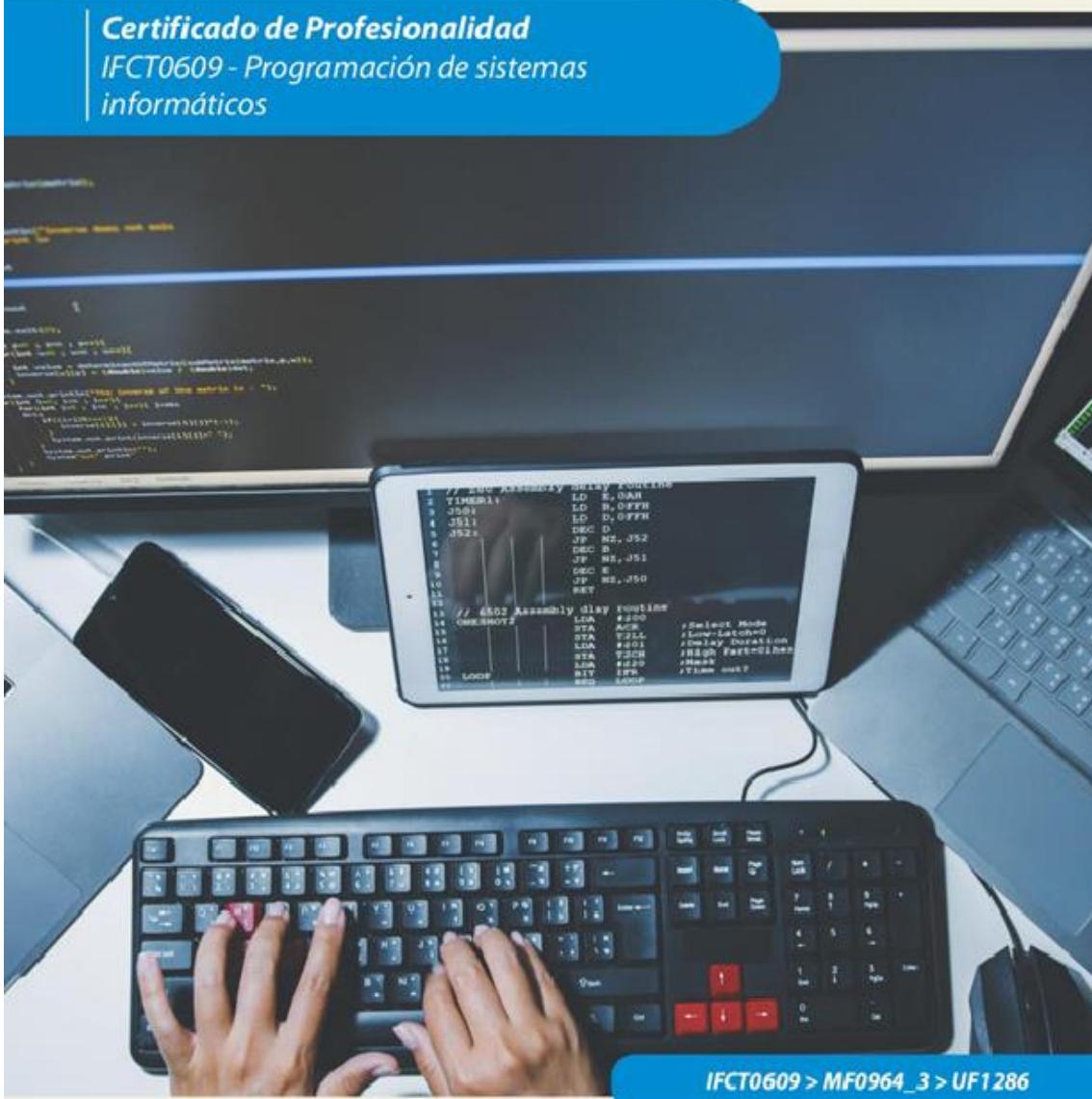




UF1286: Desarrollo y optimización de componentes software para tareas administrativas de sistemas

Certificado de Profesionalidad
IFCT0609 - Programación de sistemas informáticos



IFCT0609 > MF0964_3 > UF1286

ic editorial

José Luis Villada Romero

**Desarrollo y optimización
de componentes
software para tareas
administrativas
de sistemas
IFCT0609**

José Luis Villada Romero

ic editorial

Desarrollo y optimización de componentes software para tareas administrativas de sistemas. IFCT0609

© José Luis Villada Romero

1.^a Edición

© IC Editorial, 2023

Editado por: IC Editorial
c/ Cueva de Viera, 2, Local 3
Centro Negocios CADI
29200 Antequera (Málaga)
Teléfono: 952 70 60 04
Fax: 952 84 55 03
Correo electrónico: iceditorial@iceditorial.com
Internet: www.iceditorial.com

IC Editorial ha puesto el máximo esfuerzo en ofrecer una información completa y precisa. Sin embargo, no asume ninguna responsabilidad derivada de su uso, ni tampoco la violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Mediante esta publicación se pretende proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para **IC Editorial** ninguna forma de asistencia legal, administrativa ni de ningún otro tipo.

Reservados todos los derechos de publicación en cualquier idioma.

Según el Código Penal vigente ninguna parte de este o cualquier otro libro puede ser reproducida, grabada en alguno de los sistemas de almacenamiento existentes o transmitida por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro, sin autorización previa y por escrito de IC EDITORIAL; su contenido está protegido por la Ley vigente que establece penas de prisión y/o multas a quienes intencionadamente reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

ISBN: 978-84-1184-095-8

Presentación del manual

El **Certificado de Profesionalidad** es el instrumento de acreditación, en el ámbito de la Administración laboral, de las cualificaciones profesionales del Catálogo Nacional de Cualificaciones Profesionales adquiridas a través de procesos formativos o del proceso de reconocimiento de la experiencia laboral y de vías no formales de formación.

El elemento mínimo acreditable es la **Unidad de Competencia**. La suma de las acreditaciones de las unidades de competencia conforma la acreditación de la competencia general.

Una **Unidad de Competencia** se define como una agrupación de tareas productivas específica que realiza el profesional. Las diferentes unidades de competencia de un certificado de profesionalidad conforman la **Competencia General**, definiendo el conjunto de conocimientos y capacidades que permiten el ejercicio de una actividad profesional determinada.

Cada **Unidad de Competencia** lleva asociado un **Módulo Formativo**, donde se describe la formación necesaria para adquirir esa **Unidad de Competencia**, pudiendo dividirse en **Unidades Formativas**.

El presente manual desarrolla la Unidad Formativa **UF1286: Desarrollo y optimización de componentes software para tareas administrativas de sistemas**,

perteneciente al Módulo Formativo **MF0964_3: Desarrollo de elementos software para gestión de sistemas**,

asociado a la unidad de competencia **UC0964_3: Crear elementos software para la gestión del sistema y sus recursos**,

del Certificado de Profesionalidad **Programación de sistemas informáticos**.

Índice

Portada

Título

Copyright

Presentación del manual

Índice

Capítulo 1

Descripción de los servicios, estructura y administración de sistemas operativos

1. Introducción

2. Definición y conceptos básicos sobre sistemas operativos

3. Características estructurales de los sistemas operativos

4. Herramientas administrativas de uso común en sistemas operativos

5. Resumen

Ejercicios de repaso y autoevaluación

Capítulo 2

Programación de sistemas operativos. Lenguajes y librerías de uso común

1. Introducción

2. Las llamadas al sistema (*system calls*)

3. Programas de utilidades y comandos del sistema

4. Resumen

Ejercicios de repaso y autoevaluación

Capítulo 3

El ciclo de vida del software de gestión de sistemas

1. Introducción

2. Modelos del ciclo de vida del software

3. Descripción de las fases en el ciclo de vida del software

4. Calidad del software

5. Resumen

Ejercicios de repaso y autoevaluación

Capítulo 4

Desarrollo del software de gestión de sistemas

- 1. Introducción**
 - 2. Análisis de especificaciones para el desarrollo de *software* de gestión de sistemas**
 - 3. Técnicas de programación presentes en lenguajes de uso común aplicables al desarrollo de *software* de gestión de sistemas**
 - 4. Técnica de programación de *software* de gestión de sistemas**
 - 5. Control de calidad del desarrollo del *software* de gestión de sistemas**
 - 6. Herramientas de uso común para el desarrollo de *software* de sistemas**
 - 7. Resumen**
- Ejercicios de repaso y autoevaluación**

Bibliografía

Capítulo 1

Descripción de los servicios, estructura y administración de sistemas operativos

Contenido

1. Introducción
2. Definición y conceptos básicos sobre sistemas operativos
3. Características estructurales de los sistemas operativos
4. Herramientas administrativas de uso común en sistemas operativos
5. Resumen

1. Introducción

El sistema operativo y los servicios que este proporciona evitan que el desarrollo de aplicaciones se vuelva una tarea demasiado compleja. En los comienzos de la informática, los desarrollos eran costosos en tiempo, principalmente, porque era necesario reprogramar aspectos que hoy en día el sistema operativo proporciona de forma mucho más simple a los desarrolladores.

La evolución que ha experimentado los sistemas operativos nos hace verlos como mucho más que un programa que controla el *hardware*; se perciben como entornos con vistas sencillas y útiles mecanismos para la configuración y la ejecución de las aplicaciones.

Proporcionan una calidad elevada de servicios, a todos los niveles, y tareas como escuchar música, ver fotos o navegar por internet son ejemplos de algunos de ellos.

2. Definición y conceptos básicos sobre sistemas operativos

Un sistema operativo es un programa que actúa como una capa entre el usuario y el *hardware*. Esta capa se encarga de gestionar y administrar todas las partes del sistema.

Se puede dividir el sistema operativo en tres partes o bloques principales:

- **La capa más cercana al hardware se denomina núcleo (*kernel*)**. Contiene la funcionalidad básica del sistema operativo y las estructuras de datos necesarias.
- **La capa de los servicios que ofrece el sistema operativo**. Todas las tareas que el sistema operativo lleva a cabo son ofrecidas en forma de interfaz para el desarrollo de aplicaciones.
- **La capa de interfaz entre el usuario y el sistema operativo**. Se encarga de proporcionar una forma de diálogo entre el usuario y el sistema operativo.

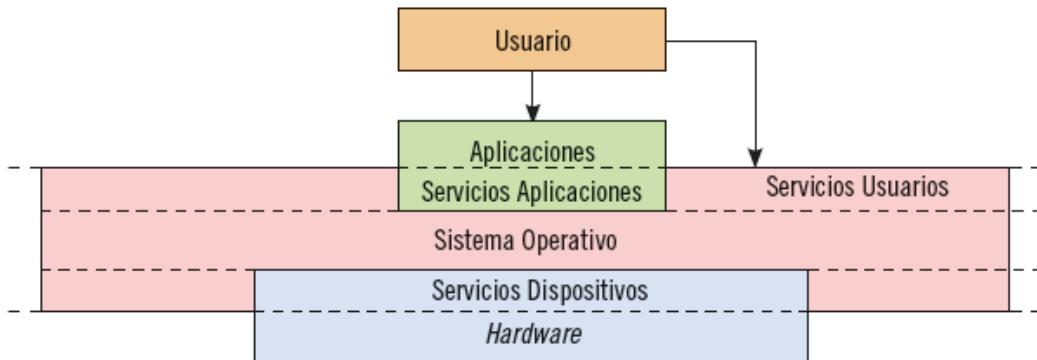
2.1. Descripción de los servicios básicos ofrecidos por un sistema operativo

El principal objetivo de un sistema operativo debe ser proporcionar un entorno para la ejecución de programas, de forma que estos programas sean capaces de usar los dispositivos de los que se compone el sistema.

Para llegar a esta meta, el sistema operativo debe ofrecer servicios a tres niveles: al usuario, a los programas y a los dispositivos. De entre todos ellos, existirá un conjunto de servicios básicos y genéricos:

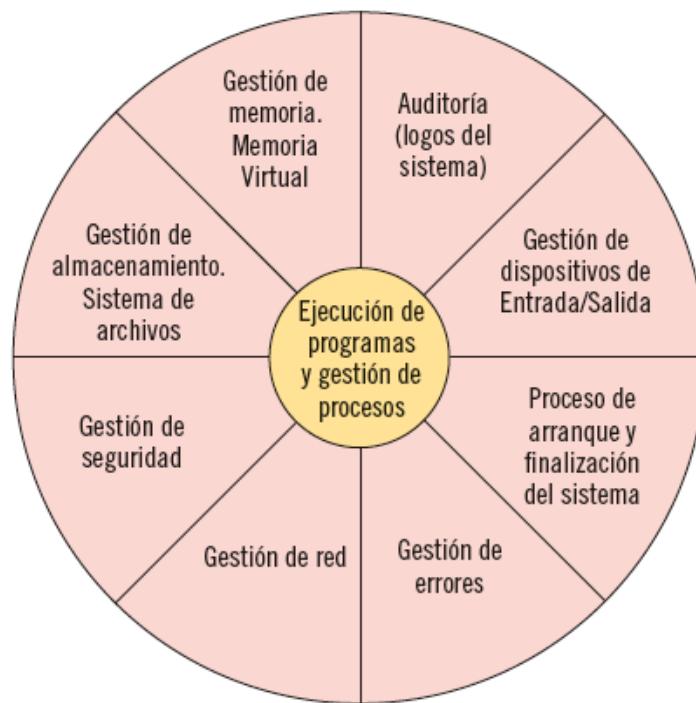
- Permitir la ejecución de aplicaciones.
- Asignar recursos del ordenador (CPU, memoria, etc.) a los programas.
- Dar acceso indirecto a los dispositivos del ordenador y a los periféricos.
- Proporcionar un sistema organizado de almacenamiento de datos.
- Comunicación interactiva con el usuario.

