Sistemas Operativos

Una computadora, sola, sin algún tipo de software, no sirve de mucho. El software es una parte indispensable del sistema, no solamente si hablamos de aplicaciones, sino que además se necesita software adicional (a veces llamado **software de sistema** o **software de base**[1]) para poder hacer que el sistema ejecute esas aplicaciones. Veamos por qué.

Sistemas de computación

Una vez que tenemos un compilador o intérprete, podemos crear **aplicaciones.** Estas aplicaciones pueden tener muchos propósitos diferentes (están las aplicaciones de cálculo científico, de cálculo financiero, los juegos, los programas utilitarios, educativos...), pero siempre están preparadas para ser ejecutadas por cierto **hardware** y son hechas para que los **usuarios** del sistema las utilicen. Estas tres entidades, hardware, aplicaciones y usuarios, son tres componentes esenciales de un **sistema de computación**. Sin embargo, a este escenario le falta un componente muy importante: el **sistema operativo**.

Algo de historia

Este componente surge de la necesidad de posibilitar y organizar el uso de los recursos del sistema de computación, y es **un concepto sumamente complejo** que fue tomando forma a través del tiempo, en sucesivos pasos evolutivos. Estos pasos evolutivos siempre se fueron orientando hacia la mayor comodidad para los usuarios y el mejor aprovechamiento de los recursos del sistema. Conocer estos pasos evolutivos permite comprender mejor de qué hablamos al referirnos a los **sistemas operativos**.

En los primeros tiempos (antes de 1960), las computadoras se asemejaban mucho al MCBE que hemos descripto. La selección y carga de las aplicaciones era manual, y no se podía ejecutar más de un programa por vez. El operador del equipo debía cargar primero el traductor, traducir el programa fuente X, descargar el traductor, luego cargar el vinculador, vincular el programa objeto de X resultante con las bibliotecas, descargar el vinculador, cargar el ejecutable del programa X, ejecutarlo, y repetir el procedimiento para la siguiente aplicación que se deseaba ejecutar. El procedimiento de ejecutar una aplicación y cargar la siguiente era lento y trabajoso. Como la computadora era muy costosa y debía ser utilizada por gran número de personas, era interesante reducir este tiempo de espera entre ejecuciones para **minimizar el tiempo ocioso** del recurso de computación.

> Decía un experto: “cada usuario tenía asignado un tiempo mínimo de 15 minutos, de los cuales invertía 10 minutos en preparar el equipo para hacer su cómputo, desperdiciando las dos terceras partes del tiempo.” Para esos usuarios, el costo del tiempo de computadora era de 146000 dólares (¡de 1954!) mensuales.
>
> *Per Brinch Hansen, The Evolution of Operating Systems (2000)*

Un primer paso adelante (década de 1960) fue automatizar la carga y ejecución de programas, al crearse cargadores especiales que podían seguir una lista de instrucciones. Estos cargadores eran programados en un sencillo lenguaje de control de tareas o **comandos**, escritos en **tarjetas perforadas**, y ejecutaban conjuntos, o **lotes** (en inglés, *batch*) de **tareas** (en inglés, *jobs*). El

1 http://es.wikipedia.org/wiki/Software_de_sistema

cargador cargaba el traductor para el lenguaje que se deseaba y lo ejecutaba. Al terminar de ejecutar, el control volvía al procesador de comandos, que seguía leyendo tarjetas y ejecutando las tareas indicadas en ellas. El operador podía reunir en un lote de tarjetas todas las tareas similares (por ejemplo, todos los programas fuente de FORTRAN que había que traducir y ejecutar). Los sistemas de computación provistos de este software básico se llamaron **sistemas batch** (de **procesamiento por lotes**).

Cargadores

En nuestra descripción del MCBE vimos que, al momento de iniciar su operación, todos los registros de la máquina contienen un valor 0, y que la memoria está *mágicamente* cargada con un programa almacenado. En una computadora real, la situación es parecida a la del MCBE, pero debido a la implementación tecnológica de la memoria, ésta se encuentra vacía. La memoria de trabajo de las computadoras, que se llama **RAM** (siglas de *Random Access Memory*, **Memoria de Acceso Aleatorio**), mantiene sus contenidos únicamente mientras está alimentada por corriente eléctrica, y pierde sus contenidos al ser apagada la computadora. Otra implementación diferente de memoria, la memoria **ROM** (*Read Only Memory* o **Memoria de Sólo Lectura**), conserva sus contenidos en forma permanente. Esta memoria se utiliza para guardar programas de diagnóstico y de carga inicial de la computadora.

