CONTENIDO

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS.

El ordenador es un sistema programable formado por un conjunto de elementos hardware que necesitan instrucciones que le indiquen cómo utilizar los recursos. El conjunto de instrucciones o programas es lo que conocemos como soporte lógico o software. Un ordenador, sin software que lo programe, es básicamente un bloque de metal inútil, pero con el software puede almacenar, procesar y obtener información, editar textos, controlar el entorno, etc.

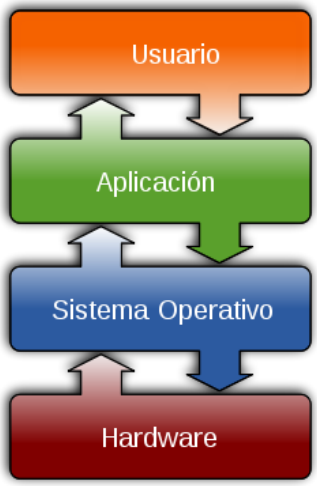
CONCEPTO

Sin duda alguna, la utilización de los recursos mediante programas es muy complicada, puesto que cada dispositivo es diferente y con gran cantidad de características a controlar. Por ello, una de las primeras acciones a llevar a cabo es el diseño y codificación del software que nos facilite el manejo de estos recursos, evitando, en lo posible, que debamos poseer profundos conocimientos del hardware, cediéndole esta tarea a un reducido número de profesionales que serán los que construyan dicho software. Una vez realizado este esfuerzo de diseño, cabe pensar por que no se completa un poco más con el fin de dotar a los usuarios de unas cuantas funciones adicionales, que no sólo faciliten el uso de estos recursos, sino que además los potencien lo más posible. Pues bien, este software así diseñado, cuya finalidad es gestionar adecuadamente los recursos para que realicen el trabajo que se les ha encomendado, y que, además, potencien las funciones de los mismos, es lo que denominaremos sistema operativo, pudiéndolo definir como:

> **Un sistema operativo es un conjunto de programas que, ordenadamente relacionados entre sí, contribuyen a que el ordenador lleve a efecto correctamente el trabajo encomendado.**

OBJETIVOS.

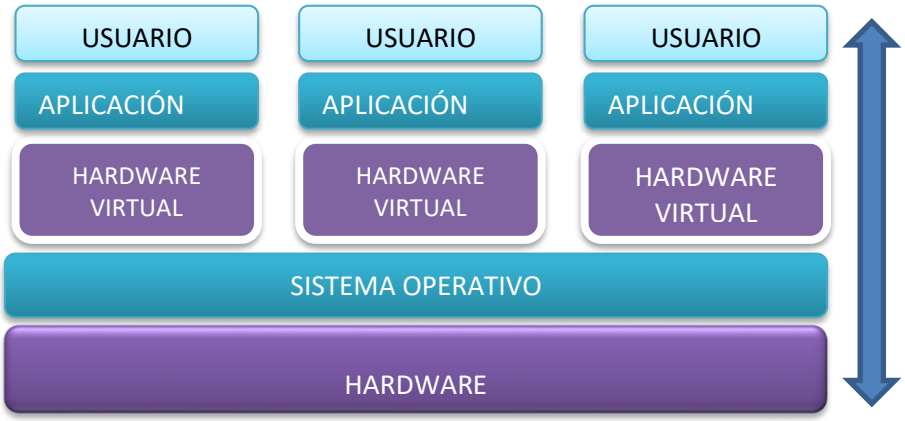
Desde el punto de vista del usuario, el sistema operativo consiste en una **serie de programas** y funciones que **ocultan** los detalles del **hardware**, ofreciéndole una vía sencilla y flexible de acceso al mismo, teniendo dos **objetivos** fundamentales:



- **Seguridad:** El sistema operativo debe actuar contra cualquier manipulación extraña, ya sea accidental o premeditada que pudiera dañar la información, perjudicar a otros usuarios o provocar un funcionamiento indeseado del sistema. Por ejemplo, hay ciertas instrucciones que pueden parar la máquina y otras que realizan operaciones directamente sobre el hardware, que debemos evitar que se utilicen por los programas.

Para ello, algunos sistemas proporcionan dos estados, llamados estado protegido (Sistema o **Kernel**), en el cual se ejecuta el **sistema operativo**, y estado no protegido (Usuario o **User**), que es el destinado a la ejecución de los **programas de usuario** y de aplicación. De esta manera se impide que los programas de los usuarios puedan tener contacto directo con el hardware, o puedan forzar un incorrecto funcionamiento del sistema.

- **Abstracción:** La tendencia actual del software y de los lenguajes de programación es ocultar lo más posible los detalles de más bajo nivel, intentando dar a los niveles superiores una visión más sencilla, global y abstracta, ofreciéndoles operaciones para manipular dichas estructuras ocultas, desconociendo por completo la gestión interna de las mismas. Sobre estas estructuras se construyen otras que abstraen a las anteriores, y así sucesivamente. Gracias a la **abstracción**, los sistemas operativos **enmascaran** los **recursos físicos**, permitiendo su manejo con funciones más generales que ocultan las básicas, constituyendo verdaderos recursos ficticios o virtuales, que mejoran y son más potentes que los físicos.



Desde el punto de vista de un programa o usuario, la máquina física se convierte, gracias al sistema operativo, en una **máquina virtual**, también conocida como máquina extendida, que presenta la ventaja respecto a la física de ofrecer más funciones de las que normalmente soportaría esta última. Desde el punto de vista del usuario, el sistema operativo proporciona servicios que no están presentes en la máquina subyacente. Estos servicios incluyen las facilidades de carga y ejecución de programas, interacción entre el usuario y los programas, permitiendo que se ejecuten varios al mismo tiempo, gestión de la contabilidad para facturar los servicios y almacenamiento de datos y programas.

Como resumen, podemos decir que el **sistema operativo** persigue alcanzar la **mayor eficiencia** posible del **hardware** y **facilitar** el **uso** del mismo a los **usuarios** y a las aplicaciones.

FUNCIONES DE LOS SISTEMAS OPERATIVOS.

Las funciones de los sistemas operativos son diversas y han ido evolucionando de acuerdo con los progresos que la técnica y la informática han experimentado. Como principales funciones, podríamos enumerar las siguientes:

- **Gestión de procesos.** Hay que diferenciar entre los conceptos programa y proceso. Un programa es un ente pasivo, que cuando se carga en memoria y comienza a ejecutarse, origina uno o varios procesos.

- **Gestión de la memoria.** La gestión de memoria suele ir asociada a la gestión de procesos. Para ejecutar un proceso es necesario asignarle unas direcciones de memoria exclusivas para él y cargarlo en ellas, cuando el proceso finalice su ejecución es necesario liberar las direcciones de memoria que estaba usando.

- **Gestión de ficheros.** Un fichero es una **abstracción** para definir una **colección de información no volátil**. Su objetivo es proporcionar un modelo de trabajo sencillo con la información almacenada en los dispositivos de almacenamiento. Estos ficheros deben tener espacio asignado en los dispositivos, deben estar protegidos entre ellos, deben organizarse según unos determinados esquemas… todo esto es la gestión de ficheros.

- **Gestión de los dispositivos de E/S.** La gestión de la entrada salida (E/S) tiene como objetivo proporcionar una interfaz de alto nivel de los dispositivos de E/S sencilla de utilizar.

- **Gestión de la red.** El sistema operativo es el encargado de gestionar los distintos niveles de red, los drivers (controladores) de los dispositivos involucrados en la red, los protocolos de comunicación, las aplicaciones de red, etc.

- **Protección y seguridad.** Mecanismos para permitir o denegar el acceso a los usuarios y a sus procesos a determinados recursos (ficheros, dispositivos de E/S, red, etc.).

Para comprender mejor porqué existen dichas funciones y cuáles son sus objetivos, las iremos estudiando mientras hacemos un breve recorrido a través de la historia de los ordenadores y la informática, ya que nos ayudara a comprender mejor el concepto de sistema operativo.

Los objetivos fundamentales de los sistemas operativos respecto a conseguir la mayor **eficiencia y facilidad de uso** posibles no son siempre compatibles, ya que cualquier sistema que deba ser eficiente, normalmente no será fácil de usar, mientras que, si es fácil de usar, se deberá ofrecer a los usuarios muchas facilidades y ayudas, incluyendo muchos pasos e información que para un usuario experto no serían necesarias, lo que implica, obviamente, una pérdida de eficiencia.

LOS PRIMEROS ORDENADORES.

Los primeros ordenadores tenían un gran tamaño, eran extremadamente caros y muy difíciles de usar. Estas enormes máquinas ocupaban normalmente amplias salas y eran gestionadas por el usuario desde una consola, único medio de acceder a dicho ordenador. Cada usuario tenía asignados períodos de tiempo durante los cuales sólo él podía utilizar el ordenador, siendo el dueño absoluto de la máquina.

Cuando a un usuario le llegaba su tiempo de máquina, tenía que apresurarse a introducir en el ordenador todas las fichas perforadas que conformaban su programa, ejecutar el programa en el ordenador, vigilar su funcionamiento y esperar a que todas las operaciones se terminaran (si había suerte, antes que se le terminara su tiempo de maquina).

Estos ordenadores se basaban en dos factores: sus dispositivos de entrada/ salida y su habilidad para ejecutar un programa, pero no disponían de recursos lógicos adicionales, como pudieran ser medios de almacenamiento secundario por lo que, los usuarios debían introducir sus programas en el ordenador cada vez que se deseaba ejecutar el trabajo correspondiente.

En el caso de que al programador se le acabara el tiempo de máquina concedido sin haber terminado el trabajo, éste debía suspenderlo en el estado en que se encontrara en ese instante, recopilar todo el material obtenido, retirarse a su mesa de trabajo, y estudiarlo hasta que tuviera otra vez la oportunidad de disponer del ordenador. Por el contrario, si el programador acababa antes del final del tiempo asignado, el ordenador quedaba inactivo hasta el siguiente período de tiempo concedido a otro programador.

Tarjetas perforadas.

Podemos deducir que el aprovechamiento de los recursos del ordenador era escasísimo, y por tanto carísimo, además de no ser satisfactorio para los usuarios; de ahí los esfuerzos para mejorar su rendimiento.

Prestando una mínima atención al procedimiento anterior, se pueden pensar varios mecanismos para obtener un mayor aprovechamiento del ordenador, principalmente incidiendo en los tiempos muertos del sistema que podrían utilizarse para llevar a cabo otros trabajos mientras se realizan las correcciones, estudios, etcétera, sobre un programa ejecutado que tuviese errores.

Para resolverlo, los propietarios de los sistemas contrataron a una o varias personas especializadas para ejecutar las rutinas de carga y descarga, con el fin de mantener el sistema con la máxima ocupación posible, recibiendo los trabajos de los usuarios para su ejecución. De esta forma, al recibir dichos trabajos, los reunía y ejecutaba secuencialmente consumiendo únicamente el tiempo que realmente necesitasen y evitando en gran medida los tiempos de inactividad del procesador.

Esta persona se conoce como el operador del ordenador, que es un técnico dedicado únicamente a su manipulación para realizar los trabajos encomendados. A partir de este momento, **el programador o usuario dejó de tener acceso directo al sistema**.

En el caso de que existiesen errores en la ejecución de los trabajos, se hacía un volcado en código binario de la memoria para su entrega al programador, y se ejecutaba inmediatamente el siguiente trabajo, disminuyendo considerablemente los tiempos muertos y, por lo tanto, mejorando el rendimiento.

Además, otra de las misiones del operador, era agrupar los trabajos que tuvieran necesidades de recursos físicos y lógicos similares para que se ejecutasen como un grupo. Supongamos que tenemos varios programas que deben ser traducidos con el compilador Fortran y otros con Cobol; si se reúnen todos los programas Fortran y se compilan como un grupo, sólo habrá que cargar el compilador una vez, obteniendo un considerable ahorro de tiempo.

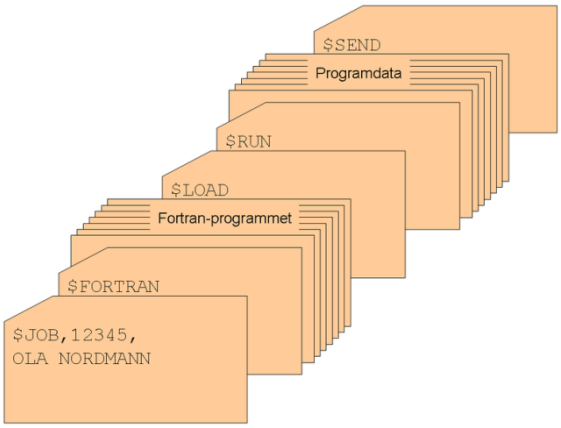
A este modo de trabajo se le conoce como “proceso por lotes” o en inglés “batch”. Es una forma de trabajo en la que los programas no pueden ser interactivos, debido a que el usuario no está presente cuando el trabajo se ejecuta.

SECUENCIA AUTOMÁTICA DE TRABAJOS.

A pesar del ahorro de tiempo inactivo y de la agrupación de trabajos, aún persistían breves períodos de inactividad, ya que, si un trabajo se paraba por algún error, el operador debía observar la consola y tomar nota de todo lo sucedido para comunicárselo al programador. Analizando el trabajo del operador se observó que era bastante mecánico, y que se podía automatizar en gran parte, pensando que podía diseñarse un programa que estuviese permanentemente residente en la memoria del ordenador y que fuese el que realizase muchas de esas operaciones, surgiendo la secuencia automática de trabajos.

Se diseñó un pequeño programa que transfería automáticamente el control de un trabajo a otro. Este programa se denominó **Monitor Residente**, que se puede considerar como el germen de un Sistema Operativo. El monitor **residía permanentemente en memoria**. En el momento de encender el ordenador se daba control al monitor, el cual, a su vez, daba control al primer trabajo, de manera que cuando terminaba su ejecución, el monitor tomaba el control de nuevo activando el siguiente trabajo, y así sucesivamente. Es decir, controlaba la secuencia de los trabajos a realizar.

