3 Conceptos de 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.

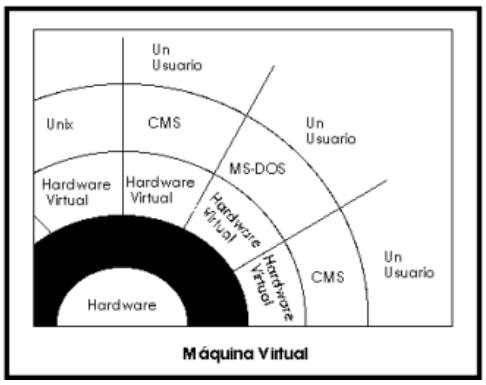
Sin duda alguna, la utilización de los recursos mediante programas es muy complicada, puesto que cada dispositivo es diferente y con gran cantidad de características a controlar. Por ello, una de las primeras acciones a llevar a cabo es el diseño y codificación del software que nos facilite el manejo 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 porqué 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.

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 directamente 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



Máquina Virtual

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.

3.1 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, puede originar una gran cantidad de 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 E/S tiene como objetivo proporcionar una interfaz de alto nivel de los dispositivos de E/S sencilla de utilizar. En algunos sistemas esta interfaz es semejante a la de los ficheros (Linux).
- **Gestión de la red.** El sistema operativo es el encargado de gestionar los distintos niveles de red, los drivers (manejadores) 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 meteórica historia de los ordenadores y la informática. 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.

3.1.1 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 maquina, tenia 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.

De todo esto 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 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.

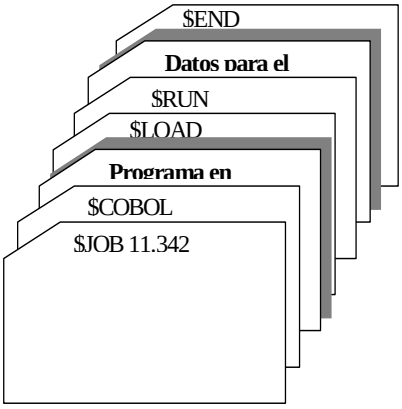
Además, otra de las misiones del operador, era agrupar los trabajos que tuvieran necesidades de recursos físicos y lógicos similares para que se ejecutasen como un grupo. Supongamos que tenemos varios programas que deben ser traducidos con el compilador Fortran y otros con Cobol; si se reúnen todos los programas Fortran y se compilan como un grupo, sólo habrá que cargar el compilador una vez, obteniendo un considerable ahorro de tiempo. Es por esto que a este modo de trabajo se le conoce como “proceso por lotes” o en ingles “batch”. Es una forma de trabajo en la que los programas no pueden ser interactivos, debido a que el usuario no esta presente cuando el trabajo se ejecuta.

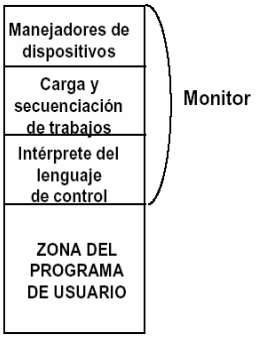
3.1.2 Secuencia automática de trabajos.

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 tratarlas 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.

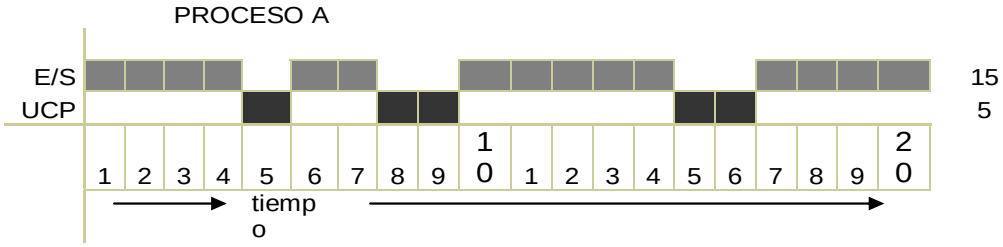
A partir de este momento, los operadores tenían a misión de cargar y descargar las tarjetas en los lectores y perforadores correspondientes, instalar y extraer las cintas magnéticas, mantener las impresoras con suficiente papel, y todas aquellas labores que son necesarias para que el sistema esté operativo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

3.2 MULTIPROGRAMACIÓN.

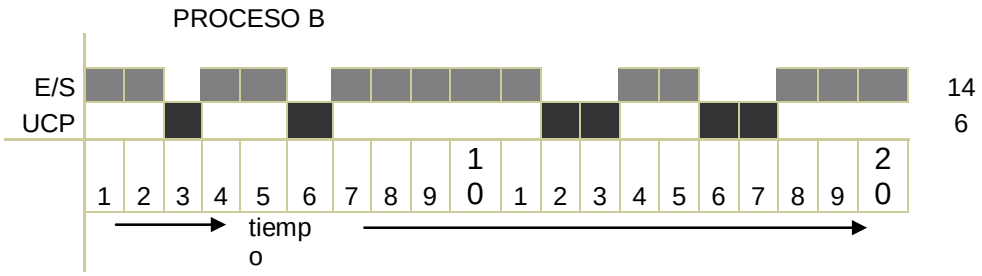
Si ejecutamos un solo programa en un ordenador, difícilmente podremos alcanzar un rendimiento del 100% ya que siempre tendrá que realizar operaciones de entrada/salida. Es decir, habrá tiempos muertos del procesador durante los que no realizará ningún trabajo, y no todo el tiempo estará realizando cálculos del programa. Esto era particularmente notorio en los sistemas de trabajos por lotes, en los cuales el valioso tiempo del ordenador, casi siempre estaba ocupado por rutinarias operaciones de entrada / salida, y no de aprovechamiento de la UCP del ordenador.

Para comprenderlo mejor, podemos tomar como ejemplo la ejecución de los programas representados 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 fueron diseñados.



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 UCP. Establezcamos ahora un segundo proceso (B).

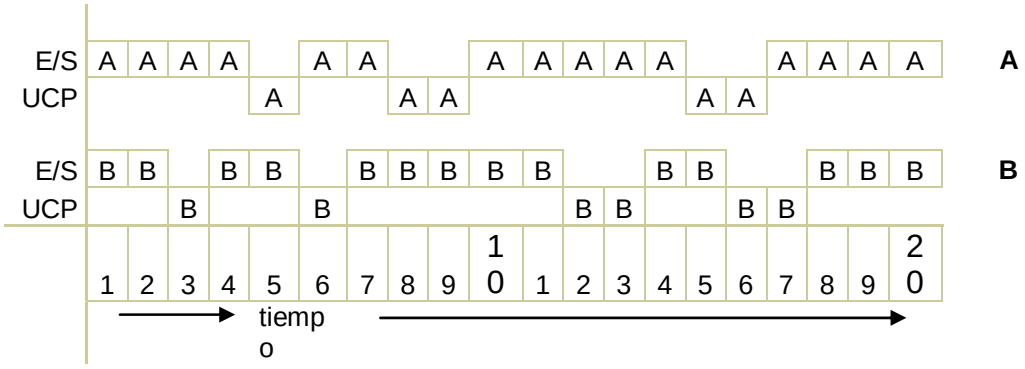
Vemos que este proceso (B) necesita también 20 unidades de tiempo para ejecutarse, de las cuales 14 van a emplearse para trabajar con las E/S, y 6 van a utilizarse para trabajar con la CPU.



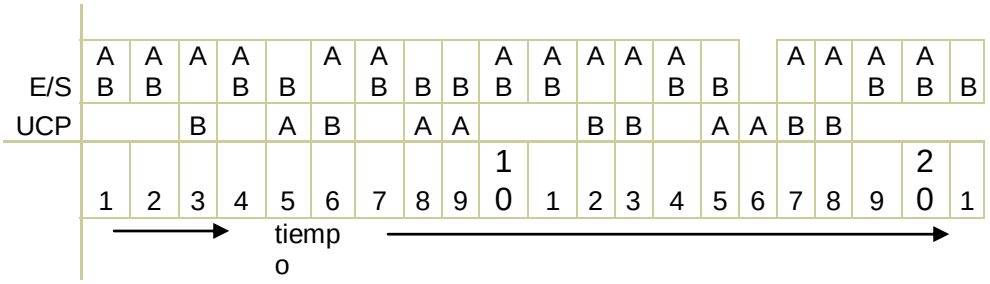
Si tenemos que ejecutar en nuestra maquina, el proceso A y luego el proceso B, el tiempo total de la ejecución será de 40 unidades de tiempo. Sin embargo… ¿no sería posible optimizar algo este tiempo?

Si estudiamos el proceso A, veremos que durante las 4 primeras unidades de tiempo, la UCP esta esperando, sin hacer nada. Y podemos comprobar como 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.



