T E M A 5

Operativo UNIX

Contenido

TEMA 5

INTRODUCCIÓN AL SISTEMA OPERATIVO UNIX

5.1. Introducción

Aunque el sistema operativo más empleado es MS-DOS, UNIX destaca por ser utilizado en sistemas que van desde computadores personales hasta supercomputadores. El interés de su estudio radica, además, en que muchos principios de diseño en sistemas operativos se aplican en él.

5.1.1. Historia

Los orígenes de UNIX se remontan a finales de los años 60, cuando los laboratorios Bell, de AT&T, participan en el desarrollo de un nuevo sistema operativo llamado Multics. Este sistema fracasó, entre otros motivos, porque estaba diseñado para soportar cientos de usuarios con un hardware similar al de un computador personal PC/AT. Era un sistema demasiado avanzado para su época.

Uno de los ingenieros de los laboratorios Bell que habían trabajado en Multics, Ken Thompson, escribió una versión reducida de Multics sobre una computadora PDP-7 que se encontraba en desuso, dando lugar a lo que fue la versión primitiva de UNIX. Su trabajo fue bien considerado y Dennis Ritchie comenzó a colaborar con él.

En esos momentos ocurrieron dos acontecimientos que fueron el origen de la difusión de UNIX. En primer lugar, el sistema fue llevado a sistemas PDP-11, que eran más avanzados que el obsoleto PDP-7. Estos sistemas fueron los dominantes en el mundo de los minicomputadores en la década de los 70. En segundo lugar, para que el sistema fuera portable, Thomson decidió escribirlo en un lenguaje de alto nivel que él mismo diseñó llamado B. Este lenguaje tenía muchas limitaciones (era interpretado y no tenía tipos de datos estructurados) y Ritchie diseñó un sucesor de B, llamado C, y escribió un compilador para él. En 1973, el sistema operativo fue reescrito en C.

En esta época, AT&T no podía comercializar productos relacionados con la informática, por lo que se decidió que UNIX podía ser cedido, con el código fuente incluido, a las universidades que lo solicitaban para uso educacional. No se ofrecía ningún tipo de soporte técnico. Por otro lado, el PDP-11 era el sistema más empleado en la mayoría de los departamentos de computación de las universidades. Este cúmulo de circunstancias originó la rápida difusión de UNIX.

UNIX fue llevado a nuevas máquinas, pero su simplicidad y claridad hizo que muchas empresas lo mejorasen a su modo, lo que originó versiones diferentes de la original. En el periodo de 1977 a 1982 los laboratorios Bell suministraron una versión comercial conocida como UNIX system III. La siguiente versión fue la UNIX system V, y AT&T comenzó a ofrecer soporte técnico en enero de 1983. Sin embargo, en la universidad de California en Berkeley se había desarrollado una variante de UNIX cuya última versión, 4.3BSD, contenía muchas mejoras (memoria virtual, gestión de redes, utilidades como vi, csh, etc.). Como consecuencia, muchos fabricantes basaron su versión de UNIX en 4.3BSD en lugar de la versión oficial de AT&T, la system V.

A finales de los 80, las dos variantes anteriores, incompatibles entre sí, eran las más empleadas. Como, además, cada fabricante introducía sus propias mejoras, el resultado es que el mundo de UNIX estaba dividido en dos y no existían normas para los formatos de los códigos binarios de los programas, lo que limitaba el éxito comercial de UNIX, ya que era prácticamente imposible que un programa desarrollado en una versión fuese ejecutable en otra. El primer intento de aunar las dos versiones fue auspiciado por IEEE y se denominó POSIX (Portable Operating System), que partió de una intersección de las dos versiones. Sin embargo, para evitar que AT&T tomase el control de UNIX, un grupo de fabricantes (IBM, DEC, Hewlett-Packard, ...) formó un consorcio llamado OSF (Open System Foundation) para producir un nuevo sistema que estuviese conforme con las normalización de IEEE y que contuviese, además, un elevado número de nuevas características, como un sistema de ventanas (X11), un interfaz gráfico de usuario (MOTIF), etc. Por su parte,

AT&T organizó su propio consorcio, UI (Unix International), con fines similares. UI desapareció en 1993, AT&T vendió UNIX a la empresa Novell y algunos miembros de UI se incorporaron a OSF. En 1995, Novell vendió los derechos de Unix a la empresa Santa Cruz Operation.

En la actualidad, sigue sin haber una versión única de UNIX. Incluso dentro de OSF existen varias versiones: AIX de IBM, Ultrix de DEC, HP-UX de HP, Solaris y SunOs de Sun, etc.).

5.1.2. Arbol genealógico de Unix

En la siguiente figura se muestra un árbol genealógico de UNIX.

