado para ejecutar, diaria, semanal y mensualmente, respectivamente, los trabajos en `/etc/cron.daily/`, `/etc/cron.weekly/` y `/etc/cron.monthly/`.) Si un trabajo no se ha ejecutado en `n` días (el periodo) o más, `anacron` ejecuta el trabajo.

E.3. Memoria virtual

Linux puede usar un **fichero de paginación** o una **partición de intercambio** (*swap*) para la memoria virtual, que funciona por paginación, si bien se prefiere una partición porque no sufre fragmentación y se puede situar en la parte más rápida de un disco duro (los cilindros exteriores). El tamaño de la partición depende de factores como la memoria requerida por los trabajos, tamaño de los programas, número de trabajos simultáneos, demanda del sistema, etc. Además, para hibernar se necesita tanto espacio de intercambio como total de memoria usada. Al montar un espacio de intercambio, la opción `pri=`*`valor`*, que podemos poner en `/etc/fstab`, establece la prioridad, siendo mayor cuanto mayor es el número.

`free`

Muestra el total de memoria física y de intercambio libre y usada en el sistema. Las columnas son:

`total` Total de memoria instalada (física y de intercambio).

`used` Memoria usada (física y de intercambio, `total` − `free` − `buff/cache`).

`free` Memoria no usada (física y de intercambio).

`shared` (Física)

`buff/cache` Suma de `buffers` y `cache` (física).

`available` Estimación de memoria disponible para nuevos procesos (física).

`vmstat [`*`PERIODO`* `[`*`NÚMERO`*`]]`

Muestra estadísticas de memoria virtual.

`PERIODO` El periodo entre actualizaciones en segundos.

`NÚMERO` Número de actualizaciones.

DESCRIPCIÓN DE LAS CELDAS

Memoria (en KiBs)

`swpd` Memoria virtual usada.

`free` Memoria sin usar.

`buff` Memoria usada como *buffers*.

`cache` Memoria usada como caché.

Espacio de intercambio (*swap*)

`si` Memoria traída del disco (/s).

`so` Memoria llevada al disco (/s).

E/S

`bi` Bloques recibidos de un dispositivo de bloques (bloques/s).

`bo` Bloques enviados a un dispositivo de bloques (bloques/s).

Sistema

`in` Interrupciones por segundo, incluyendo el reloj.

`cs` Cambios de contexto por segundo.

`mkswap` *`FICHERO`*

Crea un área de intercambio Linux en un dispositivo o fichero.

`swapon`

`swapon` *`FICHERO`*`...`
`swapoff` *`FICHERO`*`...`
`swapon` especifica un dispositivo (o fichero) para intercambio. `swapoff` desactiva el uso como área de intercambio.

Apéndice F

Arranque del sistema

F.1. El *firmware*

Cuando se pulsa el botón de encendido, se generan señales eléctricas que hacen que en los registros del procesador y de otros componentes se carguen valores predeterminados. En el procesador, el contador de programa se carga con la dirección del ***firmware***, un trozo de código almacenado normalmente en memoria no volátil de lectura y escritura (EEPROM, flash, etc.) que se encarga, entre otras cosas, de preparar el hardware.[1]

Los principales tipos de *firmware* son BIOS, que suele aparecer en ordenadores anteriores a 2010, y UEFI, que aparece en los más recientes[2]. Este busca en los dispositivos de almacenamiento un **cargador** o **gestor de arranque**. Si no lo encuentra, muestra un error. Si lo encuentra, lo copia en memoria y lo ejecuta, y este pasa a leer de disco el sistema operativo y, posiblemente, otros componentes necesarios para que este arranque, para finalmente pasarle el control.

En los sistemas BIOS, el *firmware* detecta los dispositivos de almacenamiento y toma el primero según una secuencia de arranque dada, copia su sector 0 (el MBR) a disco y pasa el control a lo que acaba de leer. Si este no tiene código de arranque, toma elfich

En los UEFI se usa particionamiento GPT, y la **partición del sistema de UEFI**, que en Linux suele montarse en `/boot/efi` y que contiene un sistema de ficheros FAT y GUID `C12A7328-F81F-11D2-BA4B-00A0C93EC93B`, contiene directorios por cada sistema operativo con todo lo necesario para la puesta en marcha: Windows usa `/EFI/Microsoft` y Fedora usa `/EFI/fedora`. También suele haber un **módulo de soporte de compatibilidad** (*Compatibility Support Module*, CSM), que simula un entorno BIOS para poder arrancar sistemas operativos sin soporte UEFI. El *firmware* almacena una entrada apuntando a cada fichero de arranque. El *firmware* busca en cada sistema de almacenamiento disponible con partición del sistema UEFI, en orden configurable, un gestor de arranque funcional, de entre los dados por una lista ordenada.