Para que el monitor supiera qué programa debía ejecutar, y qué datos iba a tratar, se añadieron al paquete de tarjetas que contenía el programa unas tarjetas de control con las directivas necesarias para dicho monitor. Esta; tarjetas debían ceñirse a un lenguaje estricto de comandos denominado Lenguaje de Control de Trabajos (JCL: Job Control Language).



Para diferenciar estas tarjetas del resto del programa, con objeto de identificarlas y poder ser tratadas adecuadamente, la primera columna debía contener algún símbolo especial que variaba de un sistema a otro, pero que normalmente era "$" o "//", tal y como se ve en la figura.

Además, debía poder gestionar la entrada y salida de datos desde el exterior al ordenador y viceversa, con el fin de poder llevar a cabo correcta y rápidamente el trabajo encomendado, sin necesidad de intervención de los usuarios u operadores.

A partir de este momento, los operadores tenían la misión de cargar y descargar las tarjetas en los lectores y perforadores correspondientes, instalar y extraer las cintas magnéticas, mantener las impresoras con suficiente papel, y todas aquellas labores que son necesarias para que el sistema esté operativo físicamente.

Vemos aquí como por primera vez en el ordenador se introducen y procesan **programas** que no tienen una utilidad “directa”, es decir, que no buscan obtener un resultado en concreto pedido por un programador, sino que **facilitan el uso** de la máquina.

TIPOS DE SISTEMAS OPERATIVOS.

En este punto vamos a describir algunas características que permiten clasificar a los sistemas operativos. Básicamente veremos tres clasificaciones: sistemas operativos por su estructura (visión interna), sistemas operativos por los servicios que ofrecen y sistemas operativos por la forma en que ofrecen sus servicios (visión externa).

SISTEMAS OPERATIVOS POR SU ESTRUCTURA (VISIÓN INTERNA).

Si estudiamos los sistemas operativos atendiendo a su estructura interna, veremos que existen dos tipos fundamentales, los sistemas de estructura **monolítica** y los sistemas de estructura **jerárquica**.

En los sistemas operativos de estructura **monolítica** nos encontramos con que el sistema operativo está formado por **un único programa** dividido en rutinas, en donde cualquier parte del sistema operativo tiene los mismos privilegios que cualquier otra. Estos sistemas tienen la ventaja de ser muy rápidos en su ejecución (solo hay que ejecutar un programa) pero cuentan con el inconveniente de carecer de la flexibilidad suficiente para soportar diferentes ambientes de trabajo o tipos de aplicaciones. Es por esto que estos sistemas operativos suelen ser hechos a medida, para solucionar un problema en concreto y no para trabajar de forma generalista.

A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software, del sistema operativo, donde una parte del sistema contenía partes más pequeñas y esto organizado en forma de niveles. Se dividió el

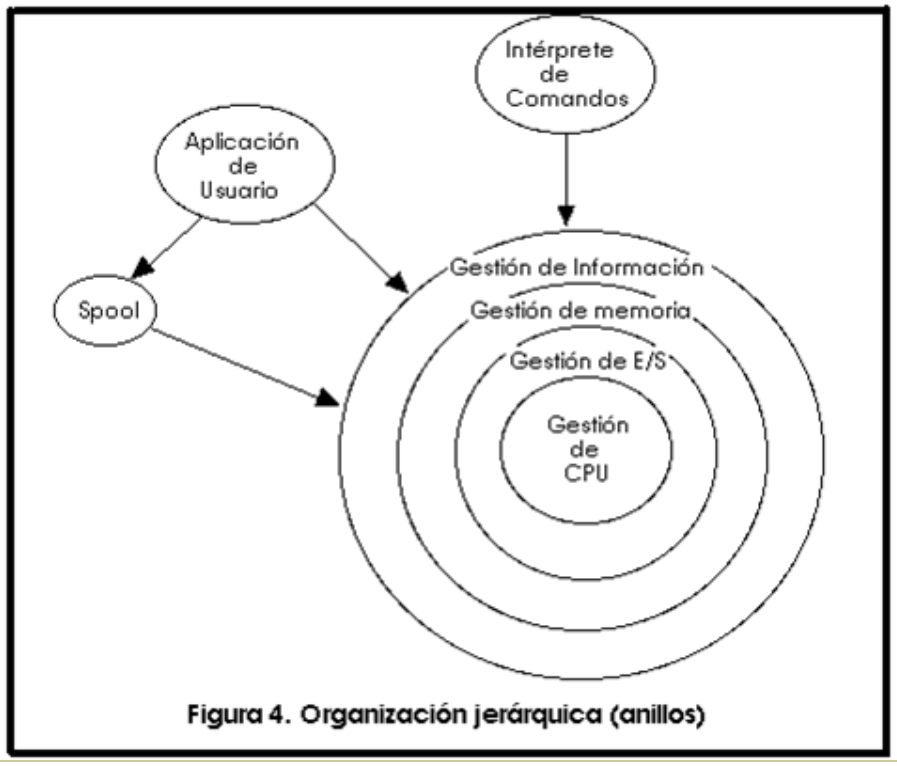
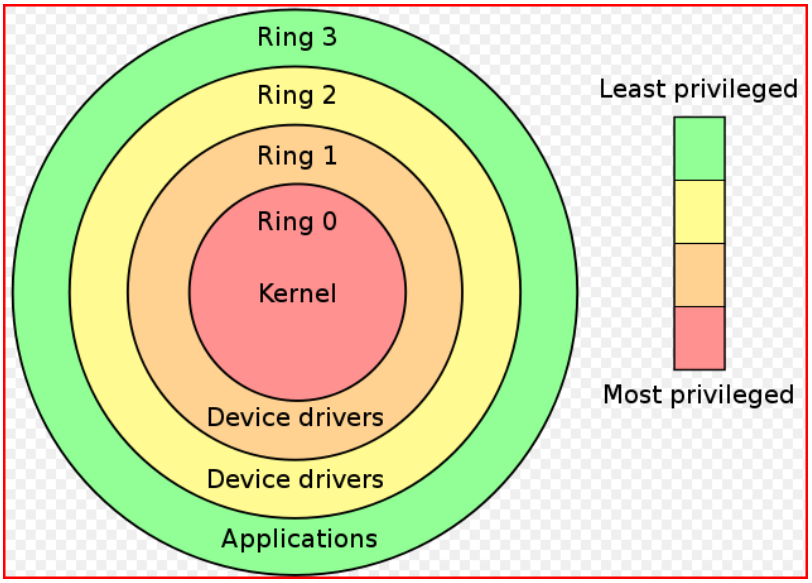


Figura 4. Organización jerárquica (anillos)

sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interface con el resto de elementos.

Se puede pensar también en estos sistemas como si fueran `multicapa'.

En la estructura anterior se basan prácticamente la mayoría de los sistemas operativos actuales. Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos o "rings"



En el sistema de anillos, cada uno tiene una apertura, conocida como puerta o trampa (trap), por donde pueden entrar las llamadas de las capas inferiores. De esta forma, las zonas más internas del sistema operativo o núcleo del sistema estarán más protegidas de accesos indeseados desde las capas más externas. Las capas más internas serán, por tanto, más privilegiadas que las externas.

Cada capa supervisa a la capa que tiene por encima, de modo que para que algo se ejecute en la capa 5, por ejemplo, debe recibir permiso y supervisión de la capa 4, que esta supervisada por la 3, y así sucesivamente. Evidentemente cuanto más al “exterior” de la estructura se ejecute un programa, más lento va a ser su funcionamiento ya que va a recibir un gran número de supervisiones. Por el contrario, cuanto más en el interior se ejecute un proceso, mayor será su velocidad.

En el centro de esta estructura se encuentra el Kernel o Núcleo del sistema operativo, que es su parte más importante.

TIPOS DE SISTEMAS OPERATIVOS SEGÚN SU VISIÓN EXTERNA.

Según el número de usuarios que soporta concurrentemente:

- **Monousuarios.** Los sistemas operativos monousuarios son aquéllos que soportan a un usuario a la vez, sin importar el número de procesadores que tenga la computadora o el número de procesos o tareas que el usuario pueda ejecutar en un mismo instante de tiempo. Las computadoras personales típicamente se han clasificado en esta sección.

- **Multiusuario.** Los sistemas operativos multiusuario son capaces de dar servicio a más de un usuario a la vez, ya sea por medio de varias terminales conectadas a la computadora o por medio de sesiones remotas en una red de comunicaciones. No importa el número de procesadores en la máquina ni el número de procesos que cada usuario puede ejecutar simultáneamente.

Según el número de tareas que puede ejecutar concurrentemente:

- **Monotareas.** Los sistemas monotarea son aquellos que sólo permiten una tarea a la vez por usuario. Puede darse el caso de un sistema multiusuario y monotarea, en el cual se admiten varios usuarios al mismo tiempo, pero cada uno de ellos puede estar haciendo solo una tarea a la vez.

- **Multitareas.** Un sistema operativo multitarea es aquél que le permite al usuario estar realizando varias labores al mismo tiempo. Por ejemplo, puede estar editando el código fuente de un programa durante su depuración mientras compila otro programa, a la vez que está recibiendo correo electrónico en un proceso en background (segundo plano). Es común encontrar en ellos interfaces gráficas orientadas al uso de menús y el ratón, lo cual permite un rápido intercambio entre las tareas para el usuario, mejorando su productividad.

Según el número de procesadores que puede gestionar:

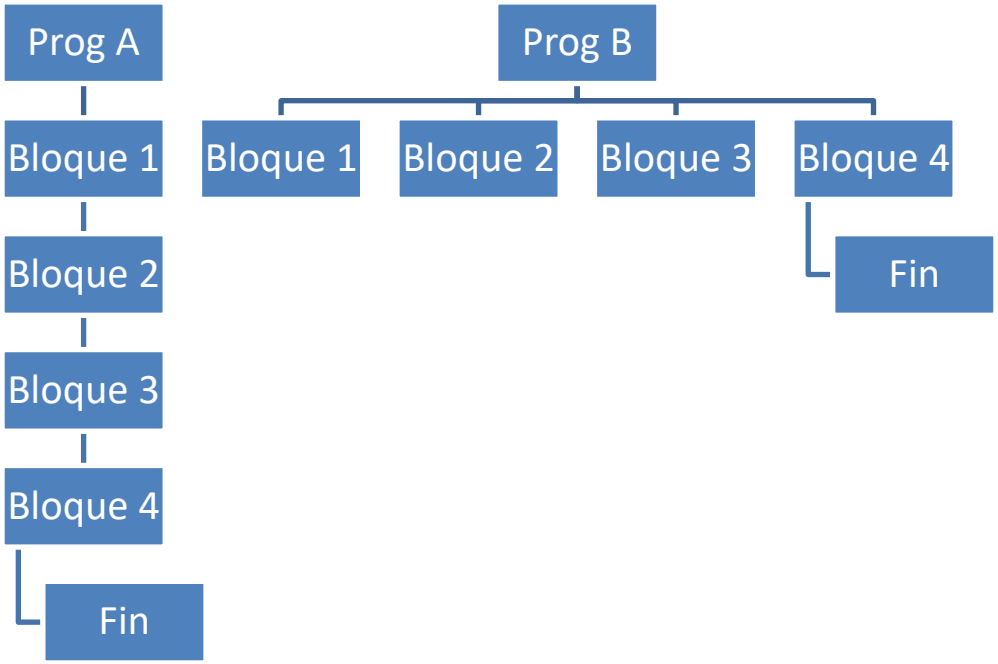
- **Uniproceso.** Un sistema operativo uniproceso es aquél que es capaz de manejar solamente un procesador de la computadora, de manera que si la computadora tuviese más de uno le sería inútil. Por ejemplo, Windows 98 es un sistema operativo Uniproceso.

- **Multiproceso.** Un sistema operativo multiproceso es capaz de manejar más de un procesador en el sistema, distribuyendo la carga de trabajo entre todos los procesadores que existan en el sistema. Generalmente estos sistemas trabajan de dos formas: simétricamente o asimétricamente.

  - Cuando se trabaja de manera **asimétrica**, el sistema operativo selecciona a uno de los procesadores el cual jugará el papel de procesador maestro y servirá como pivote para distribuir la carga a los demás procesadores, que reciben el nombre de esclavos. El único procesador que realmente tiene acceso a todos los recursos del sistema es el maestro, que relega en los esclavos los trabajos que le van llegando. Es un sistema simple de construir y donde es muy fácil añadir más procesadores esclavos.