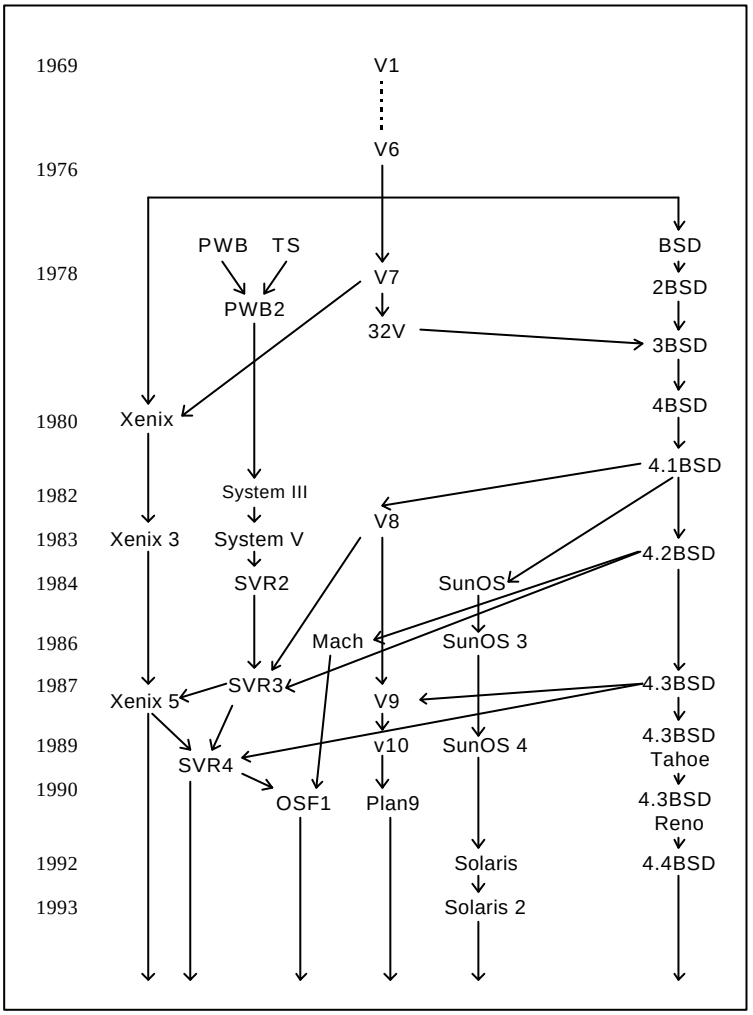

Figura 1.- Árbol genealógico de UNIX.

5.1.3. Principios de diseño

UNIX fue diseñado para ser un sistema operativo interactivo, multiusuario y multitarea:

- **Interactivo** quiere decir que el sistema acepta órdenes, las ejecuta y se dispone a esperar otras nuevas.
- **Multitarea** significa que puede realizar varios trabajos, denominados procesos, al mismo tiempo.
- **Multiusuario** significa que más de una persona puede usar el sitema al mismo tiempo.

UNIX fue diseñado por programadores para ser usado por programadores en un entorno en que los usuarios son relativamente expertos y participan en el desarrollo de proyectos de software.

Es común que varios programadores cooperen activamente para realizar un producto software, por lo que UNIX ofrece facilidades que permiten a los usuarios el trabajar en equipo y el compartir información de forma controlada. Este enfoque es muy distinto al de un sistema operativo de un computador personal, en el que un usuario no necesariamente experto en informática trabaja con un procesador de textos. Así, existen

herramientas para el desarrollo de programas como `make`, que permite sacar partido de las posibilidades que ofrece la compilación separada, y el sistema de control de código fuente (SCCS, Source Code Control System), que se usa para el mantenimiento de las diferentes versiones de los programas durante el desarrollo de los mismos.

Para un programador experto, un sistema operativo debe ser, como mínimo, simple y consistente. Así, por ejemplo, los ficheros son meras secuencias de bytes. Por otro lado, si la orden `ls A*` muestra por pantalla todos los ficheros cuyos nombres comienza por `A`, la orden `rm A*` debe de borrar todos los ficheros cuyos nombres empiezan por `A` y no el único fichero cuyo nombre es `A*`.

Un programador experto también quiere que un sistema operativo sea flexible y potente. Esto significa que el sistema debe de ofrecer un conjunto pequeño de elementos básicos que se puedan combinar de muchas maneras para construir una aplicación más compleja. Una de las líneas básicas de diseño de UNIX es que todo programa debe de hacer únicamente una cosa y hacerla bien.

Por último, a los programadores no les gusta escribir más de lo necesario. Por este motivo, las órdenes del sistema son muy simples (`cp` en lugar de `copy`, `rm` en lugar de `remove`, `du` en lugar de `disk_usage`, etc.).

5.1.4. Estructura de UNIX

La estructura de UNIX se amolda a un típico modelo en capas, de forma que cada capa únicamente puede comunicarse con las capas que se hallan en los niveles inmediatamente inferior y superior (ver Figura 2).

El núcleo (*kernel*) del sistema interactúa directamente con el hardware y proporciona una serie de servicios comunes a los programas de las capas superiores, de forma que las peculiaridades del hardware permanecen ocultas. Como los programas son independientes del hardware, es fácil (en teoría) mover programas entre sistemas UNIX que se ejecutan en hardware diferente. Programas como el intérprete de comandos (*shell*) `sh` y el editor `vi` interactúan con el kernel a través de llamadas al sistema, que permiten el intercambio de información entre el kernel y los programas.

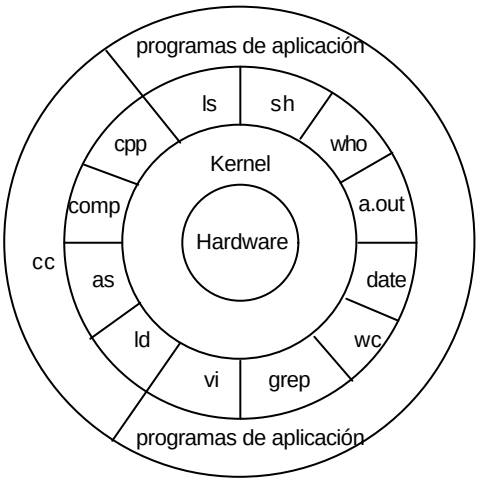

Figura 2.- Estructura de UNIX.

5.2. UNIX desde el punto de vista del usuario

UNIX se compone de tres partes principales: el núcleo (`kernel`), el sistema de ficheros y el intérprete de comandos (`shell`). Estas dos últimas son visibles al usuario, mientras que el kernel permanece oculto.

- El kernel es la parte del sistema que gestiona los recursos del computador (el hardware).
- El sistema de ficheros organiza la información en disco y es de tipo jerárquico.
- El shell es un programa de utilidad que permite al usuario comunicarse con el sistema.

denes es el siguiente:

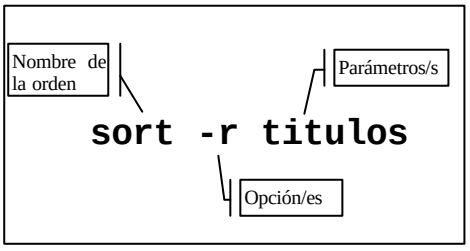

Figura 3.- Formato de las órdenes en UNIX..