En 2011, con la versión 2.3.1 de la especificación UEFI aparece *Secure Boot*, que sólo

[1] También suelen ocurrir más cosas, como un *Baseboard Management Controller* (BMC) o un *Intel Management Engine* (IME). Este último componente se encarga (v. CVE-2017-5689 y otros) de que la NSA (que lo tiene desactivado por petición especial) y otros puedan espiarte.

[2] Y lo jode todo.

permite arrancar a software firmado con una clave reconocida por el *firmware*. En los equipos certificados para Windows 8, las claves son proporcionadas por Microsoft y no son públicas. Dos opciones son desactivar *Secure Boot* en la configuración o usar un cargador firmado con una clave de Microsoft.

F.2. GRUB 2

El *GRand Unified Bootloader* versión 2 es un gestor de arranque que normalmente se usa para cargar el núcleo de Linux, si bien puede cargar otros sistemas operativos.

En BIOS, consta de 3 fases dadas por **imágenes**, que en Fedora se almacenan en `/usr/lib/grub/i386-pc` y en `/boot/grub2/i386-pc`. Algunas son:

`boot.img` Se escribe en el MBR, ocupa 512 B y se encarga de cargar y ejecutar la segunda fase.

`diskboot.img` Segunda fase y primer sector de la imagen *core*, se encarga de cargar y ejecutar el resto de la imagen *core*.

`cdboot.img` Como `diskboot.img` pero para CD/DVD.

`kernel.img` Contiene utilidades en tiempo de ejecución básicas de GRUB 2, y en general se integra en la imagen *core*.

`core.img` Imagen principal, construida dinámicamente a partir de `diskboot.img`/`cdboot.img`, `kernel.img` y una lista de módulos. Normalmente contiene módulos suficientes para acceder a la configuración y el resto de módulos (ficheros `.mod`) en el sistema de ficheros.

Las imágenes `*boot.img` no entienden ningún sistema de ficheros, por lo que las dos últimas fases se copian a la zona del disco detrás del MBR y antes de la primera partición, y se almacena el sector de comienzo de cada imagen y su tamaño.

En UEFI, el código de GRUB 2 se encuentra en `grubx64.efi`, con formato Windows PE, y generalmente contiene todo lo necesario, si bien sigue siendo posible cargar módulos dinámicamente. Fedora usa el fichero `sham.efi`, firmado por Microsoft, que se encarga de instalar una clave propia y ejecutar GRUB 2, que, al igual que el núcleo de Linux, estará firmado por esta clave.

GRUB 2 tiene 3 interfaces:

- Menú: Permite elegir el sistema operativo entre varias opciones.
- Editor: Permite editar una entrada de menú, y se accede seleccionando la entrada y pulsando `e`. Podemos usar `F10` para arrancar con los cambios o `Esc` para descartarlos y volver al menú.
- Línea de órdenes: Permite ejecutar órdenes de GRUB 2, y se accede pulsando `c` desde el menú o `F2` desde el editor.

Los dispositivos se identifican como

{tipo de dispositivo}{número de dispositivo BIOS},[particionamiento]{número de partición}

Los números de dispositivo empiezan por 0 y los de partición por 1, y los discos duros se denotan `hd`. Por ejemplo, `hd0,1` es la primera partición del primer disco duro, y escribimos `hd0,msdos1` si queremos indicar además que el tipo de particiones es MS-DOS. También podemos especificar un fichero como *dispositivo*`/`*ruta*, donde la *ruta* parte de la raíz del *dispositivo*.

Fedora almacena la configuración de GRUB 2 en `/boot/grub2/grub.cfg`. Esta se genera a partir de ficheros en `/etc/grub.d`, que se ejecutan en orden alfabético si tienen permisos de ejecución, y en Fedora son:

`00_header` Configura el modo gráfico, la entrada por defecto, la pausa antes de arrancar esta, etc., configuración que normalmente procede de `/etc/default/grub`.

`10_linux` Identifica los núcleos de Linux instalados en el dispositivo de arranque (normalmente montado en `/boot`) y les crea entradas.