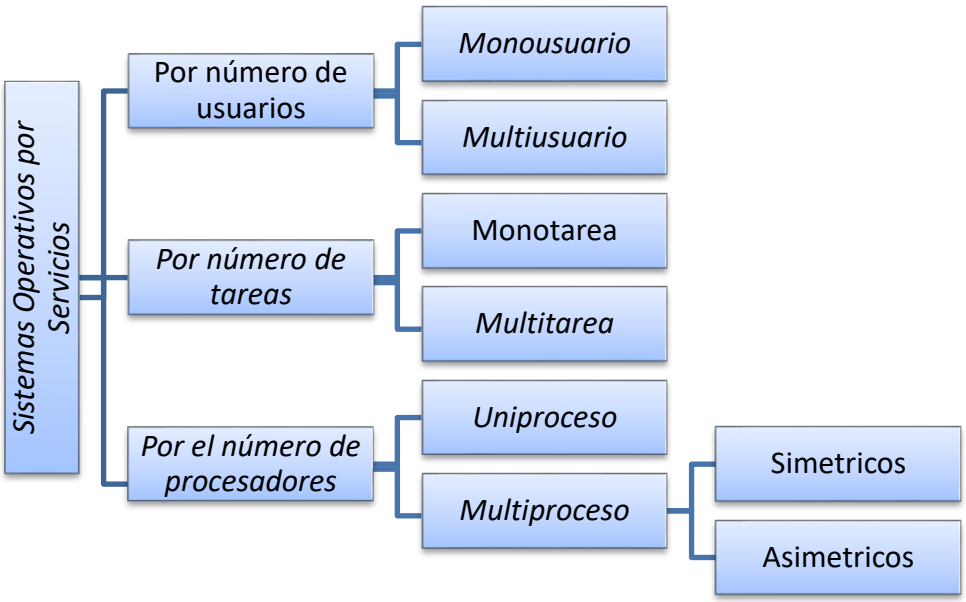
- Cuando se trabaja de manera **simétrica**, los procesos o partes de ellos (threads, hebras o hilos) son enviados indistintamente a cualquiera de los procesadores disponibles, teniendo una mejor distribución y equilibrio en la carga de trabajo bajo este esquema. Se dice que un thread es la parte activa en memoria y corriendo de un proceso, lo cual puede consistir de un área de memoria, un conjunto de registros con valores específicos, la pila y otros valores de contexto. Es un sistema mucho más difícil de construir, y es también tremendamente complicado añadir más procesadores, pero tiene la gran ventaja de ser muchísimo más práctico, ya que cada procesador tiene acceso a todos los recursos y las cargas de trabajo se pueden dividir de forma mucho más rápida. Además, este sistema simétrico permite gestionar los recursos como la memoria de forma compartida entre los distintos procesadores.

Un aspecto importante a considerar en estos sistemas es que la aplicación debe construirse específicamente para aprovechar varios procesadores. Existen aplicaciones que fueron hechas para correr en sistemas uniproceso que no aprovechan el multiproceso, ya que el código debe contener secciones de código paralelizable (que se puedan correr en paralelo), los cuales son ejecutados al mismo tiempo en procesadores diferentes.



Así, Prog A se ejecutará en 4 ciclos de reloj sin importar si se ejecuta en un sistema monoprocesador o no, mientras que Prog B se ejecutará en 4 ciclos de reloj en un sistema monoprocesador, pero en 1 ciclo de reloj en un sistema con 4 procesadores.

Resumen de los tipos de sistemas operativos según su visión interna.



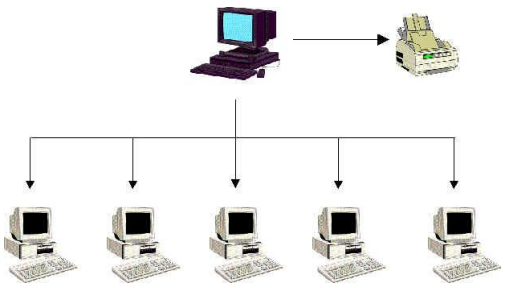
SISTEMAS OPERATIVOS SEGÚN SU FUNCIÓN.

Esta clasificación se refiere a la función que el sistema va a realizar.

SISTEMAS OPERATIVOS DE ESCRITORIO.

Estos sistemas operativos se utilizan en los equipos personales, estaciones de trabajo, portátiles, etc. También se suelen conocer como sistemas operativos clientes. Windows 7 por ejemplo, es un sistema operativo de escritorio. Suelen ser sistemas operativos preparados para permitir un uso fácil por parte del usuario, destacan en multimedia, juegos, sonido, ofimática, etc.

SISTEMAS OPERATIVOS EN RED Y SERVIDORES.



Los sistemas operativos de red se definen como aquellos que tiene la capacidad de interactuar con sistemas operativos en otras computadoras por medio de un medio de transmisión con el objeto de intercambiar información, transferir archivos, ejecutar comandos remotos y un sin fin de otras actividades. Lo importante es hacer ver que el usuario puede acceder a la información no solo de su máquina, sino a la de cualquier máquina de la red, y esto se consigue gracias a que utiliza un sistema operativo de red.

Hoy en día todos los sistemas operativos de escritorio son sistemas operativos de red también, cosa que no ocurría anteriormente. Normalmente solemos llamar sistemas operativos en red a los sistemas operativos que funcionan como servidores en una red, como es el caso del Windows Server o Linux Server.

SISTEMAS OPERATIVOS DISTRIBUIDOS.

Un sistema distribuido se define como una colección de equipos informáticos separados físicamente y conectados entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software de modo que el usuario percibe que existe un solo sistema (no necesita saber qué cosas están en qué máquinas). El usuario accede a los recursos remotos de la misma manera en que accede a recursos locales ya que no percibe que existan varios ordenadores, sino que solo es capaz de ver uno formado por todos los anteriores.

Una ventaja fundamental de los sistemas distribuidos es que permiten aumentar la potencia del sistema informático, de modo que 100 ordenadores trabajando en conjunto, permiten formar un único ordenador que sería 100 veces más potente que un ordenador convencional.

Los sistemas distribuidos son muy confiables, ya que si un componente del sistema se descompone otro componente debe de ser capaz de reemplazarlo, esto se denomina Tolerancia a Fallos.

El tamaño de un sistema distribuido puede ser muy variado, ya sean decenas de hosts (red de área local), centenas de hosts (red de área metropolitana), y miles o millones de hosts (Internet); esto se denomina escalabilidad. De hecho, si un ordenador formando por un sistema distribuido se queda "corto" para las necesidades de la empresa, basta con instalar más

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora y mainframe, mientras se mantiene la flexibilidad de trabajar en múltiples problemas más pequeños. Esta forma de computación se conoce como **grid**.

Los grandes retos de cálculo de hoy en día, como el descubrimiento de medicamentos, simulación de terremotos, inundaciones y otras catástrofes naturales, modelización del clima/tiempo, grandes buscadores de internet, el programa SETI, etc. Son posibles gracias a estos sistemas operativos distribuidos que permiten utilizar la computación distribuida.

El modelo de computación de ciclos redundantes, también conocido como computación zombi, es el empleado por aplicaciones como Seti@Home, consistente en que un servidor o grupo de servidores distribuyen trabajo de procesamiento a un grupo de computadoras voluntarias a ceder capacidad de procesamiento no utilizada. Básicamente, cuando dejamos nuestro ordenador encendido, pero sin utilizarlo, la capacidad de procesamiento se desperdicia por lo general en algún protector de pantalla, este tipo de procesamiento distribuido utiliza nuestra computadora cuando nosotros no la necesitamos, aprovechando al máximo la capacidad de procesamiento. La consola PS3 también cuenta con una iniciativa de este tipo.

Otro método similar para crear sistemas de supercomputadoras es el **clustering**. Un cluster o racimo de computadoras consiste en un grupo de computadoras de relativo bajo costo conectadas entre sí mediante un sistema de red de alta velocidad (gigabit de fibra óptica por lo general) y un software que realiza la distribución de la carga de trabajo entre los equipos. Por lo general, este tipo de sistemas cuentan con un centro de almacenamiento de datos único. Los clusters tienen la ventaja de ser sistemas redundantes, si falla un equipo se resiente un poco la potencia del cluster, pero los demás equipos hacen que no se note el fallo.

En un **cluster** normalmente todos los equipos están ubicados en una **misma red** de área **local**, mientras que en un **grid** los equipos suelen estar distribuidos por **todo** el **mundo**.

Algunos sistemas operativos que permiten realizar clustering o grid, son; Amoeba, BProc, DragonFly BSD, Génesis, Kerrighed, Mosix/OpenMosix, Nomad, OpenSSI, Plurid.

Un cluster que usamos habitualmente, es el que forma Google. Se estima que en 2010 usaba unos 450.000 ordenadores, distribuidos en varias sedes por todo el mundo y formando clusters en cada una de dichas sedes.

Cada cluster de Google está formado por miles de ordenadores y en los momentos en que se detecta que el sistema está llegando al límite de su capacidad, se instalan cientos de ordenadores más en pocos minutos, aumentado así la potencia de cada cluster. Estos equipos normalmente con ordenadores x86 como los que solemos usar nosotros, corriendo versiones especiales de Linux, modificadas por la propia Google para que permitan la formación de estos clusters. (Buscar en google “google centro de datos” para ver algunas informaciones sobre ellos).

https://www.slideshare.net/abhijeetdesai/google-cluster-architecture

https://www.google.com/about/datacenters/

SISTEMAS OPERATIVOS POR SU DISPONIBILIDAD.

Dividimos aquí los sistemas operativos por la forma en que se ponen disponibles a los usuarios.

SISTEMAS OPERATIVOS PROPIETARIOS.

Se les denomina propietarios porque son sistemas propiedad de la empresa que los desarrolla. La empresa no vende en realidad el sistema operativo, sino una licencia de uso del mismo. No se tiene acceso al código fuente del sistema, o por lo menos, no se tiene permiso para modificarlo libremente.

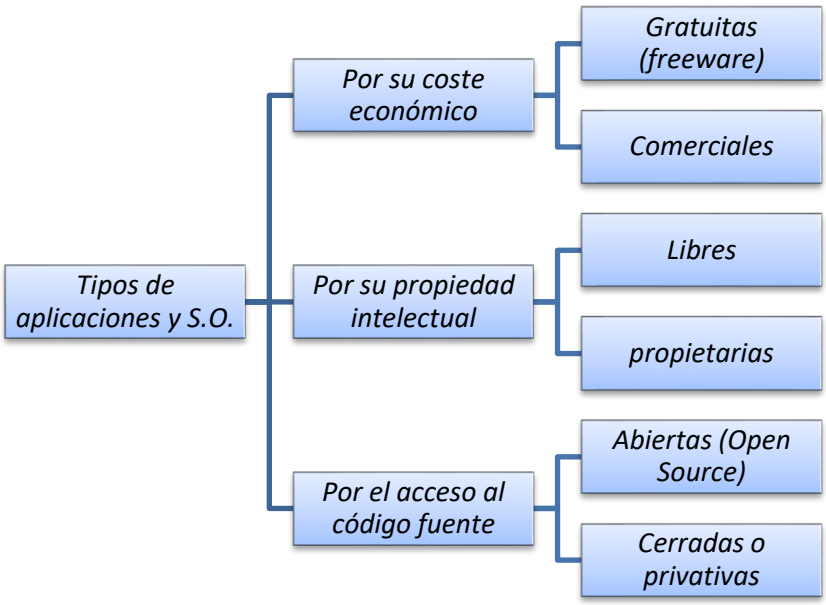
También está prohibido distribuir estos sistemas, o usarlos de formas no autorizadas por la empresa desarrolladora. Toda la familia Windows es un claro ejemplo de sistema operativo propietario.

SISTEMAS OPERATIVOS LIBRES.

Son sistemas operativos en los que se ha renunciado a cualquier tipo de propiedad intelectual. Son sistemas que pueden usarse libremente, ser distribuidos, permiten que se acceda a su código fuente y permiten que esté sea modificado de la forma que queramos.

No hay que confundir el hecho de que sean libres con el hecho de que sean gratuitos. Posteriormente trataremos en profundidad el tema de las licencias de software.

En general, tanto los sistemas operativos como las aplicaciones normales, pueden definirse según su disponibilidad en alguno de estos apartados:



SISTEMAS OPERATIVOS POR SU TIPO DE LICENCIA.

Dentro de los sistemas operativos comerciales, propietarios y privativos, nos podemos encontrar con diversos tipos de licencia de uso:

O.E.M.

OEM (abreviatura del inglés original equipment manufacturer, en español sería fabricante de equipamiento original). Este tipo de licencias se las otorga el desarrollador del sistema operativo al fabricante de hardware, de modo que cuando nosotros compramos uno de sus productos, este viene con una licencia de uso del sistema operativo de tipo OEM. La particularidad de este tipo de licencias es el que el sistema operativo viene preparado para ese hardware específicamente, de modo que no tenemos realmente una licencia de uso del sistema operativo, sino una licencia de uso del sistema operativo únicamente para ese hardware en concreto.

Estas licencias son las más económicas, y suelen poseer restricciones especiales, aparte de venir sin manuales ni caja.

RETAIL.

Es la licencia que compramos directamente del desarrollador. Somos propietarios de la licencia, podemos instarlo en cualquier tipo de hardware compatible, podemos revender la licencia o cederla, etc.

Normalmente solo permiten su uso en una sola maquina a la vez. Vienen con su caja y manuales.

En las licencias de tipo retail, normalmente podemos elegir entre una licencia completa, o una licencia de actualización, que permite actualizar un sistema anterior al nuevo, por un coste algo más reducido.

VLM (LICENCIAS POR VOLUMEN).

Para una empresa con cientos de ordenadores, es complicado controlar las licencias individuales de cada una de sus máquinas. Existe la posibilidad de contratar un tipo de licencia especial con el desarrollador, de modo que, con una única clave de licencia, podemos utilizar varias máquinas a la vez. Es habitual que existan licencias de 25 usos concurrentes, 50, etc.

Son las licencias más caras evidentemente, aunque son bastante más económicas que comprar cada una de las licencias individualmente.

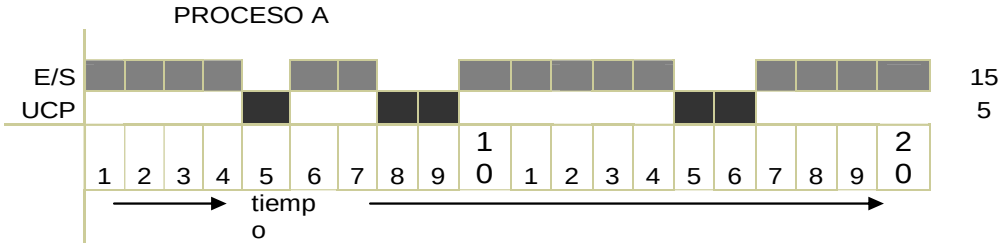
MSDN (LICENCIAS DE EDUCACIÓN.)

Son unas licencias especiales de Microsoft que permiten su uso únicamente para actividades educativas y de formación. Cualquier uso de estas licencias en equipos que desarrollen actividades fuera de este ámbito, es ilegal. Existen también licencias de este tipo para empresas de desarrollo, academias, etc.