Cuando el shell lee una línea de comandos, extrae la primera palabra, asume que ésta es el nombre de un programa ejecutable, lo busca y lo ejecuta. El shell suspende su ejecución hasta que el programa termina, tras lo cual intenta leer la siguiente línea de órdenes.

Las opciones modifican el comportamiento normal de la orden y se suelen indicar justo después del nombre de la orden y con el prefijo `-`. Si existen varias opciones se pueden agrupar. Por ejemplo, las orden `ls -a -l -s` es equivalente a `ls -als`. Se pueden escribir varias órdenes en una misma línea escribiendo un punto y coma entre ellas.

Los parámetros aportan información adicional que será usada por la orden. Los más habituales son nombres de ficheros y nombres de directorios, pero no tiene por qué ser siempre así.

La mayoría de las órdenes aceptan múltiples opciones y argumentos. Por ejemplo, la orden `ls a.c b.pas a.out` mostrará los tres ficheros que han pasado como parámetros (si existen). Esto ha de ser tenido en cuenta en algunas ocasiones, ya que de lo contrario pueden aparecer comportamientos

inesperados. Por ejemplo, en la orden `head -20 hola.c`, la opción -20 indica a la orden `head` que imprima las primeras 20 líneas de `hola.c`, en lugar del número por defecto que es 10. Sin embargo, si se `head 20 hola.c`) la orden es válida e intenta imprimir las diez primeras líneas de un fichero llamado 20 y las 10 primeras líneas del fichero llamado fichero `hola.c`. Si bien en este caso la situación no es grave, en otros casos puede dar lugar a sucesos más graves. Supongamos que queremos `.c` mediante la orden `rm *.c`. Si nos equivocamos y escribimos por error `rm * .c` (nótese el espacio en blanco entre el `*` y el punto), la orden `rm` borrará todos los ficheros y después dará un mensaje de error indicando que el fichero cuyo nombre es `.c` no existe.

UNIX es un sistema multitarea, por lo que un usuario puede ejecutar más de un programa a la vez. Si al final de un orden se escribe el símbolo `&`, se le indica al shell que la orden se ejecute en segundo plano (*background*). Una vez introducida la línea de órdenes, el shell muestra el prompt para recibir una nueva orden. Por ejemplo, `ls -al > listado &` almacenará en el fichero *listado* el contenido del directorio actual, pero lo hará en segundo plano, por lo que podremos lanzar nuevas órdenes mientras ésta se ejecuta.

El shell, como la mayoría de las órdenes de UNIX, emplean tres ficheros denominados "estándar". Cuando un programa inicia su ejecución tiene acceso de forma automática a tres ficheros llamados entrada estándar, salida estándar y error estándar (salida de errores). En UNIX los dispositivos se tratan como ficheros, por lo que estos tres ficheros, si no se indica lo contrario, son dispositivos. Concretamente, la entrada estándar es el teclado, mientras que la salida y error estándar es el monitor. Por ejemplo, la orden `sort`, sin parámetros, lee líneas del terminal hasta que el usuario pulsa CTRL-D para indicar fin de fichero, las ordena alfabéticamente y escribe el resultado por pantalla.

UNIX proporciona un mecanismo sencillo para cambiar la entrada y salida estándar. Este mecanismo se denomina **redirección de E/S**. También es posible encadenar varias órdenes de forma que la salida estándar de una orden se dirija hacia la entrada estándar de la siguiente. Esta característica se denomina (pipeline).

5.2.2.1. Redirección de salida estándar

Si el último parámetro de una orden es un nombre de fichero precedido por el carácter `>` (mayor que), la salida estándar de esa orden de redirige hacia ese fichero en lugar de aparecer por la pantalla. Si el fichero no existe, se crea. Si existe, se eliminará su contenido y se reemplazará por la salida de la orden. Para evitar este último caso, se puede redireccionar mediante los caracteres `>>`, de forma que la salida de la orden se concatena con el contenido anterior del fichero.

Ejemplos:

`ls -al > listado`	Redirige la salida de la orden `ls` al fichero *listado*.
`cat fich1 fich2 > fich3`	Almacena en *fich3* la concatenación de *fich1* con *fich2*
`cat fich1 fich2 >> fich3`	Añade al final de fich3 los contenidos de *fich1* y *fich2*

5.2.2.2. Redirección de entrada estándar

Para redirigir la entrada estándar se usa en signo `<` (menor que) seguido del fichero de entrada. Por ejemplo, `cat < fich1` mostraría por pantalla el contenido de *fich1*. En este caso se obtendría el mismo resultado mediante `cat fich`, ya que la orden *cat* muestra por pantalla el contenido de los ficheros que se le pasan como parámetros.

Es posible redireccionar la entrada y la salida en una mismo orden. Por ejemplo, `sort < fich1 > fich2` ordena el contenido del fichero `fich1` y escribe el resultado en `fich2`.

5.2.2.3. Cauces o tuberías (pipes)

Mediante redireccionamiento es posible encadenar una orden con un fichero. Mediante cauces o tuberías (pipes) se pueden encadenar varias órdenes. Para ello se emplea el símbolo `|`. Por ejemplo, la orden `who` visualiza una lista de los nombres de los usuarios que se hallan conectados al sistema y la orden `wc` cuenta líneas, palabras y caracteres.

Para saber cuántos usuarios hay conectados se puedan encadenar las órdenes de la siguiente forma: `who | wc -l` (la opción `-l` hace que `wc` cuente únicamente líneas). Otro ejemplo: `grep ter *.t | sort | head -20 | tail -5 > f_sal` selecciona todas las líneas que contienen la cadena "`ter`" de aquellos ficheros cuyos nombres terminan en `.t`, las ordena, selecciona las veinte primeras, de éstas toma las cinco últimas y las escribe en `f_sal`.