`30_os-prober` Usa `os-probler` para buscar e instalar otras distribuciones de Linux y otros sistemas operativos, y se puede deshabilitar en `/etc/default/grub`.

`40_custom` Se usa para añadir entradas personalizadas, pues su segunda línea es `exec tail -n +3 $0`.

Los ejecutables, normalmente guiones *shell*, imprimen su trozo de configuración en la salida estándar, y `grub2-mkconfig` añade `### BEGIN` *ruta* `###` antes de la salida del guión con la *ruta* dada y `### END` *ruta* `###` al final.

`/etc/default/grub` especifica opciones, en formato *CLAVE*`=`*valor*, que establecen cómo estos guiones generan las entradas de menú:

`GRUB_DEFAULT` Fija la entrada por defecto: un número (empezando por 0), el nombre de la entrada entre comillas o `saved`, que permite que `grub2-set-default` y `grub2-reboot` establezcan la entrada por defecto.

`GRUB_SAVEDEFAULT` Si está definida y vale `true`, cada vez que se seleccione una entrada del menú, se convierte en la entrada por defecto. Para que funcione debe ser `GRUB_DEFAULT=saved`.

`GRUB_TIMEOUT` Plazo en segundos para seleccionar una entrada de menú antes de arrancar la entrada por defecto. Si vale `-1`, el plazo nunca expira.

`GRUB_CMDLINE_LINUX` El contenido de esta opción se añade al final de los parámetros del núcleo de Linux.

`GRUB_DISABLE_RECOVERY` Si vale `true`, evita que, para cada núcleo de Linux instalado, aparezca una segunda entrada en la que al núcleo se le pasan parámetros para una posible recuperación del sistema.

En `grub.cfg`, las entradas tienen la forma `menuentry '`*Nombre*`' [`*OPCIÓN*`]... { `*COMANDO*`... }`, donde algunas opciones son:

`--class` *clase* Permite agrupar las entradas en clases.

`--unrestricted` Cuando la seguridad está activada, esta opción se puede arrancar sin restricciones.

Los comandos se separan por saltos de línea:

`insmod` *`módulo`* Carga un módulo de GRUB 2, para ampliar su seguridad. Algunos son `gzio` (para descomprimir `gzip`), `part_msdos` (para acceder a particiones MS-DOS), `ext2` (para leer sistemas de ficheros ext2/3/4), `chain` (para cargar otros gestores de arranque, *Chain-loading*), `ntfs` y otros para entender XFS, FAT, etc. Si GRUB 2 puede acceder al fichero de configuración, muy probablemente ya tenga cargados `part_msdos` y `ext2`, necesarios para acceder al fichero.

`set` *`nombre='valor'`* Establece una opción. La opción `root` indica la partición con los ficheros de arranque: el núcleo de Linux (`vmlinuz`) y su disco RAM (`initramfs`).

`search --no-floppy --fs-uuid --set=root ...` Busca en todos los discos, salvo disquetes, una partición con un sistema de ficheros con cierto UUID, y si la encuentra la establece como de arranque. Se usa ante posible fallo de la anterior, pues la partición de arranque puede cambiar de nombre, por ejemplo, si se inserta un nuevo disco.

`linux, linux16, linuxefi` *`RUTA`* `[`*`OPCIÓN`*`]...` Indica la ruta del núcleo de Linux en la partición `root` y las opciones que se le pasarán para el arranque. Por ejemplo, `mem=`*`N`* indica al núcleo que sólo ha de usar cierta cantidad de RAM, donde *`N`* puede tener prefijos como `G` para GiB. Otros parámetros son para el proceso de inicio (el primero en ejecutarse), que los recibe a través del núcleo.

`initrd, initrd16, initrdefi` Indica la ruta del disco RAM de Linux en la partición `root`, un disco que se carga en RAM y que Linux monta tras inicializarse a sí mismo, y es de donde obtiene lo necesario para montar el sistema de ficheros raíz. Una vez este se monta, el disco RAM no es necesario y se elimina.

`chainloader +1` Carga y ejecuta el primer sector de una partición; usado normalmente para cargar Windows, el cual no reconoce otros sistemas operativos instalados y al instalarse en sistemas BIOS, modificará el MBR con su propio gestor de arranque que no da opción a arrancar Linux.