- ¿Cómo es que llega un programa a la memoria en una computadora real? Se requiere un software adicional a las aplicaciones (un **cargador**), que sea capaz de leer la aplicación que necesitamos ejecutar, byte por byte, y escribir esas instrucciones y datos en la memoria. Esta aplicación proviene de algún medio de **almacenamiento permanente** tal como discos, cinta, tarjetas, u otros dispositivos.
- ¿Cómo **se cargará** a su vez este programa **cargador**? La computadora debe incorporar una **memoria permanente** donde residan las instrucciones del cargador, y el hardware debe estar configurado de tal manera que la primera instrucción ejecutada por la máquina al encenderse sea la primera instrucción del cargador.

La siguiente cuestión que llamó la atención fue que los sistemas normalmente gastaban gran parte del tiempo haciendo Entrada/Salida, leyendo o escribiendo programas y datos de los dispositivos de almacenamiento. Los dispositivos de Entrada/Salida son relativamente autónomos (son capaces de procesar, independientemente, grandes cantidades de datos) pero tienen una velocidad de procesamiento muchísimo menor que la de la CPU. Mientras se efectuaban operaciones de Entrada/Salida, la CPU permanecía prácticamente ociosa, y el recurso de computación seguía siendo desperdiciado. La idea para mejorar esta situación fue aprovechar esos tiempos muertos de la CPU para ejecutar otros trabajos simultáneamente. Con este esquema, la computadora tenía varios trabajos en marcha al mismo tiempo, sólo que dedicando la CPU a cada uno por vez, mientras los demás trabajos se encontraban realizando operaciones de entrada/salida.

Esto requirió programas de control mucho más complejos que los anteriores sistemas batch. Éstos fueron los primeros **sistemas operativos multiprogramados**, capaces de mantener varios programas en diferentes estados de ejecución al mismo tiempo (década de 1960). Se plantearon nuevos e interesantes problemas: no sólo había que repartir el uso de la CPU entre las tareas, sino que había que poder mantener en memoria varios programas a la vez, y además imponer alguna especie de control para que los trabajos de diferentes usuarios no invadieran espacio o recursos de los demás. De hecho aquí ya aparecen varios de los grandes problemas que deben resolver los sistemas operativos: la **administración de procesos**, la **administración de memoria** y la **protección**.

Mientras tanto, los sistemas de Entrada/Salida se hacían más sofisticados, iban apareciendo **periféricos** como discos y tambores magnéticos, de mejores características, y el tamaño y velocidad de las memorias aumentaban. Haber logrado ejecutar en forma más o menos simultánea varios

trabajos inspiró el siguiente paso: modificar ese sistema operativo multiprogramado para hacer que la CPU se repartiera entre diferentes usuarios que pudieran usar el equipo de manera interactiva, es decir, prácticamente dialogando con la computadora. Usando **terminales** (conjunto de teclado y monitor conectado al equipo central), el usuario podía interactuar con el sistema operativo mediante una **consola** donde daba órdenes al sistema, y los programas y datos se leían y escribían en medios de almacenamiento con tiempos de respuesta aceptables. Hasta ese momento (década de 1970) no se habían visto sistemas así. Estos sistemas se llamaron **sistemas de tiempo compartido** (*time-sharing systems*). El sistema operativo establecía unos intervalos de tiempo muy cortos durante los cuales entregaba la CPU a cada trabajo en el sistema, uno tras otro, y volvía a comenzar la ronda. El efecto era que, aunque los equipos seguían siendo compartidos, cada usuario tenía la ilusión de disponer completamente de una máquina para sí mismo, con la cual podía desarrollar o ejecutar programas.