Vemos aquí las necesidades de los procesos A y B, y podemos comprobar como muchas veces, B necesita la UCP y puede usarla porque A no la esta 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 UCP. Fijaros como el tiempo total para ejecutar los dos procesos es de 21 unidades de tiempo, solo 1 mas 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 como 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 UCP esta 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 UCP es de un proceso al mismo tiempo, no pudiendo ejecutar los 2 procesos a la vez).

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.

3.3 GESTIÓN DE LA MEMORIA EN LA MULTIPROGRAMACIÓN.

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 o 100 procesos).

Esto conlleva dos problemas fundamentales, la protección y la relocalización.

3.3.1 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 caso, el controlador de memoria puede solucionar el problema de dos maneras: de manera `estática' o de manera `dinámica'. La solución `estática' consiste en que todas las direcciones del programa se vuelvan a recalcular al momento en que el programa se carga a memoria, esto es, prácticamente se vuelve a recompilar el programa. La solución `dinámica' 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. Por ejemplo, suponga que el programa es cargado en una sección que comienza en la dirección 100. El programa hará referencias a las direcciones 50, 52, 54. Pero el contenido de esas direcciones no es el deseado, sino las direcciones 150, 152 y 154, ya que ahí comienza el programa. La suma de 100 + 50,..., etcétera se hará en tiempo de ejecución. Depende del tipo de programa, puede valer más la resolución estática que la dinámica. Depende de si el proceso es muy largo, de si hace muchas referencias a memoria, etc.

3.3.2 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 otros programas.

La solución a este problema también 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 al 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 (preparados para sistemas monoprogramados) 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, y pertenecen a otros programas.

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

Particiones fijas o particiones variables

En el esquema de la multiprogramación en memoria real se manejan dos alternativas para asignarle a cada programa su partición correspondiente: particiones de tamaño fijo o particiones de tamaño variable.

La alternativa más simple son las particiones fijas. Dichas particiones se crean cuando se enciende el equipo y permanecerán con los tamaños iniciales hasta que el equipo se apague. Es una alternativa muy vieja, quien hacía la división de particiones era el operador analizando los tamaños estimados de los trabajos de todo el día. Por ejemplo, si el sistema tenía 512 Kilo bytes de RAM, podía asignar 64 k para el sistema operativo, una partición más de 64 k, otra de 128k y una mayor de 256 k. Esto era muy simple, pero inflexible, ya que si surgían trabajos urgentes, por ejemplo, de 400k, tenían que esperar a otro día o reparticionar la memoria, inicializando el equipo desde cero.

La otra alternativa, que surgió después y como necesidad de mejorar la alternativa anterior, era crear particiones contiguas de tamaño variable. Para esto, el sistema tenía que mantener ya una estructura de datos suficiente para saber en dónde habían huecos disponibles de RAM y de dónde a dónde habían particiones ocupadas por programas en ejecución, ya que cuando un programa termina su ejecución, la memoria que estaba utilizando se libera. Así, cuando un programa requería ser cargado a RAM, el sistema analizaba los huecos para saber si había alguno de tamaño suficiente para el programa que quería entrar, si era así, le asignaba el espacio. Si no, intentaba relocalizar los programas existentes con el propósito de hacer contiguo todo el espacio ocupado, así como todo el espacio libre y así obtener un hueco de tamaño suficiente. Si aún así el programa no cabía, entonces lo bloqueaba y tomaba otro. El proceso con el cual se juntan los huecos o los espacios ocupados se le llama `compactación'.

Estos mecanismos ocasionan que surjan varios problemas. Por ejemplo, si el sistema tiene tres huecos, uno de 20 k, otro de 24 k y otro de 200 k y un proceso desea 20 k, ¿Cual se le asigna? Existen varios algoritmos para dar respuesta a la pregunta anterior, los cuales se enumeran aquí:

- Primer Ajuste: Se asigna el primer hueco que sea mayor al tamaño deseado.
- Mejor Ajuste: Se asigna el hueco cuyo tamaño exceda en la menor cantidad al tamaño deseado. Requiere de una búsqueda exhaustiva.
- Peor Ajuste: Se asigna el hueco cuyo tamaño exceda en la mayor cantidad al tamaño deseado. Requiere también de una búsqueda exhaustiva.
- El Siguiente Ajuste: Es igual que el `primer ajuste' con la diferencia que se deja un apuntador al lugar en donde se asignó el último hueco para realizar la siguiente búsqueda a partir de él.
- Ajuste Rápido: Se mantienen listas ligadas separadas de acuerdo a los tamaños de los huecos, para así buscarle a los procesos un hueco más rápido en la cola correspondiente.

Por otro lado, conforme el sistema va avanzando en el día, finalizando procesos y comenzando otros, la memoria se va configurando como una secuencia contigua de huecos y de lugares asignados, provocando que existan huecos, por ejemplo, de 12 k, 28k y 30 k, que sumados dan 70k, pero que si en ese momento llega un proceso pidiéndolos, no se le pueden asignar ya que no son localidades contiguas de memoria (a menos que se realice la compactación). Al hecho de que aparezcan huecos no contiguos de memoria se le llama `fragmentación externa'.

De cualquier manera, la multiprogramación fue un avance significativo para el mejor aprovechamiento de la unidad central de procesamiento y de la memoria misma, así como dio pie para que surgieran los problemas de asignación de memoria, protección y relocalización, entre otros.

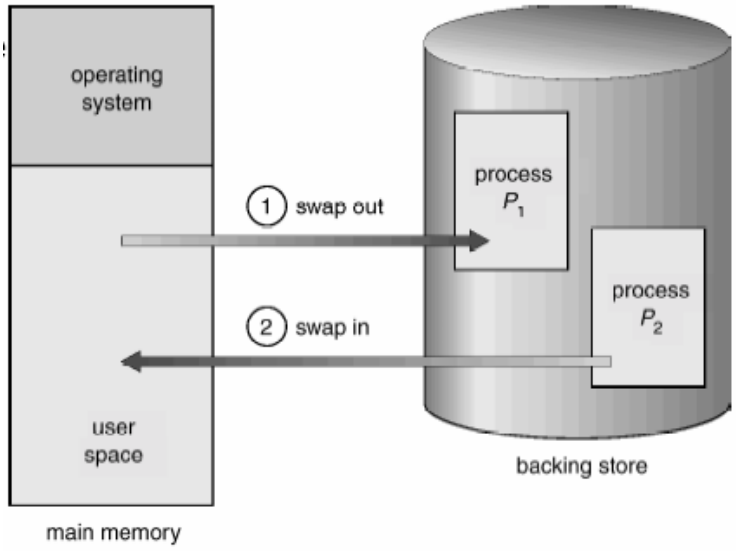
Los overlays

Una vez que surgió la multiprogramación, los usuarios comenzaron a explorar la forma de ejecutar grandes cantidades de código en áreas de memoria muy pequeñas, auxiliados por algunas llamadas al sistema operativo. Es así como nacen los `overlays'.

Esta técnica consiste en que el programador divide lógicamente un programa muy grande en secciones que puedan almacenarse el las particiones de RAM. Al final de cada sección del programa (o en otros lugares necesarios) el programador insertaba una o varias llamadas al sistema con el fin de descargar la sección presente de RAM y cargar otra, que en ese momento residía en disco duro u otro medio de almacenamiento secundario. Aunque esta técnica era eficaz (porque resolvía el problema) no era eficiente (ya que no lo resolvía de la mejor manera). Esta solución requería que el programador tuviera un conocimiento muy profundo del equipo de cómputo y de las llamadas al sistema operativo. Otra desventaja era la portabilidad de un sistema a otro: las llamadas cambiaban, los tamaños de particiones también. Resumiendo, con esta técnica se podían ejecutar programas más grandes que las particiones de RAM, donde la división del código corría a cuenta del programador y el control a cuenta del sistema operativo.

Multiprogramación en memoria virtual.

La necesidad cada vez más imperiosa de ejecutar programas grandes y el crecimiento en poder de las unidades centrales de procesamiento empujaron a los diseñadores de los sistemas operativos a implantar un mecanismo para ejecutar automáticamente programas más grandes que la memoria real disponible, esto es, de ofrecer `memoria virtual'.



La memoria virtual se llama así porque el programador ve una cantidad de memoria mucho mayor que la real, y en realidad se trata de la suma de la memoria de almacenamiento primario y una cantidad determinada de almacenamiento secundario. El sistema operativo, en su módulo de manejo 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.

La memoria virtual se apoya en varias técnicas interesantes para lograr su objetivo. Una de las teorías más fuertes es la del `conjunto de trabajo', la cual se refiere a que un programa o proceso no está usando todo su espacio de direcciones en todo momento, sino que existen un conjunto de localidades activas que conforman el `conjunto de trabajo'. Si se logra que las páginas o segmentos que contienen al conjunto de trabajo estén siempre en RAM, entonces el programa se desempeñará muy bien.