Estructuración de los servicios del sistema operativo



Existe una infinidad de tareas que el sistema operativo necesita realizar para poder desarrollar y proporcionar estos servicios básicos. Estas tareas se agrupan en distintos bloques.

Bloques funcionales de un sistema operativo



Algunas de las tareas dentro de estos bloques serían:

- Secuenciación de tareas.
- Administrar errores.
- Administrar interrupciones.
- Sistema de permisos.
- Conurrencia.
- Uso de recursos compartidos.



Sabía que...

El primer sistema operativo de la historia fue creado en 1956 para un ordenador IBM 704, y básicamente lo único que hacía era comenzar la ejecución de un programa cuando el anterior terminaba.

2.2. Gestión de memoria. Memoria virtual

Todo sistema operativo necesita proporcionar a los programas un entorno de ejecución sin interferencias sobre los otros programas que están ejecutándose a la vez. Por lo tanto, debe repartir la memoria almacenada de forma que ningún programa altere el espacio de otro programa.

En la actualidad, existen varios tipos de memorias que el sistema operativo puede utilizar y que, en función de su velocidad y tamaño, siguen la siguiente jerarquía (de mayor a menor velocidad de acceso y de menor a mayor capacidad):

- Registros de CPU del procesador.
- Caché.
- Memoria principal (RAM).
- Memoria secundaria.

El modelo de gestión de memoria de un sistema operativo debe cumplir con los siguientes objetivos:

- Ofrecer a cada proceso un espacio lógico propio.
- Proporcionar protección entre los procesos.
- Permitir que los procesos comparten memoria.
- Dar soporte a las distintas regiones del proceso.
- Maximizar el rendimiento del sistema.
- Proporcionar a los procesos mapas de memoria muy grandes.

Para conseguir alcanzar todos estos objetivos, el sistema operativo necesita llevar un registro de las partes de memoria que se están utilizando y las que no.



Importante

Un proceso no es más que un programa en ejecución. Son los elementos que maneja el sistema operativo a nivel de aplicación y son identificados por un conjunto de instrucciones, los valores de los registros de la CPU en el momento de ejecución, la memoria reservada y su contenido, e información necesaria para la planificación del sistema operativo.

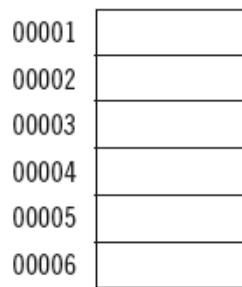
El grado de multiprogramación sería el n.º de procesos de ejecución simultánea.

Cuando se habla de memoria a nivel de sistema operativo se está haciendo referencia al concepto lógico o abstracto. Es decir, aunque se sabe que existe el

material que constituye la memoria física de un ordenador, el sistema operativo usa mecanismos lógicos para su gestión, independientemente de que la memoria sea de un material o de otro.

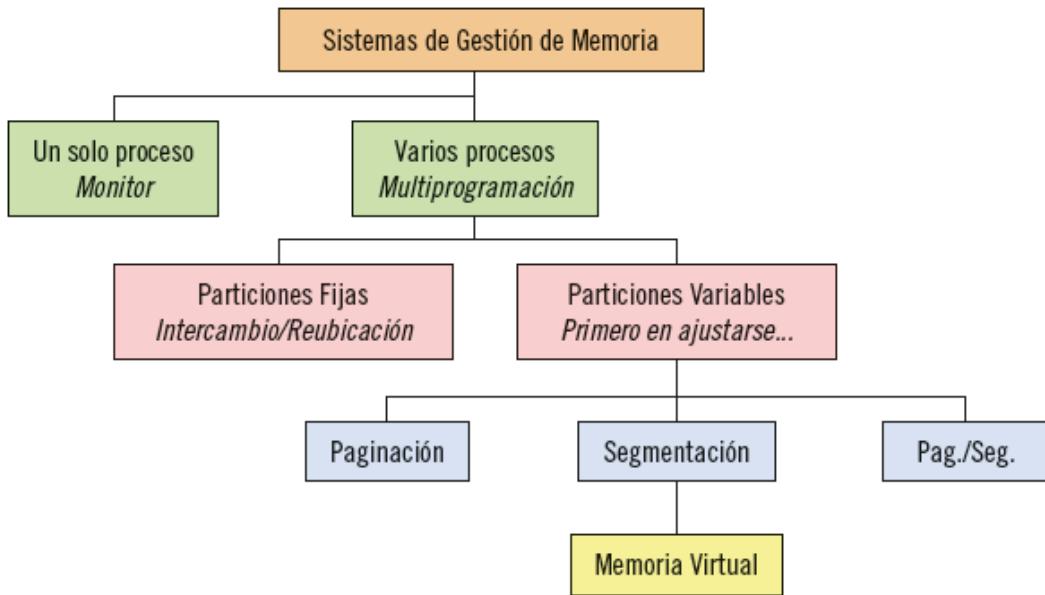
Ese concepto de memoria es el que permite abstraer la idea de almacenamiento lógico al nivel de usuario y su uso a nivel de aplicación. Desde este punto de vista, es posible entender la memoria como una secuencia de bytes de datos, donde cada byte es referenciado mediante un número decimal de 32 bits, o bien un número hexadecimal de ocho cifras. A este concepto se le denomina **direcccionamiento de la memoria**.

Bloques funcionales de un sistema operativo



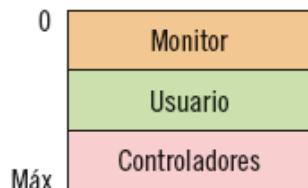
En los sistemas operativos modernos, las políticas de gestión de memoria hacen uso de técnicas muy depuradas para asignar un espacio de memoria a un proceso. Los esquemas principales de gestión de memoria adoptan la siguiente clasificación:

Clasificación de los esquemas de memoria



En el esquema de gestión de memoria de un solo proceso el usuario tiene un control total sobre el espacio de memoria. La memoria se encuentra dividida en dos partes: una reservada al sistema operativo y otra para la ejecución del proceso, que es cargado y ejecutado paso a paso. En implementaciones reales, la memoria se encuentra dividida en tres partes:

Partes reales del espacio de memoria



El esquema de multiprogramación es más complejo, ya que permite la ejecución simultánea de más de un programa. Este tipo de esquemas requieren dividir la memoria. A continuación, se detallan las características de los principales esquemas de este tipo.

Contigua: particiones fijas y variables

En este esquema, la memoria se encuentra dividida en espacios o particiones. Cuando un proceso necesita ejecutarse, se le asigna uno de estos espacios, almacenando toda la información del proceso en posiciones contiguas de la partición seleccionada.

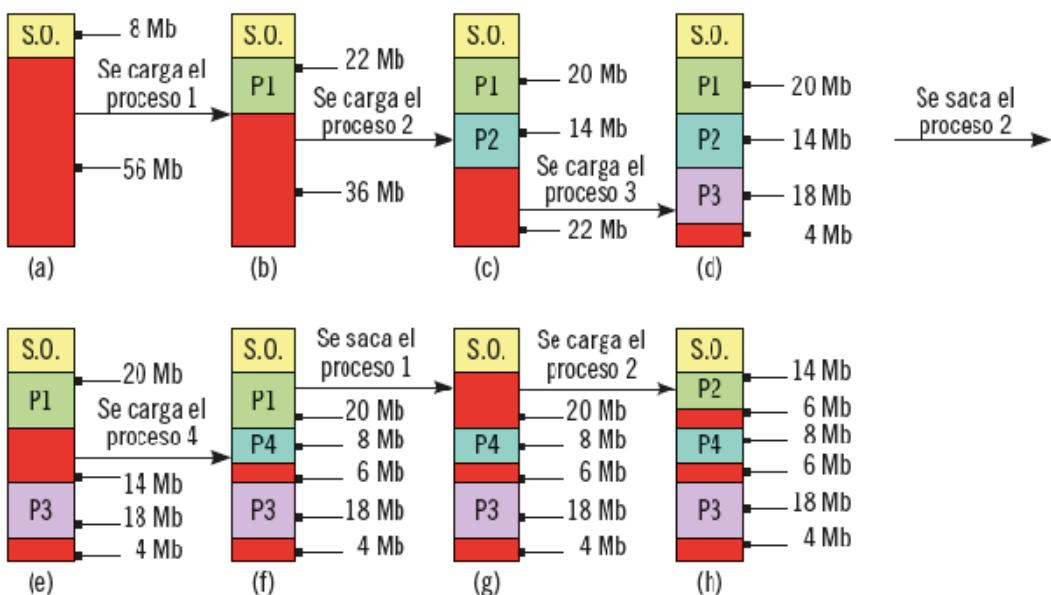
Cada partición solo puede contener un proceso.

Es posible encontrar esquemas con particiones fijas, donde el tamaño de la partición se configura cuando se enciende el ordenador, quedando fija hasta que se apaga, o bien con particiones de tamaño variables.

La principal ventaja que introdujo este esquema fue la protección del espacio de ejecución del proceso. Ningún proceso puede acceder al espacio de memoria de otro.

Los inconvenientes son obvios: se desperdicia gran cantidad de memoria que permanece oculta en cada partición cuando el tamaño es fijo y se producen huecos de memoria demasiado pequeños que no son aprovechados por otros programas en el caso de la partición variable.

Ejemplos de esquemas de memoria contigua de partición fija y variable



Actividades

1. En un sistema con esquema de asignación con múltiples particiones de tamaño fijo, ¿qué determinaría el límite del grado de multiprogramación de dicho sistema?
-



Aplicación práctica

¿Cómo quedaría la estructura de la pila de memoria después de que el sistema operativo asigne los procesos de la siguiente tabla a memoria principal, teniendo en cuenta que se está ante un esquema de memoria contigua con particiones fijas de 200 K y que la secuencia de llegada de los procesos es C, B, A, D?

Procesos	Tamaños			
A	150 K	0-74	0-99	C
B	75 K	75-149	100-199	C
C	150 K	150-224	200-299	B
D	300 K	225-299	300-399	B
		300-374	400-499	A
		375-449	500-599	A
		450-524	600-699	D
		525-599	700-799	D
		600-674	800-899	D
		675-749		
		750-824	900-999	D
		825-899		

SOLUCIÓN

0-74	C
75-149	C
150-224	B
225-299	
300-374	
375-499	
450-524	A
525-599	A
600-674	D
675-749	D
750-824	D
825-899	D

Intercambio (*swapping*)

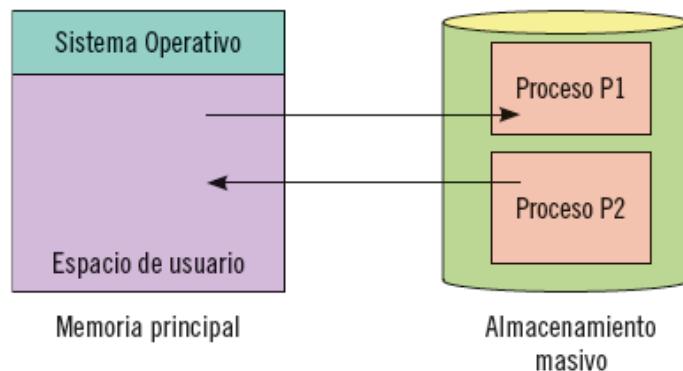
Este esquema de gestión de memoria se basa en mover procesos que están en suspensión desde la memoria principal a la memoria del disco, cuando no caben en la memoria principal, y mover procesos que deben ejecutarse desde la memoria de disco a la memoria principal, cuando se ha liberado parte de la memoria principal.

Las funciones más importantes son:

- Seleccionar procesos a eliminar de la memoria principal.
- Seleccionar procesos a cargar en la memoria principal.
- Asignación y gestión del espacio de intercambio.