GESTION DE PROCESOS.

Si ejecutamos un solo programa en un ordenador, difícilmente podremos alcanzar un rendimiento del 100% ya que siempre tendrá que realizar operaciones de entrada/salida. Es decir, habrá tiempos muertos del procesador durante los que no realizará ningún trabajo, y no todo el tiempo estará realizando cálculos del programa. Esto es así porque las unidades de E/S (impresoras, monitores, teclados, etc.) son millones de veces más lentas que la CPU del ordenador.

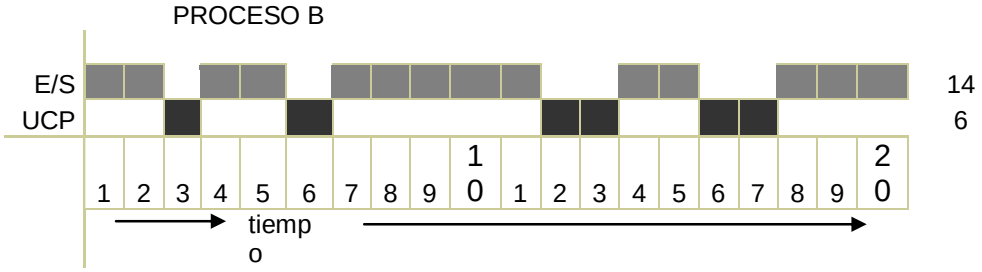
Para comprenderlo mejor, podemos tomar como ejemplo la ejecución del programa representado en la figura siguiente, donde se detalla el diagrama de tiempos de ocupación de los recursos, incluido el propio procesador, necesarios para que puedan realizar el trabajo para



el que fue diseñado.

En este primer proceso (A), vemos que se necesitan 20 unidades de tiempo para ejecutar en su totalidad el proceso, de las cuales 15 se van a usar para emplear los dispositivos de E/S (impresoras, discos, etc.) y 5 van a usarse para cálculos y procesos con la CPU. Establezcamos ahora un segundo proceso (B).

Vemos que este proceso (B) necesita también 20 unidades de tiempo para ejecutarse, de las cuales 14 van a emplearse para trabajar con las E/S, y 6 van a utilizarse para trabajar con la CPU.

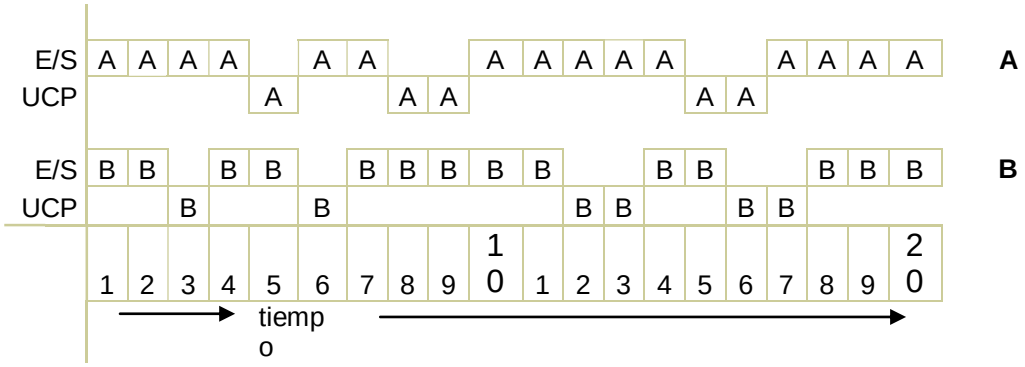


Si tenemos que ejecutar en nuestra máquina, el proceso A y luego el proceso B, el tiempo total de la ejecución será de 40 unidades de tiempo obviamente. Sin embargo… ¿no sería posible optimizar algo este tiempo?

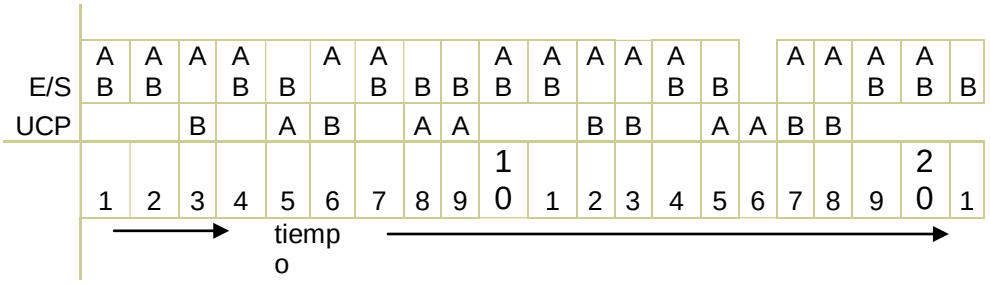
Si estudiamos el proceso A, veremos que durante las 4 primeras unidades de tiempo, la CPU está esperando, sin hacer absolutamente nada. Y podemos comprobar cómo el proceso B necesita usar la UCP en la unidad de tiempo 3.

Normalmente, las operaciones de E/S pueden ser concurrentes, es decir, desarrollarse al mismo tiempo, ya que pueden tratarse de operaciones con distintos dispositivos, o aprovechar las características de caché de los mismos.

Entonces, podemos ejecutar los dos procesos concurrentemente, alternando entre uno y otro, según la UCP vaya quedando libre, y solapando las operaciones de entrada salida. Vamos a ver como quedarían los anteriores trabajos si los ejecutamos concurrentemente, aprovechando los tiempos muertos. Recordemos las necesidades de ambos procesos.



Vemos aquí las necesidades de los procesos A y B, y podemos comprobar cómo muchas veces, B necesita la CPU y puede usarla porque A no la está usando, hagamos esta operación:



Aquí tenemos la ejecución de ambos procesos, realizando de forma **concurrente** las operaciones de entrada salida, y aprovechando los tiempos muertos de la CPU. Fijaros como el tiempo total para ejecutar los dos procesos es de **21** unidades de tiempo, solo 1 más que lo que lleva ejecutar un solo proceso, y muchísimo menos de las **40** u. que llevaría ejecutar ambos procesos de forma independiente.

Además, podemos comprobar cómo gracias a estas técnicas, aprovechamos al máximo los componentes de nuestro sistema, ya que las unidades de E/S están usándose en 20 de las 21 unidades de tiempo, y la CPU está usándose al menos 11 de las 21 unidades de tiempo.

Esta técnica se conoce como **multiprogramación** y tiene como finalidad conseguir un mejor aprovechamiento de los recursos del ordenador, ejecutando simultáneamente varios programas ofreciendo una falsa apariencia de ejecución paralela de los mismos. (Como hemos visto, la ejecución en la CPU es de un proceso al mismo tiempo, no pudiendo ejecutar los 2 procesos a la vez).

Si contamos con un microprocesador que tenga varias CPU, ahí sí sería posible una ejecución paralela real (de ahí la existencia de microprocesadores con varios núcleos, o lo que es lo mismo, varias CPU encapsuladas juntas).

La utilización de la multiprogramación ha dado lugar a diferenciar los trabajos de acuerdo con sus características y necesidades de recursos, pudiendo clasificarlos en dos tipos.

- En primer lugar los **trabajos limitados por proceso**, es decir, aquellos que consumen la mayor parte de su tiempo en el tratamiento de la información y poco en entrada / salida.
- En segundo lugar, los **trabajos limitados por entrada/salida** que basan su acción en operaciones de entrada/salida haciendo poco uso del procesador, el cual permanecerá la mayor parte del tiempo inactivo, considerándose como ideales desde el punto de vista de la multiprogramación.

Como hemos podido ver, la multiprogramación es una técnica altamente recomendable, pero al llevarla a cabo, nos vamos a encontrar con una serie de problemas que habrá que resolver. Vamos a ver algunos de estos problemas.

Propongamos otro ejemplo, esta vez de 3 procesos:

PROC. A	CPU	E/S	E/S	E/S	E/S	CPU	E/S	E/S	E/S	CPU	CPU	CPU	E/S	E/S	E/S
PROC. B	CPU	E/S	CPU	E/S	E/S	E/S	CPU	E/S	CPU	CPU	CPU	E/S	CPU	CPU	E/S
PROC. C	E/S	E/S	E/S	E/S	CPU	E/S	E/S	E/S	CPU	CPU	E/S	E/S	E/S	CPU	E/S
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ¿Cuántas unidades de tiempo tardarían los procesos A y B en ejecutarse mediante multiprogramación en un microprocesador de 1 solo núcleo?
- ¿Cuántas unidades de tiempo tardarían los procesos B y C?
- ¿Cuántas unidades de tiempo tardarían los tres procesos A, B y C?

Un microprocesador que cuente con varios núcleos, podría representarse como que cuenta con varias CPUs. Así, un microprocesador con 2 núcleos permitiría que dos procesos accediesen a la CPU al mismo tiempo.

- ¿Cuántas unidades de tiempo tardarían los procesos A y B en ejecutarse mediante multiprogramación en un microprocesador de 2 núcleos teóricos?
- ¿Cuántas unidades de tiempo tardarían los procesos B y C en ese mismo caso?
- ¿Cuántas unidades de tiempo tardarían los tres procesos, A, B y C en ese mismo caso?

ADMINISTRACIÓN DE PROCESOS.

Uno de las funciones más importantes de un sistema operativo es la de administrar los procesos y tareas del sistema. Vamos a ver en este punto, dos temas relacionados con la administración de procesos; la planificación del procesador y los problemas de concurrencia.

PLANIFICACIÓN DEL PROCESADOR.

Si recordamos cuando vimos la multiprogramación, vimos que en muchas ocasiones un proceso debía dejar paso a otro o esperarse cuando había conflictos en el uso de la CPU. Pues el planificador es el encargado de decidir qué proceso entra en cada momento.

La planificación del procesador se refiere a las técnicas que se usan para decidir cuánto tiempo de ejecución y cuando se le asignan a cada proceso del sistema en un sistema multiprogramado (multitarea). Obviamente, si el sistema es monoprogramado (monotarea) no hay mucho que decidir, pero en el resto de los sistemas esto es crucial para el buen funcionamiento del sistema.

Supóngase un ordenador que contiene un único microprocesador. Dicho microprocesador solamente puede ejecutar un programa en cada instante de tiempo. Además, cuando un programa está ejecutándose, nunca dejará de hacerlo por sí mismo. De manera que, en principio, cualquier programa monopoliza el microprocesador impidiendo que otros programas se ejecuten.

Por ello, la primera misión de un planificador es expulsar el programa en ejecución cuando decida que es pertinente. Esto se consigue de dos maneras, siempre con ayuda del propio hardware:

- Cuando expira un temporizador, que se activa a intervalos regulares de tiempo. En intervalos muy cortos, generalmente cada 250 milisegundos.
- Cuando el programa solicita una operación de entrada/salida. Dado que el programa no puede continuar hasta que termine dicha operación, es un buen momento para ejecutar otro programa.

En ambos casos, el control del microprocesador pasa a manos del planificador gracias a que el hardware genera una interrupción. En este proceso de expulsión, se guarda el estado de ejecución del programa (programa y su estado se denomina proceso).

A continuación, el planificador decide cuál será el siguiente proceso en ejecutarse. Naturalmente, solamente se escogen procesos que estén listos para hacerlo. Si un proceso sigue esperando por una operación de entrada/salida no será candidato a ejecutarse hasta que finalice tal operación.

La selección del proceso sigue alguna política de planificación preestablecida. Una vez seleccionado un proceso, se procede a ejecutarlo. Para ello, el planificador restaura su estado de ejecución (previamente salvado) y abandona el uso del microprocesador cediéndoselo a dicho proceso.

Todo este proceso se realiza en millonésimas de segundo.

Gracias a que el tiempo del microprocesador se reparte entre todos los procesos a intervalos muy cortos, el ordenador ofrece la sensación de que todos los procesos están ejecutándose a la vez.

Cuando un ordenador tiene varios microprocesadores este esquema se repite para cada microprocesador.

PROBLEMAS DE CONCURRENCIA

En los sistemas de tiempo compartido (aquellos con varios usuarios, procesos, tareas, trabajos que reparten el uso de CPU entre estos) se presentan muchos problemas debido a que los procesos compiten por los recursos del sistema.

Imaginemos que un proceso está escribiendo en el disco duro y se le termina su turno de ejecución e inmediatamente después el proceso elegido para ejecutarse comienza a escribir sobre el disco duro. El resultado es un disco duro cuyo contenido es un desastre de datos mezclados. Así como el disco duro, existen una multitud de recursos cuyo acceso debe ser controlado para evitar los problemas de la concurrencia.

El sistema operativo debe ofrecer mecanismos para sincronizar la ejecución de procesos: semáforos, envío de mensajes, 'pipes', etc.

Los semáforos son rutinas de software (que en su nivel más interno se auxilian del hardware) para lograr exclusión mutua en el uso de recursos. Para entender este y otros mecanismos es importante entender los problemas generales de concurrencia, los cuales se describen enseguida.

