

MP0483.

Sistemas informáticos

**UF2. Instalación de Sistemas
Operativos**

**2.1. Tipos y funciones de un
sistema operativo**

Índice

☰	Objetivos	3
☰	¿Qué es un sistema operativo?	4
☰	Estructura de un sistema informático	6
☰	El núcleo (kernel) del S.O.	9
☰	Diferentes estructuras de un sistema	11
☰	Los programas del sistema (S.O.)	15
☰	Ejercicio práctico: CMD y PowerShell	18
☰	Sistemas por lotes ("batch")	23
☰	Sistemas por lotes con multiprogramación	26
☰	Sistemas de tiempo compartido	28
☰	Sistemas distribuidos	29
☰	Funciones del sistema operativo	31
☰	Procesos y multiprogramación	35
☰	Ejercicio práctico: ver los procesos activos en Windows	37
☰	Ejemplos de interfaces de usuario	40
☰	Tipos de sistemas operativos	43
☰	¿Y qué es la BIOS?	47
☰	Software de sistema	49
☰	Software de aplicación	55
☰	Software de programación	59
☰	Resumen	62

Objetivos

En la actualidad existen muchos tipos de sistemas operativos. Quizás los más conocidos a nivel popular sean los sistemas Windows de Microsoft, los Mac de Apple y algo menos los sistemas Linux, descendientes del Unix original.

El conocimiento general de los sistemas más comunes, sus características y un primer acercamiento a su gestión es la intención de este capítulo.

Empezaremos viendo un poco de la arquitectura de un sistema operativo (S.O.) para luego centrarnos en sus funciones y los diferentes tipos existentes.

Los objetivos de la lección son:

- 1 Identificar los elementos funcionales de un sistema informático.
- 2 Ser capaz de analizar las características, funciones y arquitectura de un sistema operativo.
- 3 Comparar diferentes sistemas operativos en lo que se refiere a sus requisitos, características, campos de aplicación y licencias de uso.

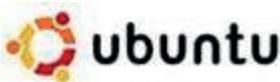
¿Qué es un sistema operativo?

Un **sistema operativo** es un conjunto de programas que sirven de soporte a toda la actividad del usuario sobre la máquina.

Normalmente este software (**sistema operativo**) se instala sobre el hardware del ordenador y es el que nos permite interactuar con él. También hace posible instalar las aplicaciones que queremos ejecutar.

Cualquier aplicación que vayamos a usar (un editor de texto, un programa de cálculo, una base de datos) necesita tener “por debajo” funcionando el sistema operativo.

Nos estamos refiriendo a los ordenadores de propósito general (los que manejamos normalmente) y no a equipos específicos, que pueden funcionar con microprocesadores y un software en código máquina instalado directamente (sin la presencia de un S.O.), como por ejemplo un equipo de medida, un controlador de temperatura, un teléfono, etc. Pero incluso muchos de ellos, aunque sean terminales de pequeño tamaño, ya incorporan un sistema operativo sobre el que podemos instalar aplicaciones. El ejemplo más claro puede estar en los modernos “smartphones”.



- i** La variedad de sistemas operativos en la actualidad es enorme, con muchas variantes, pero si tuviésemos que establecer una división general podríamos pensar en: sistemas “Windows” (Microsoft), el mundo “Apple”, el entorno de software libre liderado por todas las distribuciones de Unix/Linux, y en el caso de los smartphones tenemos que mencionar los sistemas “Android”.

Estructura de un sistema informático

Podemos ver a cualquier sistema informático de propósito general como una arquitectura que consta de una serie de elementos, de los cuales el más importante es el sistema operativo.



Funciones básicas del sistema operativo

- Hacer **accesible** el **hardware (HW)** de la máquina a los usuarios y sus aplicaciones.
- **Gestionar los recursos del sistema**, es decir:
 - Acceso controlado a cualquier recurso: CPU, memoria, dispositivos E/S, etc.
 - Compartir recursos, permitiendo su utilización por múltiples procesos de aplicación.
 - Optimizar la utilización de los recursos (memoria, discos, puertos, registros, etc.).
 - Controlar que los dispositivos que se instalan o conectan al sistema son adecuados y funcionan con los protocolos necesarios.

- Supervisar la ejecución de los procesos para evitar errores o un mal uso que pueda causar daño.
- Permitir la comunicación del equipo con el exterior a través de sus distintas interfaces (red local, WIFI, etc.)
- Hacer que sea fiable la ejecución de las aplicaciones y controlar que se ejecuten adecuadamente.
- Hacer “amigable” para el usuario la operación del sistema, normalmente a través de interfaces gráficas (GUI = Graphic User Interface).
- Efectuar labores de mantenimiento y actualización, de forma automática o bajo petición del usuario, como por ejemplo la instalación de nuevo HW y SW, etc.
- Proporcionar un conjunto básico de funcionalidades y aplicaciones incluidas en el propio S.O., generalmente para administración del propio sistema, pero también como elementos básicos de funcionamiento. Por ejemplo, la mayoría ya llevan incorporado un navegador de Internet, algún editor de texto, etc.

Estructura del sistema

El sistema operativo (S.O.) es un conjunto de elementos software (SW) que están organizados siguiendo una arquitectura de funcionamiento. Aunque cada S.O. tiene su propia arquitectura, podemos identificar en todos las siguientes partes (subsistemas):



Núcleo del S.O. (*kernel*): controla la ejecución de los programas y el acceso al HW de la máquina. Proporciona los servicios básicos del S.O. al resto de elementos. Está siempre ejecutándose mientras la máquina esté encendida.

Programas del S.O. : en conjunto ofrecen una interfaz de acceso a los usuarios, tanto para la ejecución de comandos (a través de la “*shell*”) como una interfaz de acceso para aplicaciones (API o *Application Programming Interface*), y estos a su vez se apoyan sobre los diferentes servicios del S.O.

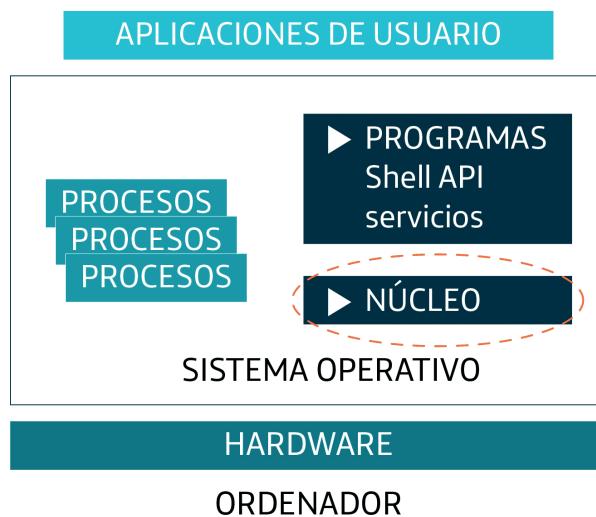
El núcleo (kernel) del S.O.

También se conoce como el “kernel” del S.O. y **gestiona directamente los recursos HW del sistema**, encargándose de las funciones más básicas, como el acceso al HW (memoria, buses internos de comunicaciones, asignación de tiempos, etc.).

Cuando el núcleo inicia la ejecución de un programa decimos que crea un “proceso” (que no es más que eso, un proceso es un programa en ejecución), y puede crear varias “instancias” (ejecuciones) de un mismo programa, es decir, varios procesos.

La ejecución de procesos por el núcleo del S.O. normalmente se hace en modo “privilegiado” o “supervisor” (también llamado modo *kernel*). El resto de programas se ejecutarán también a través de procesos, pero con un nivel de permisos menor.

Esto quiere decir que habrá acciones que solamente podrán ser ejecutadas por el núcleo y que estarán prohibidas para el resto de procesos normales de aplicación. Además, el núcleo siempre podrá ejercer el control sobre cualquier proceso que esté ejecutándose en nuestra máquina.



Algunas funciones del núcleo (*kernel*) del S.O. son, por ejemplo:

CONTROL GENERAL DEL SISTEMA

- Gestión del tiempo de uso del procesador (CPU).
- Gestión de la memoria del sistema (asignación, distribución, etc.).
- Gestión del almacenamiento de programas en el disco.
- Carga de programas en memoria para su ejecución.
- Control de errores y registro de eventos.
- Seguridad del sistema: control de acceso, autenticación de usuarios, privilegios de ejecución, etc.

SERVICIO A USUARIO Y PROGRAMAS

- Ejecución de programas (a través de la creación de procesos).
- Permitir el acceso a los dispositivos de entrada salida (E/S): discos, impresoras, teclado, monitor, redes, periféricos...
- Gestión de acceso al sistema de archivos.
- Gestionar sus comunicaciones externas hacia las redes (p.ej. Internet).
- Gestionar las comunicaciones internas entre los procesos en ejecución.
- Gestión de usuarios, grupos, privilegios de cada grupo, etc.

Diferentes estructuras de un sistema

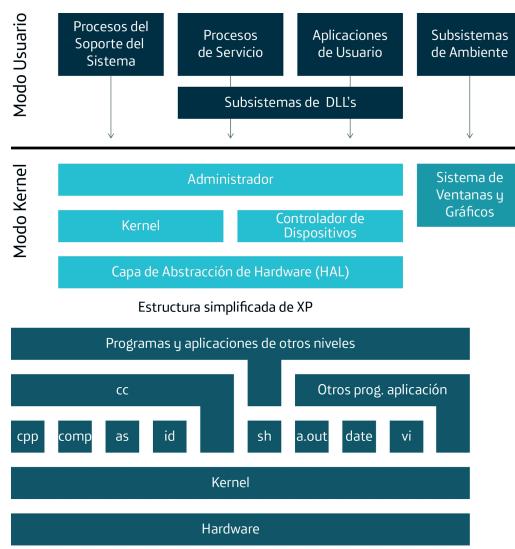
Los sistemas operativos han evolucionado en su estructura interna a lo largo de la historia, bien para aprovechar la capacidad creciente de ejecución proporcionada por el HW, la velocidad de conexión de las redes, etc. De esta forma podemos encontrar diferentes arquitecturas.

Estructura monolítica

En esta arquitectura el S.O. está construido como **un solo programa binario, ejecutable y muy extenso**, que se ejecuta en modo *kernel* (con privilegios de ejecución propios del núcleo).

En un sistema monolítico cualquier procedimiento del sistema es visible para los demás y puede comunicarse con cualquier otro. Existen cientos de procedimientos a realizar, por lo que el sistema suele resultar difícil de manejar.

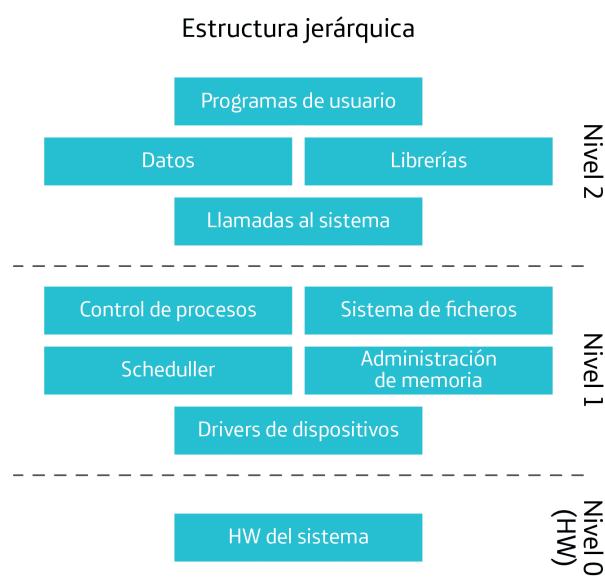
Ejemplos de este tipo pueden ser MS-DOS, UNIX / Linux y la familia Windows.



Arquitectura del Sistema Unix

Estructura jerárquica

Otra opción, en vez de tener un sistema monolítico, es **dividir las funciones en componentes (módulos) de menor tamaño**, cada uno de ellos con sus funciones bien definidas, con sus entradas y salidas determinadas. El nivel más bajo será el propio HW del equipo, y el más alto la interfaz de usuario, después cada S.O. puede tener más o menos niveles. En cada nivel habrá un conjunto de rutinas que manejen un conjunto de datos para realizar las funciones de ese nivel.



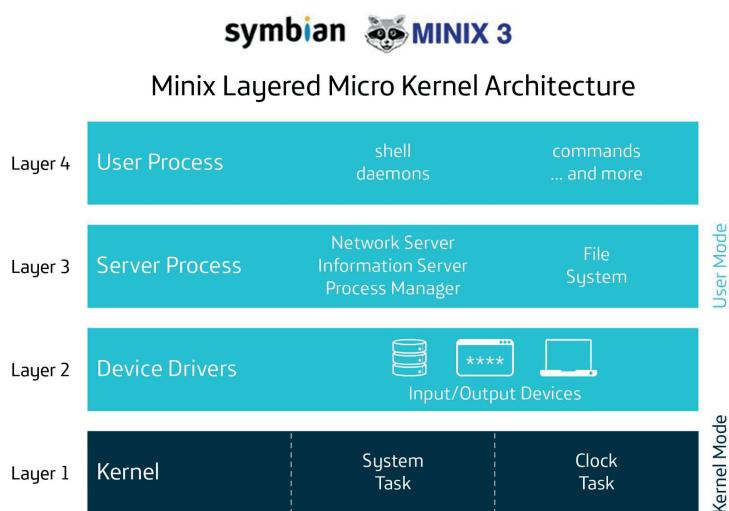
Capas o anillos

Cuando **cada nivel “envuelve” al nivel inferior y proporciona un único acceso a los niveles superiores** podemos considerar que el sistema está estructurado en capas en forma de **“anillo”** (“ring”), que se van englobando desde la más externa (la que da servicio al usuario) hasta la más profunda (la que gobierna el HW del sistema) y viceversa.