La industria del hardware de computación continuó mejorando a gran velocidad los procesos de producción, y por lo tanto fueron bajando los precios y aumentando las capacidades de los productos. En esta época surge el **microprocesador**[2], que cambió definitivamente la dinámica de la industria electrónica en todos los campos. Pronto aparecieron sistemas personales de computación, de los cuales había comenzado a hablarse ya en los años 70. En **1981** se publicó la especificación de una **computadora personal**[3], la **IBM PC (*personal computer*)**, de arquitectura abierta; es decir, al contrario de lo que había ocurrido históricamente con los productos de computación, protegidos por patentes y secretos industriales, su diseño estaba documentado y era públicamente accesible. Cualquier fabricante de hardware tenía disponible la información necesaria para producir elementos compatibles con ella y fabricar reproducciones (o **clones**) de esa computadora, lo cual garantizó rápidamente un gran mercado para esta clase de productos. Las computadoras que usamos hoy en la casa, el trabajo o las escuelas, son descendientes directos de esta arquitectura abierta.

Al contrario de los sistemas operativos anteriores para máquinas mayores, que eran **multiusuario** y **multitarea**, el sistema operativo de la PC (llamado **MS-DOS**[4], *Microsoft Disk Operating System*) estaba diseñado para ejecutar sólo un programa por vez, para un único usuario que ejecutaba programas interactivamente, usando una interfaz de consola de caracteres. Sin embargo, el modelo PC fue tan exitoso que otros sistemas operativos, del dominio de los grandes sistemas multiusuarios (como **UNIX**[5]), fueron adaptados para correr en estas computadoras. En 1991 se escribió **Linux**[6], un clon de UNIX, para la PC.

Posteriormente se sucedieron gran cantidad de avances en la productividad personal de los usuarios, al adaptarse a la PC ambientes operativos basados en la **metáfora del escritorio**, es decir, aquellos que simulan un ambiente de trabajo habitual de los usuarios, con elementos familiares como escritorios y carpetas. Estas **interfaces de usuario** (como el ambiente operativo **Windows**[7] de Microsoft y la interfaz gráfica de UNIX, el sistema **XWindow**[8]) necesitan pantallas con ciertas capacidades gráficas. Utilizan elementos gráficos como **ventanas e íconos**, y se manejan mediante el teclado y otros dispositivos de interacción como el **mouse**.

2 http://es.wikipedia.org/wiki/Microprocesador
3 http://es.wikipedia.org/wiki/IBM_PC
4 http://es.wikipedia.org/wiki/MS-DOS
5 http://es.wikipedia.org/wiki/Unix
6 http://es.wikipedia.org/wiki/Linux
7 http://es.wikipedia.org/wiki/Microsoft_Windows
8 http://es.wikipedia.org/wiki/X_Window_System

Servicios de los sistemas operativos

Por debajo de unas u otras interfaces de usuario, el sistema operativo sigue enfrentándose a los mismos problemas que ya habían aparecido anteriormente (y a otros nuevos), siendo los más importantes y visibles **la gestión de memoria, la gestión de procesos y la gestión del almacenamiento**. El sistema operativo es la única pieza de software realmente imprescindible en la computadora, y es la única realmente independiente de todas las demás. Sin embargo, existe únicamente para dar servicios a las aplicaciones, que son el objetivo por el cual queremos tener un sistema de computación. Entre estos servicios se incluyen:

- La ejecución de programas (permitiendo crear **procesos**)
- Las operaciones de entrada/salida (evitando las colisiones o conflictos entre requerimientos de entrada/salida hechos por diferentes aplicaciones)
- El manejo de archivos (ofreciendo un **sistema de archivos** con un sistema de nombres y **atributos**)
- La **protección** (evitando la interferencia entre los procesos)