Para usar este tipo de esquema, es necesario configurar en el sistema operativo el uso de un archivo de intercambio para almacenar la imagen dinámica del proceso retirado de la memoria principal. Se podrá tener un archivo global o bien un archivo por proceso.

Esquema de memoria con intercambio



Actividades

2. ¿Puede coexistir un esquema de intercambio en sistema de asignación contigua con tamaño de partición fijo de memoria? Razone la respuesta.

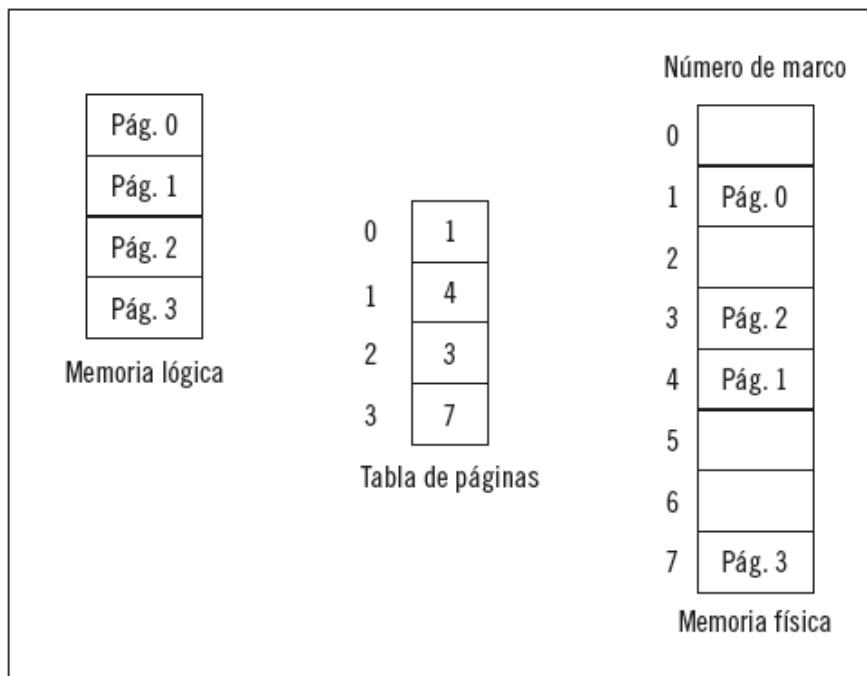
si, si no se malgastaría
mucha memoria

Paginación

En este esquema, la memoria se divide en unidades del mismo tamaño denominadas marcos de página. Mientras que los programas se dividen en unidades lógicas denominadas páginas. Las páginas tendrán el mismo tamaño que los marcos de página.

Cuando un proceso se ejecuta carga un número de páginas en la memoria física y el resto en disco.

Esquema de memoria con paginación



El sistema mantiene una tabla para la traducción del número de página al número de marco.

Las dos funciones principales son:

- Llevar a cabo la transformación de una dirección de memoria a la página correspondiente.
- Transferir páginas entre la memoria secundaria y la memoria principal y viceversa.

Entre las ventajas, se encuentra que es posible ejecutar un programa cargando solo una parte del mismo y el resto bajo demanda de ejecución, no es necesario que las páginas estén contiguas en memoria y además es fácil controlar todas las páginas porque tienen el mismo tamaño.

La principal desventaja es que debe existir un mecanismo de traducción de direcciones, con la consiguiente pérdida de eficiencia.



[Actividades](#)

3. ¿Qué ocurriría si el tamaño del marco en memoria fuera menor que el tamaño de página en un sistema de paginación de memoria?
-

Segmentación

La segmentación se basa en la división lógica del programa en partes denominadas segmentos, donde cada una de estas partes agrupa elementos relacionados lógicamente, como por ejemplo: pila, código, datos, etc.

Cuando el programa es compilado, el compilador construye los segmentos en función de la estructura del programa. Los segmentos pueden tener tamaños distintos. En este esquema, cada programa puede ocupar más de una partición y no tienen por qué ser contiguas.

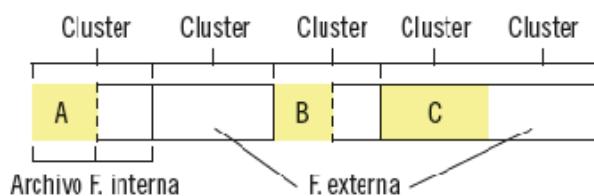
Entre las ventajas y desventajas, destaca que facilita la compactación de código y datos. No produce la fragmentación interna, pero sí fragmentación externa.



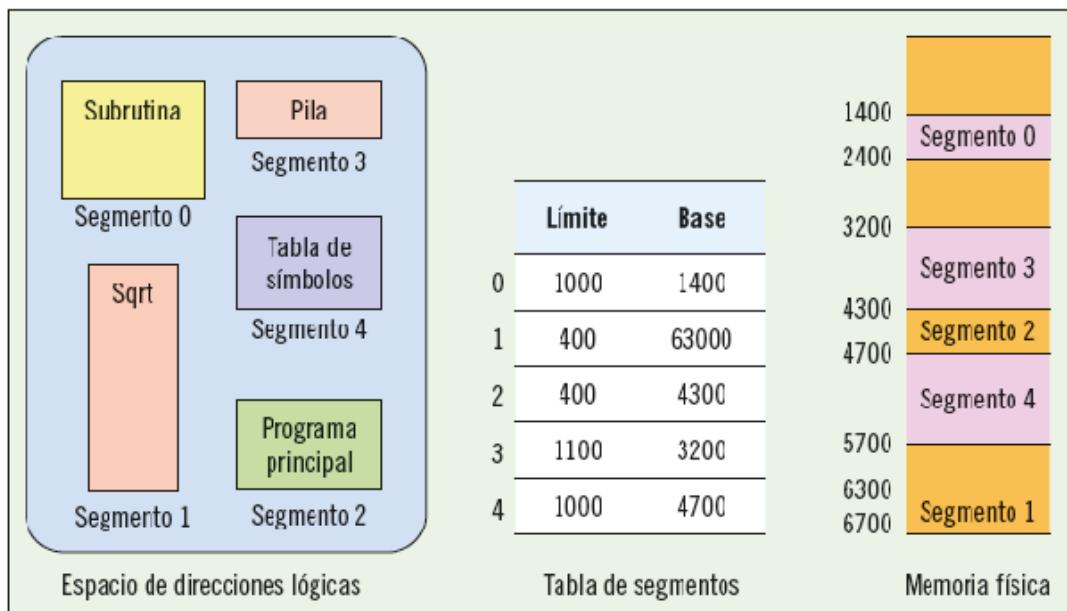
Importante

La fragmentación interna se produce cuando se intenta almacenar información con un tamaño menor que la unidad mínima de almacenamiento lógico que puede gestionar el sistema operativo. Sin embargo, la fragmentación externa se produce cuando entre particiones de memoria quedan huecos libres más pequeños que no pueden ser aprovechados por el sistema operativo cuando intenta guardar otros bloques mayores.

Tipos de fragmentación de memoria



Esquema de memoria con segmentación



Actividades

4. ¿Por qué no hay fragmentación interna en un esquema de segmentación de la memoria?
-

Segmentación paginada

Se trata de un esquema que usa la combinación de la segmentación y la paginación. En este caso, la memoria es segmentada y los segmentos se componen de páginas.

La traducción de direcciones necesita de tablas de páginas y de tablas de segmentos. Las direcciones virtuales contienen un identificador de segmento, un número de página y un desplazamiento dentro de la página.

La ventaja de este esquema es que se elimina la fragmentación externa, pero se introduce la fragmentación interna.

Memoria virtual

Cuando la memoria era muy cara y, por lo tanto, los equipos informáticos disponían de una cantidad muy reducida de memoria, era la pericia del programador la que permitía aprovechar al máximo los recursos que usaban los programas. Fue entonces cuando se originó el mecanismo de *overlay*. Se trataba de una técnica que dividía un programa en varias partes y la ejecución del mismo se realizaba como una secuencia de esas partes: primero se cargaba en memoria una parte y, tras su ejecución, continuaba con la carga de la siguiente.

Pero esta técnica no solucionaba el problema de la limitación de memoria, por lo que se intensificaron los esfuerzos por encontrar un mecanismo general que permitiese la flexibilidad necesaria para el programador. Es el origen de la memoria virtual.

La memoria virtual permite proporcionar a un proceso un mapa de memoria mucho mayor que la memoria física existente en el sistema.

Se han visto algunos ejemplos de memoria virtual en los esquemas de gestión de memoria anteriores, como es el caso de la paginación y la segmentación.

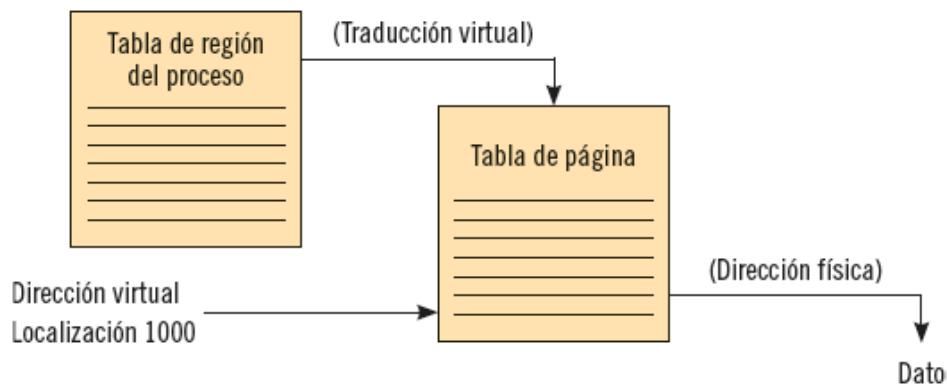
Cuando no se dispone del suficiente espacio en la memoria principal, el sistema operativo guarda aquellas partes del programa que exceden la cantidad de memoria. Cuando se libera el espacio suficiente y la ejecución del programa lo requiere, se traslada a la memoria principal el espacio de memoria necesario para la ejecución del programa.

El sistema de memoria virtual consiste en que cada proceso crea que tiene todo el espacio de memoria disponible, de manera que existe un desacoplamiento entre el espacio de direcciones que usa el programa (direcciones virtuales) y las direcciones físicas donde realmente se almacenan los datos (direcciones físicas). Los sistemas de memoria virtual utilizan la división de los programas en páginas o fragmentos. Estas páginas no necesitan encontrarse físicamente contiguas.

Por ejemplo, cuando un proceso requiere una variable almacenada en la dirección 1000, es necesaria una traducción de esta dirección a la dirección física que realmente

contiene el dato. Se utiliza una estructura denominada tabla de páginas para llevar a cabo esa traducción. Cada proceso posee varias tablas asociadas, correspondientes a los distintos segmentos del programa.

Esquema de traducción de direcciones virtuales a direcciones físicas



Actividades

5. ¿Por qué es necesaria una traducción entre direcciones virtuales y direcciones físicas en un esquema de memoria virtual? Razona la respuesta.

2.3. Ejecución de programas y gestión de procesos

El sistema operativo guarda información de cada proceso, de forma que es posible identificar sus características y todos los recursos que tiene asignados.

De toda la información que es mantenida por el sistema operativo, existe una parte muy importante, conocida como **Bloque de Control del Proceso (BCP)** que contiene, entre otra información, el valor de cada registro del proceso.

Información mantenida por el sistema operativo para un proceso

Información de identificación	Puntero (a otro PCB)
	Identificador del proceso
Información de estado de la CPU	Puntero de instrucciones
	Registros generales
Información de control del proceso	Estado
	Información de manejo de memoria
	Información de E/S
Información de uso de recursos	% de uso de CPU
	Cantidad de mem. usada
	Bytes de E/S leídos/escritos

En el sistema operativo existen dos tipos de procesos que se pueden generar: los procesos de usuario (los que son creados por el usuario) y los procesos de sistema (que forman parte del sistema operativo).

Las funciones más importantes que debe llevar a cabo el módulo de gestión de procesos son:

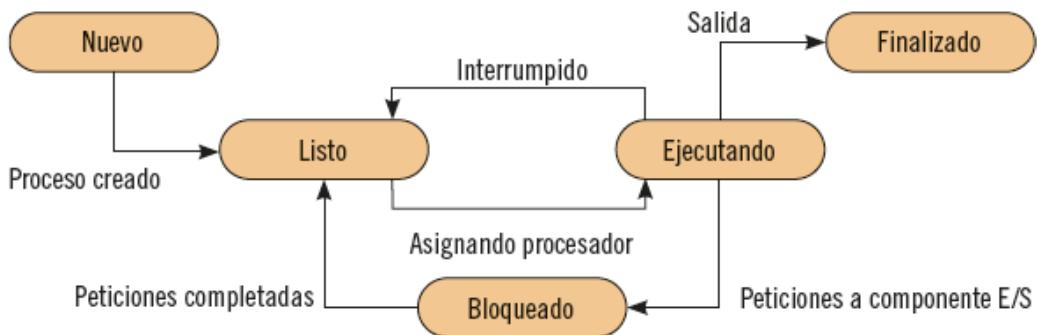
- Intercalar la ejecución de procesos para optimizar el uso del procesador.
- Dar soporte a la comunicación entre procesos.
- Proporcionar mecanismos para la creación y terminación de procesos.
- Proporcionar un mecanismo de asignación de recursos a los procesos.