Microkernels

Son sistemas orientados a lograr una **alta fiabilidad** y suelen ser diseñados para equipos con altos requerimientos de seguridad. **La idea es hacer que la parte del sistema que corre en modo kernel sea la menor posible**, dado que los errores en este tipo de operación pueden afectar gravemente al sistema. Se trata de **dividir al S.O. en pequeños módulos bien definidos, y solamente uno de ellos se ejecutará en modo kernel**, y a ese se le conoce como **microkernel**. El resto de los módulos se ejecutarán como procesos de usuario normales sin privilegios especiales, con lo que su impacto en caso de fallo será menor.



Máquinas virtuales

Una máquina virtual **es un SW que ejecuta instrucciones y simula una máquina física** con su funcionalidad completa hacia los usuarios/aplicaciones. De esta forma, sobre una misma máquina física (ordenador) podemos proporcionar varios entornos de ejecución diferentes, compartiendo los recursos reales disponibles y dando la apariencia a cada máquina virtual de usar todo el sistema y además, y no menos importante, también sirve para proteger al S.O. anfitrión de las acciones del usuario, que no tienen efecto más allá del entorno de la máquina virtual.

Ejemplos de máquinas virtuales son: VmWare, VirtualBox, Virtual PC, o la más conocida JVM (Java Virtual Machine).



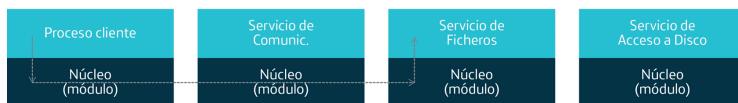
Cliente/servidor

La modularización de los componentes de los S.O. hace que cuando un determinado programa en ejecución (proceso) requiera de los servicios de otro, sea necesario la comunicación entre ambos para que el primero (**cliente**) le pida la ejecución del servicio al segundo (**servidor**). Esta filosofía es similar a la que conocemos como cliente/servidor en las redes, pero en este caso en el ámbito de los procesos que se ejecutan en un S.O. Las funciones cruciales para la máquina seguirán estando normalmente bajo control del propio *kernel*.

Proceso cliente y servidor con núcleo monolítico



Proceso cliente y servidor con núcleo modular y distribuido



Los programas del sistema (S.O.)

Cada sistema operativo contiene, además del *kernel*, una serie de componentes software que permiten ofrecer los servicios al usuario.

Por ejemplo proporcionando una interfaz para las aplicaciones (por eso se compilan para ser ejecutadas sobre Windows, Unix, etc.

y al instalarlas nos preguntan) y un entorno gráfico amigable para el usuario (normalmente a través de un entorno de ventanas).

Aunque el conjunto de servicios y componentes pueden variar de un S.O. a otro, en general encontraremos en todos ellos:

- Una interfaz a través de la cual introducir órdenes (comandos), el intérprete de mandatos (*shell*) o **interfaz de línea de comandos** (CLI = Command Line Interface).
- Un **entorno gráfico** de ventanas sobre un “escritorio” (GUI = Graphical User Interface).
- Un **sistema de ejecución por lotes** “batch” (que ya veremos más adelante).
- **Utilidades de gestión y administración** del sistema: gestión de archivos, guías de ayuda para la instalación/desinstalación de aplicaciones, etc.
- **Utilidades de protección** del sistema y de recuperación en caso de fallo.
- **Programas de soporte** para la programación y el desarrollo SW (compiladores, enlazadores, depuradores, etc.).



Esta capa de servicios ofrecerá a los procesos en ejecución una serie de posibilidades de interacción con ellos, propias de cada S.O., y a las que conocemos como "**llamadas al sistema**".

Estas permiten a los procesos en ejecución una serie de funcionalidades determinadas, y su conjunto permite a las aplicaciones interactuar con el S.O., y es lo que se conoce como **API** ("Application Programming Interface").

Independientemente de las aplicaciones que pueda utilizar, el usuario también puede introducir comandos (órdenes) para el S.O. de forma directa, a través del intérprete de mandatos (CLI) o "**shell**". Esta *shell* también es propia de cada S.O., pero varios de ellos pueden usar una misma *shell* o puede ser muy similar entre aquellos sistemas de la misma familia. Los comandos introducidos pueden ser órdenes internas para el S.O. o la llamada a un programa externo que se encuentre almacenado en el disco duro.

Además, un mismo S.O. puede soportar varias *shell* (intérpretes de mandatos), y en ellos es posible elegir la que el usuario quiere utilizar. Esta *shell* que recibe, interpreta y ejecuta las órdenes que introduce el usuario a veces no se considera parte del S.O., pero aquí la incluiremos y estudiaremos, puesto que ofrece servicios de diálogo al usuario y permite actuar directamente sobre el S.O. y los procesos en ejecución.

Ejemplo de acciones que podemos realizar como usuario a través de la interfaz del sistema:

- **Gestión de programas y procesos** , permitiendo cargar, ejecutar, detener, priorizar... procesos de aplicación.
- Utilizar programas para **desarrollo y programación** : compiladores, enlazadores, depuradores, bibliotecas de funciones para un determinado lenguaje etc.
- Gestionar **archivos/ficheros**: guardar, borrar, mover, etc., manejar ficheros en general.
- **Configurar protocolos de comunicaciones** : pilas de protocolos TCP/IP y su entorno.
- Utilizar **programas y aplicaciones clientes** : clientes de correo, navegadores de Internet, clientes FTP o Telnet, etc.
- Usar **herramientas de administración** , instalación y modificación del SW y del HW del sistema. Por ejemplo todo lo que tiene que ver con la administración de usuarios, permisos, perfiles y privilegios, etc.
- Utilizar otras **herramientas y accesorios** como el reloj, la calculadora, juegos, etc.
- Cualquier otro componente que el fabricante del S.O. nos proporcione (como simuladores, máquinas virtuales, etc.).

Ejercicio práctico: CMD y PowerShell

Importante: por favor límítate a realizar solamente lo que te pedimos en este ejercicio.

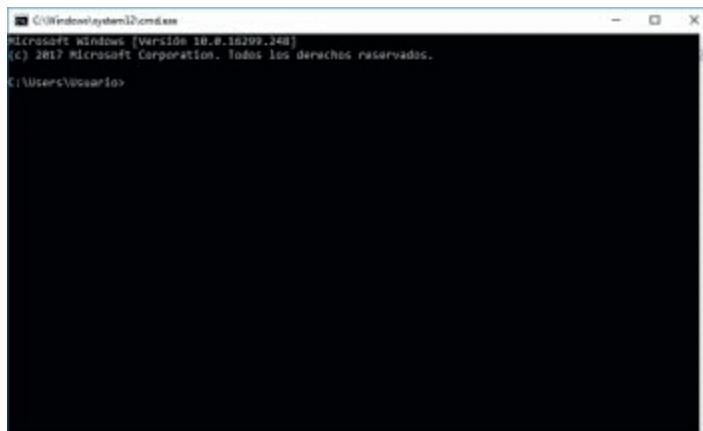
Estas consolas de comandos son herramientas muy potentes, pero introducir órdenes inadecuadas puede dañar tu sistema operativo.

Acabamos de ver que un mismo sistema operativo puede tener varias "shell" (intérpretes de mandatos), esto es muy común en el entorno Unix/Linux, pero vamos a ver un ejemplo en entorno Windows.

Ya desde los tiempos de MS-DOS tenemos disponible en entorno Windows la "consola de comandos" (cmd), pero lo que mucha gente no sabe es que también, desde hace años, Microsoft ha dotado a sus sistemas de otra *shell* más potente: Windows PowerShell.

No pretendemos que ahora aprendas a manejarlas en profundidad, simplemente que conozcas que existen, que las veas como un ejemplo de que en un mismo S.O. puede haber varios intérpretes de mandatos, y que te vayas familiarizando con ellas y que puedas visualizarlas en tu equipo. En este curso emplearemos sobre todo la consola de comandos, pero iremos aprendiendo sobre ella poco a poco.

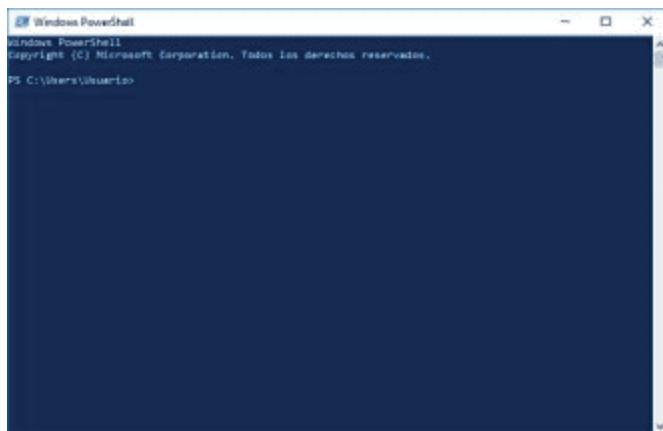
CMD y PowerShell, ¿cuál es la diferencia?



Consola de comandos de Windows.

En primer lugar la **consola de comandos** ("*cmd*") es más antigua, y su funcionalidad nos permite introducir en el sistema órdenes a través de un repertorio de comandos alfanuméricos, y también a través de ficheros donde hayamos grabado un conjunto de estas órdenes (lo que llamamos "*script*" en general y tienen la extensión ".bat" normalmente). A través de estas órdenes podemos realizar muchas tareas, que pueden ir desde copiar ficheros, crear o borrar directorios hasta configurar los parámetros del sistema, visualizar sus características, etc.

Sobre **PowerShell** te diremos simplemente que es una herramienta más reciente (aunque ya lleva años disponible), mucho más potente que la consola *cmd*, y que también necesita que el S.O. tenga "por detrás" algunos recursos más complejos (pero que ahora no vienen al caso). La principal diferencia es que PowerShell permite realizar una programación bastante más compleja que "*cmd*", y también que además de su repertorio de órdenes y comandos nos da la posibilidad de trabajar con programas de aplicación de Microsoft (como MS SQL Server, MS Exchange, etc.) y trabajar a nivel de servidor.



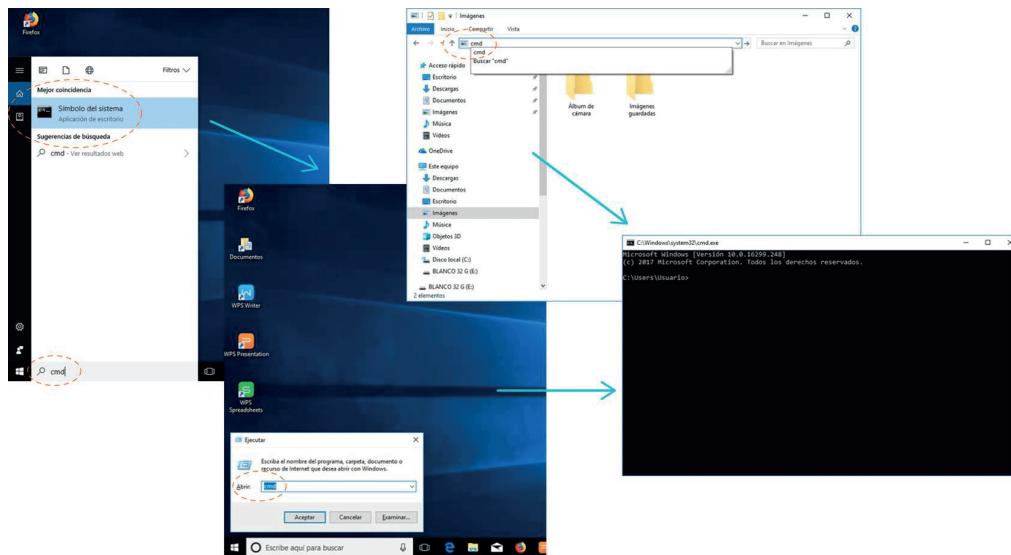
Windows PowerShell.

Ahora mismo solamente te las vamos a mostrar y te pedimos como ejercicio que las visualices e introduzcas un par de comandos.

Abrir la consola de comandos "cmd"

Si estás en Windows 10, tenemos varias opciones:

- Escribe "cmd" en el cuadro de búsqueda del sistema y te ofrecerá la opción de activarla.
- También podemos dar "*tecla de windows + R*" y nos aparece una ventana donde podemos escribir "cmd" y ejecutar.
- O bien desde cualquier ventana del administrador de archivos ponemos "cmd" en la barra de direcciones y damos *enter*.