Otro factor importante es si los programas exhiben un fenómeno llamado `localidad', lo cual quiere decir que algunos programas tienden a usar mucho las instrucciones que están cercanas a la localidad de la instrucción que se está ejecutando actualmente.

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.

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 fueron 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 el 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. De hecho, el sistema operativo VMS usa un esquema combinado para cargar páginas: cuando se hace referencia a una dirección cuya página no está en RAM, se provoca un fallo de página y se carga esa página junto con algunas páginas adyacentes. En este caso la página solicitada se cargó por demanda y las adyacentes se cargaron por anticipación.

3.4 MULTIPROGRAMACIÓN. ADMINISTRACIÓN DE PROCESOS.

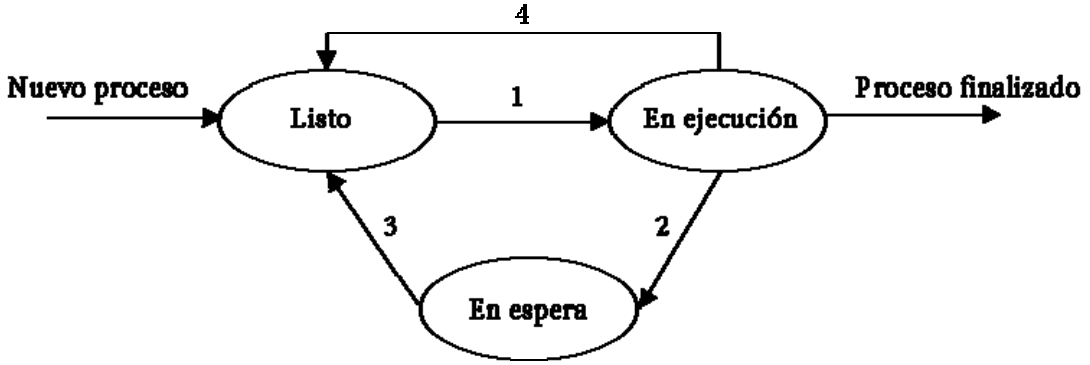
Uno de los módulos más importantes de un sistema operativo es la de administrar los procesos y tareas del sistema de cómputo. En esta sección se revisarán dos temas que componen o conciernen a este módulo: la planificación del procesador y los problemas de concurrencia.

3.4.1 Planificación del procesador.

La planificación del procesador se refiere a la manera o 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.

Niveles de planificación

En los sistemas de planificación generalmente se identifican tres niveles: el alto, el medio y el bajo. El nivel alto decide que trabajos (conjunto de procesos) son candidatos a convertirse en procesos compitiendo por los recursos del sistema; el nivel intermedio decide que procesos se suspenden o reanudan para lograr ciertas metas de rendimiento mientras que el planificador de bajo nivel es el que decide que proceso, de los que ya están listos (y que en algún momento paso por los otros dos planificadores) es al que le toca ahora estar ejecutándose en la unidad central de procesamiento. Vamos a revisar principalmente los planificadores de bajo nivel porque son los que finalmente eligen al proceso en ejecución.



Objetivos de la planificación

Una estrategia de planificación debe buscar que los procesos obtengan sus turnos de ejecución apropiadamente, conjuntamente con un buen rendimiento y minimización de la sobrecarga (overhead) del planificador mismo. En general, se buscan cinco objetivos principales:

- Justicia o Imparcialidad: Todos los procesos son tratados de la misma forma, y en algún momento obtienen su turno de ejecución o intervalos de tiempo de ejecución hasta su terminación exitosa.
- Maximizar la Producción: El sistema debe de finalizar el mayor numero de procesos en por unidad de tiempo.
- Maximizar el Tiempo de Respuesta: Cada usuario o proceso debe observar que el sistema les responde consistentemente a sus requerimientos.
- Evitar el aplazamiento indefinido: Los procesos deben terminar en un plazo finito de tiempo.
- El sistema debe ser predecible: Ante cargas de trabajo ligeras el sistema debe responder rápido y con cargas pesadas debe ir degradándose paulatinamente. Otro punto de vista de esto es que si se ejecuta el mismo proceso en cargas similares de todo el sistema, la respuesta en todos los casos debe ser similar.

Características a considerar de los procesos

No todos los equipos de cómputo procesan el mismo tipo de trabajos, y un algoritmo de planificación que en un sistema funciona excelente puede dar un rendimiento pésimo en otro cuyos procesos tienen características diferentes. Estas características pueden ser:

- Cantidad de Entrada/Salida: Existen procesos que realizan una gran cantidad de operaciones de entrada y salida (aplicaciones de bases de datos, por ejemplo).
- Cantidad de Uso de CPU: Existen procesos que no realizan muchas operaciones de entrada y salida, sino que usan intensivamente la unidad central de procesamiento. Por ejemplo, operaciones con matrices.
- Procesos de Lote o Interactivos: Un proceso de lote es más eficiente en cuanto a la lectura de datos, ya que generalmente lo hace de archivos, mientras que un programa interactivo espera mucho tiempo (no es lo mismo el tiempo de lectura de un archivo que la velocidad en que una persona teclea datos) por las respuestas de los usuarios.
- Procesos en Tiempo Real: Si los procesos deben dar respuesta en tiempo real se requiere que tengan prioridad para los turnos de ejecución.
- Longevidad de los Procesos: Existen procesos que típicamente requerirán varias horas para finalizar su labor, mientras que existen otros que solo necesitan algunos segundos.

Planificación apropiativa o no apropiativa (preemptive or not preemptive)

La planificación no apropiativa (también denominada cooperativa o not preemptive) es aquella en la cual, una vez que a un proceso le toca su turno de ejecución ya no puede ser suspendido, ya no se le puede arrebatar la unidad central de procesamiento. Este esquema puede ser peligroso, ya que si el proceso contiene accidental o deliberadamente ciclos infinitos, el resto de los procesos pueden quedar aplazados indefinidamente. Aquí, el sistema operativo y su planificador de procesos, no pueden hacer nada una vez que el proceso ha llegado a la UCP.

Una planificación apropiativa (preemptive) es aquella en que existe un reloj que lanza interrupciones periódicas en las cuales el planificador toma el control y se decide si el mismo proceso seguirá ejecutándose o se le da su turno a otro proceso. Este mismo reloj puede servir para lanzar procesos manejados por el reloj del sistema. Por ejemplo en los sistemas UNIX existen los 'cronjobs' y 'atjobs', los cuales se programan en base a la hora, minuto, día del mes, día de la semana y día del año.

Es decir, mientras en la multitarea apropiativa (preemptive) es el sistema operativo el encargado de “meter y sacar” a los procesos de la UCP, en la no apropiativa (cooperativa) es el propio proceso el que de vez en cuando debe ceder el control de la UCP al sistema.

Asignación del turno de ejecución

Los algoritmos de la capa baja para asignar el turno de ejecución se describen a continuación:

- Por prioridad: Los procesos de mayor prioridad se ejecutan primero. Si existen varios procesos con la misma prioridad, pueden ejecutarse estos de acuerdo a su orden de llegada o por 'round robin' (por turno en cola circular). La ventaja de este algoritmo es que es flexible en cuanto a permitir que ciertos procesos se ejecuten primero e, incluso, por más tiempo. Su desventaja es que puede provocar aplazamiento indefinido en los procesos de baja prioridad. Por ejemplo, suponga que existen procesos de prioridad 20 y procesos de prioridad 10. Si durante todo el día terminan procesos de prioridad 20 al mismo ritmo que entran otros con esa prioridad, el efecto es que los de prioridad 10 estarán esperando por siempre. También provoca que el sistema sea impredecible para los procesos de baja prioridad.
- El trabajo más corto primero: Es difícil de llevar a cabo porque se requiere saber o tener una estimación de cuánto tiempo necesita el proceso para terminar. Pero si se sabe, se ejecutan primero aquellos trabajos que necesitan menos tiempo y de esta manera se obtiene el mejor tiempo de respuesta promedio para todos los procesos.
- El primero en llegar, primero en ejecutarse: Es muy simple, los procesos reciben su turno conforme llegan. La ventaja de este algoritmo es que es justo y no provoca aplazamiento indefinido. La desventaja es que no aprovecha ninguna característica de los procesos y puede no servir para un proceso de tiempo real. Por ejemplo, el tiempo promedio de respuesta puede ser muy malo comparado con el logrado por el del trabajo más corto primero.
- Round Robin: También llamada por turno, consiste en darle a cada proceso un intervalo de tiempo de ejecución (llamado time slice), y cada vez que se vence ese intervalo se copia el contexto del proceso a un lugar seguro y se le da su turno a otro proceso. Los procesos están ordenados en una cola circular. Por ejemplo, si existen tres procesos, el A, B y C, el planificador daría sus turnos a los procesos en el orden A, B, C, A, B, C…. La ventaja de este algoritmo es su simplicidad, es justo y no provoca aplazamiento indefinido.
- El tiempo restante más corto: Es parecido al del trabajo más corto primero, pero aquí se está calculando en todo momento cuánto tiempo le resta para terminar a todos los procesos, incluyendo los nuevos, y aquel que le quede menos tiempo para finalizar es escogido para ejecutarse. La ventaja es que es muy útil para sistemas de tiempo compartido porque se acerca mucho al mejor tiempo de respuesta, además de responder dinámicamente a la longevidad de los procesos; su desventaja es que provoca más sobrecarga porque el algoritmo es más complejo.
- La tasa de respuesta más alta: Este algoritmo concede el turno de ejecución al proceso que produzca el valor mayor en una formula en la que se tienen en cuenta el tiempo que ha esperado y el que le queda por terminar. Es decir, que dinámicamente el valor se va modificando y mejora un poco las deficiciencias del algoritmo del trabajo más corto primero.
- Por política: Una forma de asignar el turno de ejecución es por política, en la cual se establece algún reglamento específico que el planificador debe obedecer. Por ejemplo, una política podría ser que todos los procesos reciban el mismo tiempo de uso de CPU en cualquier momento. Esto significa, por ejemplo, que si existen 2 procesos y han recibido 20 unidades de tiempo cada uno (tiempo acumulado en time slices de 5 unidades) y en este momento entra un tercer proceso, el planificador le dará inmediatamente el turno de ejecución por 20 unidades de tiempo. Una vez que todos los procesos están 'parejos' en uso de CPU, se les aplica 'round robin' (cola circular).