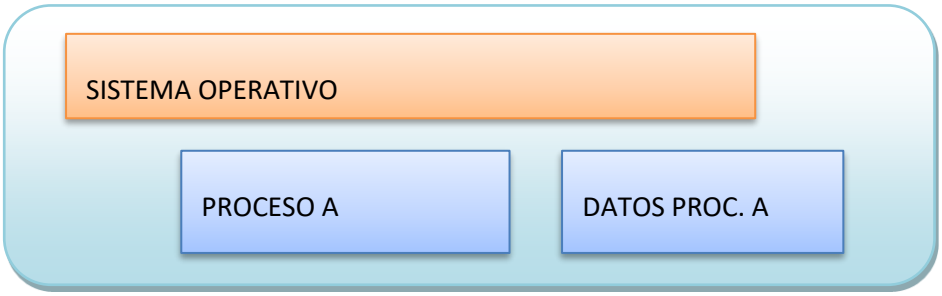
- **Condiciones de Carrera o Competencia**: La condición de carrera (race condition) ocurre cuando dos o más procesos acceden a un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada. Supongamos, por ejemplo, que dos personas realizan una operación en un banco en la misma cuenta corriente, uno A por el cajero, y uno B mediante Internet desde su casa. El usuario A quiere hacer un depósito. El B un retiro. El usuario A comienza la transacción y lee su saldo que es 1000. En ese momento pierde su turno de ejecución por parte del ordenador del banco (y su saldo queda como 1000) y el usuario B inicia el retiro: lee el saldo que es 1000, retira 200 y almacena el nuevo saldo que es 800 y termina. El turno de ejecución regresa al usuario A el cual hace su depósito de 100, quedando saldo = saldo + 100 = 1000 + 100 = 1100. Como se ve, el retiro se perdió y eso será magnifico para los usuarios A y B, pero al banquero no le haría demasiada gracia. El error pudo ser al revés, quedando el saldo final en 800.

- **Postergación o Aplazamiento Indefinido(a)**: Consiste en el hecho de que uno o varios procesos nunca reciban el suficiente tiempo de ejecución para terminar su tarea. Por ejemplo, que un proceso ocupe un recurso y lo marque como 'ocupado' y que termine sin marcarlo como 'desocupado'. Si algún otro proceso pide ese recurso, lo verá 'ocupado' y esperará indefinidamente a que se 'desocupe'.

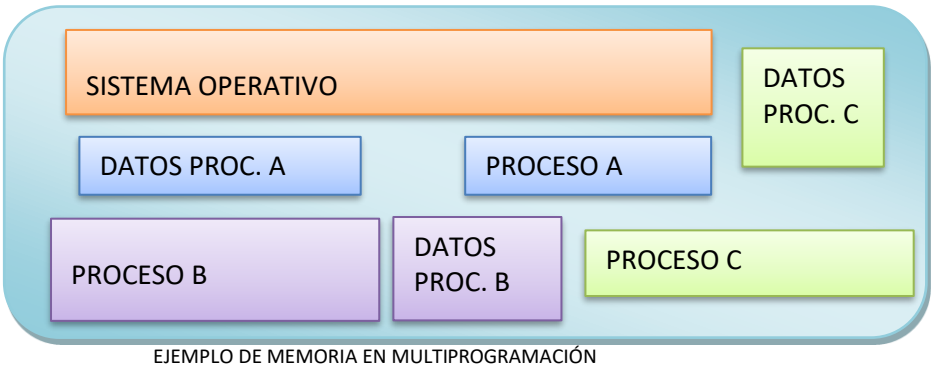
- **Condición de Espera Circular**: Esto ocurre cuando dos o más procesos forman una cadena de espera que los involucra a todos. Por ejemplo, suponga que el proceso A tiene asignado el recurso 'pantalla' y el proceso B tiene asignado el recurso 'disco'. En ese momento al proceso A se le ocurre pedir el recurso 'disco' y al proceso B el recurso 'pantalla'. Ahí se forma una espera circular infinita entre esos dos procesos.

GESTIÓN DE MEMORIA.

En un sistema monoprogramado (lo contrario que multiprogramado), en la memoria del ordenador solo hay un único programa, acompañado de sus datos y del sistema operativo. Esto hace que el uso de la memoria, y la asignación de la misma al programa sea muy simple. Sin embargo, en un sistema multiprogramado nos vamos a encontrar en memoria con varios programas a la vez (2 en el mejor de los casos, pero podemos realizar multiprogramación con 20, 100 o 700 procesos).



EJEMPLO DE MEMORIA EN MONOPROGRAMACIÓN



EJEMPLO DE MEMORIA EN MULTIPROGRAMACIÓN

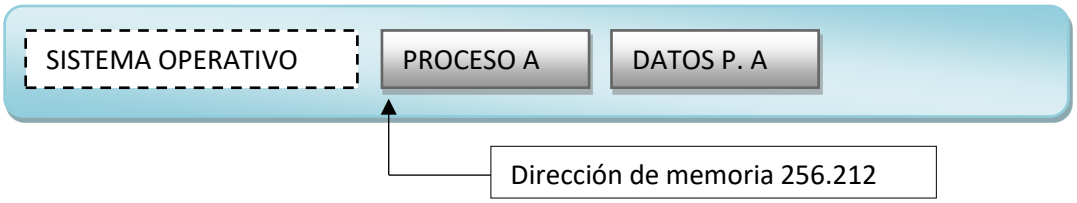
Este "lio" que vemos en la memoria usando multiprogramación hace que se presenten dos problemas fundamentales, la **relocalización y la protección**.

PROBLEMAS CON LA MEMORIA. RELOCALIZACIÓN.

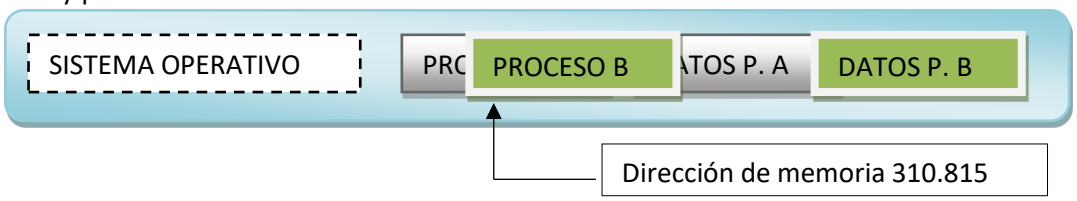
Este problema consiste en que los programas que necesitan cargarse a memoria real ya están compilados y montados, de manera que internamente contienen una serie de referencias a direcciones de instrucciones, rutinas y procedimientos que ya no son válidas en el espacio de direcciones de memoria real de la sección en la que se carga el programa. Esto es, cuando se compiló el programa se definieron o resolvieron las direcciones de memoria de acuerdo a la

sección de ese momento, pero si el programa se carga en otro día en una sección diferente, las direcciones reales ya no coinciden.

Si en memoria solo va a estar este programa, no hay problemas en cargarlo siempre en la misma dirección o sección de memoria, pero si cargamos varios programas en la memoria, esto ya no es posible, dado que varios programas podrían pedir la misma sección.



En este ejemplo teórico que vemos arriba, el proceso A está preparado para cargarse en la dirección de memoria 256.212. Mientras que solo dicho proceso esté funcionando en memoria no hay problema en esto.

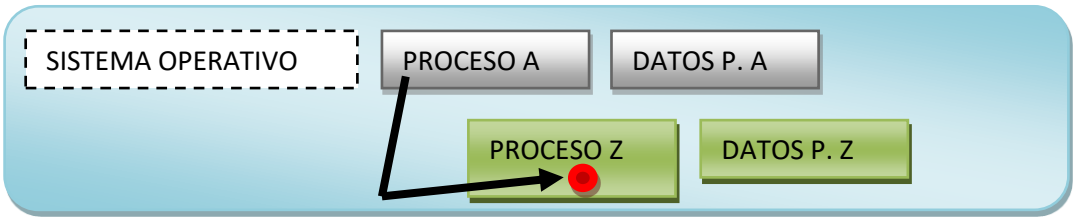


En este ejemplo teórico de arriba, vemos como al mismo tiempo que el proceso A, intentamos ejecutar un proceso B que quiere cargarse en la dirección de memoria 310.815… vemos como "machaca" al proceso A, y también sus datos. Esto conllevaría que ambos procesos dejarían de funcionar, ya que irían sobrescribiéndose el uno al otro.

El gestor o controlador de memoria del sistema operativo, puede solucionar este problema de varias formas, una de las más usadas consiste en tener un **registro** que guarde la dirección **base** de la sección que va a contener al programa. Cada vez que el programa haga una referencia a una dirección de memoria, se le suma el registro base para encontrar la dirección real. Así, en el ejemplo anterior tendríamos que cuando el proceso A quiere acceder a la memoria para instalarse en la dirección 256.212, el SO (Sistema Operativo) hace la operación (dirección + base) donde base por ejemplo es 20.000 por lo que el proceso A sería cargado en la dirección 276.212. Cuando el proceso B se quiera cargar, el SO ajusta la base por ejemplo a 500.000 (para saltarse al proceso A), de forma que en lugar de instalarse en la dirección 310.815 se instalará en la dirección 810.815.

PROBLEMAS CON LA MEMORIA. PROTECCIÓN.

Este problema se refiere a que, una vez que un programa ha sido cargado a memoria en algún segmento en particular, nada le impide al programador que intente direccionar (por error o deliberadamente) localidades de memoria menores que el límite inferior de su programa o superiores a la dirección mayor; es decir, quiere referenciar localidades fuera de su espacio de direcciones.



Obviamente, este es un problema de protección, ya que no es legal leer o escribir en áreas de memoria que pertenezcan a otros programas, y hay que proteger estas zonas de memoria.

La solución a este problema puede ser el uso de un registro base y un registro límite. El registro base contiene la dirección del comienzo de la sección que contiene el programa, mientras que el límite contiene la dirección donde termina. Cada vez que el programa hace una referencia a memoria se comprueba si cae en el rango de los registros y si no es así se envía un mensaje de error y se aborta el programa.

Muchos programas antiguos dan errores de acceso a memoria cuando son ejecutados en sistemas multiprogramados. Precisamente estos errores vienen dados por que intentan acceder a direcciones de memoria que quedan fuera de su ámbito.

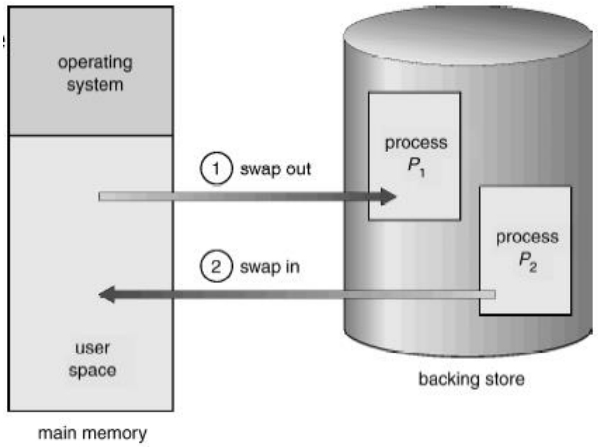
En estos casos, el sistema operativo impide que dichos programas accedan a esas direcciones, lo que provoca un error en el acceso a memoria

MULTIPROGRAMACIÓN EN MEMORIA VIRTUAL.

Las CPU de los ordenadores eran cada vez más poderosas, lo que permitía ejecutar programas cada vez más potentes, y por lo tanto más grandes.

Al mismo tiempo, el uso de la multiprogramación, hacía que el sistema operativo tuviera que colocar decenas de estos grandes procesos en la memoria RAM.

El problema es que la cantidad de memoria RAM "física" que podemos instalar en un ordenador es finita, es decir, tiene un límite. Así, si intentamos ejecutar en multiprogramación 20 procesos, y cada uno de ellos necesita 512 MB de RAM para trabajar, tendríamos que tener instalados en nuestro ordenador 10 GB de RAM.



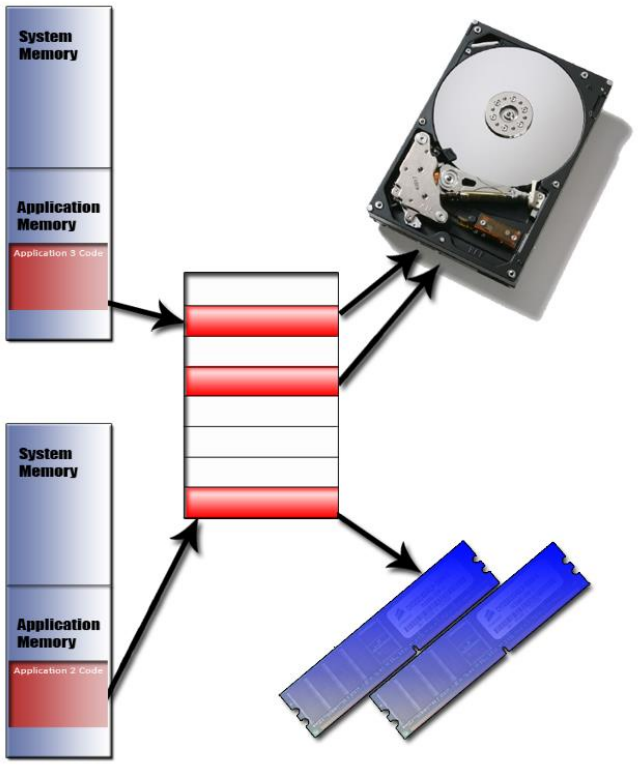
Todo esto empujó a los diseñadores de los sistemas operativos a implantar un mecanismo que permitiera ofrecer a los procesos más cantidad de memoria de la que realmente estaba instalada en la máquina, esto es, de ofrecer `**memoria virtual**'.

La memoria virtual se llama así porque el programa ve una cantidad de memoria mucho mayor que la real, y que en realidad se trata de la suma de la memoria de almacenamiento primario y una cantidad determinada de almacenamiento secundario (por ejemplo el del disco duro).

El sistema operativo, en su módulo de gestión de memoria, se encarga de intercambiar programas enteros, segmentos o páginas entre la memoria real y el medio de almacenamiento secundario. Si lo que se intercambia son **procesos enteros**, se habla entonces de **multiprogramación en memoria real**, pero si lo que se intercambian son segmentos o **páginas**, se puede hablar de **multiprogramación con memoria virtual**.

Existe una técnica en la cual, el sistema operativo divide a los procesos en pequeñas porciones, de tamaño fijo denominadas páginas, de un tamaño múltiplo de 1 K. Estas páginas van a ser pasadas de la RAM al disco y viceversa. Al proceso de intercambiar páginas, segmentos o programas completos entre RAM y disco se le conoce como `intercambio' o `swapping'.