De todas estas funciones se encarga el planificador de procesos del sistema operativo.

El objetivo de los sistemas multitarea es mantener múltiples programas en ejecución simultáneamente, pero como la CPU solo puede ejecutar un programa cada vez, hay que decidir qué se ejecuta en cada momento. A este mecanismo se le denomina planificación (*scheduling*).

Hoy en día, todos los sistemas operativos importantes se consideran SS. OO. de tiempo compartido. Se usa una fracción de tiempo denominada *quantum*, que representa el tiempo máximo que un proceso puede tener asignada la CPU. Tras este tiempo, el planificador asignará la CPU a otro proceso.

Ciclo de vida de un proceso



El proceso puede pasar por varios estados desde que se crea hasta que se destruye.

Cuando el sistema operativo **ejecuta un programa**, crea un proceso con estado “nuevo”. En este estado, el proceso no ha sido admitido todavía por el SO. En general, estos procesos no han sido cargados aún en la memoria principal. Cuando el SO acepta el proceso, cambia su estado a “listo”. Ahora el proceso está preparado para ser ejecutado, solo espera a que el planificador de corto plazo lo disponga. Cuando el planificador asigna la CPU a un proceso, este pasa al estado de “ejecución”. Solo hay en cada momento un proceso en este estado. Si el proceso realiza una operación que requiere el acceso a un dispositivo de Entrada/Salida, como por ejemplo el disco duro, el proceso pasa al estado “bloqueado” hasta que el dispositivo termina de realizar la operación solicitada por el proceso. Mientras, el procesador se mantiene ocupado con otros procesos. Cuando la operación de Entrada/Salida acaba, el proceso pasa al estado de “listo” de nuevo para que el planificador lo tenga en cuenta. En condiciones normales, un proceso termina cuando se ejecuta su última instrucción. En este caso, el SO libera todos los recursos asignados y todos los datos del proceso.



Aplicación práctica

Una aplicación informática es ejecutada en un sistema operativo. La aplicación realiza una lectura en el disco duro del sistema. Finalmente, el usuario cierra la aplicación. Describa la secuencia de estados en los que se encuentra el proceso de la aplicación en cada momento desde que se inicia hasta que finaliza.

SOLUCIÓN

La aplicación es ejecutada. Se crea el proceso. El estado del proceso es NUEVO.

Después, el proceso es inicializado y preparado para la ejecución. El estado del proceso es LISTO.

El proceso consigue el acceso al procesador y empieza la ejecución. El estado del proceso es EJECUTANDO.

El proceso realiza una petición de E/S al disco para obtener un dato. El proceso permanece en suspensión hasta que el dispositivo entrega la información. El estado del proceso es BLOQUEADO.

El disco duro devuelve los datos. El procesador está ejecutando otro proceso. El estado del proceso pasa a LISTO.

El sistema operativo vuelve a asignar el procesador al proceso y vuelve a ejecutarse. El estado pasa a EJECUTANDO.

Finalmente, el usuario cierra la aplicación. El proceso termina la ejecución liberando todos los recursos. El estado pasa ha FINALIZADO.

2.4. Gestión de almacenamiento. Sistemas de archivos

El sistema operativo necesita manejar una gran cantidad de información, para lo cual usa dispositivos de almacenamiento secundario. Por ello, es necesario que siga un modelo de gestión coherente y uniforme de la información.

Para un SO, el concepto básico de almacenamiento es el archivo. Un archivo es una unidad lógica que contiene información relacionada.

Entre las principales funciones del módulo de gestión de almacenamiento, destacan:

- Realizar la traducción del sistema de direccionamiento lógico al sistema de direccionamiento físico en los dispositivos masivos.
- Realizar la transferencia e intercambio de datos entre la memoria principal y la memoria secundaria.
- Realizar tareas de mantenimiento sobre los dispositivos de memoria secundaria: controlar el estado, la asignación de memoria física y el espacio disponible de cada dispositivo.
- Proporcionar mecanismos para proteger, compartir, recuperar y restaurar los datos almacenados en memoria secundaria.

2.5. Gestión de dispositivos de entrada/salida

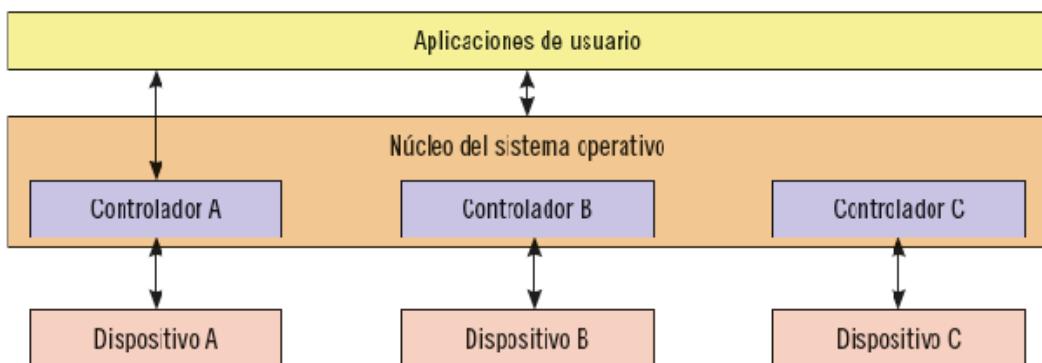
Antes de abordar el tema de la gestión de dispositivos, hay que definir exactamente qué se entiende por un dispositivo de entrada/salida. Cuando se habla de un dispositivo de entrada/salida, se hace referencia a cualquier elemento que no sea ni la memoria ni el procesador.

El problema de un gestor de dispositivos de entrada/salida radica en la existencia de múltiples tipos de dispositivos distintos. Las principales características que definen un dispositivo son:

- **Velocidad.** Existen dispositivos lentos, que transfieren unos cuantos caracteres por segundo. Pero también existen dispositivos muy rápidos, que mueven millones de caracteres por segundo, como por ejemplo los discos duros y los dispositivos de comunicaciones.
- **Unidad de transferencia.** El dispositivo puede transferir bytes como el teclado o el ratón, o bien bloques como los discos.
- **Codificación de la información.** La información que transmite un dispositivo puede ser muy diversa, incluso un mismo dispositivo puede adoptar distintos tipos de codificación en instantes diferentes.
- **Protocolo de comunicación.** El lenguaje entre el dispositivo y la CPU depende del mismo y de las conexiones.
- **Operaciones.** Existen dispositivos solo de entrada, solo de salida y de entrada/salida.
- **Errores.** Los mensajes de error son comunes y varían dependiendo del tipo de dispositivo.

Para aislar de la complejidad que significaría tratar de comunicarse con el dispositivo por parte de la CPU a un nivel muy bajo, los proveedores del dispositivo proporcionan el *software* propio para acceder a todos sus servicios de una forma más cómoda y transparente al programador y/o SO. Este *software* se denomina controlador. El SO utilizará la interfaz proporcionada por el controlador para entenderse con el dispositivo.

Sistema de controladores en un sistema operativo



Entre las principales funciones dentro del módulo de gestión de dispositivos de E/S, se encuentran:

- Realizar la comunicación, mediante comandos, sobre los dispositivos. Gestionar las interrupciones que producen dichos dispositivos y saber interpretar los errores.
- Proporcionar una interfaz simple y flexible para que los procesos del sistema puedan usar los dispositivos.
- Proporcionar servicios para la emulación de dispositivos virtuales que permitan usar la interfaz de E/S como si se estuviera tratando con un dispositivo de E/S real.
- Asegurar el funcionamiento de un dispositivo E/S cuando es conectado por primera vez.



Actividades

6. ¿Qué mecanismo es usado por los dispositivos de E/S cuando requieren el uso del procesador en un momento determinado? **interrupción**

2.6. Gestión de red

Todos los SS. OO. modernos ponen a disposición del usuario un conjunto de herramientas que permiten configurar el subsistema de red. La complejidad de este subsistema queda completamente oculta bajo la interfaz que proporcionan. En la mayoría de los casos, solo hacen falta conocimientos sobre el tipo y estructura de la red, ya que el sistema operativo se encargará del resto.

El subsistema de gestión de red permite que los procesos del sistema operativo interactúen a través de un canal de comunicación con el enfoque de una red de comunicación.

Suele estar dividido en dos partes:

- En el *kernel* se implementan las tareas de bajo nivel para la comunicación entre sistemas, pila de protocolos TCP/IP, controladores de red, etc.
- En el espacio de usuario, se encuentran programas y ficheros que configuran los parámetros relacionados con la red, dirección IP, tablas de enruteado, etc.

En *Windows*, algunos de estos programas son: *Ping*, *Nslookup*, *Netstat*, etc.

2.7. Gestión de errores

Cuando se ejecuta una aplicación en el sistema operativo, puede ocurrir que el proceso realice su trabajo correctamente o bien, que se produzca algún error. En la mayoría de los casos, un error incontrolado en un sistema puede ocasionar la pérdida de información e incluso la inestabilidad del sistema operativo si se está haciendo uso de recursos del mismo.

Los SS. OO. modernos incorporan mecanismos dirigidos fundamentalmente a minimizar el impacto que pueden ocasionar estos errores y, a la misma vez, proporcionan las herramientas necesarias para que los desarrolladores puedan probar y mejorar los programas que utilizarán el resto de los usuarios.

Cuando se produce un error, el resultado es una interrupción de la ejecución normal del programa que lo causa. A continuación, el SO otorga el control de ejecución a la rutina adecuada para que trate el error. Todo el tratamiento de estos errores se conoce como gestión de excepciones y existen sistemas operativos que lo soportan y otros que no.

Las excepciones se pueden clasificar en los siguientes tipos:

- **Fallos**, que pueden ser corregidos y que retoman la ejecución normal del programa que las generó.
- **Traps**, que son utilizadas por los programadores para depuración.
- **Aborts**, que son errores graves que ocurren cuando hay un fallo de *hardware*.

La clasificación anterior se refiere a excepciones que son detectadas por el procesador. Sin embargo, también pueden ocurrir excepciones de forma programada. Para ello, se usa la instrucción **int** o **int3**. Algunas de las excepciones más comunes en *Linux* aparecen en la siguiente tabla.

Tabla con algunas de las excepciones más comunes en Linux

#	Exception	Exception handler	Signal
0	Divide error	divide_error()	SIGFPE
1	Debug	debug()	SIGTRAP
2	NMI	nail()	None
3	Breakpoint	int3()	SIGTRAP
4	Overflow	overflow()	SIGSEGV
5	Bounds check	bounds()	SIGSEGV
6	Invalid opcode	invalid_op()	SIGILL
7	Device not available	device_not_available()	None
8	Double fault	doublefault_fn()	None
9	Compressor segment overrun	compressor_segment_overrun()	SIGFPE
10	Invalid TSS	invalid_TSS()	SIGSEGV
11	Segment not present	segment_not_present()	SIGBUS
12	Stack segment fault	atack_segment()	SIGBUS
13	General protection	general_protection()	SIGSEGV
14	Page Fault	page_fault()	SIGSEGV
15	Intel-reserved	None	None
16	Floating-point error	coprocessor_error()	SIGFPE
17	Alignment check	aligment_check()	SIGBUS
18	Machine check	machine_check()	None
19	SIMD floating point	aimd_coproceasor_error()	SIGFPE

2.8. Gestión de la seguridad

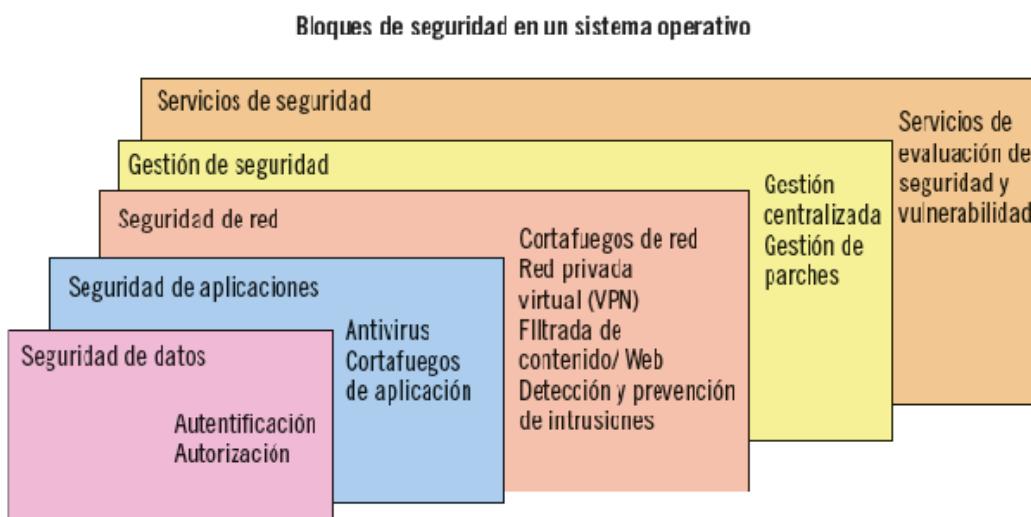
La seguridad es importante en un sistema operativo, ya que se está hablando de un *software* que controla todo el sistema a nivel de *hardware* y de *software*. Sin embargo, el concepto de seguridad es muy amplio y dependiendo del ámbito puede significar una u otra cosa.