Las órdenes que toman su entrada de la entrada estándar y su salida de la salida estándar se denominan **filtros**. Son órdenes que toman información de entrada, realizan alguna acción de sobre las mismas, y produce alguna salida. La mayoría de las órdenes de UNIX son filtros, por lo que pueden usarse como operaciones intermedias en un cauce. Algunas órdenes no son filtros, ya que es posible que su entrada o `ls` y `lp` no son filtros, ya que la primera tiene como entrada un directorio y la segunda tiene como salida la impresora. En estos casos, `ls` sólo podrá estar al comienzo de un cauce y `lp` al final.

5.2.2.4. Metacaracteres

El shell ofrece la posibilidad de utilizar una notación abreviada para operar sobre conjuntos de ficheros y directorios en una única orden. Esto se consigue mediante el uso de algunos caracteres que tienen significados especiales. Estos caracteres se denominan metacaracteres y se emplean para establecer una correspondencia con nombres o partes de nombres de ficheros. Los más empleados son:

- `*`: representa cualquier cadena de caracteres, incluyendo la cadena vacía.
- `?`: representa a cualquier carácter simple.
- `[]`: una lista de caracteres encerrada entre corchetes especifica que la correspondencia es con cualquier carácter simple de la lista.
- `-`: el guión se utiliza dentro de los corchetes para indicar un rango de caracteres.

Ejemplos: Supongamos que en un directorio existen los ficheros siguientes:

```
a1 a11   a111  a2    aA    aB    aa    b2
a10      a110  a12   a3    aA1   aG1   b1    b3
```

Las referencias siguientes seleccionan los ficheros que se muestran.

Referencia	Grupo de ficheros referenciados
`a*`	`a1 a10 a11 a110 a111 a12 a2 a3 a4 aA1 aB aG1 aa`
`a?`	`a1 a2 a3 aA aB aa`
`a??`	`a10 a11 a12 aA1 aG1`
`a?*`	`a1 a10 a11 a110 a111 a12 a2 a3 aA aA1 aB aG1 aa`
`a?1`	`a11 aA1 aG1`
`[ab]*`	`a1 a11 a111 a2 aA aB aa b2 a10 a110 a12 a3 aA1 aG1 b1 b3`
`?1*`	`a1 a10 a11 a110 a111 a12 b1`
`a[A-Z]*`	`aA aA1 aB aG1`
`[!a]*`	`b1 b2 b3`
`a[A-D]*`	`aA aA1 aB`
`? [1-9]*`	`a1 a10 a11 a110 a111 a12 a2 a3 b1 b2 b3`
`? [!1-2]*`	`a3 aA aA1 aB aG1 aa b3`

5.2.2.5. El lenguaje del shell

Es posible incluir una lista de órdenes del shell en un fichero y entonces ejecutar un nuevo shell utilizando este fichero como entrada estándar. Este segundo shell leerá las líneas del fichero en orden y las ejecutará de una en una. Los ficheros que contienen órdenes del shell se llaman *shell scripts*. Los shell scripts pueden asignar valores a variables shell para poder leerlas posteriormente. También pueden tener parámetros y tienen sentencias `if`, `for`, `while` y `case`. En realidad, un shell script es un programa escrito en el lenguaje del shell.

5.3. Fundamentos de UNIX

5.3.1. Procesos

UNIX es un sistema operativo multitarea y multiusuario, lo que implica que puede atender más de un trabajo a la vez, para cualquier usuario. Cada uno de estos trabajos se denomina proceso. Un proceso de puede definir como un programa en ejecución. Los procesos son las únicas entidades activas en UNIX. Ejecutan un único programa y tienen un único flujo o hebra (*thread*) de control (otros sistemas operativos tienen un modelo de proceso según el cual pueden existir varios flujos de control por proceso).

Los procesos se pueden ejecutar en primer plano (*foreground*), bajo el control de usuario, o en segundo plano (*background*). Los procesos interactivos se han de ejecutar forzosamente en primer plano, ya que los que se ejecutan en background no tienen como entrada estándar el teclado. Cuando se ejecuta un proceso en background el sistema devuelve un número. Este número es el identificador de proceso o PID (process-ID) que UNIX asigna al proceso. No existen PIDs iguales.

Cuando un usuario inicia una sesión mediante el programa `login`, el kernel crea un proceso *shell* para que uso. Este proceso estará activo hasta que el usuario cierre su sesión. Desde el shell el usuario comenzará la ejecución de un número indeterminado de órdenes, que se traducen en procesos. Los procesos que lanza el usuario suelen tener un tiempo de vida relativamente corto, que se corresponde con la duración de la orden. Otros procesos, sin embargo, están siempre en ejecución. Por ejemplo, existen unos procesos llamados *daemons* que son ejecutados de forma automática cuando el sistema operativo inicia su ejecución.

Un proceso sólo puede iniciar su ejecución (nacer) si es creado por otro proceso. El proceso más antiguo se denomina padre y el creado, hijo. Todo proceso hijo conoce el PID de su proceso padre, denominado PPID (*parent-process-ID*). Un proceso padre puede engendrar varios hijos, pero un hijo únicamente puede tener un padre. De esta forma se crea una estructura de procesos en árbol. Cuando un proceso padre acaba su ejecución (muere), generalmente mueren con él todos sus hijos. Sin embargo, esto no tiene por qué ser así. Para evitar que existan procesos que no tengan padre (procesos huérfanos), cuando un proceso padre muere, el PPID de sus procesos hijos toma el valor 1, que es el PID del un proceso especial llamado `init`. De esta forma, cada proceso tiene siempre un padre.