Como ves, "hay muchos caminos que llevan a Roma", y la conclusión que queremos que recuerdes es:

¡En un sistema operativo puede haber varios métodos o procedimientos para realizar una misma acción!

Una vez que tienes la consola de comandos abierta simplemente introduce los siguientes comandos en orden (los tecleas y pulsas "enter") y fíjate en el resultado (el informe) que te devuelve el sistema:

- systeminfo
- cls
- dir
- ipconfig
- cls
- exit

Habrás visto que el último comando cierra la ventana de la consola. No te preocunes, ahora solo queríamos que vieses el funcionamiento, iremos profundizando a lo largo del curso.

Te mostramos el [ejemplo en un vídeo.](#)

Abrir Windows PowerShell

Para abrir este *shell* te mostraremos solo un método: escribe "Power" en el cuadro de búsqueda de Windows y verás cómo se despliega la opción de activar PowerShell, dale clic con el ratón y te aparecerá la ventana.

¿Ves alguna diferencia con la consola de comandos tradicional?

Introduce el comando: "`get-command + enter`".

Te saldrá un informe con un montón de líneas, que son los comandos disponibles en PowerShell. Te los mostramos solamente para que te des idea de su potencia, pero **NO DEBES INTRODUCIR NINGUNO DE ELLOS** en este momento. Ya lo harás más adelante en un entorno seguro que crearemos.

Introduce el comando: "`exit + enter`". Verás que se cierra la *shell*.

Como antes, te lo mostramos [en un vídeo.](#)

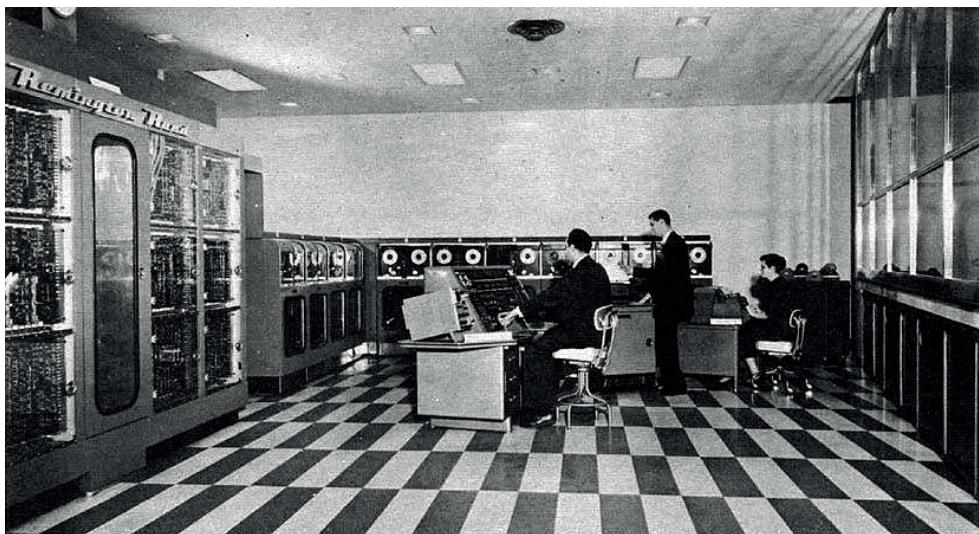
Sistemas por lotes ("batch")

Cuando interactuamos con el ordenador y le pedimos que ejecute procesos (programas en ejecución), básicamente podemos hacerlo de dos formas:

- De **forma interactiva**, ejecutándolo bajo nuestra supervisión.
- Ordenando su ejecución de **forma autónoma** por la máquina, sin nuestro control directo durante la ejecución.

A esta segunda forma de ejecución es a lo que se conoce como "**ejecución por lotes**" (o "procesamiento por lotes" = "*batch processing*"), y la forma de ordenar la ejecución de las tareas suele ser a través de ficheros de órdenes ("**scripts**"), en los que especificamos al ordenador lo que deseamos que ejecute y sobre qué recursos y dispositivos.

Realmente cualquier programa podría ejecutarse de esta forma si podemos pasarle los parámetros necesarios para su operación. Quizás conozcas los archivos “.bat”, que son ficheros con un conjunto de órdenes secuenciales que podemos ejecutar ordenando al S.O. la ejecución del archivo. Sin embargo, cuando se habla de “procesamiento por lotes” podemos referirnos también a la realización de tareas agrupadas y de forma automática ordenadas por un programa o por el propio S.O., y que normalmente son tareas repetitivas o que se almacenan temporalmente para luego enviarlas a un servidor, por ejemplo.



UNIVAC -1107

FUENTE: <https://commons.wikimedia.org/wiki/File:UNIVAC-I-BRL61-0977.jpg>

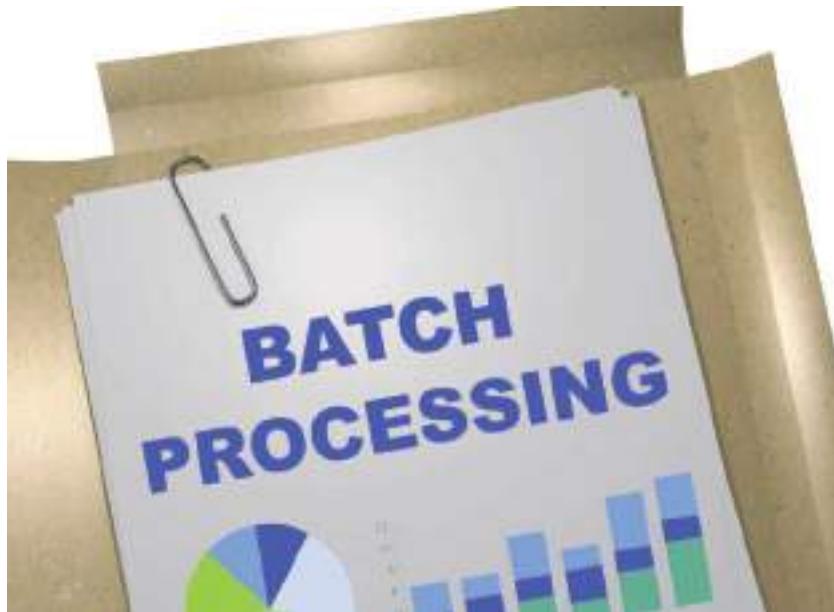
i Este tipo de procesamiento era muy común en los grandes ordenadores “mainframes”. Ejemplos de S.O. diseñados para funcionar por lotes fue el “EXEC II”, desarrollado para el ordenador UNIVAC -1107, o también el S.O. SCOPE de la CDC6600.

Los sistemas operativos con procesamiento por lotes (**sistemas “batch”**) aparecieron en los primeros tiempos (años 50) para aprovechar la capacidad de procesamiento aunando tareas similares y ejecutándolas de forma agrupada.

En estos sistemas el usuario no tenía una interfaz directa con el procesador, sino que preparaba las órdenes y las introducía a través de un sistema de E/S (entrada/salida), como por ejemplo un lector de tarjetas perforadas, y luego dejaba al sistema que lo realizase de forma autónoma y le devolviese el resultado cuando lo hubiese terminado. Cuando el sistema trabaja en modo “batch”, al programar la ejecución de las tareas es necesario planificar cuidadosamente cómo van a realizarse, pues no vamos a estar “presentes” mientras se están ejecutando.

El procesamiento por lotes se emplea normalmente para **tareas voluminosas y repetitivas**, que requieren mucho tiempo para su ejecución, o que no deben ser interrumpidas.

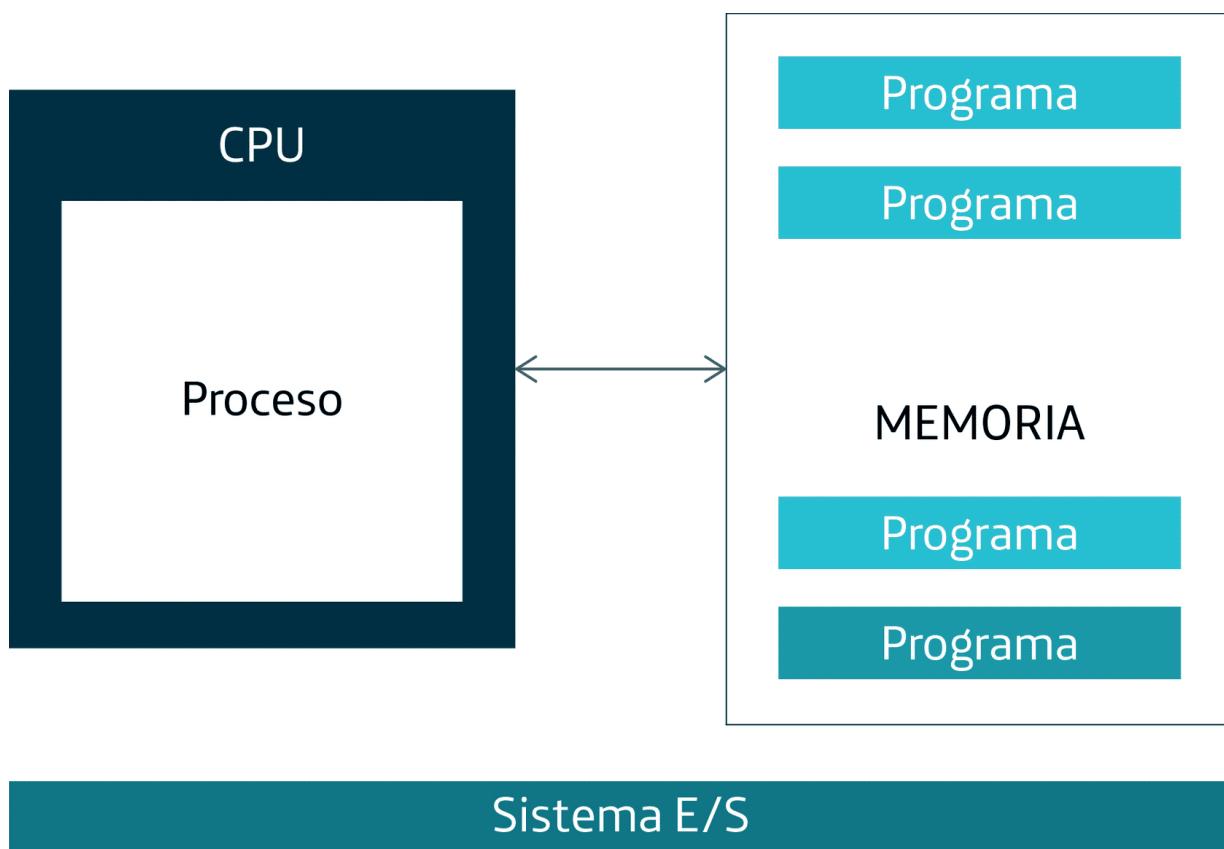
Un ejemplo de tarea típica para realizar en modo “batch” puede ser el cálculo y generación de las nóminas de una empresa, o la ejecución de un programa de cálculo estadístico o de acceso a múltiples bases de datos y que deba hacerse en franjas de poca actividad.



Sistemas por lotes con multiprogramación

Sistemas “batch” con multiprogramación

Unos años más tarde, después de los primeros sistemas de procesamiento por lotes (*batch*), en los que las tareas a realizar se cargaban en memoria y ejecutaban secuencialmente (cuando acaba una se cargaba la siguiente), aparecieron los **sistemas “batch” con multiprogramación**, en los cuales en la memoria del sistema se podían cargar varias tareas de forma simultánea, y la ejecución se realizaba distribuyendo el tiempo de CPU entre ellas.



Esta nueva forma de ejecución implicaba:

- **Administrar la memoria** del sistema para permitir reservar espacio para varios programas y sus datos.
- **Administrar el tiempo de la CPU** para dividirlo entre las tareas a ejecutar de forma concurrente.
- **Administrar los dispositivos de E/S (entrada/salida)** para poder ser usados por la tarea en ejecución en cada momento.

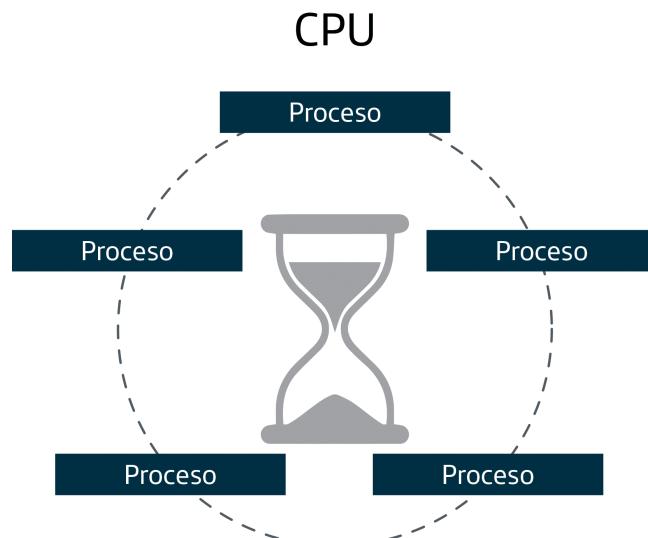
Sin embargo, todavía estos sistemas solían recibir la carga de los programas a través de cintas o tarjetas perforadas, la cual muchas veces era realizada por un operador y no directamente por el usuario de los programas.