En términos informáticos, se dice que la seguridad es equivalente a garantizar una serie de propiedades. En el caso de un sistema operativo, esas propiedades se aplicarían sobre los usuarios del mismo:

- **Consistencia:** el sistema debe comportarse como se espera y mantener ese comportamiento sin que se produzcan cambios inesperados.
- **Servicio:** el sistema ofrece los servicios de una manera confiable, constante y consistente.
- **Protección:** el entorno de ejecución de un proceso no debe afectar al de otro. De esta manera, si un programa tiene errores y detiene su ejecución, no debe afectar al sistema.
- **Control de acceso:** el sistema debe ofrecer un mecanismo de acceso a los datos que garantice la confidencialidad de los mismos con respecto a otros usuarios.
- **Autenticación:** el sistema debe ofrecer un mecanismo para identificar al usuario que accede al sistema y proporcionar los privilegios adecuados en función del tipo de usuario.

El sistema operativo debe encargarse de garantizar la seguridad a todos los niveles, implementando un modelo de gestión de la seguridad centralizado, en la mayoría de los casos apoyado por un sistema de gestión de actualizaciones que las principales compañías ponen a disposición conforme encuentran agujeros de seguridad en sus sistemas.

Para hacerse una idea de cuáles son los principales mecanismos que afectan de manera directa a la seguridad de todo el sistema, se puede observar el siguiente diagrama:



2.9. Auditoría (logs del sistema)

Todos los sistemas operativos poseen mecanismos para detectar errores y registrar la información referente a los mismos para que los administradores puedan analizarla y actuar en consecuencia. Estos registros se almacenan en ficheros denominados *logs*.

En las distribuciones *Linux*, estos *logs* se encuentran en las siguientes ubicaciones del sistema de ficheros:

- /var/log/message: registro de mensajes generales del sistema.
- /var/log/auth.log: *log* de autenticación.
- /var/log/kern.log: registro del *kernel*.
- /var/log/cron.log: registro de *crond*.
- /var/log/maillog: registro del servidor de *mails*.
- /var/log/qmail/: registro de *Qmail*.
- /var/log/httpd/: registro de errores y accesos a *Apache*.
- /var/log/lighttpd: registro de errores y accesos a *Lighttpd*.
- /var/log/boot.log: registro de inicio del sistema.
- /var/log/mysqld.log: registro de la base de datos MySQL.
- /var/log/secure: *log* de autenticación.
- /var/log/utmp o /var/log/wtmp: registro de *logins*.

En el caso de *Windows*, depende de la versión del sistema operativo. Pero, por regla general, se encuentra accediendo a las herramientas administrativas desde el panel de control. Dentro de la ventana de herramientas administrativas, al pulsar sobre el visor de eventos.

2.10. Procesos de arranque (*boot*) y finalización del sistema (*shutdown*)

Todo sistema operativo necesita un proceso de arranque y parada. El proceso de arranque prepara el sistema para que los usuarios puedan usarlo. Este proceso se divide en dos partes: arranque *hardware* y arranque del SO. El arranque *hardware* se realiza bajo la supervisión de la BIOS y es propio del *hardware*, no del sistema operativo. Se realiza un test de *hardware* y se carga en memoria el registro de arranque maestro (MBR), alojado en el primer sector del disco de inicio. El MBR contiene información sobre las particiones del disco de inicio, así como sobre cuál es la partición activa.

Tras esto, el control se pasa al MBR. El MBR busca en la partición activa el sector de arranque (que se encuentra en los primeros sectores de la partición) y lo carga en memoria. Estos sectores contienen el cargador del SO. Una vez cargado en memoria, se le otorga el control y comienza la carga real del SO con la siguiente secuencia de

pasos: test del sistema de ficheros, creación de las estructuras de datos internas, completado de la carga del SO residente y creación del proceso de *login* para la autenticación del usuario.

Tras esta fase, comienza la de funcionamiento normal del SO.

En el caso de *Linux*, la secuencia de pasos del cargador sería la siguiente:

- Se carga en memoria el *kernel*.
- Se cargan los módulos necesarios y se monta la partición root en modo solo lectura. En concreto, se busca la imagen initrd, se descomprime, se monta y se cargan los controladores necesarios.

```
[ 0.000000] Linux version 5.15.0-53-generic (buildd@lcy02-amd64-047) (gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #59-Ubuntu SMP Mon Oct 17 18:53:30 UTC 2022 (Ubuntu 5.15.0-53.59-generic 5.15.64)
[ 0.000000] Command line: BOOT_IMAGE=/vmlinuz-5.15.0-53-generic root=/dev/vda1 ro
[ 0.000000] KERNEL supported cpus:
[ 0.000000]   Intel GenuineIntel
[ 0.000000]   AMD AuthenticAMD
[ 0.000000]   Hygon HygonGenuine
[ 0.000000]   Centaur CentaurHauls
[ 0.000000]   zhaoxin Shanghai
[ 0.000000] Disabled fast string operations
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
```

- Se inicializan los ficheros antes de desmontar initrd.
- Se crea el dispositivo "root", se monta la partición "root" (solo lectura) y se libera la memoria no usada. El *kernel* se encuentra cargado en memoria y operativo.
- El *kernel* transfiere el control del proceso de arranque a /sbin/init.
- /sbin/init carga servicios y herramientas de espacio del usuario y monta las particiones listadas en /etc/fstab.

```
Welcome to Ubuntu 22.04.1 LTS

[ 12.062709] systemd[1]: Hostname set to <ubuntu>.
[ 12.469544] systemd[1]: Queued start job for default target Graphical Interface.
[ 12.487733] systemd[1]: Created slice Slice /system/modprobe.
[ OK ] Created slice Slice /system/modprobe.
[ 12.496049] systemd[1]: Created slice Slice /system/systemd-fsck.
[ OK ] Created slice Slice /system/systemd-fsck.
[ 12.505086] systemd[1]: Created slice User and Session Slice.
[ OK ] Created slice User and Session Slice.
[ 12.512688] systemd[1]: Started Forward Password Requests to Wall Directory Watch.
[ OK ] Started Forward Password Requests to Wall Directory Watch.
[ 12.520584] systemd[1]: Set up automount Arbitrary Executable File Formats File System Automount Point.
[ OK ] Set up automount Arbitrary Executable File Formats File System Automount Point.
[ 12.532573] systemd[1]: Reached target Slice Units.
[ OK ] Reached target Slice Units.
[ 12.540498] systemd[1]: Reached target Swap.
[ OK ] Reached target Swap.
[ 12.547894] systemd[1]: Reached target Local Verity Protected Volumes.
[ OK ] Reached target Local Verity Protected Volumes.
[ 12.555966] systemd[1]: Listening on Device-mapper event daemon FIFOs.
[ OK ] Listening on Device-mapper event daemon FIFOs.
[ 12.564189] systemd[1]: Listening on LVM2 poll daemon socket.
[ OK ] Listening on LVM2 poll daemon socket.
[ 12.573363] systemd[1]: Listening on multipathd control socket.
[ OK ] Listening on multipathd control socket.
[ 12.581780] systemd[1]: Listening on Syslog Socket.
[ OK ] Listening on Syslog Socket.
[ 12.590039] systemd[1]: Listening on fsck to fsckd communication Socket.
[ OK ] Listening on fsck to fsckd communication Socket.
[ 12.598198] systemd[1]: Listening on initctl Compatibility Named Pipe.
[ OK ] Listening on initctl Compatibility Named Pipe.
[ 12.606194] systemd[1]: Listening on Journal Audit Socket.
```

- Finalmente, se presenta la interfaz de acceso de usuario.

```
[ OK ] Finished Terminate Plymouth Boot Screen.
[ OK ] Finished Set console scheme.
[ OK ] Created slice Slice /system/getty.
[ OK ] Started Getty on tty1.
[ OK ] Reached target Login Prompts.
[ OK ] Started Authorization Manager.
      Starting Modem Manager...
[ OK ] Finished Record successful boot for GRUB.
      Starting GRUB failed boot detection...
[ OK ] Started OpenBSD Secure Shell server.
[ OK ] Started LSB: automatic crash report generation.
[ OK ] Finished GRUB failed boot detection.
[ OK ] Finished Remove Stale Online ext4 Metadata Check Snapshots.
[ OK ] Finished Rotate log files.
[ OK ] Started User Login Management.
[ OK ] Started Unattended Upgrades Shutdown.
[ OK ] Started Disk Manager.
[ OK ] Started Modem Manager.
[ OK ] Started Dispatcher daemon for systemd-networkd.
[ OK ] Started Snap Daemon.
      Starting Wait until snapd is fully seeded...
      Starting Time & Date Service...
[ OK ] Started Time & Date Service.
[ OK ] Finished Wait until snapd is fully seeded.
      Starting Apply the settings specified in cloud-config...
[ OK ] Finished Service for snap application lxd.activate.
[ 21.761416] cloud-init[1027]: Cloud-init v. 22.3.4-0ubuntu1~22.04.1 running 'modules:config' at Wed, 23 Nov 2022 09:01:34 +0000. Up 21.70 seconds.
[ OK ] Finished Apply the settings specified in cloud-config.
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.
      Starting Execute cloud user/final scripts...
      Starting Record Runlevel Change in UTMP...
[ OK ] Finished Record Runlevel Change in UTMP.
[ 22.268819] cloud-init[1033]: Cloud-init v. 22.3.4-0ubuntu1~22.04.1 running 'modules:final' at Wed, 23 Nov 2022 09:01:35 +0000. Up 22.21 seconds.
```

Para *Windows*, la secuencia de arranque es la siguiente:

1. La BIOS accede al MBR indicando cuál es la partición activa.
2. Se accede al sector de arranque o *boot* de dicha partición, que ha sido creado durante la instalación del SO.
3. El sector de arranque dirige al fichero NTLDR (para *XP*), o bien BOOTMGR sustituto del fichero NTLDR en sistemas posteriores a *Windows XP*.
4. NTLDR o BOOTMGR accede al fichero BOOTBCD, que contiene información de la ubicación de otros sistemas instalados en el equipo. Si los hay, aparece un menú de arranque para elegir con cuál se quiere iniciar el equipo.
5. Comienza el inicio del sistema ejecutando el *kernel* ubicado en el fichero NTOSKRNL.EXE.

El proceso de finalización o parada del sistema operativo lleva a cabo la liberación de todos los recursos del sistema, así como la parada y destrucción de todos los procesos que se encuentran en ejecución.



Aplicación práctica

Imagine que su sistema operativo está instalado en la única partición del disco duro del sistema y que, por lo tanto, en los primeros sectores del disco se encuentra el programa que se encarga de cargar el sistema operativo. Describa la secuencia de pasos genéricos que se producen desde que arranca el equipo hasta que se produce la petición de autenticación en el sistema operativo.