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. Imagine que un proceso está escribiendo en la unidad de cinta y se le termina su turno de ejecución e inmediatamente después el proceso elegido para ejecutarse comienza a escribir sobre la misma cinta. El resultado es una cinta cuyo contenido es un desastre de datos mezclados. Así como la cinta, 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 'cinta' 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 'cinta'. Ahi se forma una espera circular entre esos dos procesos que se puede evitar quitándole a la fuerza un recurso a cualquiera de los dos procesos.

Los sistemas operativos cuentan con varias técnicas que permiten minimizar estos problemas. Algunas de estas técnicas que pueden usarse son:

Asignar recursos en orden lineal: Esto significa que todos los recursos están etiquetados con un valor diferente y los procesos solo pueden hacer peticiones de recursos 'hacia adelante'. Esto es, que si un proceso tiene el recurso con etiqueta '5' no puede pedir recursos cuya etiqueta sea menor que '5'. Con esto se evita la condición de ocupar y esperar un recurso.

Asignar todo o nada: Este mecanismo consiste en que el proceso pida todos los recursos que va a necesitar de una vez y el sistema se los da solamente si puede dárselos todos, si no, no le da nada y lo bloquea.

Algoritmo del banquero: Este algoritmo usa una tabla de recursos para saber cuántos recursos tiene de todo tipo. También requiere que los procesos informen del máximo de recursos que va a usar de cada tipo. Cuando un proceso pide un recurso, el algoritmo verifica si asignándole ese recurso todavía le quedan otros del mismo tipo para que alguno de los procesos en el sistema todavía se le pueda dar hasta su máximo. Si la respuesta es afirmativa, el sistema se dice que está en 'estado seguro' y se otorga el recurso. Si la respuesta es negativa, se dice que el sistema está en estado inseguro y se hace esperar a ese proceso.

3.5 TIPOS DE SISTEMAS OPERATIVOS.

En este punto vamos a describir las características que clasifican 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).

3.5.1 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 esta 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 subpartes y esto organizado en forma de niveles. Se dividió el sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interfase con el resto de elementos.

Se constituyó una estructura jerárquica o de niveles en los sistemas operativos, el primero de los cuales fue denominado THE, que se utilizó con fines didácticos. 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"

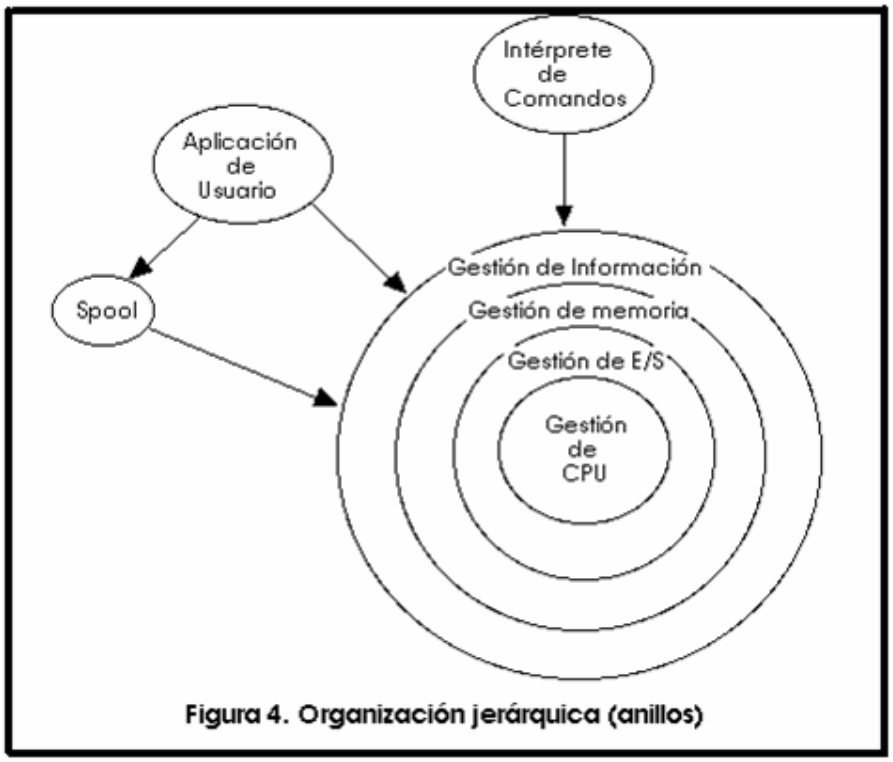


Figura 4. Organización jerárquica (anillos)

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.

3.5.2 Sistemas Operativos por Servicios.

Esta clasificación es la más comúnmente usada y conocida desde el punto de vista del usuario final. Esta clasificación se comprende fácilmente con el cuadro que a continuación se muestra

- **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.
- **Multiusuarios.** Los sistemas operativos multiusuarios 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.

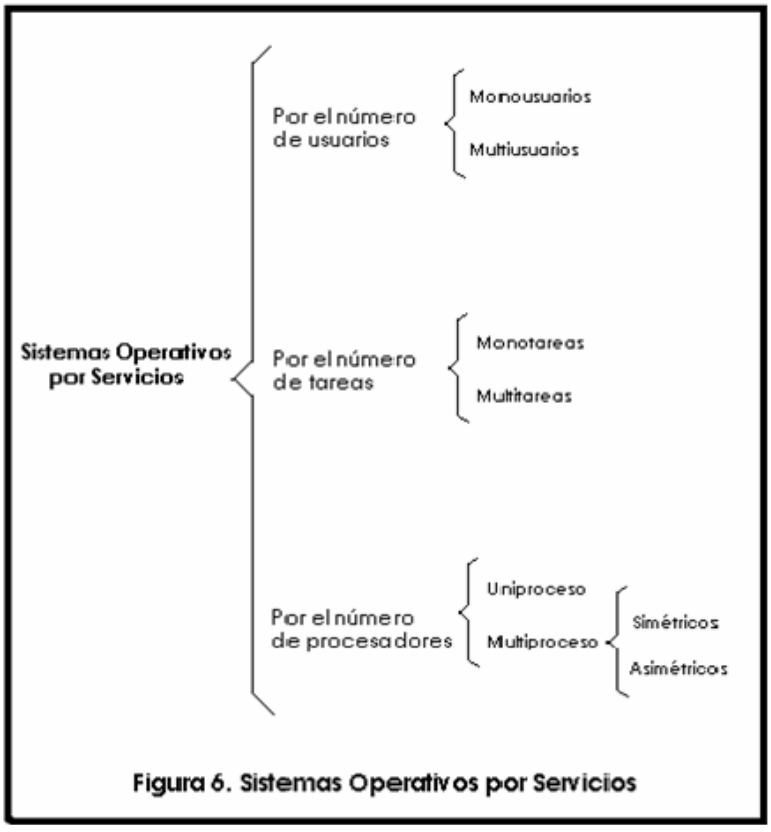


Figura 6. Sistemas Operativos por Servicios

- **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.
- **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: simétrica 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. 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, teóricamente, 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. Un aspecto importante a considerar en estos sistemas es la forma de crear aplicaciones para aprovechar varios procesadores. Existen aplicaciones que fueron hechas para correr en sistemas uniproceso que no toman ninguna ventaja a menos que el sistema operativo o el compilador detecte secciones de código paralelizable, los cuales son ejecutados al mismo tiempo en procesadores diferentes. Por otro lado, el programador puede modificar sus algoritmos y aprovechar por sí mismo esta facilidad, pero esta última opción las más de las veces es costosa en horas y muy tediosa, obligando al programador a ocupar tanto o más tiempo a la paralelización que a elaborar el algoritmo inicial.