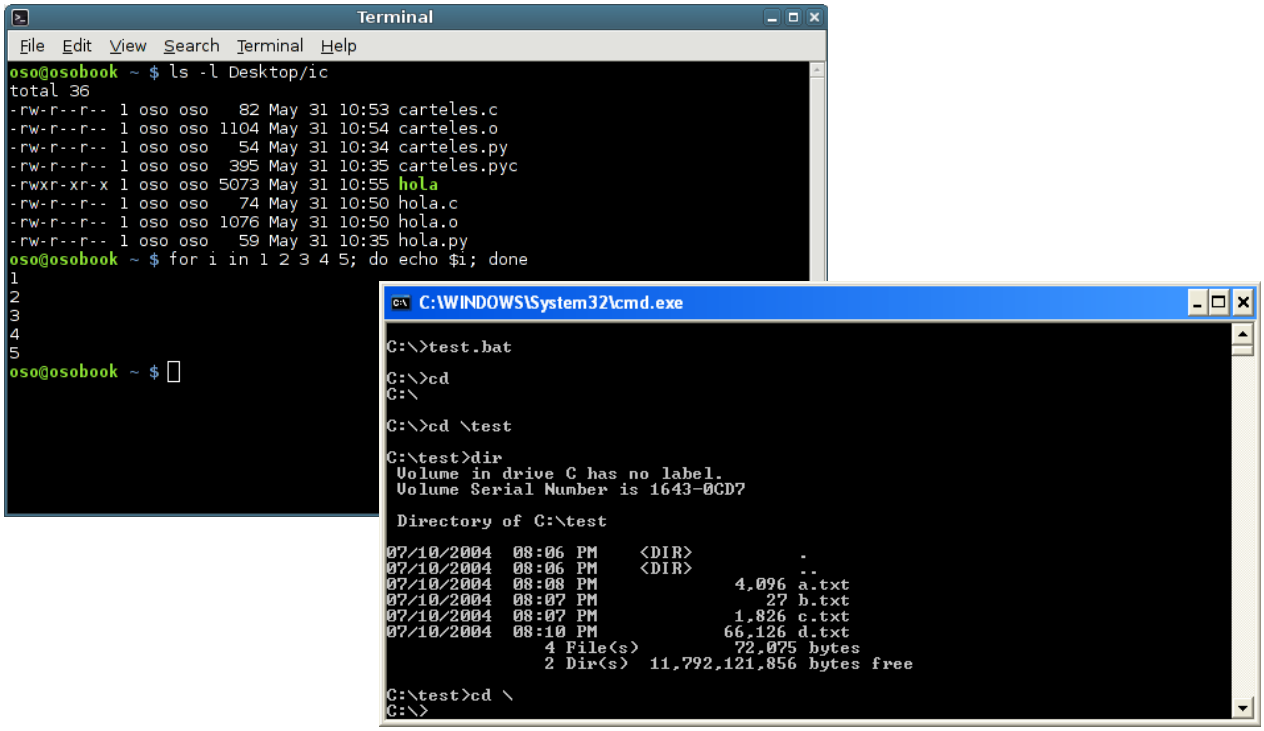
El sistema operativo está siempre **residente** en la memoria de la computadora, y recibe el control del procesador cada vez que debe cumplir alguno de estos servicios.

Ejecución de programas

Un programa ejecutándose recibe el nombre de **proceso**. Un sistema operativo **multiprogramado** puede ejecutar varios procesos **concurrentemente** (simulando que todos operan al mismo tiempo), repartiendo la CPU entre todos los procesos activos. Durante la ejecución de los procesos, éstos pueden recibir más o menos tiempo de CPU por cada ronda, dependiendo de alguna estrategia de **planificación** (o **scheduling**) implementada por el sistema operativo, a veces estableciendo un orden de **prioridades** entre los procesos. Cuando el sistema de computación cuenta con más de un procesador (es decir, si es **multiprocesador**) y **si además** el sistema operativo puede aprovecharlo (es decir, si es de **multiprocesamiento**), puede asignarse **un procesador a cada proceso**, y así la ejecución de varios procesos pasa a ser **paralela**.

Un proceso se origina normalmente cuando el sistema operativo lee el código ejecutable de un programa, contenido en algún **archivo** alojado en algún **dispositivo de almacenamiento**; y lo carga en una región de memoria asignada. Esta acción de **lanzar** un proceso puede ocurrir porque el sistema operativo está configurado para automáticamente durante el arranque del sistema (como ocurre con un servidor web) o porque lo solicita el usuario (como ocurre con aplicaciones de procesadores de texto o juegos).

El usuario que necesita lanzar un proceso da las órdenes correspondientes desde la **interfaz de usuario**, que puede ser una consola o una interfaz gráfica). Ambas son formas de **intérpretes de comandos** o **shells**. A veces se suele identificar el intérprete de comandos con el sistema operativo, porque es la **interfaz** o forma de comunicación más visible al usuario, pero es más apropiado pensar en el sistema operativo como un conjunto de rutinas y procesos, y considerar que el shell es una aplicación más. Un intérprete de comandos de texto suele ofrecer un **lenguaje de comandos** que puede llegar a ser muy poderoso y permitir la automatización de tareas (como el shell de UNIX).