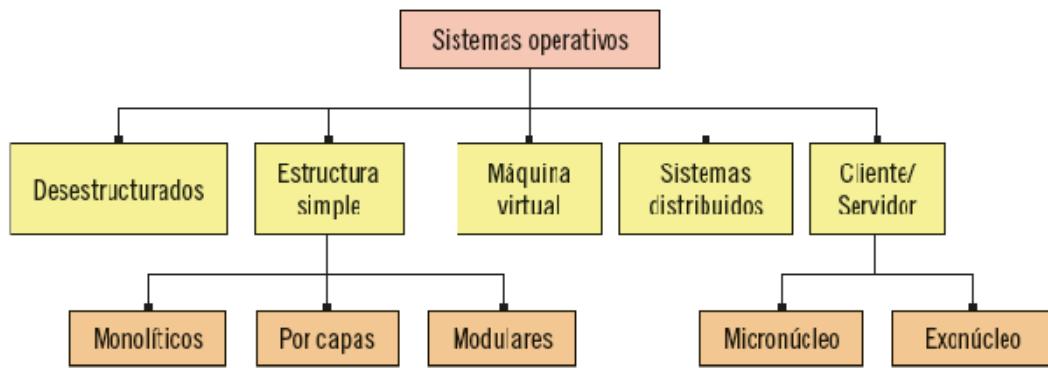
SOLUCIÓN

- I. La BIOS accede al MBR, que se encuentra en los primeros sectores del disco.
 - II. El MBR determina cuál es la partición activa.
 - III. Se accede a los primeros sectores de la partición activa donde se encuentra el programa de arranque.
 - IV. Se ejecuta el programa dentro del sector de arranque. Este programa carga en memoria el embrión del sistema operativo.
 - V. Después, el programa de arranque transfiere el control de ejecución al embrión del sistema operativo para que termine la carga de los demás módulos.
 - VI. Cuando el sistema operativo ha alcanzado un nivel de carga suficiente, se ejecuta el programa de autenticación de usuario.
-

3. Características estructurales de los sistemas operativos

La influencia final que tiene la arquitectura del sistema operativo sobre todos los aspectos funcionales del mismo es tan grande que es necesario un estudio detallado de los modelos básicos para entenderla. A lo largo del tiempo, la estructura interna de los principales sistemas operativos ha evolucionado hasta adaptarse a un modelo que permite un mejor desarrollo de aplicaciones para el sistema operativo. Una primera clasificación que es posible hacer es la siguiente:

Clasificación de los modelos estructurales de un sistema operativo

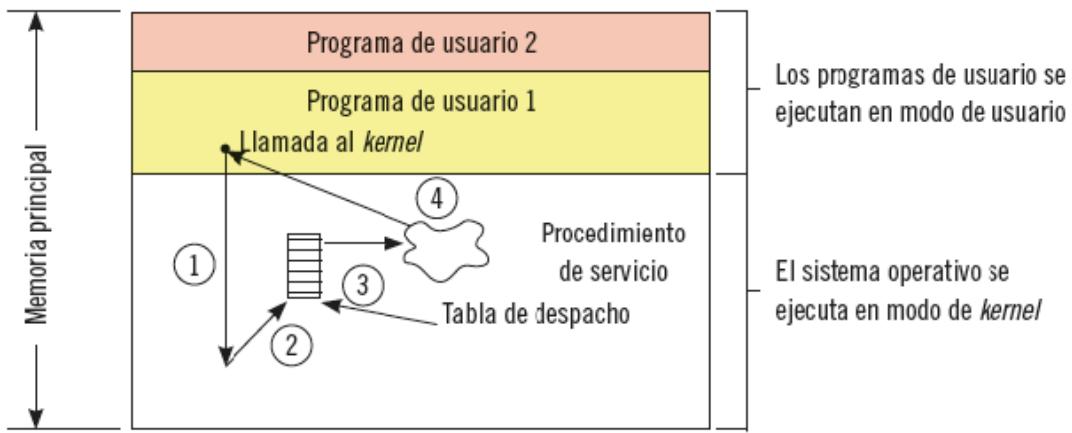


3.1. Sistemas monolíticos

En un sistema operativo con este tipo de arquitectura, todas las funciones se encuentran codificadas en un único módulo. Está compuesto de un conjunto de procedimientos, que se pueden invocar entre ellos según la necesidad y sin restricciones. Cada procedimiento tiene una firma bien definida, es decir, los parámetros de entrada y el resultado de salida son conocidos.

Hay que tener en cuenta que este tipo de modelo no es lo mismo que un modelo sin estructura. Realmente, existe una estructura, aunque sea muy primitiva. Todo sistema operativo puede ejecutarse en dos modos: modo núcleo (*modo kernel*) o modo usuario. En un sistema monolítico, los procedimientos correspondientes a los servicios del sistema operativo (llamadas al sistema) se pueden solicitar estableciendo los parámetros de entrada y realizando una llamada especial (llamada al *kernel*) para ejecutar el procedimiento.

La llamada especial realiza el cambio entre el modo usuario y el modo *kernel* y transfiere el control de la ejecución al sistema operativo.



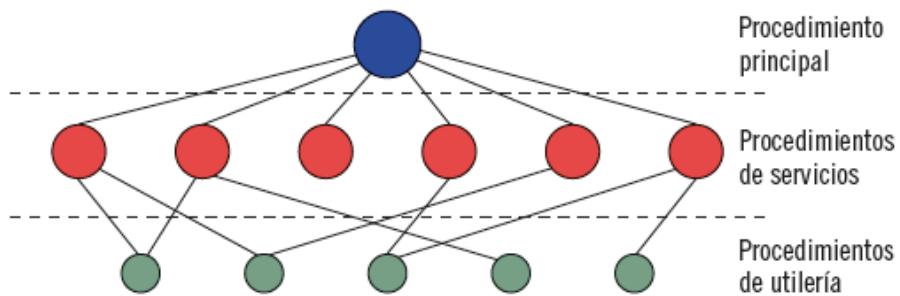
Cómo puede realizarse una llamada al sistema: (1) El programa de usuario entra en el kernel por una trampa mediante una operación Trap. (2) El sistema operativo determina el número de servicio requerido. (3) El sistema operativo invoca el procedimiento de servicio. (4) Se devuelve el control al programa de usuario.

El sistema operativo comprueba los parámetros de entrada y, en función de ellos, busca en una tabla interna la referencia al procedimiento que se debe ejecutar en función de estos parámetros. Se ejecuta y el resultado se transfiere al programa de usuario que realizó la llamada. Por lo tanto, se vuelve a pasar al modo usuario.

Este tipo de arquitectura se puede ver como un modelo de tres niveles:

- Un programa principal que realiza la llamada al servicio del SO.
- Un conjunto de procedimientos de servicio que realizan las llamadas al sistema.
- Un conjunto de procedimientos auxiliares que ayudan a los anteriores y que suelen ser comunes para varios procedimientos de servicio.

Modelo de tres niveles de un sistema monolítico



Entre los principales inconvenientes, se encuentran:

- Falta de modularidad.

- Dificultad para modificaciones.
- Dificultad en la detección y corrección de errores.
- Necesidad de recompilación en cada cambio.

3.2. Microkernels

La idea de esta arquitectura es que un pequeño módulo (*microkernel*) proporcione la funcionalidad básica del SO. Y que ese módulo sea extendido con ampliaciones. De esta manera, solo las funciones fundamentales y esenciales deben permanecer en el *microkernel*. Todas las demás funciones del SO se ejecutarán fuera del *microkernel* en modo usuario.

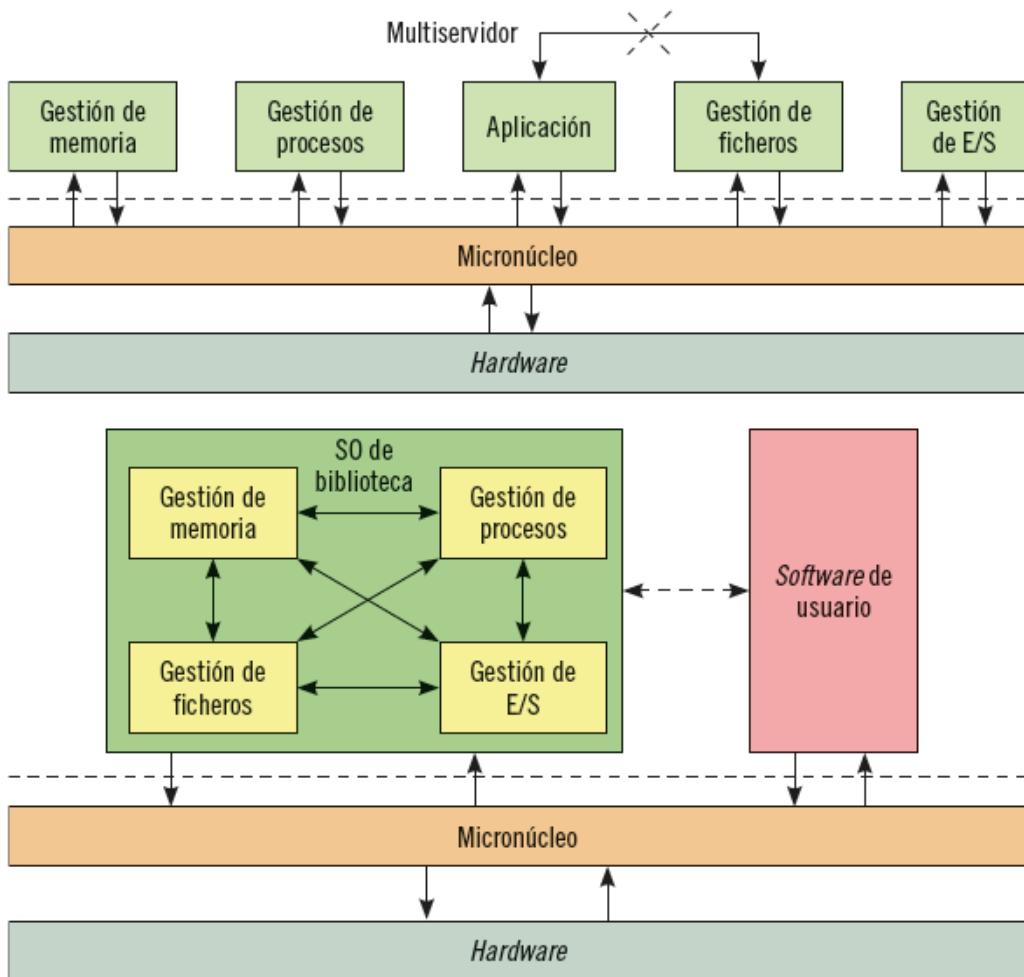
Por lo general, este tipo de arquitectura está compuesto de módulos o servidores que proporcionan la funcionalidad tradicional del SO. Algunas ventajas son:

- Uniformidad en las interfaces: la solicitud de servicios está bastante bien definida.
- Es fácil añadir nuevos servicios (extensibilidad).
- Existe un alto grado de configuración del sistema (flexibilidad).
- La detección y corrección de errores se puede realizar a nivel de *microkernel*, lo que se traduce en un sistema muy más seguro.

Y, por lo que respecta a sus desventajas:

- Sobrecarga en las comunicaciones, lo que implica menos eficiencia.

Modelo de estructura basada en *microkernels*



Actividades

7. ¿Cuáles son las principales diferencias entre un sistema operativo monolítico y un sistema operativo basado en *microkernels*?

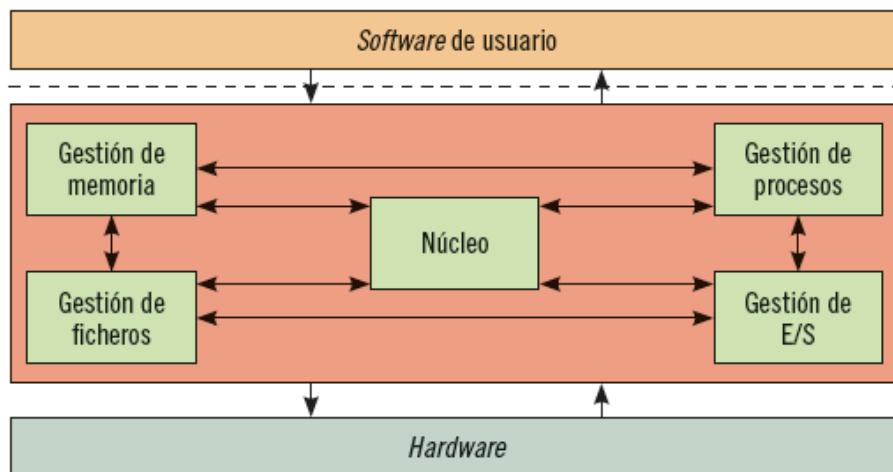
3.3. Sistemas modulares y por capas

La estructura modular, como su propio nombre indica, se basa en que la funcionalidad del sistema operativo se reparte en módulos y/o procesos distintos. Se

trataría de componentes lógicos independientes con interfaces muy bien definidas. Los módulos o procesos pueden apoyarse en un pequeño módulo básico que proporciona la funcionalidad mínima: E/S básica, memoria, etc.

Por lo general, este modelo es aceptable cuando se habla de sistemas operativos de propósito general. Sin embargo, no es adecuado cuando se necesita optimizar al máximo la eficiencia para un propósito más específico.