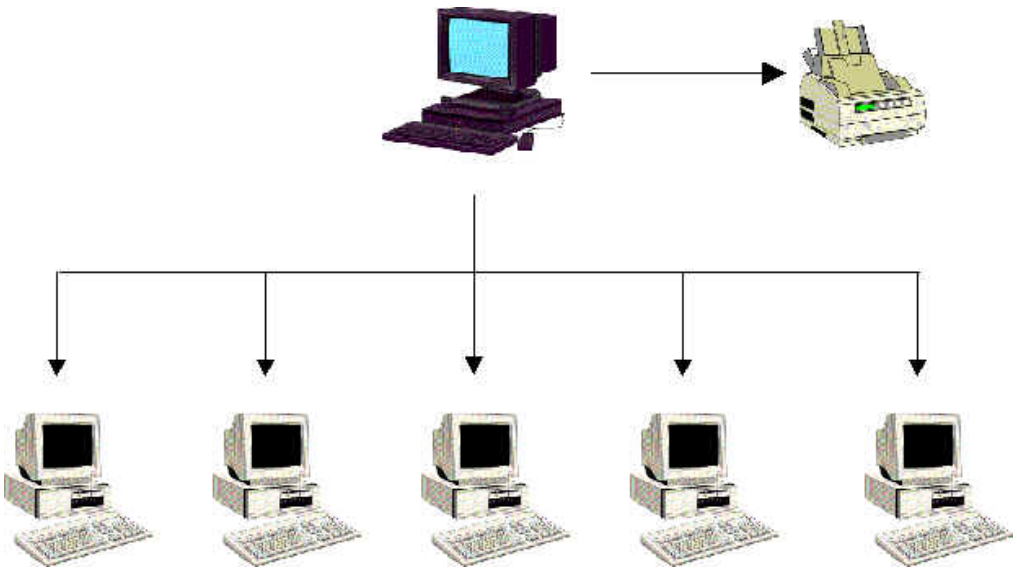
3.5.3 Sistemas Operativos por la Forma de Ofrecer sus Servicios

Esta clasificación también se refiere a una visión externa, que en este caso se refiere a la del usuario, el cómo se accede a los servicios. Bajo esta clasificación se pueden detectar dos tipos principales: sistemas operativos de red y sistemas operativos distribuidos.

Sistemas Operativos de Red.

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. El punto crucial de estos sistemas es que el usuario debe saber la sintaxis de un conjunto de comandos o llamadas al sistema para ejecutar estas operaciones, además de la ubicación de los recursos a los que desea acceder. Por ejemplo, si un usuario en la computadora TIERRA necesita el archivo MATRIZ.C que se localiza en el directorio /software/código en la computadora SOL bajo el sistema operativo UNIX, dicho usuario podría copiarlo a través de la red con los comandos siguientes: RCP SOL:/software/código/MATRIZ.C Si en vez de trabajar con el sistema operativo de red UNIX, trabajáramos con el sistema operativo WINDOWS 2000 la orden seria: COPY \\SOL\software\codigo\matriz.c

Lo importante es hacer ver que el usuario puede acceder a la información no solo de su maquina, sino a la de cualquier maquina de la red, y esto se consigue gracias a que utiliza un sistema operativo de red.



Sistemas Operativos distribuidos.

Los sistemas operativos distribuidos abarcan los servicios de los de red, logrando integrar recursos (impresoras, unidades de respaldo, memoria, procesos, unidades centrales de proceso) en una sola máquina virtual que el usuario usa de forma transparente. Es decir, ahora el usuario ya no necesita saber la ubicación de los recursos, sino que los conoce por su nombre y simplemente los usa como si todos ellos fuesen locales a su lugar de trabajo habitual. Pero es muy complicado distribuir los procesos en varias unidades de procesamiento, reintegrar resultados parciales, resolver problemas de concurrencia y paralelismo, recuperarse de fallos de algunos recursos distribuidos y consolidar la protección y seguridad entre los diferentes componentes del sistema y los usuarios. Estos sistemas operativos, aunque existen desde hace ya varios años, no se muestran especialmente fiables, debido a la complejidad antes mencionada, y a problemas internos de arquitectura. Sin embargo, algunos aspectos de la gestión distribuida si se están usando ampliamente hoy en día.

Así, los sistemas de ficheros distribuidos tienen una gran aceptación, y también se están realizando prácticas en el ámbito del proceso bruto distribuido.

3.6 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: el sistema 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.

Como se dijo más arriba, un fichero no es más que una colección de informaciones (datos) relacionadas entre sí y definidas por su creador. Normalmente, los ficheros representan programas (tanto en forma fuente como en forma objeto) y datos. Los ficheros de datos pueden ser numéricos, alfabéticos o alfanuméricos. Pueden tomar forma libre (como los ficheros de texto) o pueden estar rígidamente formateados. En general, un fichero es una secuencia de bits, bytes, líneas o registros cuyo significado ha sido definido por su creador y usuario.

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, 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, 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. En los capítulos siguientes nos centraremos en los ficheros regulares y en los directorios.

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.

3.6.1 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 y, quizá escondido, hasta algún código deontológico.

Ahora supongamos que todos estos documentos se almacenan en una enorme montaña de papel en el centro de una habitación: la locura está garantizada. La palabra mágica es archivadores. Si el bufete dispone 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.

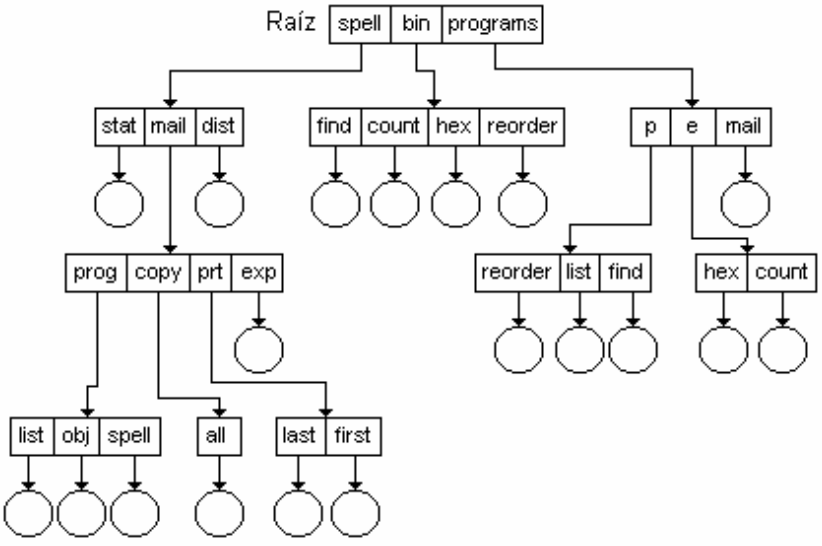
El árbol es de raíz única, de modo que cada fichero tiene un único nombre de ruta de acceso. El nombre de ruta de acceso en un directorio de esta forma es la concatenación de los nombres de directorio y subdirectorios desde el directorio raíz hasta el directorio donde se encuentra alojado el fichero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

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. 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 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).

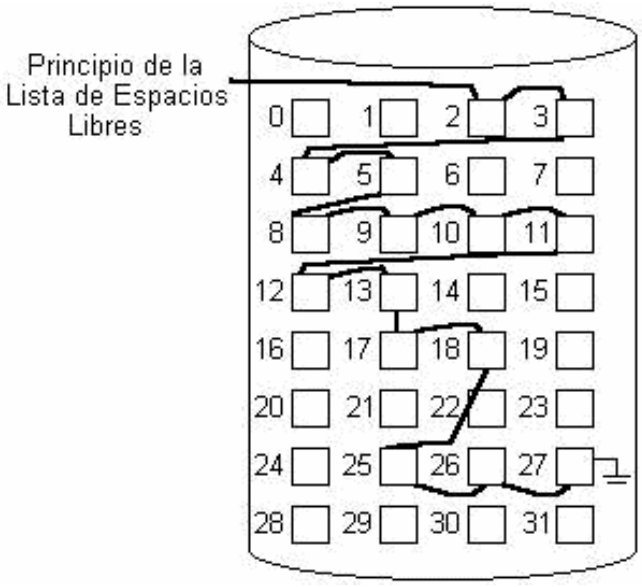
3.6.2 Métodos de asignación

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: la asignación contigua, la asignación enlazada y la asignación indexada. Algunos sistemas soportan los 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 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. (Obviamente, estos bloques de los que estamos hablando son clusters).



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

La lista de bloques libres, además de ocupar 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.