La seguridad de GRUB 2 se activa con `grub2-setpassword`, que pide una contraseña y hace que, a partir del siguiente arranque, para modificar una entrada sea necesario introducir el nombre de usuario `root` y la contraseña dada.

`grub2-install` *`DISPOSITIVO`*

Instala GRUB en un dispositivo. Esto incluye copiar las imágenes de GRUB en el directorio destino (en Fedora, `/boot/grub2`) e instalar GRUB en un sector de arranque.

`grub2-mkconfig`

Genera un fichero de configuración de GRUB.

`-o` *`FICHERO`* Escribe la salida generada al *`FICHERO`*.

`grub2-set-default` *ENTRADA*

Establece la entrada por defecto del menú de GRUB. Sólo funciona para ficheros de configuración de GRUB creados con `GRUB_DEFAULT=saved` en `/etc/default/grub`.

`grub2-reboot` *ENTRADA*

Como `grub2-set-default`, pero sólo para el siguiente arranque.

`grub2-setpassword`

Genera el fichero con la contraseña cifrada de GRUB.

F.3. Parámetros del núcleo de Linux

`kernel-command-line`

`root=` Sistema de ficheros raíz.

`dracut.cmdline`

`init=` Ruta del programa de inicio a ejecutar tras terminar con el disco RAM.

F.4. Arranque de Linux

En Fedora y otras muchas distribuciones, el arranque y parada lo realiza `systemd`, el primer proceso en modo usuario tras terminar con el disco RAM, y que tiene PID 1.

Reemplaza al demonio `init` de los sistemas UNIX System V, que se basaba en guiones *shell*, y aprovecha muchas características de Linux como *cgroups*, *namespaces*, etc. Fedora 15 fue una de las primeras distribuciones en usar `systemd`, que por entonces convivía con un gran número de guiones de System V. Ventajas de `systemd`:

- El arranque y parada pasa a ser en paralelo, aprovechando los sistemas multinúcleo.
- La mayoría de dependencias pasan a establecerse automáticamente.
- Los *cgroups* (*control groups*, grupos de control) permiten limitar, contabilizar y aislar el uso de recursos por parte de un grupo de procesos. Como un proceso hijo pertenece obligatoriamente al mismo *cgroup* que su proceso padre (salvo procesos privilegiados), es posible usar esto para asegurarse de que los servicios realmente terminan.
- Permite ejecutar servicios bajo demanda (cuando se necesiten), si bien algunos servicios están activos siempre para dar una respuesta rápida.
- Permite montar sistemas de ficheros solo cuando se necesitan, comprobar su consistencia y activar sus cuotas de disco. Para ello usa `autofs`, que permite montar sistemas de ficheros bajo demanda (por ejemplo, al acceder a un CD, DVD, *pendrive* o sistema de ficheros remoto) y desmontarlo si lleva un tiempo sin usarse. Se debe añadir

`x-systemd.automount` en las opciones de `/etc/fstab` (y generalmente también `noauto`) para activar esta opción.

Para más información, ver `http://0pointer.de/blog/projects/systemd.html`.

`systemd`

Proporciona un sistema de dependencias entre **unidades** de 11 tipos:

1. **Unidades de servicio** (*service*), que comienzan y controlan los demonios y procesos de los que consisten. `cups.service` controla el demonio de impresión, y `avahi.service` permite el acceso a ciertos servicios de red.
2. **Unidades de objetivo** o **destino** (*target*): Para agrupar unidades, o proveer puntos de sincronización durante el arranque y parada, incluyendo el estado final. Cuando se llega a `graphical.target`, el objetivo por defecto en Fedora, el sistema queda en modo multiusuario con entorno gráfico.

La mayoría de dependencias las crea y mantiene `systemd` automáticamente.

LÍNEA DE COMANDO DEL NÚCLEO

`systemd.unit=`*`UNIDAD`* Para arrancar a una unidad distinta.

`systemd.unit`

`/etc/systemd/system/*`
`/lib/systemd/system/*`
Esta página lista las opciones de configuración comunes a todos los tipos de unidades.

OPCIONES DE SECCIÓN `[UNIT]`

`Requires=` Si esta unidad se activa, las unidades listadas aquí se activarán también. Si una de las otras falla, y hay una dependencia de orden `After=` en la unidad fallida, esta unidad no se iniciará.