Modelo estructural basado en módulos

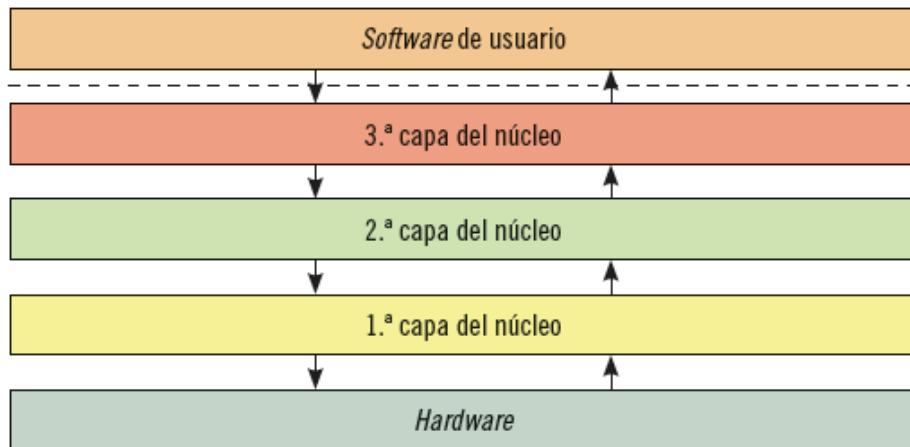


Entre los diversos inconvenientes, destacan:

- Falta de protección y fiabilidad.
- Menos flexible que la arquitectura monolítica.

En la arquitectura por capas, también llamada por anillos concéntricos, el sistema operativo se compone de niveles. Una jerarquía de niveles, donde cada nivel envuelve a otros dispuestos más internamente o más cercanos al nivel inferior, que suele ser el que corresponde con el *kernel* o núcleo del SO.

Modelo estructural basado en capas



Sus principales ventajas son:

- Modularidad.
- Fácil depuración y verificación de cada capa por separado.

Entre los inconvenientes, destacan:

- Falta de protección y fiabilidad.
- Menos flexible que la arquitectura monolítica.
- Alto costo en la definición de cada capa en la etapa de diseño.

Dentro de este tipo de arquitectura, existen varias clases específicas:

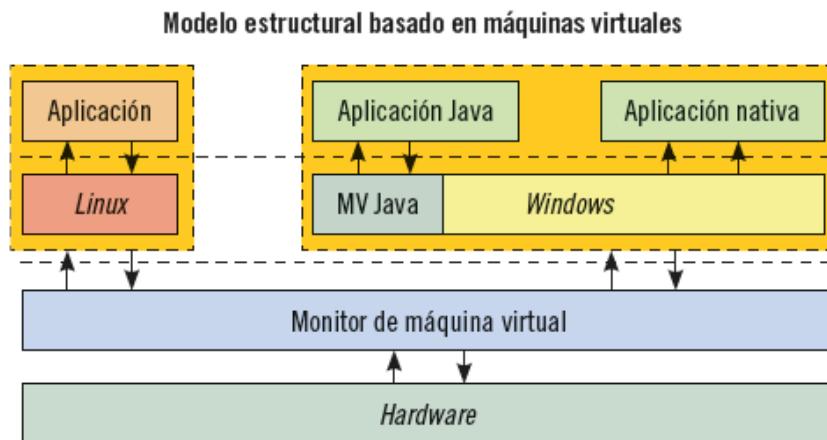
- Máquinas virtuales.
- *Exokernels*.
- Modelos cliente-servidor.

La diferencia fundamental entre ellos radica en los servicios que proporciona el *microkernel*.

3.4. Máquinas virtuales

Las máquinas virtuales son un tipo de arquitectura basada en *microkernel*. En este modelo, el *microkernel* se denomina monitor de máquina virtual y se ejecuta sobre el *hardware* directamente, con soporte para multiprogramación. Además proporciona una visión de múltiples máquinas virtuales a la capa superior. Cada máquina virtual es una copia exacta del *hardware*. Esto consigue que se puedan ejecutar varios sistemas operativos a la vez.

La idea principal es que los procesos se ejecutan limitados por los recursos y abstracciones que proporciona cada máquina virtual.



Sus principales ventajas son:

- Existe una perfecta protección entre componentes.
- Se aprovecha mejor el *hardware*.
- La reutilización de código es máxima.

En cuanto a sus inconvenientes, destaca que:

- La simulación del *hardware* real es costosa, lo que implica que la ejecución sea poco eficiente.



Aplicación práctica

Desea diseñar un sistema operativo en el que pueda intercambiar módulos que serán programados en diferentes tipos de lenguaje. Imagine que es uno de los ingenieros informáticos responsables de determinar el diseño arquitectónico. Presente el diseño del sistema operativo adecuado y defienda su propuesta.

SOLUCIÓN

Podría escoger un diseño basado en *microkernel* o bien un diseño modular. En ambos casos se sugiere una arquitectura basada en entidades lógicas independientes con una interfaz bien definida, por lo que sería fácil elaborar las funciones del sistema operativo en diferentes lenguajes con total compatibilidad entre ellas.

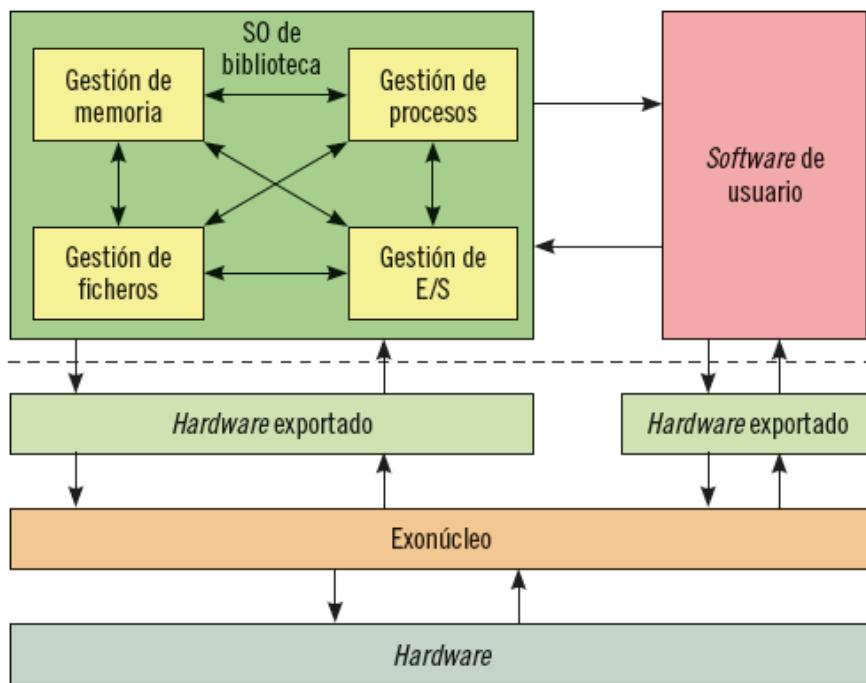
En la arquitectura *microkernel*, el micronúcleo proporciona la funcionalidad básica del sistema operativo y debería estar implementado en el mismo lenguaje para evitar perder consistencia y eficiencia. Los demás módulos actuarán como procesos servidores en el espacio de usuario. Por lo tanto, la implementación de cada módulo puede ser considerada independiente. El único requisito es que se permita la comunicación entre los diferentes módulos.

En el caso de la arquitectura basada en módulos, se puede utilizar un enfoque orientado a objetos. La estructura del sistema operativo se compone de módulos totalmente independientes entre ellos, de forma que, si alguno falla, no afecta a ningún proceso iniciado por los otros módulos. Incluso se pueden cargar dinámicamente. Esto permite aplicar un desarrollo basado en componentes sobre el sistema operativo e implementar cada componente con un lenguaje diferente sin afectar a la arquitectura.

3.5. Exokernel

En esta arquitectura, apenas existe SO. Se basa en un módulo que actúa como un gestor de recursos. Este modelo permite que las aplicaciones accedan directamente al *hardware* exportado directamente por el *kernel*. En concreto, el *exokernel* presenta el *hardware* como abstracciones de alto nivel, permitiendo a las aplicaciones tomar tantas decisiones como sea posible sobre esas abstracciones. El *exokernel* se dedica exclusivamente a garantizar la protección y el multiplexado de los recursos.

Modelo de estructura basada en *exokernels*



Definición

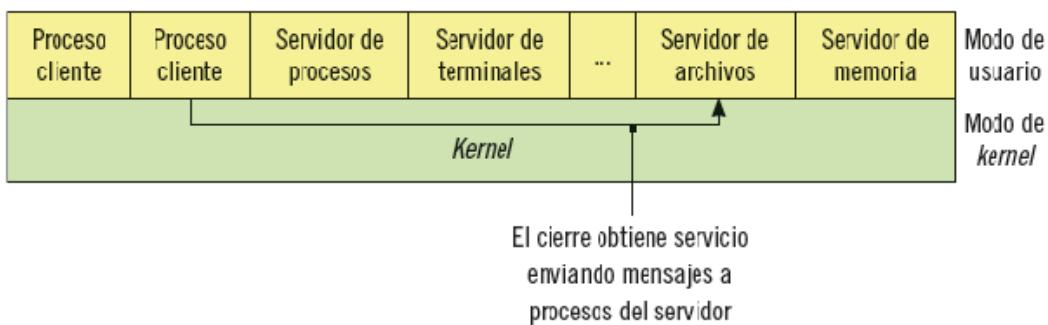
Multiplexar

Combinar más de un canal de información a través del mismo medio de transmisión usando algún dispositivo o mecanismo que lo permita.

3.6. Modelo cliente-servidor

Como se ha dicho antes, este modelo se basa en una arquitectura de *microkernel* donde el enfoque consiste en que la mayor parte de las funciones del sistema se encuentran distribuidas en procesos de usuario. Estos procesos envían solicitudes de servicios del SO a procesos servidores. El *kernel* se encarga de la comunicación entre los procesos clientes y los procesos servidores y su ejecución se realiza en modo usuario, por lo que no tienen acceso directo al *hardware*.

Modelo estructural basado en cliente/servidor



3.7. Sistemas distribuidos

Esta arquitectura apuesta por descomponer el sistema operativo en partes y ejecutar cada una de ellas en un nodo de red distinto. Sin embargo, debe proporcionar la transparencia adecuada para que el usuario no perciba este hecho. Proporciona las mismas funciones que un sistema operativo normal. Su principal objetivo es facilitar el acceso y la gestión de los recursos distribuidos en la red.

En el sistema operativo distribuido, un usuario accede a los recursos remotamente sin percatarse de ello. Tanto los trabajos, las tareas y los procesos se encuentran distribuidos en un conjunto de procesadores.

Los sistemas distribuidos están basados en las siguientes claves:

- **Transparencia:** la ejecución de programas y tareas se realiza como si solo existiera una máquina física, aunque la eficiencia se ve incrementada por el uso de los elementos distribuidos.
- **Eficiencia:** el hecho de usar más de un procesador para la ejecución incrementa sustancialmente la eficiencia de los programas.
- **Flexibilidad:** permite realizar cambios y actualizaciones sin ver reducida su funcionalidad ni su eficiencia.
- **Escalabilidad:** el despliegue del SO no está sujeto a un número determinado de máquinas remotas. Por lo tanto, se podrá usar un número arbitrario de elementos de red.
- **Seguridad:** al igual que en los sistemas operativos tradicionales, la seguridad es garantizada con el control de permisos adecuado. En este caso, se hace más hincapié en los recursos compartidos.

Aunque, debido a que todos estos aspectos suelen ser contrarios en parte, se debe llegar a un compromiso para su ejecución.

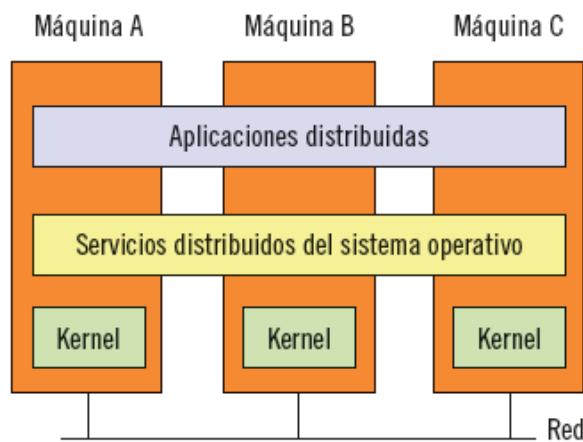
Sus principales ventajas son:

- La velocidad de ejecución es superior.
- Alto grado de fiabilidad. Si falla un elemento de red, el sistema no cae.

En cuanto a sus inconvenientes:

- Hay poco software adaptado a la ejecución distribuida.
- La saturación de la red influye muy negativamente en el rendimiento del sistema.

Modelo estructural de un sistema operativo distribuido



Actividades

8. ¿Por qué son más fiables los sistemas operativos distribuidos que los sistemas operativos no distribuidos? Razona la respuesta.