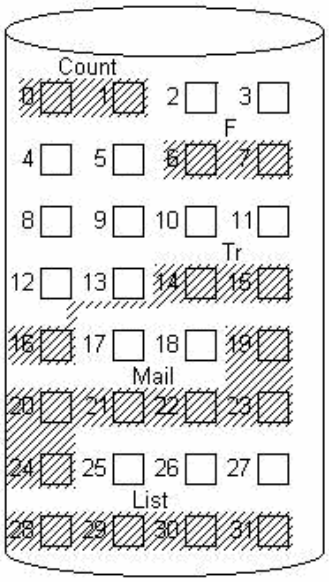
Existen otras alternativas, como son la de mantener una lista de bloques libres ligeramente modificada: se almacenan en el primer bloque libre las referencias a n bloques libres; de éstos, únicamente n - 1 están realmente libres: el último almacena otras n referencias a otros tantos bloques libres. Esta 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, ... y 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)

Aunque vimos que la fragmentación interna era algo inevitable (el espacio que se pierde en un cluster o bloque), la fragmentación externa sí se puede resolver. Una solución se llama compactación. 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) es así. 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.

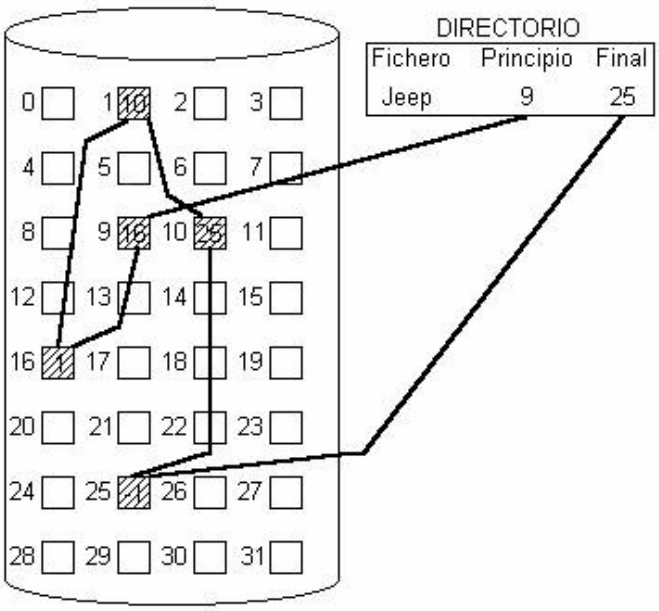
Otra solución podría estar en localizar un hueco mayor cuando el hueco actual se quede pequeño, copiar todo el fichero al nuevo hueco y liberar el anterior. Sin embargo, esto ralentiza excesivamente al sistema. La preasignación del espacio necesario puede no ser una solución válida aunque se conozca 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, pero es excesivamente engorrosa a la hora de asignar el espacio a nuevos ficheros.

Asignación enlazada

Podríamos pensar en no asignar el espacio de forma contigua. La asignación enlazada podría ser la estrategia elegida. 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 necesitamos 4 bytes 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 por este orden: 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.

BLOQUE 9	16

BLOQUE 16	1

BLOQUE 1	10

BLOQUE 10	25

BLOQUE 25	*

La asignación enlazada no provoca fragmentación externa, pues cualquier bloque de la lista de bloques libres es candidato a ser ocupado: basta con tomar el primer bloque que se encuentre. 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. (En nuestro caso bastaría cambiar el puntero que esta en el bloque 25 para que señalara al siguiente bloque, y cambiar el bloque final en la entrada de directorio).

Por supuesto, no todo van 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. Como cada lectura de un puntero requiere un acceso de lectura, 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 32 bits) sólo se desperdicia un 0.09765625%, pero tamaños menores de bloque y mayor número de bloques en disco (léase, mayor tamaño del puntero) derivan en un mayor desperdicio de espacio útil, y un mayor número de bloques requeridos por cada fichero.

Otro problema relativamente 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.

La solución a este problema puede pasar por emplear una lista doblemente enlazada o por almacenar el nombre del fichero y el número relativo del boque en cada bloque del fichero, pero esto requiere espacio adicional.

Asignación indexada

Hemos visto hace un momento cómo la asignación enlazada resolvía los problemas de la fragmentación externa y de la declaración del tamaño 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.

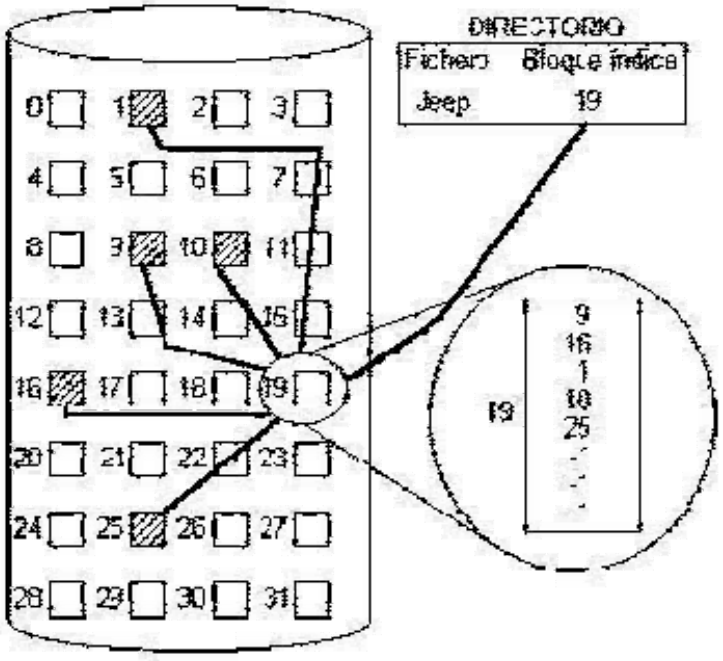
La asignación indexada viene a resolver estos problemillas reuniendo a todos los punteros en un mismo lugar: el bloque índice.

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).

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. 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. 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 dos bloques (dos punteros efectivos dentro del índice y el resto conteniendo el valor null). Es decir, que si tenemos un cluster de 32 Kilo bytes, o lo que es lo mismo, un bloque de 32 Kilo bytes, y almacenamos un fichero de 10 Kilo bytes, estaremos usando 64 Kilo bytes, 32 para almacenar el fichero y 32 para almacenar su bloque índice. Esto no puede suceder, así que hay que disminuir el tamaño del bloque índice.

Ahora bien, ¿qué tamaño debe tener el bloque índice? Visto el problema anterior, parece lógico que debería ser tan pequeño como fuese posible para minimizar la cantidad de espacio desaprovechado en tareas de administración. Sin embargo, para un fichero lo suficientemente grande este bloque índice podría resultar excesivamente pequeño.

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 apuntador a otro bloque índice (para un fichero grande). En el caso de un fichero muy grande, este segundo bloque índice podría apuntar 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 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, que (asumiendo un tamaño de 1 Kb para cada uno) suponen un fichero de 67 108 864 bytes.

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. El antiguo sistema de ficheros MINIX de las primeras versiones de UNIX/LINUX empleaba un esquema análogo a éste.

3.6.3 FAT16

El sistema de ficheros FAT (File Allocation Table), que es el usado en todas las versiones de MS-DOS hasta ahora y en las dos primeras versiones de OS/2 (1.0 y 1.1), además de ser soportado actualmente la mayoría de los sistemas operativos, posee una doble herencia de los primeros lenguajes de programación de Microsoft y del sistema operativo CP/M de Digital Research. Heredó características de sus dos ancestros que se han ido convirtiendo progresivamente en handicaps en esta era de la multitarea, modo protegido, memoria virtual y grandes discos duros.

El sistema de ficheros FAT trabaja sobre la Tabla de Localización de Ficheros (File Allocation Table) de la cual toma su nombre. Cada volumen lógico tiene su propia FAT, que sirve para dos importantes funciones:

Contener la información de localización para cada fichero en el volumen, en forma de listas enlazadas de unidades de almacenamiento (clusters, que son un número de sectores múltiplos de potencias de 2).

Indicar qué unidades de asignación están libres para asignar a un fichero que está creándose o expandiéndose.

Cuando el sistema de ficheros FAT 16 fue concebido, era una solución excelente al manejo del disco ya que los disquetes en los que se usaba eran raramente más grandes de 1Mb. En esos discos, la FAT (tabla de localización de ficheros) era suficientemente pequeña para mantenerse en memoria todo el tiempo, permitiendo un acceso aleatorio muy rápido a cualquier parte de un fichero.

Sin embargo, cuando se aplicó a los discos duros, la FAT se convirtió en un fallo más que en un acierto, ya que se hizo muy grande como para mantenerla totalmente en memoria y tenía que ser paginada en memoria por trozos. Esta paginación dio como resultado muchos movimientos innecesarios de la cabeza del disco mientras un programa leía a través de un fichero, por lo que se degradaba el rendimiento del sistema. Por otra parte, la información sobre el espacio libre en disco se dispersaba entre varios sectores de la FAT, y no era práctico asignar espacio a un fichero de forma contigua, con lo que la fragmentación se convirtió en otro obstáculo para un buen rendimiento. Además, el uso de clusters relativamente grandes en discos duros dio como resultado un montón de espacio desaprovechado.