5.3.1.1. Comunicación entre Procesos

Existen varios mecanismos mediante los cuales los procesos se pueden comunicar. En primer lugar, es posible crear canales unidireccionales para la comunicación entre dos procesos, de forma que uno escribe una secuencia de bytes que es leída por otro. Estos canales se llaman `pipes` (tuberías). El uso normal de los pipes es la comunicación entre un proceso padre y otro hijo, de forma que uno escribe y el otro lee. Los procesos se pueden sincronizar mediante pipes, ya que cuando un proceso intenta leer de un pipe vacío, permanece bloqueado hasta que hay información disponible.

`named pipes` o `FIFOs`) son similares a los pipes pero tienen la particularidad de que pueden ser usados por procesos no relacionados. Sólo existen en UNIX System V.

Los `sockets` son un mecanismo de intercomunicación entre procesos que permiten la comunicación entre procesos, aunque éstos se encuentren en máquinas diferentes.

5.3.1.2. Señales

Las señales son un mecanismo que se usa para notificar a los procesos la ocurrencia de sucesos asíncronos. Estos sucesos pueden ser una división por cero, la muerte de un proceso hijo, la solicitud de terminación del proceso por parte del usuario, etc. Cuando se produce alguno de estos sucesos, el proceso en cuestión recibe una señal, lo que causa la interrupción de la ejecución del proceso con el fin de que la señal pueda ser tratada. Las señales pueden ser enviadas por otros procesos o por el sistema operativo.

Existe un identificador para cada tipo de señal. Estos identificadores varían de una versión de UNIX a otra. En BSD UNIX varían de 1 a 31. Por ejemplo, si un proceso realiza una división por cero, recibirá por *floating-point exception*), que se corresponde con el identificador de señal número 8.

Los procesos puede decirle al sistema la acción a tomar cuando reciban una determinada señal. Las opciones son ignorar la señal, aceptar la señal o dejar que sea el sistema el que trate la señal, que es la opción por defecto. Si la elección es aceptar la señal, debe especificar un procedimiento para el tratamiento de la misma. Cuando la señal llega, el flujo de control del proceso cambia a la rutina manejadora de la señal, y cuando ésta termina, devuelve el control al proceso.

5.3.2. El sistema de ficheros

Un fichero UNIX es una secuencia de 0 o más bytes. El sistema no distingue entre ficheros ASCII, binarios, o cualquier otro tipo. La interpretación del contenido de los ficheros se deja a los programas que los utilizan.

La longitud de los nombres de los ficheros estaba limitada originariamente a 14 caracteres arbitrarios, pero BSD UNIX la incrementó hasta 255. Las mayúsculas y las minúsculas son significativas, lo que significa que `RESUMEN`, `resumen`, y `ResumeN` serían nombres diferentes.

Normalmente, muchos programas asumen que los nombres de los ficheros consistan en un nombre seguido por una extensión, separados por un punto. De esta forma, `hola.c` es normalmente un programa en C, `hola.p` un programa en Pascal, etc. Sin embargo, estas convenciones no son impuestas por el sistema operativo.

UNIX distingue tres tipos de ficheros:

- **Ficheros ordinarios o regulares**: Son secuencias de bytes agrupadas bajo un nombre, como se vio anteriormente.
- **Directorios**: Son ficheros que contienen listas de otros ficheros y directorios, lo que permite que el sistema de ficheros sea jerárquico. A un directorio contenido dentro de otro directorio se le denomina subdirectorio y el directorio que lo contiene se llama directorio padre.
- **Ficheros especiales o de dispositivo**: UNIX trata los dispositivos como si fuesen ficheros. Esta característica es una peculiaridad que distingue a UNIX de otros sistemas operativos. De esta forma, cuando un programa quiere, por ejemplo, mostrar información por pantalla, lo que tiene que hacer es realizar una operación de escritura en el fichero que representa el dispositivo de pantalla.

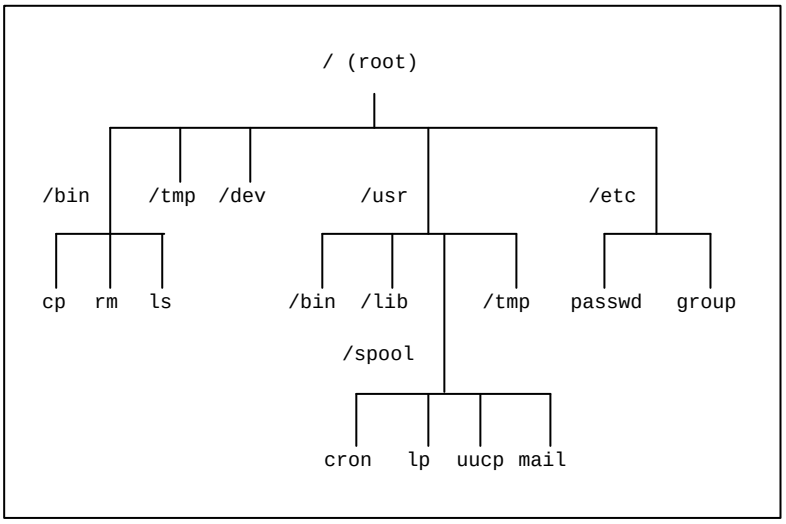

Figura 4.- Estructura de un sistema de ficheros.

El directorio que es la cima de la estructura de directorios se denomina directorio raíz (*root*) y se representa con el carácter barra inclinada (/). A partir de él se pueden encontrar todos los ficheros y directorios del sistema de ficheros. En la Figura 4 se muestra parte de una estructura de directorios y ficheros de un sistema UNIX típico.

Para localizar un fichero en el árbol de directorios es necesario incluir el directorio en el que se encuentra como parte de su nombre. Si existen varios subdirectorios se utiliza el carácter / como separador. Esta secuencia de directorios se denomina camino (*path*). El nombre de camino completo, partiendo del directorio raíz, es una especificación exacta de dónde está un fichero en el sistema de ficheros. El directorio en el que nos encontramos en un momento dado se llama directorio de trabajo o directorio actual, que en UNIX se especifica mediante el carácter punto (.). Los dos puntos (..) hacen referencia al directorio padre del directorio actual.