4. Herramientas administrativas de uso común en sistemas operativos

Desde el punto de vista del usuario, se necesita una herramienta adecuada para proporcionar órdenes al SO o, lo que es lo mismo, el usuario debe poder acceder a los servicios que proporciona el SO. Estas herramientas se conocen como interfaz de

usuario. La idea fundamental de este tipo de herramientas es la de mediar entre el hombre y la máquina, facilitar la comunicación, la interacción, entre dos sistemas completamente diferentes.

Desde un punto de vista técnico, la interfaz de usuario se define como un conjunto de componentes que son empleados por el usuario para comunicarse con el ordenador. El usuario controla el funcionamiento del ordenador mediante instrucciones, denominadas entradas. Estas entradas pueden ser introducidas por varios tipos de dispositivos: teclado, ratón, etc. De esta forma, el sistema las procesa y emite el resultado a otro dispositivo, como la impresora o la pantalla.

A lo largo de la historia, los sistemas operativos más importantes han desarrollado interfaces que se han convertido en señas de identidad propias. Es el caso de *Windows* y *Linux*.

Pero, en términos generales, se puede hablar de los siguientes tipos:

- **Interfaces de línea de comandos.** Se trata de que el control del sistema se realiza a través de comandos de texto, que son programas que realizan operaciones sobre el sistema.
- **Interfaces gráficas de usuario (GUI, Graphics User Interfaces).** Son controles visuales que permiten la configuración del sistema de una forma fácil e intuitiva.
- **Interfaces táctiles.** Se trata del tipo de interacción que está de moda actualmente. Podría verse como una evolución del tipo de interfaz gráfica, pero añadiendo que el control se realiza a través de las manos.

4.1. Interfaces gráficas de usuario

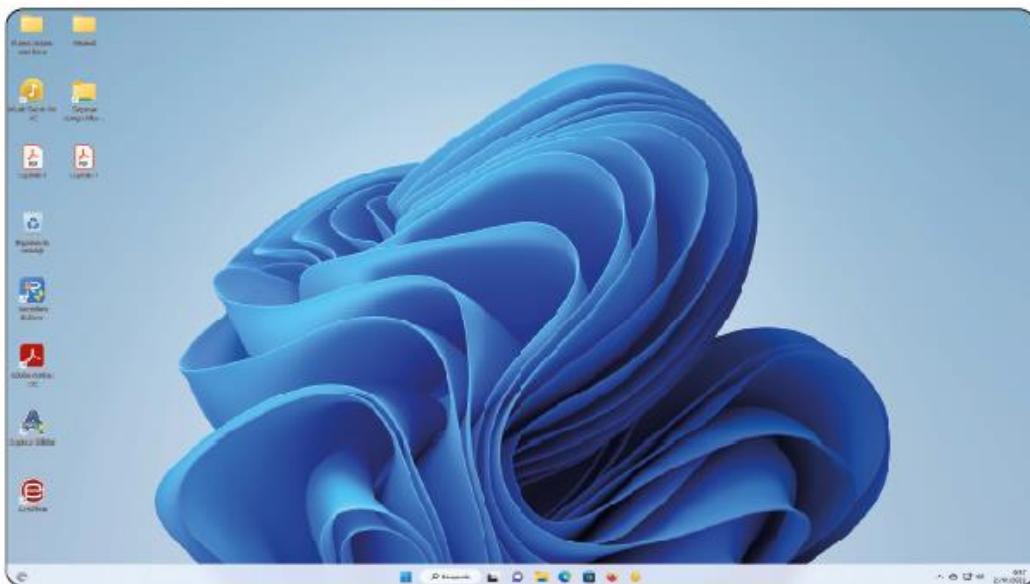
La interfaz de usuario en modo gráfico se ha convertido en el sistema de comunicación preferido entre el usuario y la máquina para configurar y controlar los sistemas operativos modernos. Se basa en un conjunto de componentes gráficos: ventanas, iconos, menús desplegables, etc. que permiten realizar acciones al usuario de una manera intuitiva. Este tipo de interfaces evita que el usuario final necesite tener grandes conocimientos informáticos para el manejo del SO. En la mayoría de los casos, la interfaz gráfica de usuario realiza una traducción de la acción realizada por el usuario a un conjunto de comandos que son ejecutados por el sistema operativo de forma similar a como se realizaría desde un intérprete de comandos.

El ejemplo más típico de este tipo de interfaces se puede encontrar en los sistemas operativos de Microsoft: *Windows*.



Actividades

9. ¿Se puede considerar el uso del ratón como parte de una interfaz del sistema operativo? Rzone la respuesta.
-

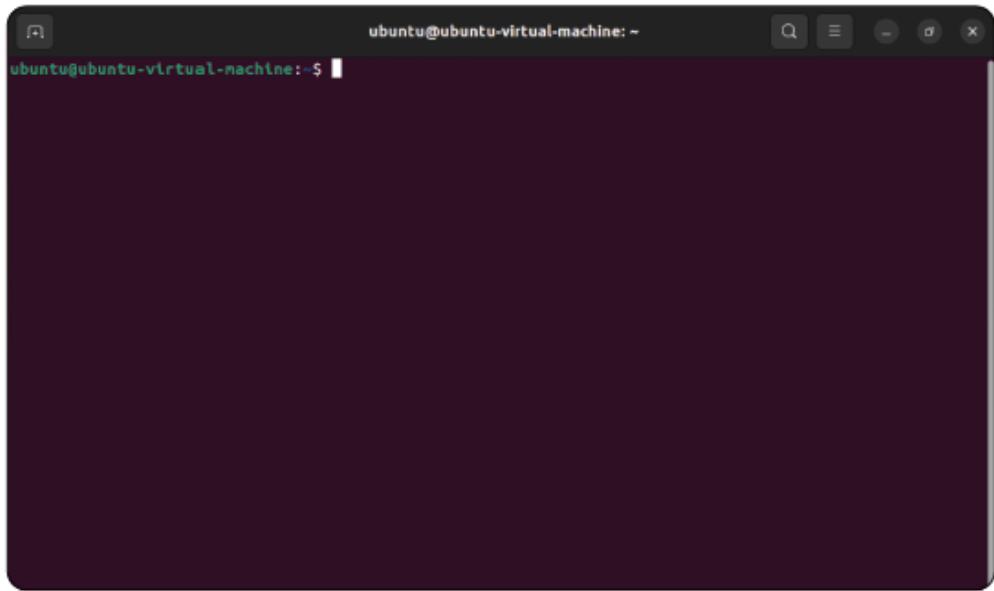


Interfaz gráfica de Windows 11 de Microsoft

4.2. Intérpretes de comandos

En este tipo de interfaces las entradas son comandos, es decir, programas informáticos que son capaces de traducir las órdenes que se introducen a un conjunto de instrucciones que realizan llamadas al sistema. Por lo general, cada comando posee un conjunto de parámetros y valores para dichos parámetros que permiten configurar el mandato o la acción a realizar mediante el comando.

El ejemplo más característico de este tipo de interfaces es el terminal de *Linux*. Las primeras distribuciones de *Linux* solo utilizaban un intérprete de comandos para el control del sistema. Hoy en día, también poseen interfaz gráfica.



Interfaz de usuario del sistema operativo Linux

En este caso, cuando se introduce un comando, el intérprete realiza los siguientes pasos:

1. El comando es buscado dentro de la lista de comandos almacenados en memoria principal. Es decir, el sistema comprueba que se trata de un comando interno del sistema.
2. Si no lo es, el sistema comprueba si se trata de un alias a otro comando. Si lo fuera, ejecutaría el comando interno asociado al alias.
3. Si tampoco es un alias, el sistema busca en la memoria secundaria. Es decir, busca un programa almacenado en el disco. Para ello, busca en las ubicaciones que tiene configuradas por defecto.
4. Si tampoco encuentra un programa en el disco, considera que, la orden introducida es errónea y se muestra el mensaje de error correspondiente.

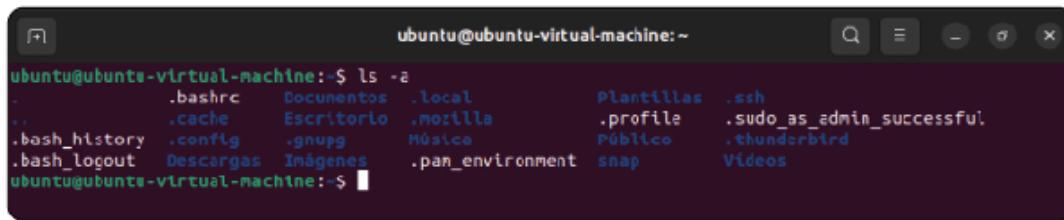
El formato general de una orden en *Linux* se compone por la palabra identificativa del comando, las opciones de mismos y una serie de parámetros que se utilizan para establecer el elemento sobre le que se realiza la acción.

Para ejecutar un comando, hay que tener en cuenta una serie de reglas básicas:

- El texto escrito del comando debe ser exactamente igual al identificador de este comando para que el sistema pueda encontrarlo y ejecutarlo.
- La búsqueda de comandos es sensible a mayúsculas y minúsculas.

- La línea de comandos de *Unix* posee un conjunto de caracteres que identifican los estados de listo y en espera para introducir nuevos comandos. A esta cadena se la conoce como prompt.

En *Linux*, se puede activar un autocompletado de comandos para revisar si existen o no. Para ello, se pulsa tabulador después de escribir algún carácter que corresponda al comando que se busca. Aparecerá un listado con los posibles comandos que empiezan por el texto que se ha escrito.



```
ubuntu@ubuntu-virtual-machine:~$ ls -a
. .bashrc Documentos .local Plantillas .ssh
.. .cache Escritorio .mozilla .profile .sudo_as_admin_successful
.bash_history .config .gnupg Música Pública .thunderbird
.bash_logout Descargas Imágenes .pan_environment snap Videos
ubuntu@ubuntu-virtual-machine:~$
```

Ejemplo de autocompletado

Otro ejemplo sería el *MS-DOS*, sistema operativo de Microsoft de modo texto en el que únicamente se disponía de un intérprete de comandos para la comunicación con el usuario.



```
E:\>command
Microsoft(R) MS-DOS(R) Version 6.22
(C)Copyright Microsoft Corp 1981-1994.

E:\>ver
MS-DOS Version 6.22

E:\>dir command.com
Volume in drive E is MSDOS6SETUP
Directory of E:\

COMMAND.COM      54,645 05-31-94   7:22a
    1 file(s)        54,645 bytes
                      0 bytes free

E:\>
```

Interfaz de usuario del sistema operativo MS-DOS de Microsoft

En MS-DOS, se podía obtener un listado de todos los comandos existentes escribiendo el comando **help**. Y, si se necesitaba conocer las distintas opciones que proporcionaba un comando en particular, tan solo debía introducirse la entrada al comando seguida por el modificador **/?**



Ejemplo

Si se querían saber las opciones que permitía configurar el comando dir para mostrar los archivos y directorios del sistema de ficheros, se usaba: dir /?

```
E:\>command  
  
Microsoft(R) MS-DOS(R) Version 6.22  
      (C)Copyright Microsoft Corp 1981-1994.  
  
E:\>ver  
  
MS-DOS Version 6.22  
  
E:\>dir command.com  
Volume in drive E is MSDOS6SETUP  
Directory of E:\  
  
COMMAND.COM      54,645 05-31-94   7:22a  
    1 file(s)      54,645 bytes  
                  0 bytes free  
  
E:\>
```



Actividades

10. ¿Cuál es el objetivo principal del uso de una interfaz en un sistema operativo?

5. Resumen

En general, los sistemas operativos facilitan el uso de equipos informáticos, móviles, tablets e incluso electrodomésticos. Si no existieran, la era de las comunicaciones sería poco más que un suspiro.

Se ha hablado de los servicios y los aspectos estructurales. Actualmente, se ha pasado de la mejora computacional en cuestiones referentes a la administración de procesos a un intento de que el sistema operativo encaje en un marco donde prevalece el diseño y la simplicidad.

Pese a conocer diferentes estructuras de sistemas operativos, la idea que está por encima de complejidades estructurales es que se trata de un sistema donde los servicios y las tareas se dividen y agrupan en módulos que permiten reducir su complejidad.

Las investigaciones en el uso de la memoria han impulsado el crecimiento de aspectos como la multiprogramación y han permitido que los sistemas operativos se hayan vuelto más seguros y eficientes.

Por último, se ha tratado la interacción entre el usuario y los equipos. Esta interacción no ha cambiado mucho con respecto a los orígenes del sistema operativo. Por el contrario, cada sistema operativo ha intentado hacer uso de estos conceptos para crear una identidad propia y venderse al mundo. Estos son los casos de *Windows* con su interfaz de ventanas, y *Linux*, con su terminal y sus infinitas posibilidades desde ella.