Sistemas de tiempo compartido

En los años 60 aparecieron los sistemas que funcionaban “**a tiempo compartido**”, en los cuales la CPU, además de repartir su tiempo entre varias tareas, podía administrarlas cargándolas en memoria o devolviéndolas a disco, según hiciese falta.

Estos sistemas normalmente ya tenían una interfaz con el usuario a través de un teclado que permitía la interacción en línea y la inserción directa de comandos. También eran conocidos como sistemas “**multitarea**” (aunque recordemos que **una sola CPU solamente puede estar ejecutando una tarea al mismo tiempo**) porque como la CPU “comutaba” entre los trabajos con una frecuencia elevada, al usuario podía darle la impresión de tenerla dedicada para su tarea.

Estos sistemas requerían de un “**planificador**” (“*scheduller*”) del tiempo de la CPU que lo distribuyese entre los diferentes usuarios/programas. Una de sus principales ventajas era la reducción del tiempo de inactividad del procesador.



Sistemas distribuidos

Un “**sistema distribuido**” realiza la ejecución de las tareas distribuyendo el esfuerzo de su ejecución entre varios procesadores (CPU) dispersos en varios equipos, incluso geográficamente distantes.

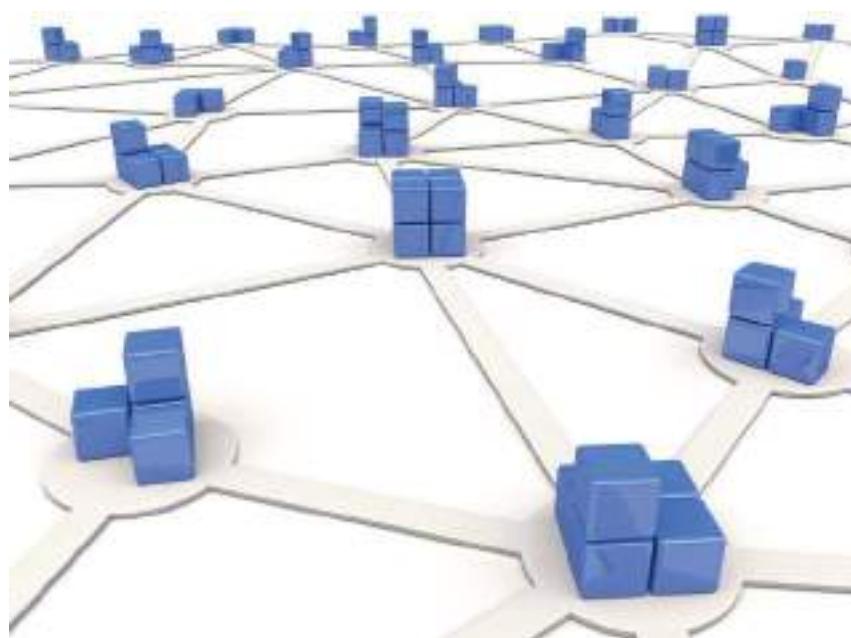
Cada uno de los procesadores puede tener toda la capacidad de ejecución (memoria, recursos de E/S a disco, etc.) y para actuar de forma coordinada con los demás necesita algún medio de comunicación (red, bus, línea telefónica, etc.).

Las principales ventajas de un sistema distribuido son:

- Se comparten los recursos entre varios procesadores.
- Al distribuirse la carga se aumenta la velocidad de ejecución de los trabajos. Esto también puede ser una ventaja si uno de los procesadores se encuentra sobrecargado y puede transferir parte de sus tareas a otro procesador del sistema.
- Puede aumentar la fiabilidad frente al fallo de alguno de los procesadores; sus tareas pueden ser asumidas por otro. Esto es lo que se conoce como funcionamiento “activo-reserva”.

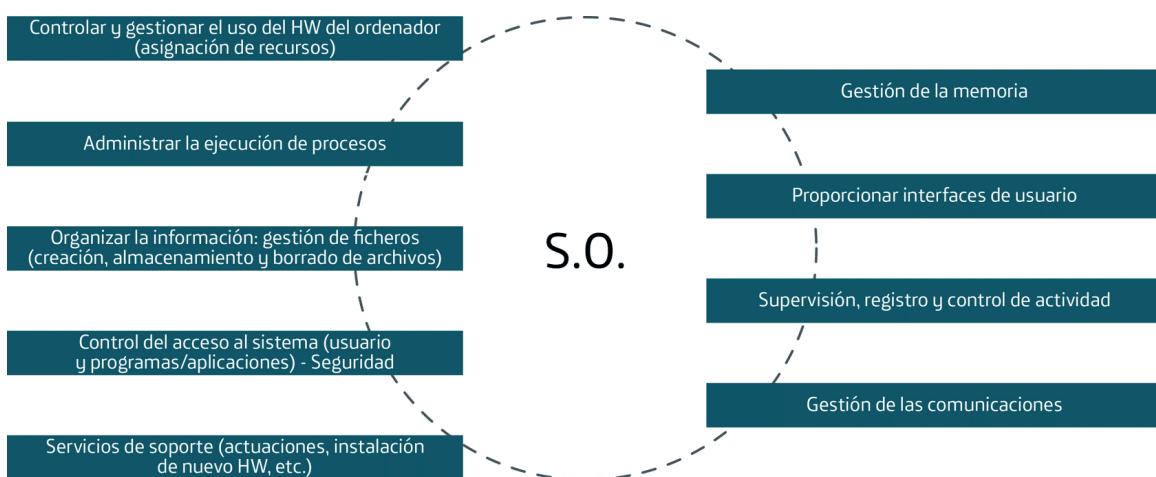
Existen básicamente dos estrategias de gestión de un sistema distribuido: **control centralizado** y **control distribuido**, según haya un nodo principal de control de todo el sistema o bien cada nodo disponga de sus propios mecanismos de control y mantenimiento.

Como comentamos, la existencia de procesos distribuidos y separados en diferentes procesadores implica que debe haber un sistema de comunicaciones, que puede ser una infraestructura más o menos compleja (una pequeña red local, una WAN, Internet) y las máquinas del sistema distribuido deben disponer de un acceso a ella.



Funciones del sistema operativo

Sabemos que un S.O. proporciona el entorno adecuado para ejecutar tareas (programas), y esto lo hace a través de una serie de servicios y funcionalidades que podemos resumir en:



Funciones del sistema operativo

Controlar y gestionar el uso del HW del ordenador (asignación de recursos)

En cualquier máquina (ordenador) los recursos disponibles son limitados. Será **el sistema operativo el encargado de distribuir la capacidad existente entre todos los usuarios y programas** que se encuentren utilizándolo en un determinado momento, asegurando un buen uso de esos recursos y evitando bloqueos o pérdidas de rendimiento .

Cada **proceso** (programa) requerirá la asignación de algunos recursos HW del sistema: espacio en disco o en memoria, puertos de comunicaciones, periféricos, etc. Es el S.O. el encargado de que sean compartidos de forma adecuada y de ir asignándolos según las necesidades. Las operaciones sobre los dispositivos de E/S no son realizadas directamente por las aplicaciones, sino que pasan a través del S.O., que las ejecuta a través de sus **rutinas específicas de E/S** y de los **manejadores de dispositivos** ("drivers").

Administrar la ejecución de procesos

El S.O. será el responsable de **cargar en la memoria principal el código de los programas** a ejecutar (todo o en parte) y **distribuirá el tiempo de uso de la CPU** entre todos aquellos procesos (programas en ejecución) que se encuentren activos en un determinado instante.

El S.O. dispone de complejos **algoritmos de planificación** y también se encarga de **establecer prioridades entre los procesos** en ejecución (algunos se ejecutarán con privilegios del “*kernel*” y otros como simples aplicaciones de usuario).

Organizar la información: gestión de ficheros

La **gestión del almacenamiento** es también una de las funciones principales a realizar por cualquier S.O. Esta gestión incluye la creación, modificación, movimiento y borrado de los archivos de información en los múltiples formatos usados por las aplicaciones.

La gestión de los archivos normalmente se basa en el establecimiento de un “sistema” (organización) con una **estructura jerárquica de directorios o carpetas**, cada una con su nombre identificativo, dentro de las cuales se depositan los ficheros de código o datos. Llamamos “**ruta**” a través de la cual accedemos a un determinado archivo a la secuencia de carpetas/directorios a recorrer para llegar a la carpeta final que lo contiene.

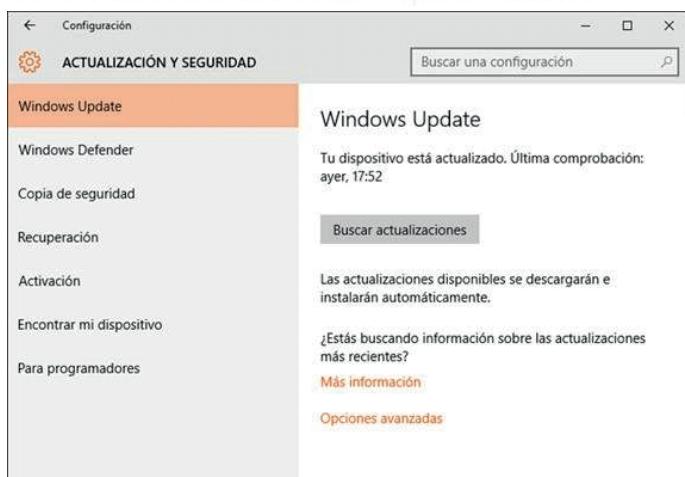
Control de la actividad y el acceso al sistema

Una primera medida para asegurar este control es que los procesos de aplicación se ejecuten en modo "normal" (**modo usuario**) con acceso limitado a los recursos, y los pertenecientes al S.O. lo hagan en modo "supervisor" (o modo "*kernel*") con acceso prácticamente ilimitado a los recursos del sistema.

Además, el servicio de seguridad del S.O. se ocupa también de evitar accesos de usuario no permitidos, para lo cual se establece, por ejemplo, un **control de sesión y autenticación** de los usuarios con una **clave** (“*password*”) asignada, y se controlan los dispositivos de E/S a través de los cuales se podrían producir ataques.

Servicios de soporte

Todos los S.O. modernos incorporan **servicios de soporte y actualización**, bien para modificar la configuración del sistema si instalamos algún nuevo HW (y necesitamos el driver asociado) o para mantener actualizado el propio S.O. e ir cargando las sucesivas mejoras que nos proporciona el fabricante.

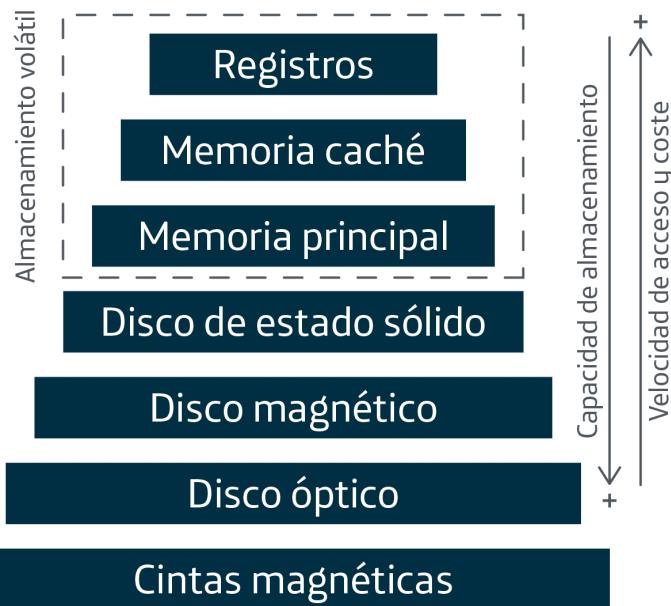


Gestión de la memoria

Para poder ejecutar un programa en el procesador **debe estar cargado (al menos una parte)** en **la memoria principal** del sistema (memoria **RAM**). Pero no podemos tener siempre **todo** el código en la memoria RAM, básicamente porque:

- Por mucha memoria RAM que tenga nuestro equipo, no será capaz de almacenar todos los posibles programas y datos que podemos ejecutar en la máquina.
- La RAM es un dispositivo de almacenamiento volátil, es decir, que pierde su contenido cuando se apaga el sistema o se queda sin alimentación.

Lo anterior hace necesario disponer de **elementos de almacenamiento no volátiles** (discos, cintas, etc.) y a una continua gestión de carga, descarga y sustitución de código y datos en memoria (“**swapping**”) según las necesidades del sistema. Esta labor es dirigida y gestionada también por el S.O.