Al lanzar un proceso, el sistema operativo le asigna un conjunto de **recursos** indispensables para funcionar. Entre ellos, una región de **memoria** que será ocupada por el **código** (es decir, las instrucciones) y los **datos** (es decir, las variables y constantes) del proceso. La ejecución de programas implica así la función de **administración de memoria** del sistema operativo. Esta función puede ser muy simple, como en los antiguos sistemas operativos monousuario, o volverse sumamente complicada, para alcanzar objetivos muy importantes.

Los sistemas operativos modernos utilizan algún esquema de **memoria virtual**[9], que es una técnica que **mapea**, o hace corresponder, un espacio de **direcciones virtuales**, grande, sobre un espacio de **direcciones físicas** de memoria, más pequeño. Cada proceso en ejecución dispone de un espacio de direcciones virtuales asignado por el sistema operativo. **Con apoyo del hardware**[10], el sistema operativo traduce las direcciones virtuales usadas por el proceso a direcciones físicas de memoria. Cada vez que un proceso lee una instrucción, o lee o escribe un posición de memoria, lo hace **usando una dirección de su espacio virtual**; pero los accesos a la memoria física necesariamente son hechos a las direcciones traducidas por el sistema operativo.

Los procesos no necesitan acceder siempre todo su espacio de direcciones virtuales, ni necesitan acceder muchas posiciones de memoria al mismo tiempo. Así el sistema operativo, manipulando el **mapa de memoria** que determina la traducción de direcciones virtuales a físicas, puede reutilizar trozos de memoria física en diferentes momentos para diferentes procesos. Para esto, utiliza parte del **almacenamiento secundario** (espacio en discos u otros medios permanentes) para guardar los contenidos de aquellos trozos de memoria física que debe reutilizar, como si se tratara de una extensión de la memoria física. De esta manera el sistema operativo puede ofrecerle a cada proceso un espacio virtual de memoria que **parece ser más grande que la memoria física**, y puede mantener en ejecución un conjunto de procesos **que demandan más que la memoria física disponible**.

9 http://es.wikipedia.org/wiki/Memoria_virtual
10 http://es.wikipedia.org/wiki/Unidad_de_Manejo_de_Memoria

Espacio virtual de los procesos y propiedad de localidad

Los procesos ocuparán una cierta cantidad de memoria virtual, o **espacio virtual**, cuyo tamaño será aproximadamente equivalente a la memoria ocupada por el código ejecutable más la memoria ocupada por las estructuras de datos usadas por el programa. Por ejemplo, un programa que multiplique dos matrices, al ser ejecutado ocupará la cantidad de bytes correspondiente a las instrucciones de código máquina que forman el programa, más la cantidad de bytes para alojar en memoria cada una de las matrices que intervienen en el cómputo.

Sin embargo, los procesos no utilizan toda esta memoria al mismo tiempo, sino que acceden a los datos e instrucciones de a unos pocos por vez. Por ejemplo, si una instrucción de un proceso consiste en cargar un contenido de memoria en un registro (tal como ocurre con la instrucción **LD 20** del MCBE), ejecutar esta instrucción implica acceder a **dos** posiciones de memoria durante el ciclo de instrucción: aquella donde está almacenada la instrucción, y aquella donde se encuentra el dato (dirección 20). Mientras se ejecuta esta instrucción, solamente hace falta tener acceso a estas dos posiciones de memoria. Más aún, con mucha probabilidad en el caso general, la siguiente instrucción ejecutada por el proceso será la ubicada en la posición siguiente de memoria, y el siguiente dato accedido estará probablemente ubicado cerca del anterior (fenómeno de **localidad**).

La memoria virtual se basa en el hecho de que los procesos, aunque tengan un **espacio virtual** grande, normalmente utilizan sólo un conjunto reducido de posiciones de memoria en un espacio de tiempo dado. A esta propiedad se la llama **localidad**. Por lo tanto, sólo se requiere que esté en memoria, en cada momento, una porción relativamente pequeña de este espacio virtual.

Además de la memoria, otros recursos que el sistema operativo debe conceder serán una cierta cantidad de tiempo de CPU (o **quantum**) que el proceso recibirá periódicamente, ciertas estructuras

de datos en la memoria del sistema operativo, etc. Al terminar el proceso, el sistema operativo recupera esos recursos, ya sean **físicos** (memoria, CPU) o **lógicos** (como estructuras de datos) para ser reutilizados.