En la paginación, se debe cuidar el tamaño de las páginas, ya que si éstas son muy pequeñas el control por parte del sistema operativo para saber cuáles están en RAM y cuales en disco, sus direcciones reales, etc. crece y provoca mucha `sobrecarga' (overhead). Por otro lado, si las páginas son muy grandes, el overhead disminuye pero entonces puede ocurrir que se desperdicie memoria en procesos pequeños. Debe haber un equilibrio entre ambos conceptos.



Otro aspecto importante es la estrategia para cargar páginas (o segmentos) a la memoria RAM. Se usan más comúnmente dos estrategias: **cargado de páginas** por **demanda** y **cargado de páginas anticipada**. La estrategia de cargado por demanda consiste en que las páginas solamente son llevadas a RAM si son solicitadas, es decir, si se hizo referencia a una dirección que cae dentro de ellas.

La carga anticipada consiste en tratar de adivinar qué páginas serán solicitadas en el futuro inmediato y cargarlas de antemano, para que cuando se pidan ya no ocurran fallos de página. Ese `adivinar' puede ser que se aproveche del fenómeno de localidad y que las páginas que se cargan por anticipado sean aquellas que contienen direcciones contiguas a la dirección que se acaba de referenciar. En el caso de Windows, se usa una conjunción de ambos métodos, y se utiliza un fichero en disco duro donde se va almacenando toda la información sobre índices de páginas en HD, en RAM, etc. Este fichero se denomina **pagefile.sys** y lo podéis encontrar normalmente en la raíz de vuestro volumen de sistema.

Un problema que tenemos con la memoria virtual, es la diferencia de velocidad enorme que existe entre la RAM y la memoria de almacenamiento secundario. Si cargamos muchos procesos, y agotamos nuestra memoria RAM real, el SO permitirá que todo siga funcionando usando el HD, pero tardará muchísimo en pasar las páginas de RAM a HD y viceversa. En muchas ocasiones, nos parecerá incluso que el SO se ha quedado “colgado”. Esto es muy habitual en sistemas con poca memoria, donde vemos que de repente la luz indicadora de actividad en los discos duros se queda encendida, y el SO deja de responder durante un buen rato.

Un intento de solucionar esto es usar memorias de estado sólido en lugar del HD para paginar ya que son memorias mucho más rápidas, esto fue implementado desde Windows Vista.

GESTIÓN DE DATOS. SISTEMAS DE FICHEROS.

Un fichero es un mecanismo de abstracción que sirve como unidad lógica de almacenamiento de información. El fichero agrupa una colección de informaciones relacionadas entre sí y definidas por su creador. A todo fichero le corresponde un nombre único que lo identifique entre los demás ficheros.

Es necesario que el sistema operativo cuente con un sistema que se encargue de administrar la información almacenada en los dispositivos en forma de ficheros: de esto se encargan los sistemas de ficheros.

Un sistema de ficheros es el aspecto más visible de todo sistema operativo y existe por razones tecnológicas, ya que no hay memoria principal lo suficientemente grande como para no necesitar de almacenamiento secundario. Surge debido a la necesidad del sistema operativo de poder gestionar la información de forma eficiente y estructurada, además de establecer unos parámetros de seguridad y protección en entornos críticos.

El sistema operativo ofrece una visión lógica y uniforme del almacenamiento de información realizando una abstracción de las propiedades físicas de sus dispositivos de almacenamiento. Para ello, define el concepto lógico de fichero. El sistema operativo se debe encargar del acoplamiento entre los ficheros y los dispositivos físicos, por medio del sistema de ficheros, que debe ser independiente del soporte físico concreto sobre el que se encuentre.

Los objetivos principales de todo sistema de ficheros deben ser los siguientes:

- Crear, borrar y modificar ficheros.
- Permitir el acceso controlado a la información.
- Permitir intercambiar datos entre ficheros.
- Poder efectuar copias de seguridad recuperables.
- Permitir el acceso a los ficheros mediante nombres simbólicos.

Hay otros objetivos secundarios, entre los que destacan:

- Optimizar el rendimiento.
- Tener soportes diversos para E/S (para poder seguir utilizando los mismos ficheros aunque cambie el soporte).
- Ofrecer soporte multiusuario.
- Minimizar las pérdidas de información.

Normalmente los ficheros se organizan en directorios (también llamados carpetas) para facilitar su uso. Estos directorios son ficheros que contienen información sobre otros ficheros: no son

más que contenedores de secuencias de registros, cada uno de los cuales posee información acerca de otros ficheros.

La información que contiene un fichero la define su creador. Hay muchos tipos diferentes de información que puede almacenarse en un fichero: programas fuente, programas objeto, datos numéricos, textos, registros contables, fotografías, videos, etc. Un fichero tiene una cierta estructura, definida según el uso que se vaya a hacer de él. Por ejemplo, un fichero de texto es una secuencia de caracteres organizados en líneas (y posiblemente en páginas); un fichero fuente es una secuencia de subrutinas y funciones, un fichero gráfico es una secuencia que permite dibujar pixeles en pantalla, etc.

Cualquier sistema operativo distingue entre varios **tipos básicos de ficheros**, que será la clasificación que consideremos nosotros:

- **Regulares o Normales**: Aquellos ficheros que contienen datos (información).
- **Directorios**: Aquellos ficheros cuyo contenido es información sobre otros ficheros, normalmente un vector de entradas con información sobre los otros ficheros.
- **De dispositivo**: Existen dispositivos cuya E/S se realiza como si fuesen ficheros, por lo tanto es razonable asociarles ficheros para simplificar y hacer más transparente el intercambio de información con dichos dispositivos.

Aunque se imponga al sistema operativo el desconocimiento del tipo de ficheros que manipula, sí se hace una distinción del mismo de forma transparente: a través de las extensiones del nombre. Mediante la extensión del nombre del fichero (una cadena de caracteres de pequeña longitud) se puede determinar el tipo del fichero. Algunos sistemas de ficheros consideran a la extensión como una parte del nombre (y, de hecho, admiten que un mismo fichero posea varias extensiones anidadas), y otros la diferencian del nombre a nivel interno.

De este modo, aunque el sistema operativo no conozca internamente la estructura de los ficheros, si es capaz de manejarlos eficientemente gracias al uso de estas extensiones. Esta es la aproximación de los sistemas operativos de Microsoft.

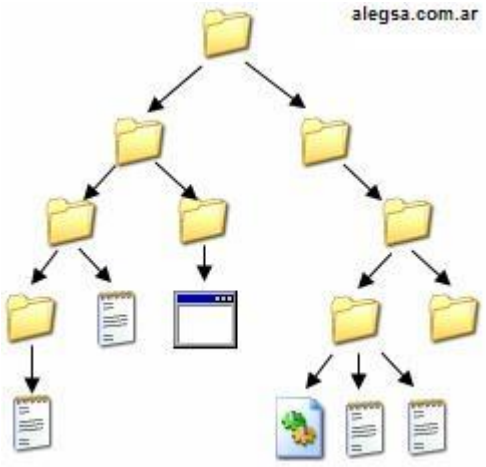
Unix y sus variantes (Linux) sin embargo, optan por la no utilización de extensiones, lo que implica que el usuario es el único encargado de saber lo que se puede realizar o no con un fichero dado.

ESTRUCTURAS DE DIRECTORIOS.

Veamos el siguiente ejemplo: Imaginemos un bufete de abogados que dispone de una ingente cantidad de información en papel: casos judiciales, precedentes, historiales de abogados, historiales de clientes, nóminas, cartas recibidas, copias de cartas enviadas, facturas del alquiler del local, albaranes de compra de lapiceros, procedimientos, etc.

Ahora supongamos que todos estos documentos se almacenan en una enorme montaña de papel en el centro de una habitación: la locura está garantizada. Obviamente el bufete debe disponer de un armario de archivadores, la información se podrá almacenar de forma lógica para poder acceder a ella rápidamente cuando sea necesario.

Lo mismo ocurre en un sistema de ficheros informático: conviene guardar la información (los ficheros) en archivadores. Los archivadores serán lo que llamaremos directorios, un tipo especial de ficheros donde se almacena información relativa a otros ficheros.



Así, en un directorio se almacenarán ficheros relacionados entre sí, y ficheros totalmente independientes irán alojados en distintos directorios. Evidentemente, esta organización es puramente lógica: todos los ficheros estarán almacenados físicamente en el mismo lugar.

Hay varias formas de organizar los directorios sobre un disco:

- Directorio de un nivel. En este tipo de organización solo se permite un nivel de directorio.
- Directorio de dos niveles. En este tipo de organización, un directorio puede incluir dentro otro directorio, pero esté ya no puede incluir otro más.
- Directorio con estructura arborescente. Prácticamente no tiene limitaciones. Un directorio puede incluir otros directorios, sin importar su número, y estos nuevos directorios pueden contener otros directorios.

De esta forma, cada usuario puede crear sus propios directorios para organizar sus ficheros a su gusto. Un ejemplo de estructura de directorios de esta forma es la considerada por los sistemas de ficheros de Unix, MS-DOS, Windows, etc.

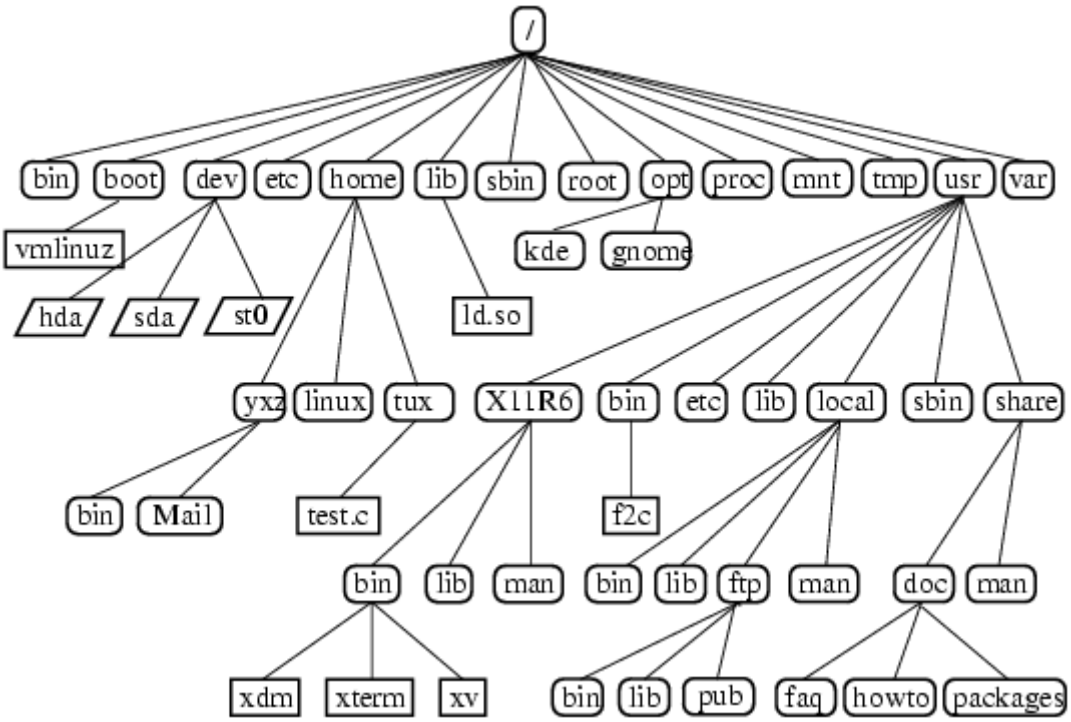
El árbol es de raíz única, de modo que cada fichero tiene un único nombre de ruta de acceso. El nombre de ruta de acceso en un directorio de esta forma es la concatenación de los nombres de directorio y subdirectorios desde el directorio raíz hasta el directorio donde se encuentra alojado el fichero a través del camino único, culminando con el propio nombre del fichero dentro del directorio.

Un directorio (o subdirectorio) contiene a su vez ficheros y/o subdirectorios, y todos los directorios poseen el mismo formato interno. Las entradas del directorio indican si el objeto referenciado es un fichero o un subdirectorio.

Se define **directorio padre** de un fichero o subdirectorio como el directorio en el que se encuentra su entrada de referencia Cada fichero o directorio (a excepción del directorio raíz) posee un único directorio padre. El directorio padre suele ser referenciado por los sistemas operativos con punto punto (..).

Se define directorio hijo de un directorio como el directorio que tiene por padre al primero. Un directorio puede contener múltiples directorios hijos, y cada directorio (a excepción del raíz) es hijo de algún otro.

Se define directorio actual como aquel en el que trabaja el usuario por defecto. Suele ser referenciado por los sistemas operativos con un punto (.). Cuando el usuario hace referencia a un fichero por nombre (no por nombre de ruta de acceso), el sistema inicia la búsqueda siempre en el directorio actual. Si no lo encuentra, comienza a recorrer el camino de búsqueda hasta dar con él. El usuario puede referirse a un fichero también por su nombre de ruta de acceso, en cuyo caso no se da lugar a emplear el camino de búsqueda. El usuario también puede cambiar su directorio actual, especificando un nombre de directorio.



En este caso, los nombres de ruta de acceso pueden adoptar dos formas alternativas. La primera es la del nombre de ruta absoluto, la que hemos definido, por la cual se nombra a cada fichero con respecto al directorio raíz.

Por ejemplo, un nombre de ruta absoluto válido es C:\Documentos\Jose\Privado\Carta.txt. Normalmente, se denota el directorio raíz por medio de un símbolo especial dependiente de cada sistema de ficheros, aunque los más habituales son los símbolos '/' y '\'. En una estructura de directorio de este tipo, el nombre de ruta de acceso absoluto de un fichero o directorio debe ser único.

La segunda opción es la del nombre de ruta relativo. En este caso, se nombra al fichero con respecto al directorio actual. Para esta labor se definen en cada directorio dos entradas de directorio especiales: "." (Un punto) que representa al propio directorio y ".." (Dos puntos), que representa a su directorio padre. Así, se puede considerar un camino único desde el directorio actual hasta cualquier fichero o directorio del sistema. Un ejemplo de ruta relativa valida podría ser por ejemplo ..\Privado\Carta.txt o Jose\Privado\Carta.txt.