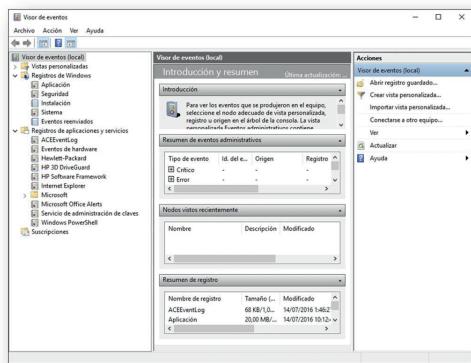
Proporcionar interfaces de usuario

Las interfaces de usuario (UI = "user interface") pueden ser básicamente de varios tipos:

- **Interfaz de proceso por lotes:** las órdenes se introducen mediante archivos de comandos que luego ejecuta el sistema de forma inmediata o programada. Eran comunes en los primeros sistemas, a través de lectores de cinta o tarjetas perforadas.
- **Interfaces en modo texto:** los CLI o "command line interface", interfaz por línea de comandos que admite órdenes en modo texto (alfanumérico) y combinaciones de teclas con significado especial (p. ej. "ctrl+c", "ctrl+x").
- **Interfaces gráficas: GUI ("graphical user interface")**, interfaces gráficas de usuario basadas normalmente en un entorno de ventanas y menús desplegables, sobre los que podemos actuar con el ratón y con el teclado para introducir texto cuando es necesario, o directamente en las modernas pantallas táctiles.

Supervisión, registro y control de actividad

Los S.O. modernos normalmente también incorporan **mecanismos de supervisión, control, registro y seguimiento de la actividad** realizada sobre el equipo. Los datos a recoger por el S.O. sobre la actividad realizada pueden configurarse e incluir, por ejemplo, el número y autoría (identidad) de los accesos al sistema, tiempos de utilización, aplicaciones ejecutadas, etc.



Gestión de las comunicaciones

La gestión de las comunicaciones por parte del S.O. se refiere a dos aspectos:

- El servicio que presta el S.O. estableciendo un **método de intercambio de información** tanto **entre/con los procesos** que se encuentran en ejecución en la máquina en un momento dado.
- A la gestión de las **comunicaciones entre dichos procesos y los dispositivos de E/S** con el exterior del sistema (a través de líneas o redes de comunicación).

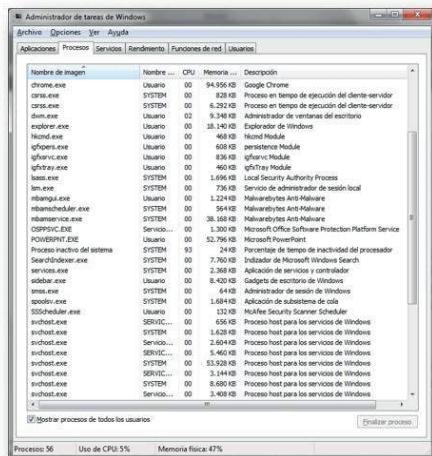
Procesos y multiprogramación

Decimos que un **proceso** es básicamente un programa en ejecución, y que para esa ejecución el proceso tendrá asociados una serie de recursos.

Estos recursos son:

- Un **tiempo de uso de la CPU** para ejecutar sus instrucciones, gestionado por el S.O.
- Un **espacio de memoria** (direcciones) en el que se ejecuta su código, otro para almacenamiento de datos y variables y una zona de pila (*stack*).
- La posibilidad de acceso a **ficheros y dispositivos de E/S** que pueda estar usando en un determinado momento.
- Unos **datos asociados al proceso**, que lo identifican y que son manejados por el S.O. (PID, tamaño en memoria, etc.).

En un momento dado en el sistema estarán coexistiendo los propios procesos del sistema operativo con los procesos de las aplicaciones que se estén ejecutando.



En los sistemas modernos normalmente hay muchos procesos activos en un determinado instante, los cuales comparten los recursos del sistema, pero hemos de tener en cuenta que para cada CPU en un determinado instante de tiempo solamente puede estar ejecutándose una instrucción de un proceso (programa), y es el núcleo del sistema operativo quien se encarga de administrar el tiempo y los recursos que asigna a cada uno de los procesos activos en función de sus necesidades y de la disponibilidad en el sistema.

Concepto de “multiprogramación”

Puesto que cada CPU solamente puede estar ejecutando un proceso (programa en ejecución) en un determinado instante de tiempo, para dar la sensación de que se están ejecutando varios programas a la vez el S.O. procede a ir **asignando tiempo de la CPU cambiando y distribuyéndolo muy rápidamente entre todos los procesos “en ejecución”**, de forma que parece que se ejecutan a la vez.

Esto también es un mecanismo de aprovechamiento de la CPU, pues por ejemplo cuando un programa tenga que hacer una lectura sobre el disco duro del sistema, el tiempo invertido en hacer la petición, realizar la lectura y devolver los datos al programa que los solicitó es un tiempo enorme para las velocidades a las que funcionan las CPU modernas. Por eso, en vez de esperar sin hacer nada a recibir los datos, el S.O. puede retirar ese proceso de ejecución momentáneamente y otorgar ese tiempo de CPU a otro proceso, mientras el primero espera.

Estas commutaciones de procesos en ejecución se realizan a una velocidad muy alta, y es el S.O. el que tiene siempre el control (con el llamado **algoritmo de schedulling**) y asigna los tiempos teniendo en cuenta, tanto las prioridades de los procesos, como otros muchos parámetros, de forma que va dividiendo el tiempo “de uso” de la CPU (tiempo de ejecución real) entre los procesos activos en cada momento.

La impresión que produce al usuario es que todos los procesos se ejecutan a la vez, y además se optimiza el aprovechamiento de los recursos de la máquina. A este mecanismo es a lo que llamamos “**multiprogramación**”.

Ejercicio práctico: ver los procesos activos en Windows

¿Te has preguntado alguna vez cuántos procesos están corriendo (ejecutándose) en tu sistema?

Vamos a verlo.

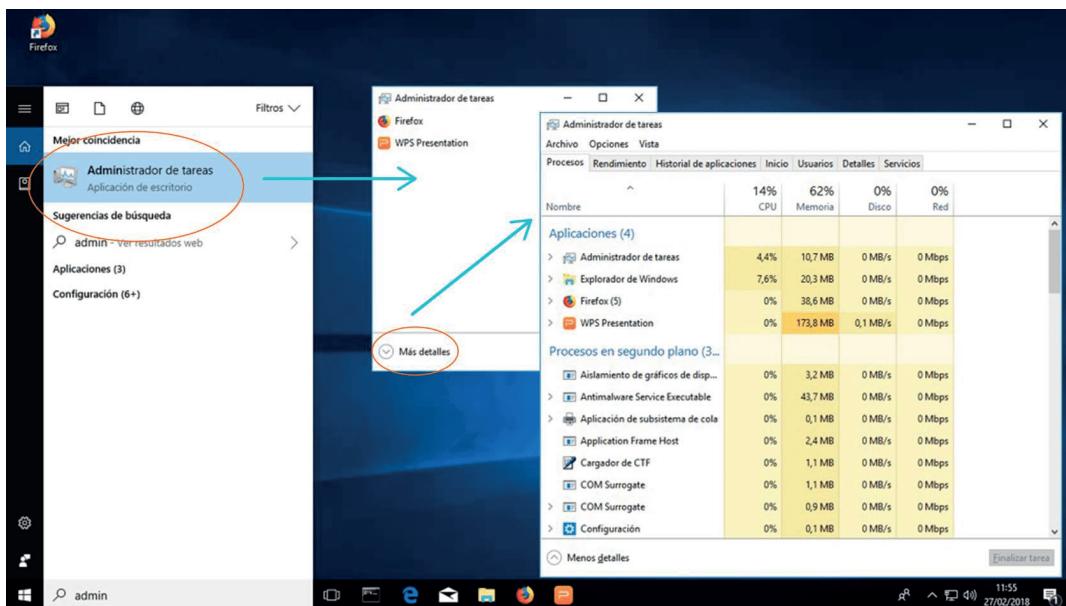
Te vamos a mostrar cómo puedes visualizar los procesos que están presentes y activos en tu sistema en un momento dado. Esto puede resultar útil, por ejemplo, cuando tu ordenador vaya muy lento, podrás ver si hay algún proceso que esté consumiendo muchos recursos (tiempo de CPU, memoria, etc.) o bien si se están realizando muchos accesos al disco, o si la memoria está muy ocupada...

Te mostraremos una forma sencilla de verlo a través el "administrador de tareas" de Windows. De momento se trata solo de visualizar el estado de los procesos y no debes realizar ninguna acción sobre ellos.

El administrador de tareas

Tanto si estás en una u otra versión de Windows, esta herramienta del sistema te permite visualizar los procesos que están corriendo en la máquina y, llegado el caso, actuar sobre ellos (de momento solo vamos a verlos).

Para activar el administrador de tareas puedes ir al cuadro de búsqueda al lado de Inicio y teclear "admin". Verás como antes de que sigas escribiendo te sale un menú con la opción de activar la herramienta de administración de tareas, y haciendo clic sobre ella se activa y aparece la ventana correspondiente.



Administrador de tareas					
Archivo Opciones Vista					
Procesos	Rendimiento	Historial de aplicaciones	Inicio	Usuarios	Detalles
Nombre	14%	62%	0%	0%	0%
	CPU	Memoria	Disco	Red	
Aplicaciones (4)					
> Administrador de tareas	4,4%	10,7 MB	0 MB/s	0 Mbps	
> Explorador de Windows	7,6%	20,3 MB	0 MB/s	0 Mbps	
> Firefox (5)	0%	38,6 MB	0 MB/s	0 Mbps	
> WPS Presentation	0%	173,8 MB	0,1 MB/s	0 Mbps	
Procesos en segundo plano (3...)					
Aislamiento de gráficos de disp...	0%	3,2 MB	0 MB/s	0 Mbps	
Antimalware Service Executable	0%	43,7 MB	0 MB/s	0 Mbps	
Aplicación de subsistema de cola	0%	0,1 MB	0 MB/s	0 Mbps	
Application Frame Host	0%	2,4 MB	0 MB/s	0 Mbps	
Cargador de CTF	0%	1,1 MB	0 MB/s	0 Mbps	
COM Surrogate	0%	1,1 MB	0 MB/s	0 Mbps	
Configuración	0%	0,1 MB	0 MB/s	0 Mbps	

Una vez dentro del administrador de tareas verás que puedes tener una vista con más o menos detalles (se cambia en la parte inferior) y que puedes elegir entre ver información sobre los procesos que están ejecutándose en ese momento (y los recursos que consumen cada uno), o bien información sobre el rendimiento del equipo, el historial de aplicaciones, los programas que arrancan al inicio, los usuarios con sesión iniciada y otros detalles sobre los servicios activos (que **te recomendamos que visualices y explores, pero no modifiques**).

Para salir basta con cerrar la ventana como cualquier otra.

Te mostramos [en un vídeo](#) resumida la actividad que te pedimos, pero tú puedes tomarte tu tiempo para visualizar todo en detalle.

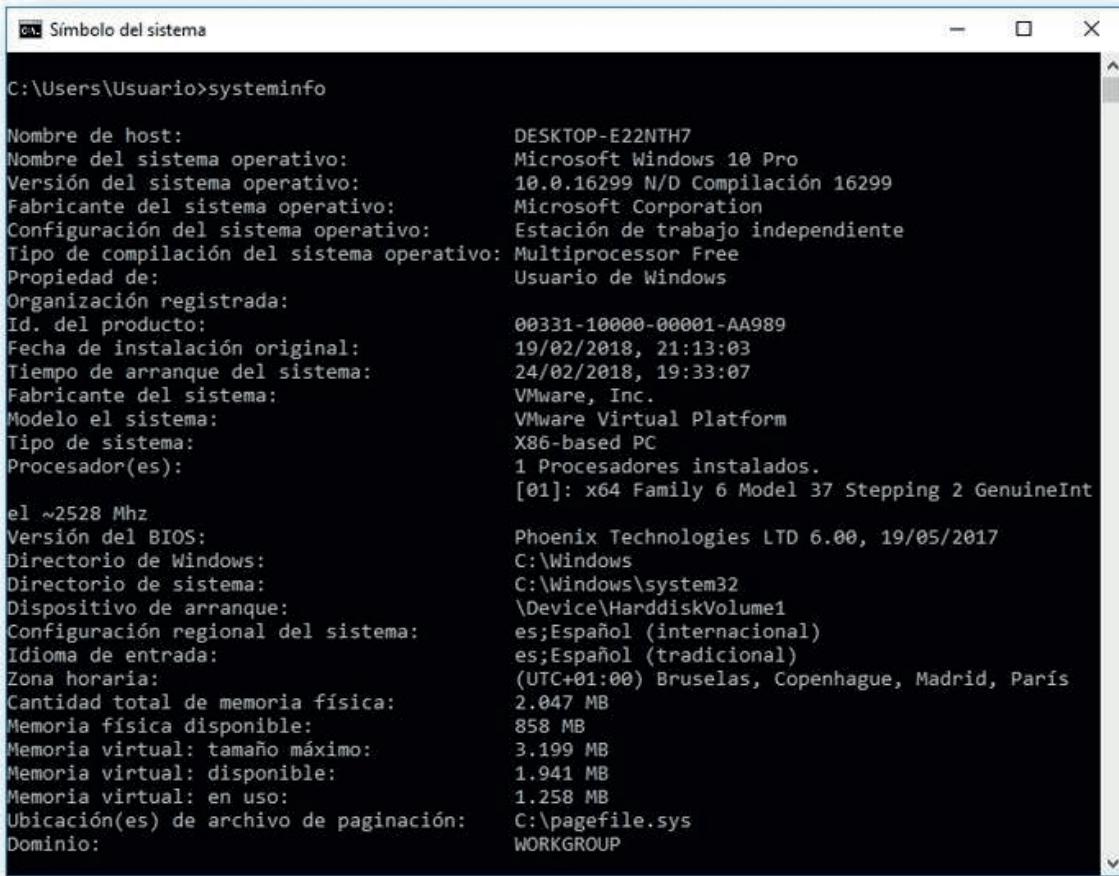
Ejemplos de interfaces de usuario

Ejemplo de CLI: intérprete de mandatos en Windows

La “**consola**” de Windows es el intérprete de mandatos incluido por defecto en la instalación del S.O.