Operaciones de Entrada/Salida

Las operaciones de Entrada/Salida (**E/S**) que ofrece el sistema operativo se realizan a un nivel más alto que las que ejecuta la máquina. Un procesador hace E/S a nivel de bytes, palabras, o regiones de memoria, desde o hacia componentes conectados a los buses del sistema; pero los programadores de aplicaciones prefieren hablar en un vocabulario de mayor nivel, por ejemplo en términos de **archivos**.

El sistema operativo contiene rutinas, subprogramas o funciones, que trabajan sobre los dispositivos en forma coordinada, y los procesos las utilizan haciendo **llamadas al sistema** ***(system calls)***, nunca accediendo directamente a los dispositivos, sino siempre por medio del sistema operativo. Una llamada al sistema es una instrucción especial, que lleva unos argumentos o parámetros que indican qué se está pidiendo al sistema operativo. El control pasa del proceso al sistema operativo, que ejecuta esa rutina o subprograma, asegurándose de que lo solicitado es válido, y luego devuelve el control al proceso. De esta manera el sistema operativo actúa como un coordinador centralizado de las operaciones sobre el hardware, y mantiene control sobre la actividades de E/S de todos los procesos.

Las operaciones de E/S deben ser controladas así por el sistema operativo, por varias razones. En primer lugar, en un ambiente **multiprogramado** pueden ocurrir condiciones de acceso simultáneo a un mismo archivo o dispositivo, y el sistema operativo, gracias a que mantiene una noción de los procesos que están en ejecución, puede **arbitrar** ese acceso para que no ocurran colisiones. Éstas podrían ocurrir, por ejemplo, cuando dos procesos concurrentes quieren imprimir algo por una impresora; o escribir líneas de texto diferentes en un mismo archivo. Si el acceso no fuera **coordinado**, los procesos estropearían el trabajo de los otros al mezclarse la información que escribieran.

En segundo lugar, los procesos son programas en ejecución que han sido escritos por diferentes personas con diferentes objetivos, y es natural que quieran disponer de los recursos (memoria, almacenamiento, CPU) para sus propias tareas, aun a costa de los demás. Esto se expresa diciendo que los procesos **compiten** por los recursos. Un usuario malintencionado podría atacar a los procesos de los demás usando sus recursos y haciéndolos fallar. Aun sin mala intención, un error de programación puede causar el mismo efecto si no se controla. Por ejemplo, un proceso podría (intencional o accidentalmente) modificar o borrar un archivo privado de otro proceso y hacerlo fallar.

> La estrategia más segura para que el sistema se mantenga funcionando correctamente es ¡suponer que los usuarios son enemigos! Esta estrategia se realiza haciendo que ningún proceso pueda usar recursos si no los solicita al sistema operativo, y puede considerarse como otro servicio aparte, el de **protección**.

Manejo de archivos

Una parte muy visible del sistema operativo es la ofrecida por un **sistema de archivos**. Un **archivo** es un conjunto de información almacenado en un medio permanente (como discos o cintas), que

representa un contenido dado. Un archivo puede guardar la clase de información que se desee: textos, datos en cualquier formato (como imágenes o videos), programas fuente, programas ejecutables, etc.

Las aplicaciones son las que tienen el conocimiento del contenido y la estructura de los archivos; pero, para el sistema operativo, cada archivo se trata simplemente de una **sucesión de bloques de bytes** que está agrupada bajo un **nombre de archivo**. Usando ese nombre, el usuario o las aplicaciones pueden manipular ese conjunto de información como un todo (creando archivos, copiándolos, borrándolos, etc.). El trabajo del sistema de archivos es permitir el acceso a estas secuencias de bloques para leerlos o escribirlos, y mantener esas secuencias de bloques en orden. El trabajo diario de los usuarios del sistema operativo siempre implica, de alguna manera, manipular archivos. Para utilizar los archivos, los procesos realizan **llamadas al sistema** para escribir, leer, crear, borrar, etc., los archivos. El sistema operativo dispone de la lógica para organizar los accesos y mantener la integridad de los archivos.