Los ficheros se van almacenando en el dispositivo, y por cada uno de ellos se apunta una entrada de directorio, donde almacenamos información sobre el tipo de fichero, nombre y demás. Hay que notar que por cada fichero se almacena por un lado el propio fichero, los datos, y por otro lado se almacena esta entrada de directorio.

Pero, ¿qué se almacena realmente en una entrada de directorio? La siguiente tabla muestra algunas de estas informaciones, aunque en un sistema de ficheros concreto pueden no estar todas las que son ni ser todas las que están:

Nombre del fichero	El nombre simbólico del fichero.
Tipo de fichero	Para aquellos sistemas que contemplan diferentes tipos.
Ubicación	Un puntero al dispositivo y a la posición del fichero en el dispositivo.
Tamaño	Tamaño actual del fichero (en bytes, palabras o bloques) y el máximo tamaño permitido.
Posición actual	Un puntero a la posición actual de lectura o escritura sobre el fichero. Su lugar exacto en el dispositivo.
Protección	Información referente a los permisos de acceso al fichero.
Contador de uso	Número de procesos que están utilizando simultáneamente el fichero.
Hora y fecha	De creación, último acceso, etc.
Identificación	Del proceso creador del fichero, del último que accedió al fichero, en qué forma lo hizo, etc.

Según el sistema concreto, una entrada de directorio puede ocupar desde una decena hasta varios miles de bytes. Por esta razón, la estructura de directorios suele almacenarse en el propio dispositivo que se está organizando, como un fichero especial.

La forma de almacenar los directorios en el dispositivo (estructura de datos del directorio) varía también de un sistema a otro. La forma más sencilla es la de una lista lineal, de forma que cada entrada se encuentra inmediatamente a continuación de la precedente. Esto sin embargo es muy caro en ejecución, pues la búsqueda de una entrada concreta suele exigir pasar por todas las anteriores (búsqueda lineal). Hay otros problemas, como el de qué hacer cuando se elimine

una entrada (algunos sistemas la marcan como borrada para su posterior utilización, otros la incluyen en una lista de entradas libres).

Otra opción es mantener una lista lineal ordenada, pero exige el mantener permanentemente ordenada la lista -lo que complica excesivamente las operaciones de borrado y creación de entradas- y la complicación del algoritmo de búsqueda. Puede pensarse entonces en emplear un árbol binario de búsqueda, que simplifica las operaciones antes mencionadas (y complica el algoritmo de búsqueda).

MÉTODOS DE ASIGNACIÓN

Nos apartamos en este punto de la definición de fichero como tipo abstracto de datos y pasamos a considerar un aspecto bastante crítico: la forma de **ubicar los ficheros físicamente sobre el disco** o, dicho de otro modo, los distintos métodos existentes para asignar espacio a cada fichero dentro del disco. Por supuesto, éste es un aspecto totalmente transparente al usuario: al usuario no le interesa (o no tiene porqué interesarle) en absoluto la forma en que se almacenan físicamente los ficheros. Esto sólo interesa al desarrollador del sistema operativo o al programador de sistemas que necesita estar en contacto con las peculiaridades físicas del dispositivo.

Parece totalmente lógico que la asignación del espacio a los ficheros deba hacerse de modo que ésta sea tan eficiente como sea posible y que las operaciones a realizar sobre los ficheros sean rápidas. Contemplaremos tres métodos de asignación diferentes:

- Asignación **contigua**
- Asignación **enlazada**
- Asignación **indexada**

Algunos sistemas soportan las tres, pero lo más normal es que un sistema determinado sólo soporte uno.

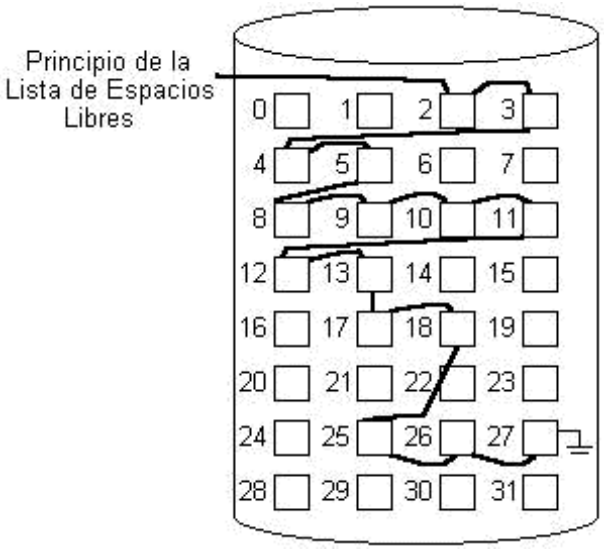
Para empezar, es conveniente conocer cómo se administra el espacio libre del disco.

ADMINISTRACIÓN DEL ESPACIO LIBRE

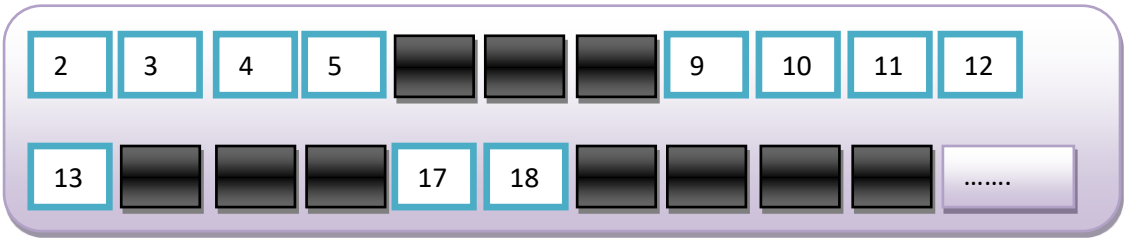
En un sistema informático, los ficheros se crean y se destruyen con frecuencia. Debido a que el espacio de disco no es ilimitado, se hace necesario reutilizar el espacio ocupado por ficheros que han sido borrados para almacenar nuevos ficheros.

El sistema operativo debe mantener, pues, una lista de bloques (clusters) libres (no asignados a ningún fichero).

A la hora de crear un fichero, el sistema operativo examina esta lista en busca de bloques libres, los asigna y elimina dichos bloques de la lista. Cuando se borra un fichero de forma efectiva, los bloques que éste ocupaba se añaden a la lista de bloques libres.



Dicha lista de bloques libres se puede implementar de una forma similar a la que ilustra la figura. El sistema sólo mantiene un puntero al primer bloque libre (*), este primer bloque libre apunta al siguiente, y así sucesivamente hasta llegar al último bloque libre. (Lista encadenada).



Una forma similar de no perder de vista a los bloques libres, sin el engorro (en cuanto a espacio ocupado) que supone el mantener una lista en memoria, es la de implementar un mapa de bits (también llamado vector de bits o, en inglés, bitmap). En este esquema, cada bloque del disco se representa por medio de un bit, que estará puesto a un valor lógico concreto si el bloque está asignado a algún fichero y a su complementario cuando el bloque esté libre. En el disco del ejemplo mostrado en la figura anterior, estaban libres los bloques 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 y 27; el mapa de bits de espacios libres correspondiente sería (suponiendo que el valor lógico 1 indica los bloques ocupados):

11000011000000111001111100011111

Ahora guardaríamos este bitmap en nuestro dispositivo de almacenamiento, y solo ocuparía uno o dos bloques normalmente.

La lista de bloques libres, además de ocupar mucho más que un bitmap, exige leer cada bloque libre para obtener la posición del siguiente, lo cual requiere una cantidad de tiempo considerable de E/S.

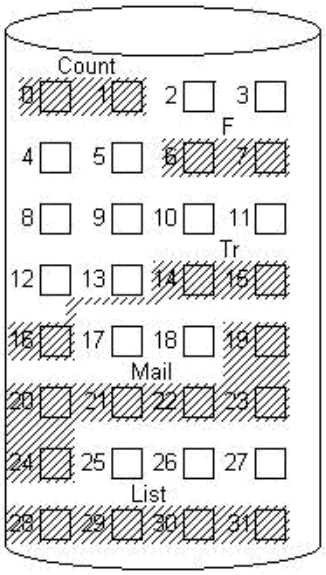
Existen otras alternativas, como son la de mantener una lista de bloques libres ligeramente modificada: se almacenan en el primer bloque libre las referencias a n bloques libres; de éstos, únicamente n - 1 están realmente libres: el último almacena otra n referencias a otros tantos bloques libres. Esta filosofía permite hallar un gran número de bloques libres de forma muy rápida.

Otra alternativa podría consistir en aprovechar que usualmente se asignan o liberan varios bloques contiguos de forma simultánea (sobre todo si se emplea la asignación contigua). Bastaría con almacenar la dirección del primer bloque liberado y el número de bloques libres que le siguen en un contador.

ASIGNACIÓN CONTIGUA

El método de asignación contigua funciona de forma que cada fichero ocupe un conjunto de bloques consecutivos en el disco. Como se dijo en apartados anteriores, cada bloque del disco posee una dirección que confiere una organización lineal al conjunto de bloques (los bloques están seguidos uno detrás de otro). De esta forma el acceso al bloque i+1 desde el bloque i no requiere normalmente movimiento alguno de la cabeza de lectura/escritura y, cuando sí lo requiere, se trata sólo de saltar a la pista siguiente.

Con la asignación contigua, la situación de un fichero queda perfectamente determinada por medio de la dirección de su primer bloque y su longitud. Sabiendo que un fichero tiene una longitud de n bloques y que comienza en el bloque b, entonces se sabe que el fichero ocupa los bloques b, b+1, b+2,... hasta b+n-1. Así pues, la única información relativa a la posición del fichero que es necesario mantener en la entrada del directorio consiste en el par dirección del primer bloque, longitud. En la figura podemos ver un ejemplo de este tipo de asignación.



DIRECTORIO

Fichero	Principio	Longitud
Count	0	2
Tr	14	3
Mail	19	6
List	28	4
F	6	2

Como puede verse, el acceso a un fichero cuyo espacio ha sido asignado de forma contigua es bastante sencillo.

El verdadero problema aparece a la hora de encontrar espacio para un nuevo fichero. Si el fichero a crear debe tener n bloques de longitud, habrá que localizar n bloques consecutivos en la lista de bloques libres. Así, en la figura anterior sería imposible almacenar un fichero que tuviera un tamaño de 7 bloques, aunque en realidad tenemos libres 15 bloques. (**Fragmentación externa**).

Vimos anteriormente que la fragmentación interna era algo inevitable (el espacio que se pierde en un cluster o bloque), sin embargo la fragmentación externa sí se puede resolver. Una solución se llama **compactación** o **desfragmentación**. Cuando aparece una pérdida significativa de espacio en disco por culpa de la fragmentación externa, el usuario (o el propio sistema de forma automática) puede lanzar una rutina de empaquetamiento que agrupe todos los segmentos ocupados generando un único gran hueco. El único coste es el tiempo empleado en llevar a cabo esta compactación.

La asignación contigua presenta algunos otros problemas: esta estrategia parte del hecho de conocer el espacio máximo que ocupará un fichero en el instante mismo de su creación, algo que no siempre (mejor dicho, casi nunca) ocurre. Se puede exigir al usuario (humano o proceso) que señale explícitamente la cantidad de espacio requerido para un fichero dado en el momento de crearlo. Sin embargo, si este espacio resultara ser demasiado pequeño, el fichero no podría

ampliarse en el futuro, y concluiríamos con un mensaje de error o una solicitud de mayor espacio. Normalmente, el usuario sobreestimará la cantidad de espacio necesaria, con lo que se derivará en un despilfarro de espacio muy importante.

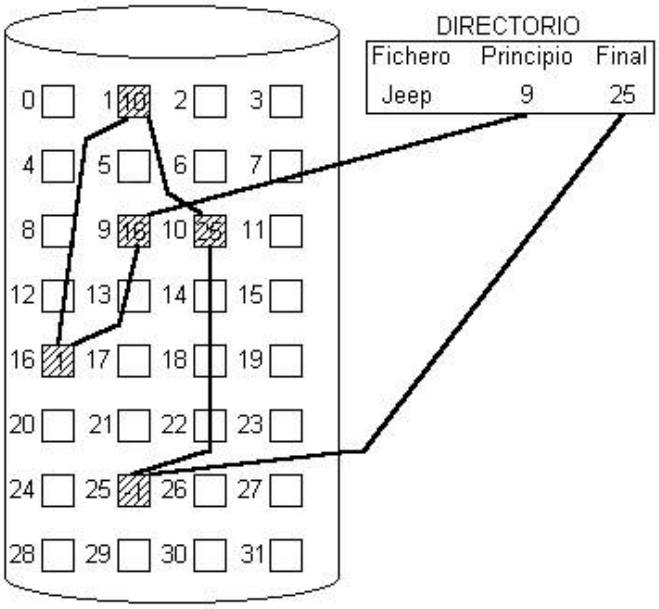
Otra solución podría estar en localizar un hueco mayor cuando el hueco actual se quede pequeño, copiar todo el fichero al nuevo hueco y liberar el anterior. Sin embargo, esto ralentiza excesivamente al sistema. La pre asignación del espacio necesario puede no ser una solución válida aunque se conozca exactamente el tamaño que alcanzará el fichero. Puede que el fichero vaya creciendo lentamente en el tiempo hacia su tamaño total, pero el espacio asignado no se utilizaría durante todo ese tiempo.

Concluyendo, la asignación contigua es **muy eficiente a la hora de acceder a los ficheros**, (de hecho es la asignación más rápida en este aspecto) pero es excesivamente **engorrosa a la hora de asignar el espacio a nuevos ficheros**.