Las restricciones en los nombres de ficheros de la FAT son heredadas de CP/M. Cuando se escribía 86-DOS (sistema operativo donde surgió FAT 16), uno de los objetivos prioritarios era hacer que los programas fueran fácilmente portables desde CP/M a 86-DOS, por lo que adoptó los límites en nombres de fichero y extensiones, haciendo que los campos críticos en los bloques de control de ficheros (FCB) de 86-DOS parecieran casi idénticos a los de CP/M. El tamaño de los campos de nombre y extensión de los FCB también se adoptó para la estructura de las entradas de directorio.

Mientras tanto, 86-DOS se convirtió en MS-DOS y las aplicaciones para MS-DOS tuvieron un crecimiento espectacular. La mayoría de los primeros programas para este nuevo sistema

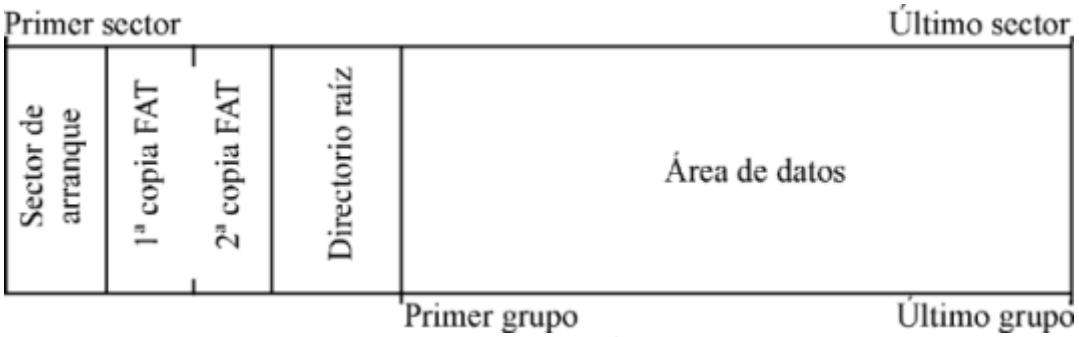
dependían de la estructura de los FCB (Bloque de Control de Fichero), por lo que el formato 8.3 para los nombres de ficheros se estancó irremediablemente en este sistema.

El diseño original de la FAT, establecía que cada entrada de la tabla de localización de ficheros poseía un número de 12 bits, lo que permite un total de 4.096 entradas en la tabla. Como cada entrada en esta tabla, referencia realmente a un cluster, con esta FAT 12 sólo se podían acceder a 4.096 clusters. Si trabajamos en disquetes, no hay problemas con esta cantidad de clusters, pero en un disco duro se necesitan muchísimos más clusters. Para permitir el manejo de discos mayores, se introdujo una FAT de 16 bits, llevando el tamaño máximo hasta aproximadamente 32 Mb. (Por eso se denomina FAT 16. 2 elevado a 16 es 65.536, y si tenemos que cada cluster es de 1 sector, es decir, medio Kilo byte obtenemos 32.768 Kilo bytes, o 32 Megas).

Hoy en día no se fabrican discos tan pequeños, por lo que se hizo obligatorio que el tamaño del cluster fuera mayor de medio Kilo byte, es decir, se hizo que el cluster agrupará varios sectores. Si, por ejemplo, se hace que el tamaño del clúster sea de 8 Kb (16 sectores), se pueden manejar discos de tamaños hasta los 512 Mb (65.536 * 8 / 1024). El problema es que todos los tamaños de los ficheros deben redondearse a un número entero de clusters, y 8 Kb es un tamaño bastante grande. Esto implica que, por ejemplo, un pequeño fichero de procesamiento por lotes con unas pocas docenas de caracteres ocupe 8 Kb de espacio en el disco y en el disco puede haber un gran número de ficheros, por lo que el espacio desperdiciado es muy grande.

La disposición de un volumen FAT se muestra en la figura siguiente. Hay dos copias de la FAT que deberían ser idénticas. Esto puede parecer una medida de seguridad, pero solamente funciona en caso de daño físico y no por errores lógicos. Es decir, si aparece un sector defectuoso en alguno de los ocupados por la FAT, se utiliza el otro de forma automática. Así pues, si el sistema de ficheros se corrompe y las dos copias no son iguales, no hay forma de determinar cuál de las dos es la copia correcta.

Cada disquete o partición bajo el sistema de ficheros FAT es distribuido en cuatro áreas separadas. Estas áreas, son guardadas en el siguiente orden:



Veamos ahora el contenido de cada una de estas áreas.

Sector de arranque

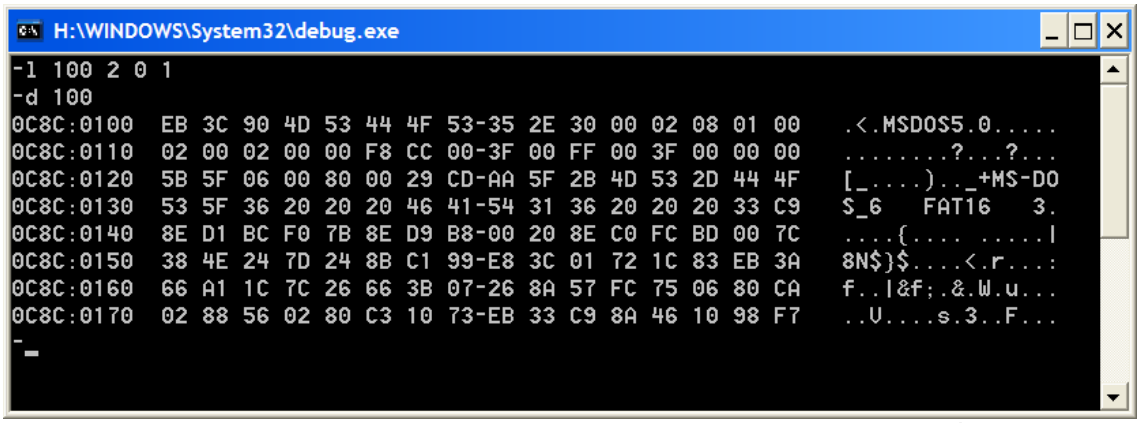
Ocupa el primer sector del disquete o partición. Este sector dispone de un pequeño programa que busca los ficheros ocultos IO.SYS y MSDOS.SYS en el directorio raíz (este es el núcleo o kernel del MS-DOS). Si los encuentra, carga el sistema operativo y si no, muestra alguna variante del conocido mensaje "Error, de disco de sistema. Reemplace y presione cualquier tecla".

El sector de arranque de cada disco posee, además, las siguientes informaciones:

- Nombre y versión del sistema operativo.
- Tipo de disco.
- Número de caras del disco.
- Número de sectores por pista.

- Número de bytes por sector.
- Número total de sectores.
- Número de sectores en cada grupo.
- Número de FAT's.
- Número de entradas máximas en el directorio raíz.
- Número de sectores ocupados por cada FAT.
- Número de serie del disco.
- Etiqueta del disco.

Si queremos ver este sector, para comprobar su contenido, podemos hacerlo mediante la orden de MSDOS (presente también en los sistemas Windows) DEBUG. Para ello, ejecutamos la orden DEBUG, y escribimos lo siguiente:



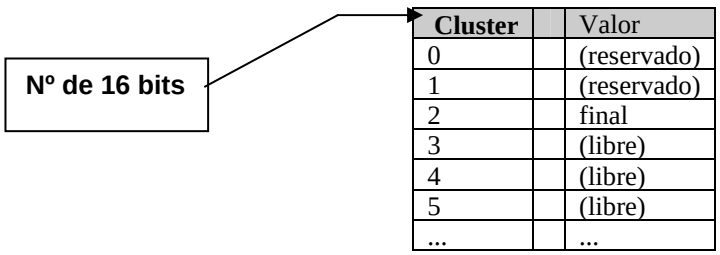
El 2 nos mostrara el sector de arranque de la unidad C, un 1 nos mostraría el sector de arranque de la unidad B, un 0 el sector de arranque de la unidad A, etc.

La FAT

Es el índice del disco. Almacena los clusters utilizados por cada fichero, los clusters libres y los defectuosos. Un fichero puede usar varios clusters no consecutivos. Para seguir el rastro del fichero por el disco, el dos emplea la FAT.

La FAT o tabla de asignación de ficheros es de tal importancia en un disco que se graba por duplicado. Si la FAT se estropea, el dos no reconocerá la información del disco.

Vamos a verlo con un ejemplo. En un disquete vacío de 3½ HD creamos un fichero de 251 bytes llamado ROMANCE.TXT. Como el cluster es de 512 bytes, ROMANCE ocupa el primero disponible (cluster 2). La FAT contiene lo siguiente:



Cluster		Valor
0		(reservado)
1		(reservado)
2		final
3		(libre)
4		(libre)
5		(libre)
...		...