Existen dos formas de referirse a un fichero dentro el sistema de directorios. Si se emplea el nombre de camino completo se habla de direccionamiento absoluto, mientras que si el nombre de camino parte del directorio actual se habla de direccionamiento relativo.

Ejemplos:

`/usr/bin/ls`	direccionamiento absoluto
`../../bin/ls`	direccionamiento relativo
`/home/zeus/users/practicas/.cshrc`	direccionamiento absoluto
`.cshrc`	direccionamiento relativo

Con frecuencia aparecen situaciones en las que un usuario necesita usar continuamente un fichero que se encuentra en un directorio diferente al de su directorio de trabajo, por lo que tendrá que hacer uso de direccionamiento absoluto o relativo. Para evitar que la necesidad de usar continuamente nombres de camino largos, UNIX permite que un usuario cree una entrada en su directorio de trabajo que apunte al fichero requerido. Esta entrada se denomina **enlace**. De esta forma el árbol de directorios se transforma en un grafo.

5.3.2.1. Seguridad y protección de ficheros

UNIX proporciona un medio de asignar permisos para trabajar con ficheros y directorios de forma que se puede limitar el rango de usuarios que pueden acceder a los mismos y las operaciones que pueden llevar a cabo.

Todos los ficheros en UNIX tienen un propietario, que normalmente es el usuario que los crea. Una vez creado, el propietario puede asignar permisos que permitan o impidan el acceso al fichero. Se distinguen tres tipos de usuarios:

- **propietario**: la persona que crea el fichero, aunque es posible ceder la propiedad a otro usuario.
- **grupo**: varios usuarios pueden formar parte de un mismo grupo.
- **público**: el resto de los usuarios del sistema.

Los ficheros se protegen mediante la asignación a cada uno de ellos de 9 bits que indican los derechos de acceso del fichero. Los tres primeros se refieren a los derechos del propietario, los tres siguientes a los de grupo y los tres últimos a los del resto de los usuarios. Cada grupo de tres bits controlan la lectura, escritura y ejecución del fichero. Combinando los tres tipos de permisos con los tres tipos de usuario se obtienen nueve permisos para cada fichero. Estos nueve permisos se escriben normalmente como una cadena de nueve caracteres (figura 3) que se denomina **modo**. La falta de un permiso se denomina protección y se representa por el signo - (menos). El modo también se puede representar mediante tres dígitos en octal.

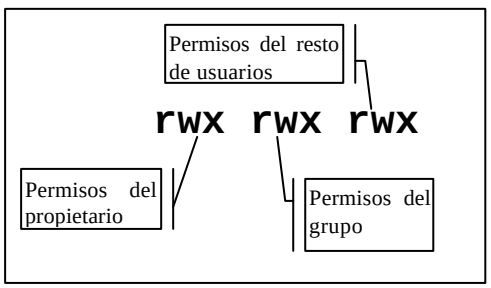

Figura 5.- Bits de derechos de acceso a un fichero.

Los bits de derechos de acceso suelen ir acompañados de un carácter adicional que indica

el tipo del fichero. Este carácter puede ser `-`, para ficheros ordinarios, `d` para directorios y `b` o `c` para ficheros especiales (dispositivos de bloques o caracteres).

Ejemplos:

`r--r--r--` $(444_{(8})$ Fichero de sólo lectura

`rw-r-----` $(640_{(8})$ Permisos de lectura y escritura para el propietario y lectura para el grupo.

`rwxrwxrwx` $(777_{(8})$ El fichero no tiene restricciones de uso.

Existen unos valores que se asignan por defecto a los ficheros y directorios cuando son creados. Normalmente estos valores son $666_{(8}$ (`rw-rww-rw`) para ficheros y $777_{(8}$ (`rwxrwxrwx`) para directorios. Existe una orden llamada `umask` que permite modificar estos valores por defecto. El formato es:

```
umask [mascara]
```

`mascara` es un número octal que, mediante la operación OR exclusiva con los valores originarios por defecto, establece nuevos permisos de creación por defecto. Por ejemplo, para que los ficheros de nueva $644_{(8}$, habrá que usar `umask 022`. El valor establecido por `umask` afectará únicamente a los ficheros de nueva creación.

5.3.3. Entrada/Salida

Para que los programas puedan acceder a los dispositivos físicos, UNIX los integra dentro del sistema de ficheros mediante los ficheros especiales, de forma que a cada dispositivo se le asigna un fichero que se encuentra, normalmente, en el subdirecorio `/dev`. Por ejemplo, la impresora suele estar representada por el fichero `/dev/lp`.

Los ficheros especiales son accedidos de la misma forma que los ficheros ordinarios. Esto significa que se pueden abrir, leer y escribir. Por ejemplo, `cp fich /dev/lp` copia `fich` a la impresora, lo que provoca su impresión. La orden `cp` no tiene constancia de que el fichero destino es un dispositivo. De esta forma no es necesario ningún mecanismo adicional para realizar la entrada/salida. Otra ventaja es que las reglas de protección de ficheros se aplican a los dispositivos, lo que permite controlar los accesos.

Los ficheros especiales se dividen en dos categorías, dispositivos de bloques y dispositivos de caracteres. Los dispositivos de bloques consisten en una secuencia numerada de bloques, de forma que cada bloque puede ser accedido de forma individual. Se usan para discos. Los dispositivos de caracteres se usan normalmente para dispositivos que leen o escriben secuencias de bytes, tales como terminales, impresoras, ratones, etc.