ASIGNACIÓN ENLAZADA

Para resolver los problemas de la asignación contigua, había que crear una estrategia de asignación que permitiera almacenar los ficheros de forma no continua, una forma de conseguir esto es mediante la asignación enlazada. Siguiendo este esquema, cada fichero no es más que una lista enlazada de bloques, que pueden encontrarse en cualquier lugar del disco. La **entrada del directorio** posee únicamente **un puntero al primer bloque** y **un puntero al último**. Cada bloque, a su vez, contendrá un puntero al siguiente bloque. Si suponemos bloques de 4 KB (4.096 Bytes), y usamos 4 Bytes (hasta 4.294.967.296 bloques podríamos usar) para almacenar cada puntero, cada bloque tendría un espacio útil de 4.092 Bytes.

En la figura siguiente podemos ver un fichero almacenado siguiendo esta estrategia. El fichero ocupa 5 bloques que son por este orden el 9, 16, 1, 10 y 25:



El proceso de creación de un fichero es muy fácil: sencillamente se crea una nueva entrada en el directorio dando a sus punteros de principio y final el valor null (valor nulo), indicando que el fichero está vacío.

El proceso de escritura simplemente escribe sobre un bloque libre, eliminándolo de la lista de bloques libres. Si se necesita otro bloque, se busca otro en la lista de bloques libres, se elimina a éste de la lista y se enlaza al anterior, actualizando el puntero al último de la entrada del directorio. La lectura del fichero únicamente consiste en ir siguiendo la cadena de punteros bloque a bloque. En el último bloque podemos colocar una marca en el puntero indicando que es el último.

BLOQUE 9	16

BLOQUE 16	1

BLOQUE 1	10

BLOQUE 10	25

BLOQUE 25	*

Vemos como la asignación enlazada aprovecha al máximo el tamaño de nuestro volumen, pues cualquier bloque libre es susceptible de ser utilizado en cualquier momento.

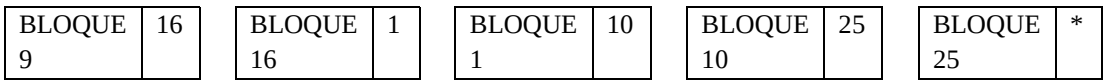
Tampoco es preciso conocer el tamaño de un fichero en el momento de su creación: el fichero puede crecer mientras existan bloques libres en el disco. Por este motivo no es indispensable compactar el disco como ocurría en la asignación continua, aunque si conveniente de vez en cuando desfragmentarlo para agilizar las operaciones de E/S.

Por supuesto, no todo va a ser ventajas. La asignación enlazada sólo es eficiente cuando se acceda a los ficheros de forma secuencial, pues el acceso al bloque i de un fichero exige recorrerlo desde el principio siguiendo la cadena de punteros hasta llegar al bloque deseado.

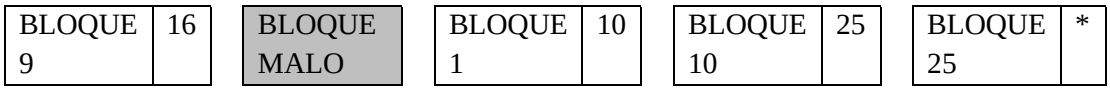
Puesto que no podemos acceder directamente a un bloque sin haber leído los anteriores, no se puede soportar el acceso directo a ficheros siguiendo esta filosofía. (No podemos empezar a leer un fichero por cualquier punto, siempre hay que empezar desde el principio).

Otra desventaja reside en el espacio desaprovechado para almacenar los punteros. En el ejemplo que vimos antes (4 KB por bloque y punteros de 4 Bytes) sólo se desperdicia un 0.09%, lo cual es muy poco, pero aun así es un 0.09% más del sitio que desperdicia la asignación contigua.

El mayor problema, y el más grave subyace en el aspecto de la fiabilidad del sistema: si, por cualquier causa, se dañara un único puntero en un bloque asignado a un fichero, el resto del fichero sería ilocalizable, y cabría la posibilidad de acceder a bloques no asignados o, peor aún, a bloques asignados a otros ficheros.

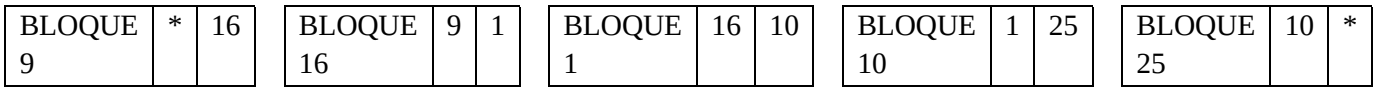


Imaginemos que en el ejemplo anterior, que vemos arriba, se pierde un bloque del disco duro, en este caso el 16 por ejemplo, lo que quedaría:



De nuestro fichero, solo sería posible leer el primer bloque (el 9) ya que a partir de ahí no sabríamos a donde saltar. De esta manera, el fallo de un solo bloque de nuestro volumen de datos, podría hacer que se perdieran ficheros gigantescos. Y lo que es aún peor, el resto de bloques del fichero no quedarían marcados como libres, sino que sería basura que se quedaría en nuestro volumen para siempre.

La solución a este problema puede pasar por emplear una lista doblemente enlazada. Esta lista consiste en utilizar dos punteros en cada bloque, uno que apuntaría al anterior bloque, y otro que apuntaría al siguiente bloque. De esta manera podríamos leer un fichero desde el principio hasta el final, o desde el final hasta el principio. El ejemplo anterior quedaría así:



Puesto que en la entrada de directorio viene marcado el primer bloque y el ultimo, podríamos empezar a leer desde el final, así que aunque se estropeara el bloque 16 como ocurría antes, solo perderíamos un bloque del fichero realmente, ya que podríamos realizar una lectura hacia delante de 1 bloque, y un lectura hacia atrás de los otros 3 bloques del fichero.

La desventaja de esta lista doblemente enlazada, es que se duplica el espacio desaprovechado en el volumen de datos, que pasa del 0,09% al 0,18% que ya es mucho.

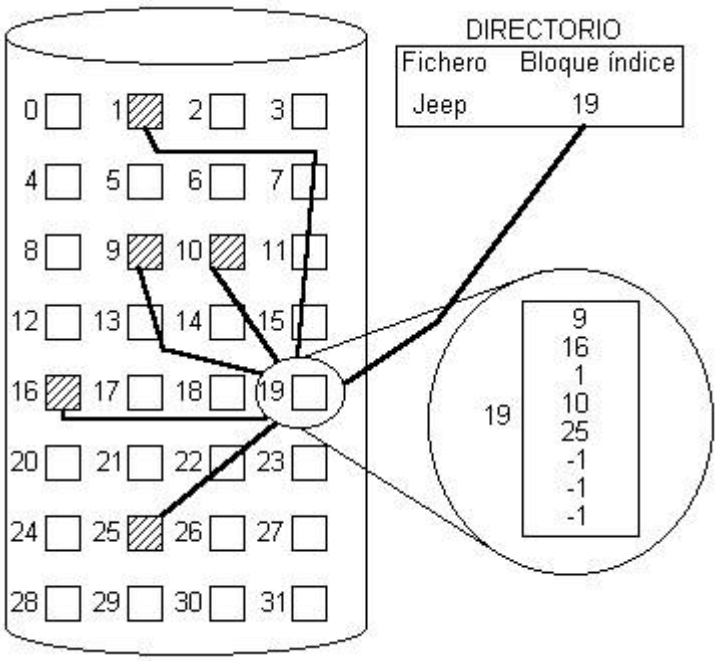
ASIGNACIÓN INDEXADA

Hemos visto hace un momento cómo la asignación enlazada resolvía los problemas de la fragmentación externa grave y de la declaración del tamaño máximo del fichero en el momento de su creación. Sin embargo, vimos también que no daba soporte al acceso directo por estar todos los bloques del fichero (y sus punteros) dispersos por todo el disco.

La asignación indexada viene a resolver estos problemas reuniendo a todos los punteros en un mismo lugar: **el bloque índice**.

A cada fichero le corresponde su propio bloque índice, que no es más que una tabla de direcciones de bloques, donde la entrada i apunta al bloque i del fichero. La entrada de un fichero en el directorio sólo necesita mantener la dirección del bloque índice para localizar todos y cada uno de sus bloques, como puede verse en la figura siguiente:

Ahora sí se permite el acceso directo, pues el acceso al bloque i sólo exige emplear la i-esima entrada del bloque índice. El proceso de creación de un fichero implica inicializar todos los punteros de su bloque índice a null para indicar que éste está vacío. (-1 lo consideramos valor Null), y a continuación ir apuntando los bloques que va usando.



La escritura del bloque i-esimo por primera vez consiste en eliminar un bloque cualquiera de la lista de bloques libres y poner su dirección en la entrada i-esima del bloque índice.

Como se ve, la asignación indexada soporta el acceso directo sin provocar fragmentación externa grave. Cualquier bloque libre en cualquier lugar del disco puede satisfacer una solicitud de espacio.

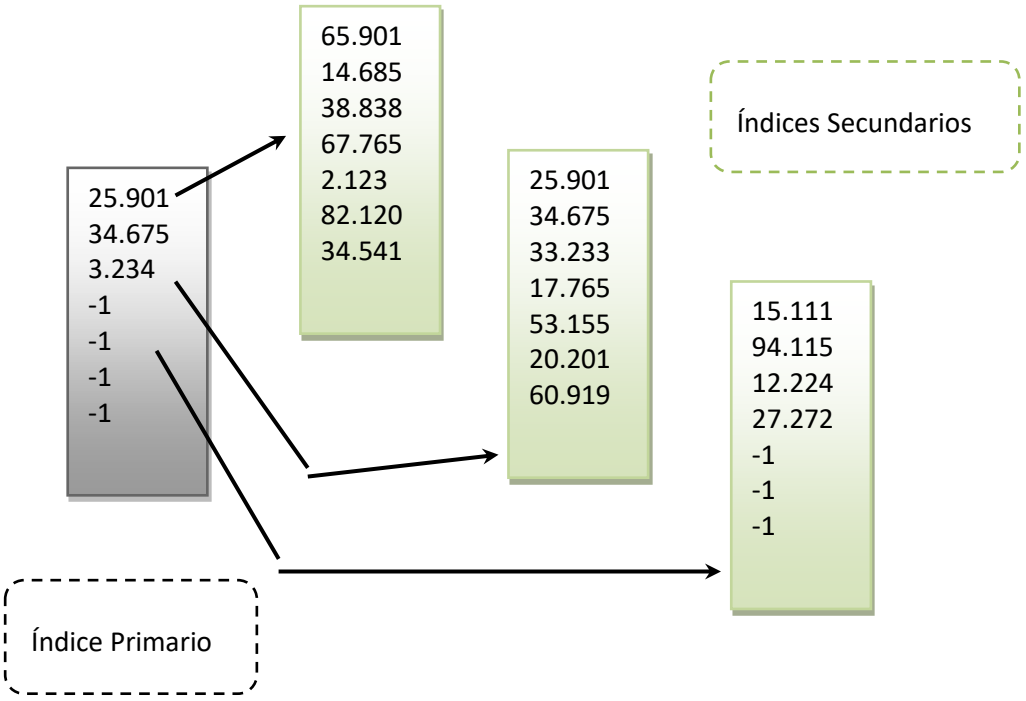
Sin embargo, también hay desventajas: los bloques índices son también bloques de disco, y dejan de estar disponibles para almacenar datos, así que perdemos algo de la capacidad del volumen. Como la mayoría de los ficheros existentes en un sistema son pequeños, el despilfarro puede ser bastante considerable, pues se exige reservar todo un bloque como índice para cada fichero, aunque éste sólo ocupe uno o dos bloques (uno o dos punteros efectivos dentro del índice y el resto conteniendo el valor null).

Si usamos por ejemplo un cluster de 32 KB, y almacenamos un fichero de 10 KB, estaremos usando realmente 64 KB, 32 para almacenar el fichero y 32 para almacenar su bloque índice.

Y supongamos ahora un fichero realmente grande, es tan grande que su lista de bloques usados no cabe en un único bloque índice. Como un bloque índice es también un bloque de disco, podríamos enlazar varios de estos bloques índices para conseguir uno mayor. Por ejemplo, un bloque índice podría contener un pequeño encabezamiento con el nombre de fichero, y las cien primeras direcciones de bloques de disco. La siguiente dirección (la última palabra en el bloque índice) es null (para un fichero pequeño) o bien un puntero a un segundo bloque índice. En el caso de un fichero muy muy grande, este segundo bloque índice podría apuntar al final a un tercero, etc.

Esta filosofía emplea una asignación enlazada para los bloques índices, lo que podría no resultar conveniente en términos de acceso directo. Como alternativa, podríamos pensar en emplear una estructura arborescente (de árbol binario) en dos o tres niveles. Un fichero pequeño sólo emplearía direcciones del primer nivel. A medida que los ficheros crecen puede acudirse a los niveles segundo y tercero.

El acceso a un bloque de datos exige acceder al puntero del primer nivel contenido en el bloque índice primario, el cual apunta a su vez a otro bloque índice secundario, etc. Por ejemplo, si podemos tener 256 punteros en un bloque índice, entonces dos niveles de índice nos permiten referenciar hasta 65.536 bloques de datos (256 x 256).



En este ejemplo de arriba vemos un sistema de asignación indexada de 2 niveles. Cada bloque índice puede almacenar 7 punteros, de modo que podemos asignar un máximo de 49 (7 x 7) bloques a un fichero. Podrimos crear un 3º nivel, haciendo que los punteros de los índices

secundarios apuntaran a un 3º nivel de bloques índice, con lo que podríamos asignar un máximo de 343 (7 x 7 x 7) bloques a un fichero.

Otra alternativa consiste en mantener los primeros punteros del bloque índice, digamos unos 15, en el directorio de dispositivo. Si un fichero requiere de más de 15 bloques, un décimo sexto puntero apunta a una lista de bloques índice. De esta manera, los ficheros pequeños no precisan de un bloque índice.