Un sistema de archivos reside en algún medio de almacenamiento y presenta alguna forma de organización, generalmente en forma de **árbol de directorios** (o **carpetas**) que pueden contenerse unos a otros. Los archivos, además de distinguirse por su nombre, tienen algunos **atributos** que pueden ser importantes para el usuario o las aplicaciones, como su fecha de creación, la cantidad de bytes que contienen, o la identidad de aquellos usuarios que pueden usarlos. Estos atributos son mantenidos por el sistema de archivos en espacios de almacenamiento especiales, llamados zonas de **metadatos**, ya que son **datos acerca de los datos**.

Diferentes sistemas de archivos mantienen diferentes conjuntos de atributos. Por ejemplo, los sistemas UNIX, para ejecutar un programa contenido en un archivo, necesitan que este archivo tenga el atributo de **ejecutable** (bit de “ejecutable” activo). El sistema de archivos que se usa normalmente en las memorias externas (pendrives, o drives USB) no mantiene este atributo, por lo cual no podemos **ejecutar** un archivo almacenado en este tipo de sistema de archivos.

Protección

En realidad la **protección** es algo que está presente en cada actividad del sistema operativo. Como dijimos antes, para garantizar que los procesos no interfieran unos con otros se necesita tener la coordinación de las operaciones y validar que las llamadas al sistema se hagan con los parámetros adecuados.

Por ejemplo, un archivo que es privado de un proceso porque contiene información confidencial, no debe poder ser leído por procesos de otros usuarios. Para poder preservar esta confidencialidad, cuando un proceso quiere acceder a un archivo (usando una **llamada al sistema**), el sistema operativo comprueba que las operaciones de E/S que se desea hacer sobre este archivo sean válidas, comparando la identidad del usuario que lanzó el proceso contra los **metadatos** del archivo. Si el usuario **dueño del proceso** figura en las **listas de acceso** correctas en los metadatos del archivo, la **llamada al sistema** para acceder al archivo tendrá éxito, y de lo contrario se generará una **situación de error** que puede hacer que el sistema operativo **termine** al proceso anómalo. Lo mismo ocurre con los demás recursos, como la **memoria**. Si un proceso pudiera escribir libremente en el espacio de memoria de otro, podría modificar el programa o corromper los datos. La protección comprende todas estas comprobaciones.

Para poder aplicar esta política de control, el sistema operativo requiere **apoyo del hardware**. Al

surgir los primeros sistemas operativos multiusuario y multiprogramados se rediseñaron los procesadores, introduciendo el llamado **modo dual** de funcionamiento. Un procesador actual cuenta con un **modo usuario**, en el cual se pueden ejecutar instrucciones aritméticas, de transferencia de control, de transferencia de datos, etc., y un **modo monitor o supervisor**. En el modo monitor se pueden ejecutar ciertas instrucciones de código máquina, llamadas **privilegiadas** (por ejemplo, las operaciones de E/S), que no se pueden ejecutar en el otro modo. El procesador cuenta con un bit de estado, que representa el estado (usuario o monitor) en que está ejecutando en cada momento.

El procesador pasa a modo monitor durante la carga del sistema operativo, en los primeros momentos de la ejecución. A partir de ese momento, una parte del sistema operativo queda residente en memoria y opera siempre en este modo. En cambio, un proceso de usuario ejecuta en modo usuario, y no puede entrar voluntariamente en modo monitor por sí mismo. Sin embargo, las **llamadas al sistema** fuerzan un ingreso al modo monitor, de forma que solamente se puede ejecutar una instrucción del procesador privilegiada si el proceso de usuario ha hecho previamente una llamada al sistema. Durante la llamada al sistema, el proceso ejecuta código del sistema operativo en modo monitor, posiblemente ejecutando E/S u otras operaciones críticas, pero **no puede hacer otra cosa** que ese código, que ya está escrito en forma correcta y segura en el sistema operativo. Al terminar la llamada al sistema, el código del sistema operativo vuelve el bit de estado del procesador a modo usuario y devuelve el control al código del proceso. La lógica necesaria para brindar protección entre los procesos está en el código del sistema operativo que se ejecuta durante esta llamada al sistema. De esta manera se garantiza que un proceso jamás hará un acceso a dispositivos de E/S, o a ningún otro recurso, sin conocimiento y supervisión del sistema operativo. Sin esta característica del hardware, este control no sería posible.