Para invocarlo podemos hacerlo a través del comando "**cmd**", introducido en la línea de comandos del menú "Inicio" o eligiendo "**ejecutar**" e insertando el comando.

Para salir de la interfaz de mandatos simplemente debemos teclear "**exit + enter**" (con ello le damos la orden al S.O. de terminar el proceso).



```
C:\Users\Usuario>systeminfo

Nombre de host: DESKTOP-E22NTH7
Nombre del sistema operativo: Microsoft Windows 10 Pro
Versión del sistema operativo: 10.0.16299 N/D Compilación 16299
Fabricante del sistema operativo: Microsoft Corporation
Configuración del sistema operativo: Estación de trabajo independiente
Tipo de compilación del sistema operativo: Multiprocessor Free
Propiedad de: Usuario de Windows
Organización registrada:
Id. del producto: 00331-10000-00001-AA989
Fecha de instalación original: 19/02/2018, 21:13:03
Tiempo de arranque del sistema: 24/02/2018, 19:33:07
Fabricante del sistema: VMware, Inc.
Modelo el sistema: VMware Virtual Platform
Tipo de sistema: X86-based PC
Procesador(es):
    el ~2528 Mhz
    Versión del BIOS: Phoenix Technologies LTD 6.00, 19/05/2017
    Directorio de Windows: C:\Windows
    Directorio de sistema: C:\Windows\system32
    Dispositivo de arranque: \Device\HarddiskVolume1
    Configuración regional del sistema: es;Español (internacional)
    Idioma de entrada: es;Español (tradicional)
    Zona horaria: (UTC+01:00) Bruselas, Copénague, Madrid, París
    Cantidad total de memoria física: 2.047 MB
    Memoria física disponible: 858 MB
    Memoria virtual: tamaño máximo: 3.199 MB
    Memoria virtual: disponible: 1.941 MB
    Memoria virtual: en uso: 1.258 MB
    Ubicación(es) de archivo de paginación: C:\pagefile.sys
    Dominio: WORKGROUP
```

Ejemplo de CLI: intérprete de mandatos en Unix



```
kermit@kermit: ~ $ pwd
/home/kermit
kermit@kermit: ~ $ cd /usr/portage/app-shells/bash
kermit@kermit: /usr/portage/app-shells/bash $ ls -al
total: 130
drwxr-xr-x  3 portage portage 1024 Jul 25 10:06 .
drwxr-xr-x 30 portage portage 1024 Aug  7 22:39 ..
-rw-r--r--  1 root   root    35800 Jul 25 10:06 ChangeLog
-rw-r--r--  1 root   root    27882 Jul 25 10:06 Ranlibrest
-rw-r--r--  1 portage portage 4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r--  1 portage portage 5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r--  1 portage portage 6151 Apr  5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r--  1 portage portage 5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r--  1 portage portage 5640 Apr  5 14:37 bash-4.0_p10-r1.ebuild
-rw-r--r--  1 portage portage 6230 Apr  5 14:37 bash-4.0_p10.ebuild
-rw-r--r--  1 portage portage 5640 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r--  1 portage portage 5532 Apr  8 10:21 bash-4.0_p17.ebuild
-rw-r--r--  1 portage portage 5660 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r--  1 root   root    5560 Jul 25 09:43 bash-4.0_p25.ebuild
drwxr-xr-x  2 portage portage 2048 May 30 03:35 metadata.xml
-rw-r--r--  1 portage portage 468 Feb  9 04:35 metadata.xml
kermit@kermit: /usr/portage/app-shells/bash $ cat metadata.xml
xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/xml/pkgmetadata.dtd">
<pkgmetadata>
  <herd>basic-system</herd>
  <use>
    <flag name="bashlogger">Log ALL commands typed into bash; should ONLY be
      used in restricted environments such as nonroot</flag>
    <flag name="net">Enable /dev/tcp/host/port redirection</flag>
    <flag name="plugins">Rdt support for loading builtins at runtime via
      'enable'</flag>
  </use>
</pkgmetadata>
kermit@kermit: /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
```

En Unix las “*shell*” (intérprete de mandatos/comandos) pueden ser varias, en la figura se muestra un ejemplo de la *shell* “*bash*”.

Existen otras *shell* dentro del entorno Unix, como por ejemplo:

- Sh
- Ksh
- Csh
- Tcsh
- Bash
- Zsh

Es posible también utilizar los comandos disponibles para la *shell* para construir archivos de comandos (“scripts”) para ser ejecutados como si se tratase de programas de aplicación, y por esta razón a veces se habla de la posibilidad de realizar “**programación shell**” (“*shell programming*”).

Ejemplos de GUI (“Graphical User Interface”)

Los entornos gráficos de ventanas son algo masivo y común a prácticamente todos los sistemas operativos actuales.

Las opciones de utilización y de gestión de las ventanas pueden diferir en ciertos detalles de unos a otros (dónde están colocados los “botones”, la combinación de teclas de acceso rápido, etc.), pero en general todos ellos facilitan enormemente al usuario la realización de acciones sobre el sistema, la ejecución de muchas aplicaciones simultáneas y la visualización de datos aprovechando al máximo las capacidades de las pantallas.

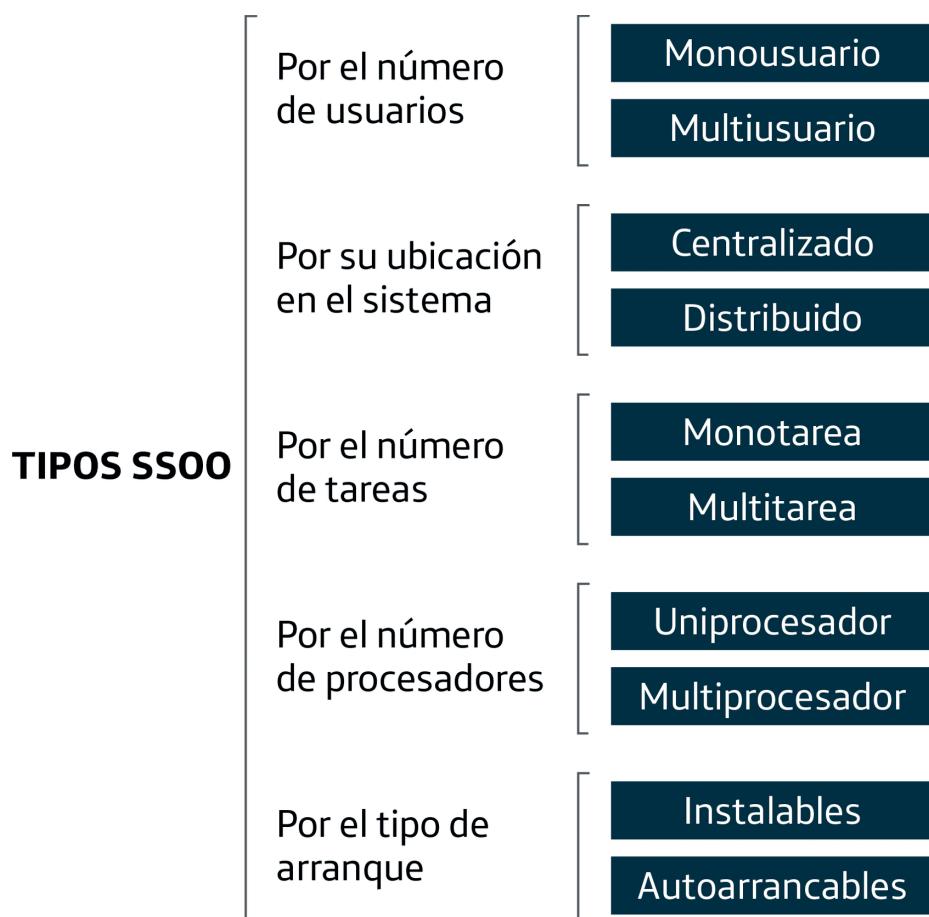


Tipos de sistemas operativos

A la hora de enumerar los diferentes tipos de sistemas operativos pueden hacerse muchas clasificaciones.

Ya hemos visto alguna división en función de su arquitectura (monolítica, jerárquica, en capas, etc.), pero comúnmente podemos encontrar otras, como por ejemplo las que te mostramos en la figura.

Veamos un poco sus diferencias y características:



Monousuario / multiusuario

Monousuario

Como su propio nombre indica, los sistemas operativos “monousuario” son aquellos que **sólamente atienden a un usuario a la vez**, y esto no tiene que ver con el número de procesadores que tenga la máquina, ni el número de procesos que pueda correr la CPU de forma simultánea (que pueden ser muy elevados).

Al principio los ordenadores personales entraban dentro de esta clasificación, pero actualmente sabemos que sobre ellas podemos instalar complejos S.O. que atiendan a más de un usuario a la vez.

Multiusuario

Los sistemas operativos “multiusuario” **pueden dar servicio simultáneo a varios usuarios**, bien porque el ordenador admita la conexión de varios terminales conectados a él, o bien porque se establezcan sesiones de trabajo remota sobre el S.O. a través de una red de comunicaciones.

Evidentemente, en función del número de procesadores que tenga la máquina, su potencia y el esfuerzo demandado (número de procesos y lo “pesados” de ejecutar que sean) el sistema podrá atender a más o menos usuarios, aunque también podrá estar limitado el número de usuarios o de sesiones en un determinado instante por configuración en el sistema.

Centralizados / distribuidos

Centralizado

Los S.O. “centralizados” son aquellos cuya **ejecución y recursos se encuentran en una sola máquina**. La mayoría de los S.O. de los ordenadores personales son de este tipo, puesto que están pensados para ser instalados y ejecutados en un solo ordenador.

Esto no quiere decir que no puedan acceder a recursos externos remotos (desde una impresora local a un recurso a través de Internet), pero los dispositivos sobre los que tiene el control el S.O. son los que se encuentran en la propia máquina.

Distribuido

Los “sistemas operativos distribuidos” son aquellos cuyos **recursos están repartidos entre varios ordenadores interconectados**, ya sea en la misma ubicación física o en sitios distantes geográficamente.

La evolución de estos sistemas ha estado fuertemente influida por el aumento de potencia de los procesadores, y también por la evolución de las redes de comunicaciones.

Algunas veces también se habla de “sistemas operativos en red”, pero estos pueden ser S.O. normales interconectados y funcionando de forma autónoma.

En los S.O. distribuidos la gestión de los recursos (físicos, tiempo de CPU, dispositivos, etc.) es global y (en principio) transparente para el usuario, que accede a ellos como si se tratase de recursos locales.

Monotarea / multitarea

Monotarea

Los S.O. “monotarea” son aquellos que **solamente permiten la ejecución de un proceso por usuario al mismo tiempo**.

Hay que tener en cuenta que un sistema operativo puede ser “monotarea” y “multiusuario” a la vez, es decir, que puede atender a varios usuarios de forma simultánea pero cada uno de ellos solamente puede ejecutar una tarea a la vez.

Multitarea

Un sistema operativo “multitarea” **permite al usuario ejecutar varios trabajos “a la vez”**. Por ejemplo, puede estar editando un fichero de texto y al mismo tiempo haber ordenado el procesamiento estadístico de un conjunto de datos a través de otro programa.

Tengamos en cuenta que si en el ordenador solamente hay una CPU, en realidad solamente podrá estar ejecutándose un programa en un determinado instante, pero la capacidad de multiprogramación de los S.O. nos permiten manejar varios y dar la sensación al usuario de que se ejecutan “al mismo tiempo”. La multitarea real solamente la encontraremos si tenemos varios procesadores o varios núcleos en el sistema.

Es bastante frecuente que estos sistemas ofrezcan al usuario interfaces gráficas de ventanas (GUI), donde cada “ventana” será un proceso que atiende a una tarea (y que a su vez puede desencadenar varias tareas más).

Uniprocesador / Multiprocesador

Uniprocesador

De nuevo, como su propio nombre indica, un sistema operativo “uniprocesador” es aquel que **solamente puede gestionar un procesador en el ordenador** (si hubiese más procesadores estarían desaprovechados).

Un ejemplo típico de este sistema es el MS-DOS o el MAC OS (anterior al MAC OS X).