5.3.4. Modelo de memoria

El modelo de memoria de UNIX es muy simple, con el objetivo de que pueda ser implementado en sistemas con sistemas de manejo de memoria muy diferentes. Todo proceso UNIX posee un espacio de direcciones que consiste en cuatro segmentos: texto, datos, *heap* y pila (*stack*).

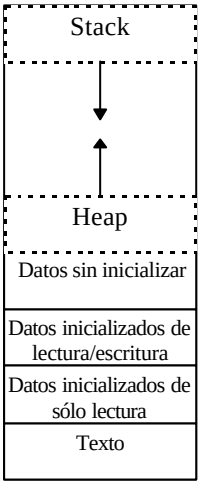

Figura 6.- Espacio de direcciones de un proceso.

El segmento de texto contiene las instrucciones máquina que constituyen el código del programa ejecutable. Es generado durante el proceso de compilación de un programa. Este segmento es normalmente de sólo lectura y no cambia de tamaño durante la

El segmento de datos contiene espacio para las variables del programa y se divide en tres partes, una con los datos inicializados de sólo lectura, otra con los datos inicializados de lectura/escritura y otra con los datos sin inicializar. La primera contiene las variables y constantes cuyo contenido no se puede modificar pero que poseen un valor inicial cuando el programa inicia su ejecución. La segunda contiene los datos que son inicializados por el programa y cuyo valor puede variar durante la ejecución del programa. La tercera se usa para almacenar aquellas variables que no han sido inicializadas por el programa, pero que son inicializadas a cero cuando el programa inicia su ejecución

El heap se usa para asignar memoria de forma dinámica.

El segmento de pila se usa para almacenar la variables locales de los procedimientos y funciones que el programa invoca durante su ejecución, por lo su tamaño cambia continuamente. Al inicio de la ejecución de un programa, la pila contiene las variables de entorno del shell y los argumentos introducidos mediante la línea de órdenes.

Normalmente, el segmento de pila comienza en la cima del espacio virtual de direcciones y crece descendiendo hacia la dirección cero. Si la pila crece por debajo de la dirección inferior del segmento de pila,

5.3.5. Llamadas al sistema

Las llamadas al sistema son un conjunto de instrucciones que constituyen un interfaz entre el sistema operativo y los programas de usuario.

En UNIX, las llamadas al sistema toman la forma de funciones del lenguaje C, y tienen un formato común. Por ejemplo, la llamada al sistema que permite cambiar el directorio actual tendría la forma `status = chdir(dirname) ;`. La llamada devuelve un código de salida que indica si la llamada al sistema ha sido satisfactoria o no. Un valor 0 indica que la llamada se ha llevado a cabo sin error y un valor -1 indica que la llamada ha fallado. Si ha habido fallo, en la variable global `errno` se coloca un código del error. Los códigos de error tendrán un significado determinado dependiendo de la llamada al sistema que los produzcan.

Las llamadas al sistema se agrupan, básicamente, en llamadas para la gestión de procesos, llamadas para la gestión de memoria, llamadas para la gestión de ficheros y directorios y llamadas para la gestión de entrada/salida.

5.4. Ordenes de UNIX más comunes

A continuación se enumeran algunas de las órdenes más empleadas de UNIX. Los formatos de las órdenes

5.4.1. Ordenes para el manejo de directorios

```
pwd
```

Muestra por pantalla el nombre de camino completo del directorio actual

```
cd [directorio]
```

Cambia el directorio de trabajo. Si no especifica ningún parámetro, establece como directorio de trabajo el *home*) del usuario.

```
ls [-aAcCdfFgilLqrRstu1] [fichero(s)]
```

Muestra el contenido de un directorio. Algunas de las opciones más comunes son:

- *F*: Si el fichero es ejecutable o un directorio muestra un asterisco(*) o una barra (/) detrás del nombre, respectivamente.
- *R*: Listado recursivo. Lista ficheros y subdirectorios.
- *a*: Lista todas las entradas. Normalmente, los ficheros que empiezan por punto (.) no se muestran.
- *l:* Listado en formato largo. Muestra el modo, número de enlaces, propietario, tamaño en bytes y tiempo de última modificación de cada fichero.

```
mkdir [-p] directorio
```

Crea un directorio. La opción *-p* permite que los directorios padres que falten sean creados. Por ejemplo, *mkdir -p source/code/practicas* creará los subdirectorios *source* y *code*, si no existen, y tras ellos creará el subdirectorio *practicas*.

```
rmdir directorio
```

Borra un directorio, siempre y cuando esté vacío.

5.4.2. Ordenes para el manejo de ficheros

```
cat [-benstuv] [fichero(s)]
```

Lee cada fichero especificado como parámetro y muestra sus contenidos por pantalla. Si no se introduce ningún fichero como parámetro, lee de la entrada estándar.

```
cp [-ip] fichero1 fichero2

cp -rR [-ip] directorio1 directorio2

cp [-iprR] fichero(s) directorio
```

Copia el contenido de *fichero1* en *fichero2*. El segundo modo permite copiar recursivamente *directorio1*, junto con sus ficheros y subdirectorios, a *directorio2*. Si éste último no existe, se crea. Si existe, se realiza una copia de *directorio1* dentro de *directorio2* (será un subdirectorio). Con el tercer modo, cada fichero se copia en el directorio indicado. Las opciones *-r* y *-R* indican comportamiento recursivo.

```
rm [-fir] fichero(s)
```

Borra ficheros y directorios. La opción *-r* indica comportamiento recursivo y se emplea para borrar directorios.

```
mv [-fi] fichero1 fichero2

mv [-fi] directorio1 directorio2

mv [-fi] fichero(s) directorio
```

Mueve ficheros y directorios dentro del sistema de ficheros. Equivale a renombrar un fichero o directorio.

```
ln [-fs] fichero [enlace]

ln [-fs] camino directorio
```

Crea una nombre adicional, llamado enlace, a un fichero. Un fichero puede tener varios enlaces.

```
find lista_de_directorios
expresion_de_busqueda
```

Busca ficheros recursivamente a partir de los directorios señalados en *lista_de_caminos*, buscando aquellos ficheros que safisfacen una *expresión_de_búsqueda*. No se siguen los enlaces simbólicos hacia otros ficheros o directorios.