`Wants=` Las unidades listadas en esta opción se iniciarán si se inicia esta. Su inverso es `WantedBy=`. Si una unidad `foo` contiene una opción `WantedBy=bar`, activar `foo` creará un enlace simbólico en el directorio `bar.wants` para expresar esta dependencia. Si contiene `Wants=bar`, el enlace se crea en `foo.wants`.

`Conflicts=` Configura dependencias negativas. Si una unidad tiene `Conflicts=` sobre otra, empezar la primera detendrá la segunda y viceversa.

`Before=, After=` Si una unidad `foo` contiene una opción `Before=bar` y ambas se inician, el inicio de `bar` esperará a que `foo` haya terminado de iniciarse. `After=` es el inverso de `Before=`.

OPCIONES DE SECCIÓN `[INSTALL]`

`WantedBy=` Se crea un enlace simbólico en el directorio `.wants/` de cada unidad listada. Esto crea una dependencia de tipo `Wants=` de la unidad listada a la actual.

systemd.service

Unidad de servicio. Las siguientes dependencias se añaden salvo que se establezca `DefaultDependencies=no`:

- Dependencia de tipo `After=` con `basic.target`, así como de tipo `Conflicts=` y `Before=` con `shutdown.target`.

systemd.special

`default.target` El que se inicia en el arranque. Normalmente un enlace simbólico a `multi-user.target` o `graphical.target`.

`emergency.target` Inicia una *shell* de emergencia. No toma ningún servicio o punto de montaje (`/` está en sólo lectura).

`graphical.target` Inicio de sesión gráfico. Toma `multi-user.target`.

`multi-user.target` Establece un sistema multiusuario (no gráfico).

`poweroff.target` Apaga el sistema.

`reboot.target` Reinicia el sistema.

`rescue.target` Toma el sistema base (incluyendo puntos de montaje) y lanza una *shell* de rescate. Para administrar el sistema en modo de un usuario sin servicios en ejecución, salvo los más básicos (tampoco la conexión de red).

systemd-analyze

`systemd-analyze critical-chain` imprime un árbol de la cadena de unidades con tiempo crítico. El tiempo tras el cual la unidad está activa se imprime tras el caracter «`@`».

`systemd-analyze plot` imprime un gráfico SVG con detalles de qué servicios se han iniciado en qué tiempo.

systemctl [*OPCIÓN*]... [*COMANDO*] [*UNIDAD*]...

`-t, --type=`*`TIPO`* Al listar las unidades, limitarse a ciertos tipos de unidad.

`-p` *`Propiedad`* Al mostrar las propiedades de una unidad con `show`, limitarse a la indicada.

`--kill-who=`*`PROCESO`* Al usarse con `kill`, elige a qué procesos enviar una señal. `main` la envía sólo al proceso principal.

`-s` Al usarse con `kill`, elige la señal a enviar a los procesos. Por defecto es `SIGTERM`, que avisa a los procesos de que deben terminar. `SIGKILL` fuerza la terminación. `SIGHUP` (o `HUP`), al enviarse a `crond.service`, le indica que debe releer su configuración.

COMANDOS

Comandos de unidad

`list-units` Lista las unidades que tiene `systemd` en memoria. Comando por defecto.

`start` *`UNIDAD`* Inicia una unidad.

`stop` *`UNIDAD`* Detiene una unidad.

`isolate` *`UNIDAD`* Inicia la unidad especificada y sus dependencias y detiene todas las demás.

`kill` *`UNIDAD`* Envía una señal a procesos de una unidad (por defecto a todos).

`status [`*`UNIDAD`*`]` Muestra información de estado resumida sobre una unidad, o sobre el sistema si no especifica la unidad.

`show` *`UNIDAD`* Muestra las propiedades de una unidad.

Comandos de ficheros de unidad

`enable` *`UNIDAD`* Habilita una unidad. La añade en varios sitios sugeridos (por ejemplo, para que se inicie al arrancar).

`disable` *`UNIDAD`* Deshabilita una unidad.

`is-enabled` *`UNIDAD`* Comprueba si una unidad está habilitada.

`mask` *`UNIDAD`* Enmascara una unidad, haciendo imposible iniciarla.

`unmask` *`UNIDAD`* Desenmascara una unidad.

`get-default` Devuelve el objetivo de arranque por defecto.

`set-default` *`OBJETIVO`* Establece el objetivo de arranque por defecto.

Comandos de sistema

`rescue` Entra en modo rescate. Equivale a `systemctl isolate rescue.target`.

`halt` Para el sistema. Es casi equivalente a `systemctl start halt.target`.