Multiprocesador

Un sistema operativo “multiprocesador” admite la gestión de **más de un procesador en la máquina**, y es capaz de distribuir la carga entre ellos (bien especializándolos por tareas o por gestión de determinados recursos, o simplemente dividiendo el número de procesos a ejecutar entre ellos).

Existen en general dos modos de funcionamiento en los sistemas multiprocesador:

- **Simétrico**: los procesos (o los “hilos” de ejecución de un proceso, llamados “*threads*”, y puede tener varios) son repartidos sobre todos los procesadores disponibles en función de la carga.
- **Asimétrico**: uno de los procesadores (“maestro”) es el encargado de distribuir la carga entre los demás (“esclavos”).

Instalables / autoarrancables

Instalables

Cualquier sistema operativo, para poder ejecutarse, debe estar cargado en la memoria del ordenador. La encargada de hacer la primera carga cuando encendemos nuestra máquina es la **BIOS** del sistema.

Lo normal es tener el S.O. instalado en el disco duro del ordenador y para ello, como es lógico, hemos de haber seguido un proceso de instalación a través de un paquete SW instalable del S.O., que normalmente cuenta con guías de ayuda para el usuario en forma de ventanas y menús de opciones.

Este S.O. una vez instalado arrancará automáticamente cuando encendamos el ordenador y la BIOS se encargue de cargarlo en la memoria del sistema.

Autoarrancables

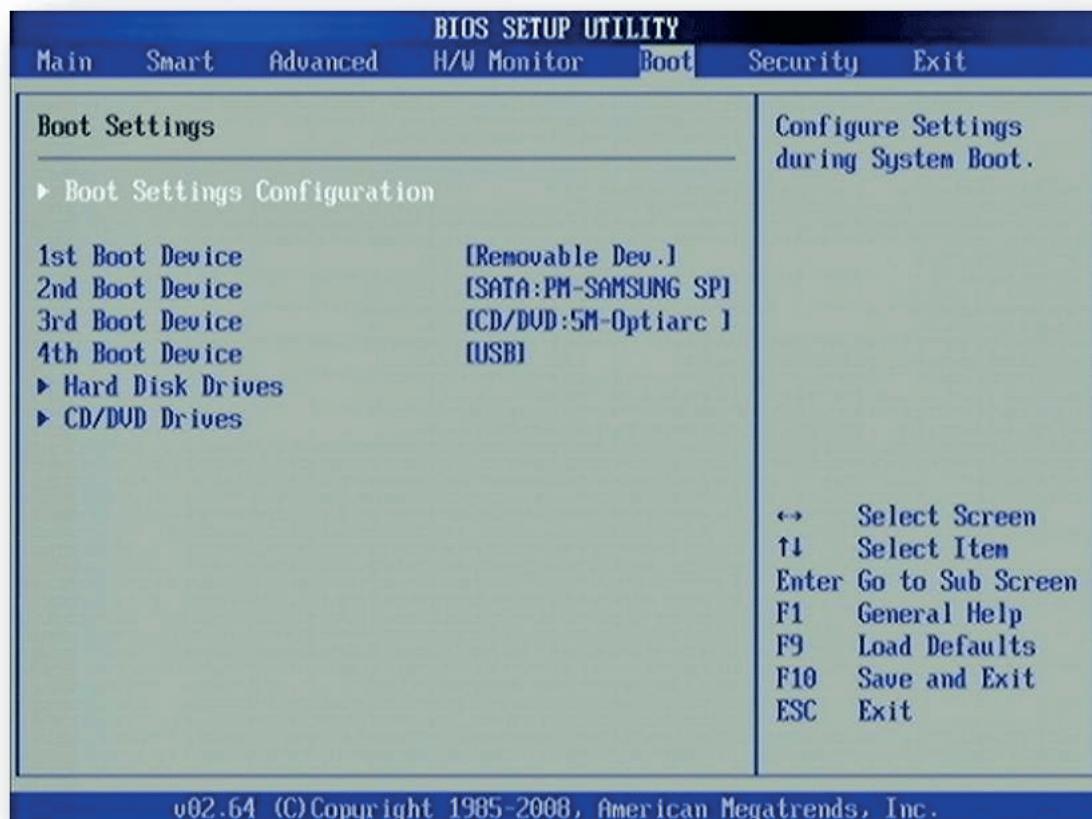
Algunas veces podemos tener una “instalación” del S.O. sobre un dispositivo externo a nuestra máquina, típicamente un CD/DVD o una memoria USB (o un disco duro externo).

Si en el orden de búsqueda de la BIOS está que lea primero del dispositivo externo, cargará el S.O. desde allí, sin ir a buscarlo al disco duro.

A estas configuraciones se les suele llamar “**live CD**” o “**live USB**”, y son útiles si queremos correr sobre el ordenador un S.O. diferente al que tiene instalado, por ejemplo porque se haya corrompido el SW por algún fallo, algún ataque/virus lo haya dañado, o para realizar un análisis forense informático sobre el ordenador sin tocar la instalación en su disco duro.

¿Y qué es la BIOS?

La **BIOS** es una rutina (programa) grabada en una memoria no volátil (firmware - ROM) y que cuando encendemos el ordenador se encarga de realizar una primera **rutina de arranque sobre la CPU.**



En esta rutina de arranque (“**boot**”) irá a buscar el S.O. a una serie de dispositivos según un orden definido, por ejemplo al disco duro, luego al CD/DVD, luego a la unidad de USB, etc.

El S.O. debe estar instalado en alguno de los dispositivos, de forma que el primero que encuentre será el que cargue en el sistema.

Si queremos arrancar nuestro ordenador desde un CD o un USB con un S.O. instalado en él deberemos configurar la BIOS para que vaya a buscarlo a ese dispositivo en primer lugar, y entonces no cargará el que tengamos instalado en el disco duro (si tenemos alguno en él).

No te preocupes, te volveremos a hablar de la BIOS un poco más adelante.

Software de sistema

Dentro de todos los posibles programas que podemos ejecutar sobre el ordenador el principal será el **sistema operativo**, que como hemos comentado permitirá la ejecución de los demás programas de usuario y optimizará los recursos de la máquina.

El S.O. puede considerarse el principal “**software de sistema**” en la máquina, pero además existe una serie de programas (que a veces vienen incluidos dentro de la distribución del S.O. y otras los tenemos que instalar específicamente) que complementan y ayudan al S.O. en la gestión de los recursos del ordenador, y que es interesante conocer.

Además, si nos dedicamos al desarrollo SW en nuestra máquina podemos instalar una serie de programas específicos para esta actividad, que nos ayuden con la generación (escritura) del código fuente en un lenguaje de programación, con la compilación y el “linkado” (enlazado) de los módulos del programa e incluso proporcionarnos entornos específicos de trabajo sobre un determinado S.O.

Por último, también podemos “correr” (ejecutar) sobre el ordenador toda una serie de aplicaciones de muy diversa índole en función del trabajo a desarrollar, desde las conocidas aplicaciones ofimáticas hasta cualquier otro programa generado para ser montado sobre nuestro S.O.

Software de sistema

A estos programas también se les conoce como “**software de base**”, y suelen **interactuar directamente con el S.O.** para ayudar en la gestión de los recursos de la máquina, controlando dispositivos y dando soporte a otros programas. De hecho, muchos de ellos suelen venir incluidos en el propio paquete de instalación del S.O. Estos programas son una ayuda al usuario para que este haga un uso más eficiente de la máquina, no para resolver la problemática de una aplicación.

Algunos tipos de programas incluidos dentro de esta denominación pueden ser:

Sistemas operativos

Es el principal software de sistema. Además de todos los tipos nombrados ya, dentro de ellos podemos encontrar los “**sistemas operativos en tiempo real**” y los “**sistemas empotrados**”. En ambos casos (a veces se denominan de igual forma) se trata de realizar tareas que tienen que ser ejecutadas “en tiempo real”, es decir, con unas restricciones de tiempo de respuesta muy concretas.

Por ejemplo tareas de control de un determinado HW o aplicaciones específicas, sin contemplar normalmente (algunos sí lo permiten) la posibilidad de instalación y ejecución de aplicaciones de usuario como las de los ordenadores de propósito general.

Consolas de entrada de comandos

Normalmente todos los S.O. ofrecen al usuario una interfaz para enviar órdenes al “**intérprete de mandatos**” (“*shell*”) del sistema. Los comandos son introducidos en modo texto y con una sintaxis específica para cada *shell* (pueden existir varias, como hemos visto).

Ventajas:

- Permiten interactuar directamente con el S.O a través de la *shell*.
- Los comandos posibles permiten una gran potencia de ejecución.

Desventajas :

- Su sintaxis suele ser compleja y hay un montón de parámetros que hay que conocer.
- Suelen ser para usuarios experimentados.
- Si cometemos errores hay un riesgo real de causar algún problema al S.O., sobre todo si se introducen con privilegios de administrador o “superusuario”.

BIOS

Ya te hemos comentado la importante labor de la **BIOS** (“**Basic Input Output System**”) en la carga del S.O. durante el arranque del sistema. Sin embargo, además de esta labor, la BIOS tiene a menudo otras funciones:

- Probar el HW del sistema.
- Proporcionar una primera capa de abstracción sobre el HW y permitir el uso de dispositivos como el teclado, la pantalla, etc.
- Enviar mensajes de error que se puedan producir durante el arranque.

Los protocolos de la BIOS fueron sustituidos hace algunos años por la especificación **UEFI** (“**Unified Extensible Firmware Interface**”), que define la interfaz entre el S.O. y el propio HW de la máquina. Este protocolo incorpora mejoras sobre las BIOS anteriores, como por ejemplo el ofrecer un entorno más amigable antes de que sea cargado el S.O. o soportar capacidades de red.



Entornos de escritorio graficos (GUI)

Como ya has leído, una “**interfaz gráfica de usuario**” (**GUI** – “**Graphical User Interface**”) es una interfaz de visualización y acceso para el usuario que se basa en mostrar en pantalla un conjunto de iconos, imágenes y ventanas que sirven para representar la información y recibir órdenes del usuario.

Las GUI aparecieron como un gran avance sobre las interfaces tipo “consola”, en las cuales las órdenes se introducían en forma de comandos de texto con opciones, con sintaxis normalmente difíciles para usuarios poco avanzados.

En las GUI normalmente también podemos arrancar ventanas que sirvan de interfaces en modo texto, bien para la inserción de órdenes o para editar textos, etc. pero sus capacidades de edición e interacción gráficas son bastante más elevadas, sobre todo desde la aparición de las pantallas táctiles.

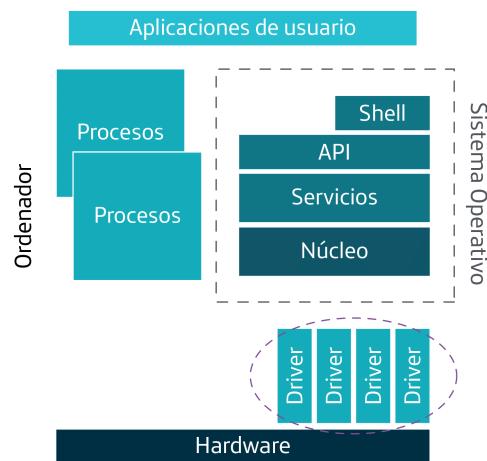
Algunos de los entornos gráficos más comunes (además del de Windows y Apple por supuesto) son: GNOME, Plasma Workspace, Xfce, Trinity, Cinnamon, Unity, Pantheon, etc.

Controladores de dispositivos

Un controlador de dispositivo tiene como función **aislar a la CPU de la problemática de operación de un determinado dispositivo físico**, es decir, encargarse del manejo de ese dispositivo realizando sobre él las acciones que se le ordenan .

Hay que tener en cuenta que se suele llamar “ **controlador de dispositivo** ” (o también “ **adaptador de dispositivo** ”) a un **conjunto de HW+SW que se encarga de controlar el funcionamiento del dispositivo físico** , y que sirve de interfaz con el SW manejador del dispositivo incorporado a la máquina, proporcionando registros para recibir órdenes, *buffers* para el almacenado temporal de los datos, generación de interrupciones, etc.
En este caso, al estar hablando de “ **SW de sistema** ” nos referimos a la parte SW del conjunto del controlador, al que a menudo también se le llama “ **manejador de dispositivo** ” o “ **driver** ”.

Los sistemas operativos pueden integrar ya algunos manejadores estándar en su paquete de instalación o bien puede ser necesaria la instalación posterior del *driver* adecuado para ese dispositivo determinado. Lo más usual es que estos *drivers* se instalen como parte del S.O. del ordenador, y cada fabricante los proporciona específicos para los S.O. más populares. Por ejemplo, cuando se inserta por primera vez una memoria USB (“ *pendrive* ”) en alguno de los puertos USB del ordenador suele aparecer el mensaje: “Instalando software controlador del dispositivo”, y en cambio las siguientes ocasiones en las que se vuelve a utilizar ese *pendrive* ya no nos saldrá dicho mensaje.



Cargadores de programas

Los “**cargadores**” (“*loaders*”) suelen formar parte del paquete del sistema operativo, y su función es **cargar los programas ejecutables** en la memoria RAM del ordenador desde los dispositivos de almacenamiento en donde se encuentren (CD/DVD, USB, etc.).