La expresión de búsqueda consta de una o más expresiones primarias, cada una de las cuales describe una propiedad de un fichero, aunque algunas indican una acción a tomar. Las expresiones primarias se *!* (NOT), *-a* (AND, que se asume por defecto) y *-o* (OR).

Algunas expresiones primarias son:

Expresión	Acción
-name fichero	Verdadero si *fichero* coincide con el nombre del fichero actual. Pueden usarse metacaracteres, pero en este caso *fichero* debe de ir entre comillas.
-user nombre	Verdadero si el fichero pertenece al usuario *nombre*
-size n	Verdadero si el fichero tiene una longitud de mayor de *n* bloques (512 bytes por bloque) si *n* es positivo, y menor que n bloques si *n* es negativo.
-mtime n	Verdadero si el fichero ha sido modificado en *n* dias
-atime n	Verdadero si el fichero ha sido accedido en *n* dias
-print	Siempre verdadero. Imprime el camino completo del fichero
-exec orden	Ejecuta la orden sobre el fichero actual. Al final de *orden* es necesario introducir los caracteres de escape y punto y coma (*\;*). Si la orden es sequida de {}, el nombre del fichero actual es pasado como argumento a *orden*.

-newer fichero Verdadero si el fichero actual ha sido modificado más recientemente que *fichero*.

Ejemplos:

- Buscar todos los ficheros cuyo nombre termine en .c a partir del directorio actual:

```
find . -name "*.c" -print
```

- Buscar todos los ficheros del sistema cuyo tamaño sea mayor de 100 bloques:

```
find / -size +100 -print
```

- Borrar todos los ficheros cuyo nombre es core a partir del directorio *home*:

```
find ~ -name core -exec rm {} \;
```

- Imprimir, a partir del directorio actual, todos los ficheros cuyo nombre termine en *.c%* o .bak y no hayan sido modificados en 20 días:

```
find . \(-name "*.c%" -o -name "*bak" \) -mtime +20 -print
```

file [-f ffich] [-cL] [-m mfile] fichero ...

Determina el tipo de un fichero examinando su contenido.

du [-s] [-a] fichero ...

Muestra el número de bloques de disco usados por un conjunto de ficheros y directorios

df [-a] [-i] [-t type] [filesystem ...] [fichero ...]

Muestra el espacio libre en el sistema de ficheros

5.4.3. Ordenes para el control de procesos

ps [-acCegklnrStuvwxU]

Muestra el estado de los procesos del sistema. Sin argumentos, muestra información sobre los procesos

kill [-señal] pid ...

Envía una señal a un proceso. Por defecto, se envía la señal de terminación del proceso (SIGTERM). Esta señal puede ser ignorada por el proceso, por lo que para eliminar de forma segura un proceso es necesario enviarle la señal de terminación incondicional (señal 9).

nice [-numero] orden [argumentos]

Permite modificar la prioridad con la que se ejecutará un proceso. La prioridad del proceso se aumentará en la *numero*. A mayor valor de *numero*, menor prioridad. Por defecto, *numero* toma el valor de 10. Es posible aumentar la prioridad de un proceso si *numero* es un valor negativo, pero en esta posibilidad únicamente puede ser usada por el superusuario.

5.4.4. Ordenes para seguridad y protección

chmod [-fR] modo fichero ...

Cambia los permisos (modo) de uno o varios ficheros o directorios. Unicamente puede ser usado por el propietario del fichero (o el superusuario). El modo del fichero se puede especificar de forma absoluta o de forma simbólica. El modo absoluto es un número octal que indica los permisos del fichero (ej.: *chmod 444 datos*). El modo simbólico es una cadena que tiene la forma

[quien] operador permiso [operador permiso]

donde quien es una combinación de las letras *u* (usuario), *g* (grupo) y *o* (otros) ó *a* (que equivale a *ugo*), operador es + (añade un permiso), - (elimina un permiso) ó = (establece un permiso); y permiso es una combinación de *r*, *w* ó *x*.

Ejemplos:

- Retirar el permiso de escritura, para el propietario, del fichero *datos*:

  chmod u-w datos

- Establecer el permiso de lectura al fichero *datos*:

  chmod ugo=r datos

- Retirar el permiso de lectura de *datos* a todos menos al propietario:

  chmod go-r datos

- Asignar permisos de lectura a todos los usuarios y de escritura para el propietario:

  chmod a=r, u+w dataos

chown propietario fichero ...

Cambia el propietario de un fichero. Sólo puede ser empleada por el propietario del fichero o por el superusuario. En la versión de UNIX SunOS, únicamente puede ser empleada por el superusuario.

passwd

Permite cambiar la palabra de paso del usuario.

5.4.5. Ordenes varias

diff

Muestra las diferencias, línea por línea, entre dos ficheros

grep

Busca las líneas de un fichero que contienen una determinada cadena o expresión regular

head

Muestra las primeras líneas de un fichero

tail

Muestra las últimas líneas de un fichero

wc

Cuenta las líneas, palabras y caracteres de un fichero

who

Muestra los usuarios que se encuentran conectados al sistema

whoami

Muestra el nombre del usuario de la sesión activa.

man

Muestra las páginas del manual de referencia. Por ejemplo, *man ls* muestra las páginas del manual referentes a la orden *ls*.

5.5. Referencias

- “Modern Operating Systems”. A.S. Tanenbaum. Prentice-Hall. 1992.
- "Operating System Concepts". A. Silberschatz, P. B. Galvin. Addison-Wesley. 1994.
- "UNIX for Programmers and Users". G. Glass. Prentice-Hall. 1993.
- SunOS Reference Manual.