`poweroff` Apaga el sistema. Es casi equivalente a `systemctl start poweroff.target.`

`reboot` Reinicia el sistema. Es casi equivalente a `systemctl start reboot.target`.

/etc/nologin

Si existe y es legible, `login` solo permitirá el acceso a `root`. Se borra automáticamente al reiniciar el sistema.

`journalctl [`*OPCIÓN*`]... [`*PATRÓN*`]...`

Muestra el contenido del registro de `systemd`. Sin parámetros, muestra todo el registro, empezando por la entrada más antigua (cuando se arrancó por primera vez, si bien la cantidad de mensajes que se pueden guardar está limitada para no ocupar mucho espacio).

`-n` *N* Muestra los eventos más recientes y limita el número de eventos mostrados.

`-b [`*ID*`][±`*desplazamiento*`]` Muestra los mensajes de un arranque específico. Si se omite el ID, un *desplazamiento* menor o igual a 0 mostrará los arranques desde el final del registro. Si se indica el *ID*, puede ser seguido opcionalmente por el *desplazamiento*. Valores negativos significan arranques anteriores y positivos significan posteriores.

`--list-boots` Muestra una lista de todos los arranques (relativo al actual), sus IDs y las fechas del primer y último mensaje del arranque.

`-k` Muestra sólo mensajes del núcleo.

`-u` *UNIDAD*`|`*PATRÓN* Muestra los mensajes para la *UNIDAD* dado, o para cada unidad que coincide con el *PATRÓN*.

F.5. Dependencias automáticas

Hay varias formas de establecer dependencias entre unidades aparte de las descritas:

- Un *socket* tipo UNIX es un canal de comunicación entre dos procesos en el que un proceso envía información por un extremo y el otro la extrae por el otro. Al igual que una tubería, tiene una capacidad máxima, con lo que si un proceso intenta leer de un *socket* vacío o escribe en uno lleno, se bloquea. Si dos servicios se comunican por un *socket*, `systemd` crea el *socket* y luego lanza los dos servicios en paralelo, pasándoles como parámetro el *socket* creado a través del cual se sincronizarán.
- La sincronización por mensajes de D-Bus funciona de forma similar, pero el servicio que depende de otro espera a que en el bus aparezca el proveedor que necesita, que será ofrecido por el otro servicio, y D-Bus proporciona la funcionalidad necesaria para esto.

F.6. `udev`

El *userspace device manager* es un servicio (demonio `systemd-udevd`) que, cuando se conecta o desconecta un dispositivo, recibe un evento del núcleo, usa información de `/sys` para ver si el *driver* proporciona un fichero `dev` con los números mayor y menor del fichero especial a crear y usa **ficheros de reglas** (*rules*) en `/lib/udev/rules.d/` y `/etc/udev/rules.d/` para:

- Crear el fichero especial de dispositivo en `/dev` y los enlaces simbólicos al mismo.
- Establecer los permisos y propietarios de este fichero.
- Si es necesario, cargar los módulos y *firmware* para poder usarlo.
- Si alguna regla lo indica, ejecutar los programas especificados para inicializar el dispositivo.

F.7. Parada de Linux

Se notifica a los usuarios, se envía la señal `SIGTERM` a los procesos, se paran los demonios, se echa del sistema a los usuarios que quedan conectados, se envía la señal `KILL` a los procesos que queden en ejecución, se sincronizan los discos, se desmontan los sistemas de ficheros y, según el tipo de parada, se apaga o se reinicia el sistema.

`halt`

`halt`, `poweroff`, `reboot` pueden usarse para detener, apagar o reiniciar la máquina (respectivamente).

`shutdown [`*OPCIÓN*`]... [`*TIEMPO*`] [`*MENSAJE*`]...`

Detiene, apaga o reinicia la máquina.

El tiempo puede estar en formato *hh*`:`*mm* indicando el tiempo en el que ejecutar el apagado, en formato de 24 horas. La sintaxis `+`*m* se refiere al número indicado de minutos desde ahora. `now` es un alias para `+0`. Si no especifica el tiempo, se usa `+1`.

`-H` Detiene la máquina.

`-P` Apaga (por defecto).

`-r` Reinicia.

`-h` Equivale a `-P` salvo que se indique `-h`.

`-k` Sólo escribe el *MENSAJE*.

`-c` Cancela un apagado en curso (sólo si no ha empezado).