Algunos programas no pueden cargarse “completos”, o bien necesitan ser ejecutados en diferentes posiciones de memoria y hay que “ubicarlos”, e incluso pueden ser “enlazados” sus diferentes módulos durante la carga. El programa cargador puede ser en encargado de realizar estas funciones para el S.O.

Gestores de arranque (bootloaders)

Un “**gestor de arranque**” es un programa que está pensado para cargarse inmediatamente en el arranque del ordenador (evidentemente después de ejecutarse la **BIOS**) y proporcionar ciertas funcionalidades básicas, además de preparar todo lo necesario para que se cargue luego el S.O.

Algunas de las posibles opciones que nos pueden ofrecer son:

- Elección entre cargar un S.O. u otro (por ejemplo Windows/Linux).
- Elección de cargar el S.O. desde un dispositivo externo.
- Cargar el S.O. desde una unidad de red.

Ejemplos de estos programas: “**LILO**” (“Linux Loader”) o “**GNU GRUB**” (“GNU GRand Unified Bootloader”).

Hipervisores

Un “**hipervisor**” (a menudo también conocido como “**monitor de máquina virtual (VMM)**”) es un programa que sirve para “ocultar” las características reales del HW de la máquina y **ofrecer un entorno virtual de funcionamiento** (“HW virtual”) hacia un sistema operativo, de forma que funcione sobre un “entorno virtualizado” (como S.O. virtualizado). Al mismo tiempo pueden ofrecer otras funcionalidades, que normalmente tienen que ver con el control de la ejecución del S.O. virtualizado.

Al “independizar” la ejecución del S.O. del HW es posible hacer correr varios sistemas operativos “a la vez” sobre una misma máquina, compartiendo los recursos HW reales evidentemente. Existen varios tipos de hipervisores:

- **Hipervisores nativos o de “tipo 1”**: se ejecutan directamente sobre el HW físico de la máquina.
- **Hipervisores “hosted” o de “tipo 2”** : se ejecutan sobre un S.O. anfitrión, y a su vez sobre el hipervisor se ejecutan las máquinas virtuales.
- **Hipervisores “híbridos”** : se ejecutan con un S.O. anfitrión y ambos pueden acceder directamente al HW de la máquina.

Existen muchas plataformas de virtualización, en el caso de Microsoft por ejemplo: Microsoft Virtual PC, Windows Virtual PC, Microsoft Windows Server e Hyper-V. Otras pueden ser Citrix XEN Server y VMWare ESX-Server. Volveremos sobre el tema de la virtualización más adelante y verás en la práctica el funcionamiento de una máquina virtual.

Software de aplicación

Es aquel software que normalmente conocemos como “**aplicaciones**”, es decir, está dedicado a resolver una necesidad específica del usuario e interactuar con él para ofrecerle un “**servicio de usuario**”.

Editores gráficos y multimedia

Paquetes de aplicaciones de ofimática

Programas de cálculo

Paquetes de diseño asistido por ordenador (CAD)

Programas de comunicaciones

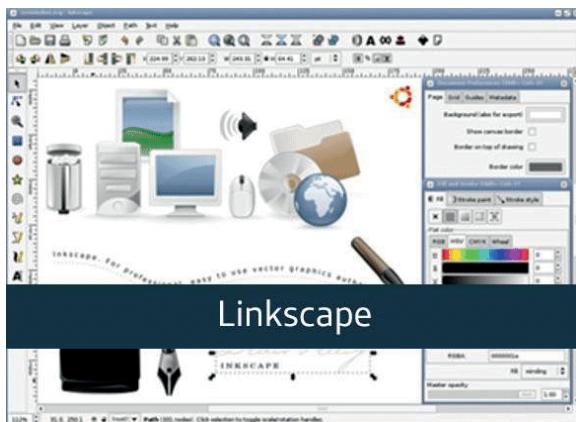
Sistemas de gestión de bases de datos

Ejemplos de programas de aplicación podrían ser los que te mostramos y en general cualquier aplicación o programa de propósito general, juegos, etc. que instalamos sobre el ordenador con la finalidad de dar algún servicio al usuario.

Editores gráficos y multimedia

Casi todos hemos empleado algún programa para realizar alguna presentación (como MS PowerPoint o similar) y hemos utilizado sus recursos para crear imágenes o combinar contenido gráfico.

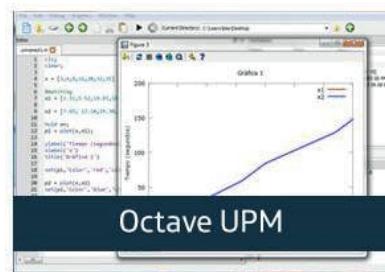
Existen muchos programas que nos permiten la edición de material audiovisual de gran calidad. Algunos de ellos están especializados en el retoque fotográfico, o bien en la creación de imágenes (rasterizadas o vectoriales) y también están los dedicados a la edición de audio, vídeo, o producción audiovisual completa. Algunos ejemplos:



Programas de cálculo

Cuando nos referimos a programas de cálculo no solamente hacemos referencia a las “hojas de cálculo” que incorporan los paquetes o fílmaticos (como “Excel” por ejemplo), sino que existen multitud de programas especializados para el cálculo en muchos entornos, desde el cálculo matemático avanzado en general, hasta áreas de trabajo específicas como el cálculo de infraestructuras, la contabilidad, el análisis de variables financieras, etc.

Muchos de ellos combinan su potencia de cálculo con funciones avanzadas de presentación gráfica de la información. Ejemplos:



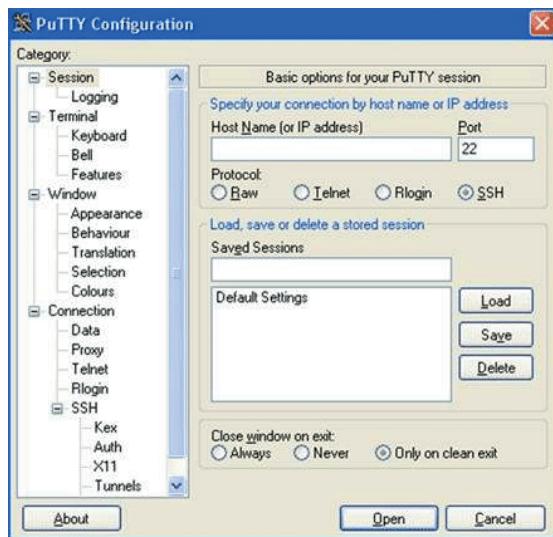
Programas de comunicaciones

Son programas dedicados específicamente al intercambio de información (en uno u otro formato) a través de las redes de comunicaciones, o para el acceso a servicios o servidores externos.

Algunas funcionalidades directamente relacionadas con el control del HW de comunicaciones vendrán dentro del paquete del S.O., pero para otras tendremos que recurrir a programas específicos.

Ejemplos pueden ser:

- Mensajería de voz o videoconferencia (p. ej. Skype).
- Clientes de correo electrónico.
- Conexión remota a servidores (p. ej. Telnet, FTP).
- Navegadores web.
- Clientes para conexión a redes sociales (p. ej. Twitter).



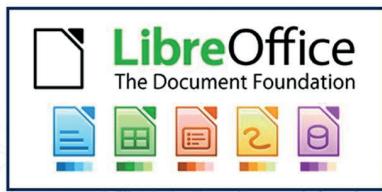
Paquetes de aplicaciones de ofimática

En general conocidos por todos, son conjuntos de programas que se distribuyen para atender a las principales necesidades de un puesto de trabajo en una oficina o a nivel personal.

Algunos fabricantes incorporan en su oferta de venta algún paquete ofimático junto con su sistema operativo, y otros nos los ofrecen por separado o de forma gratuita.

Los programas más comunes que suelen incorporar son:

- Un editor de textos.
- Un editor de presentaciones.
- Una hoja de cálculo.
- Un cliente de correo electrónico.
- Un cliente de base de datos.
- Programas accesorios y utilidades.



Paquetes de diseño asistido por ordenador (CAD)

El software de diseño asistido por ordenador (“Computer Aided Design” – CAD SW) está formado por todo un conjunto de programas y herramientas especialmente dedicado a ayudar a los diseñadores industriales, ingenieros, arquitectos, etc.

Normalmente incorporan entornos gráficos avanzados con herramientas muy potentes para el dibujo, la representación en 3D, el uso de bibliotecas y librerías de material gráfico, herramientas de cálculo, etc.

Ejemplos:



Sistemas de gestión de bases de datos (SGBD)

Estos programas y paquetes SW son un conjunto de programas **específicamente dedicados a la gestión de bases de datos (SGBD)**, es decir, al almacenamiento y extracción de la información en la BD, su modificación y análisis, control de su integridad, visualización y filtrado, etc.

Los modernos SGBD proporcionan interfaces de usuario muy visuales, a través de las cuales se pueden formular peticiones de información y obtener informes personalizados. Además, por supuesto, suelen disponer de sistemas de control de acceso, permisos en función de los perfiles de usuario definidos y almacenamiento histórico de la actividad del usuario.

Algunos ejemplos:



Software de programación

Son programas de ayuda a la labor del desarrollador de SW, y pueden ir desde la definición de requisitos del SW a desarrollar, hasta el modelado, la edición del código, su compilación, la realización de pruebas, etc.

Algunas de las herramientas de programación más comunes son:

Compiladores y enlazadores

Un “**compilador**” es un programa capaz de **traducir un programa escrito en código fuente** (en lenguaje de alto nivel) **a un fichero o programa en código máquina** (que es el que entiende y puede ser ejecutado en un procesador). A este proceso de “traducción” es a lo que llamamos “compilación”. Un ejemplo en el entorno Unix/Linux:



Ensambladores

El lenguaje “**ensamblador**” (“**assembler**”) es un **lenguaje de programación “de bajo nivel”**, pero aún así este lenguaje hay que traducirlo en un código binario para ser ejecutado por el procesador, y de ello se encargan los programas “**ensambladores**”.

Algunos de estos programas pueden funcionar en “**modo intérprete**”, es decir, que no compilan y ejecutan el programa completo, sino que permiten la ejecución de las instrucciones paso a paso, lo cual permite un seguimiento más cercano del proceso.

Editores de código fuente

Un editor de código fuente es un programa de edición **específico para editar el código fuente de un programa**. Puede ser un programa editor independiente o bien ser parte de un entorno de desarrollo integrado (IDE).

Estos editores normalmente incorporan funcionalidades para ayudar a simplificar y facilitar la escritura del código fuente, como por ejemplo la posibilidad de verificar la sintaxis de las líneas de código, autocompletar sentencias, eliminación de espacios en blanco no necesarios, conversión de formatos (decimal a binario o hexadecimal) etc.

Algunos de los más conocidos:



Depuradores

Un **depurador** ("debugger"), es un programa **utilizado para detectar, corregir y eliminar los errores del código del programa** que estamos generando (a lo que llamamos "depurar" el programa).

Estos depuradores pueden ejecutar el programa e informarnos al detectar un error de ejecución (fallo en el programa) o cierta condición que hayamos preconfigurado. De esta forma nos dicen en qué punto de la ejecución se ha producido el fallo, señalando la instrucción en la cual sucedió. Algunos ejemplos:

FireBug
web debugging evolved

GDB: The GNU Project Debugger

The screenshot shows the Winpdb interface. The top menu includes File, Breakpoints, Control, Window, Help. The toolbar has icons for file operations. The main window has tabs for Namespace, Threads, and Stack. The Namespace tab shows a list of variables and their types. The Threads tab shows Thread-1 running. The Stack tab shows a call stack with frames from core.py, rpdb2.py, and rpdb2b.py. On the right, there's a code editor with Python code for winpdb, a breakpoints list, and a console window displaying debug messages.

IDE

Un entorno integrado de desarrollo (**IDE** – "Integrated Development Environment") es un paquete SW que incluye una serie de programas cuyo objetivo es facilitar la programación y el desarrollo de SW al programador.

Los IDE pueden ser específicos para desarrollar programas empleando un solo lenguaje de programación o usar varios de ellos. También **pueden ser para un solo S.O. o multiplataforma**. Lo normal es que contengan las siguientes funcionalidades:

- Un editor de código especializado.
- Un compilador.
- Un depurador.
- Un intérprete.
- Sistemas de ayuda para generar interfaces gráficas de usuario.
- Algunos incorporan también sistemas de control de versiones.

Algunos ejemplos:



Resumen

Has finalizado esta lección.

En esta lección hemos tratado un montón de conceptos, a cuál más importante para tu conocimiento y trabajo sobre un sistema informático.

Te recomendamos que te asegures de tener claro lo que es un sistema operativo y cuáles son sus funciones. Después verás que en la práctica existen sistemas muy diferentes, tanto en su apariencia y manejo como en su estructura interna, y deberemos estudiar en profundidad aquel sistema con el que vayamos a trabajar, intentando entender bien cómo funciona. Nosotros en este curso verás que te mostraremos dos de los principales sistemas que más se usan en la actualidad, el sistema Windows 10 y el Linux Ubuntu, pero esto será un poco más adelante.



PROEDUCA