Luego, creamos otro fichero llamado SONETO.TXT de 632 bytes. Como no cabe en un sólo cluster, ocupa dos. LA FAT cambia automáticamente:

Es decir, SONETO empieza en el grupo 3, continúa en el 4 y finaliza.

Cluster		Valor
0		(reservado)
1		(reservado)
2		final
3		4
4		final
5		(libre)
...		...

Ahora, borramos el fichero ROMANCE.TXT. El dos marca como libres los grupos ocupados por el fichero (en este caso, el cluster 2).

Cluster		Valor
0		(reservado)
1		(reservado)
2		final
3		4
4		final
5		(libre)
...		...

Para finalizar, creamos un nuevo fichero con 606 bytes de tamaño y de nombre RECUERDO.TXT. MS-dos busca el primer grupo disponible y lo utiliza. Como no cabe en un sólo grupo, busca otro y también lo utiliza. La FAT queda de esta forma.

Cluster		Valor
0		(reservado)
1		(reservado)
2		5
3		4
4		final
5		final
...		...

El fichero RECUERDO ha quedado fragmentado en dos grupos no contiguos: el 2 y el 5.

Si os fijáis bien, veréis que en la FAT no se guarda la entrada de directorio del fichero (su nombre, longitud, fecha de creación, etc.). La FAT solo lleva el control de los bloques (clusters) libres, reservados y ocupados, y lleva el control de la secuencia de clusters que ocupa un fichero).

El directorio raíz

Esta área almacena las entradas del directorio raíz. Cada entrada consta de los siguientes campos:

Offset	Descripción	Tamaño
0h	Nombre del fichero	8
8h	Extensión	3
0Bh	Atributos	1
0Ch	Reservado	10
16h	Hora	2
18h	Fecha	2
1Ah	Cluster Inicial	2
1Ch	Tamaño del Archivo	4

Cada entrada es almacenada en disco como un conjunto de 32 bytes. Todos los campos menos el nombre y extensión se codifican para ahorrar espacio.

Por ejemplo, el campo de la hora con sólo 2 dígitos almacena horas, minutos y segundos.

El directorio raíz (descodificado) del último ejemplo es el siguiente:

Nombre	Ext.	Atrib.	Hora	Fecha	1º Cluster	Tamaño
RECUERDO	TXT	A	17:00:06	31/08/95	2	606
SONETO	TXT	A	16:58:02	31/08/95	3	632

Los subdirectorios del raíz se comportan como ficheros. Aunque parezca extraño, para el dos un directorio es un fichero cuyo nombre es el nombre del directorio y cuyo contenido son las entradas del directorio. Para diferenciar a los subdirectorios de los ficheros, el dos coloca un atributo especial a los subdirectorios (D).

Además, el directorio raíz contiene la etiqueta del disco. La etiqueta ocupa una entrada más y posee un atributo característico (E). Recuerda como el nombre de una etiqueta no puede sobrepasar los 11 caracteres (8 + 3).

Nota: La etiqueta de un disco se almacena en dos lugares distintos: sector de arranque y directorio raíz.

Si al disco del ejemplo anterior, le creamos un subdirectorio llamado TEXTOS y además, le ponemos la etiqueta "CAPITULO-14", el directorio raíz será el siguiente:

Nombre	Ext.	Atrib.	Hora	Fecha	1º Cluster	Tamaño
RECUERDO	TXT	A	17:00:06	31/08/95	2	606
SONETO	TXT	A	16:58:02	31/08/95	3	632
TEXTOS		D	17:20:10	31/08/95	6	0
CAPITULO	-14	E	17:22:36	31/08/95	0	0

Un gran problema de esta forma de usar el directorio raíz, es que nos limita el número de ficheros que podemos tener en un volumen, dado que cuando esta lista se llene, no puede situarse en ninguna otra parte.

El área de datos

Almacena todos los subdirectorios y ficheros del disco. El área de datos se divide en un número fijo de clusters dependiendo del tipo de disco. Cuando alguien habla del tamaño de un disco, en realidad, se refiere al tamaño del área de datos. En el área de datos de un disquete de 3½ HD caben 1,44 Mb.

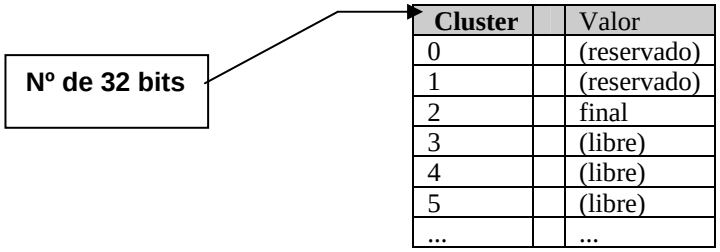
3.6.4 FAT 32.

Con la aparición del sistema operativo Windows 95, se produjo una actualización del sistema de ficheros FAT, como un intento de mejorar su rendimiento. Esta actualización conllevó un cambio de nombre, pasando a llamarse FAT32. Las características principales son:

- Soporte para volúmenes de hasta 2 Terabytes de tamaño.
- Aprovechamiento más eficiente del espacio de disco.
- FAT32 es un sistema de ficheros más robusto y flexible.
- Minimiza el efecto de la fragmentación de archivos.
- Permite nombres de ficheros más largos que 11 caracteres.

Al ser una actualización, todo lo dicho anteriormente para el sistema de archivos FAT se puede aplicar en este caso, ya que su filosofía y su estructura son las mismas, salvo las mejoras incorporadas.

El principal cambio de una FAT a otra radica en la ampliación del tamaño de las entradas, que pasa de 16 a 32 bits. Esto permite mucho más espacio de disco, ya que con una entrada de FAT de 32 bits se permitirían discos de casi 4 Tb., lo que hoy en día es espacio más que suficiente. (2 elevado a 32 es 4.294.967.296).



Otra de las ventajas es la flexibilidad y robustez del sistema, ya que FAT32 usa clusters más pequeños, en concreto de 4Kb, por lo que la eficiencia aumenta y el espacio desperdiciado disminuye. (Es normal que pueda usar clusters más pequeños, ya que puede usar más de cuatro mil millones de clusters en cada volumen).

FAT32 tiene la habilidad de recolocar el directorio raíz y utilizar la copia de seguridad de la FAT de forma efectiva. Además, el Boot record (Registro de arranque) ha sido expandido para incluir una copia de seguridad de los datos críticos del sistema. Esto significa que FAT32 es menos susceptible ante un posible fallo, al contrario que su antecesor, FAT16.

El directorio raíz en un volumen FAT32 es ahora una simple cadena de clusters, que puede ser localizada en cualquier lugar de la unidad. Por esta razón, la limitación en FAT16 sobre el número de entradas del directorio raíz ahora ha desaparecido. Además, la copia espejo de la FAT puede ser deshabilitada, permitiendo que otra copia de una FAT sea considerada como activa. Esto permite que se redimensione el tamaño de una partición FAT32 dinámicamente, aunque Microsoft no lo implementará inicialmente en la primera versión.

Para poder mantener la compatibilidad con aplicaciones, sistemas de redes, sistemas operativos, dispositivos de drivers, etc., FAT32 se implementó con pequeños cambios sobre FAT16. Sin embargo, aunque los cambios fuesen pequeños, eran bastante importantes. Sobre todo el redimensionamiento del tamaño del cluster, pasando a 4Kb, ha hecho que algunas aplicaciones y drivers de dispositivos cambien su estructura para poder manejar este sistema de ficheros.

Todas las herramientas que se utilizaban con FAT16 (chkdsk, format, etc.) han sido revisadas para que puedan manejar volúmenes con FAT32, aunque existen todavía pequeños inconvenientes (por ejemplo, una unidad FAT32 no será reconocida por el sistema operativo Windows NT, o una unidad FAT32 no puede ser comprimida con las utilidades de compresión DriveSpace).

Otra de las grandes ventajas de la FAT32 es la gran disminución de la fragmentación, ya que al ser el cluster más pequeño, el espacio asignado a cada fichero se gestiona más eficientemente. Ahora, un fichero de unos pocos Kbytes no ocupará un cluster de 8 Kb como en FAT16, sino que como mucho ocupará uno de 4Kb, por lo que el espacio desperdiciado se reduce casi a la mitad.

Como comentario final, indicar que este sistema de ficheros solamente está disponible en la versión OSR 2 (Operative System Revision 2) de Windows 95 y posteriores, ya que no se incluyo en la primera versión del mismo.

Aunque hemos visto como FAT32 soluciona gran parte de los principales inconvenientes de FAT16, FAT32 no deja de ser una versión del antiguo sistema de ficheros FAT, muy limitado en cuestiones de seguridad, rendimiento, etc.

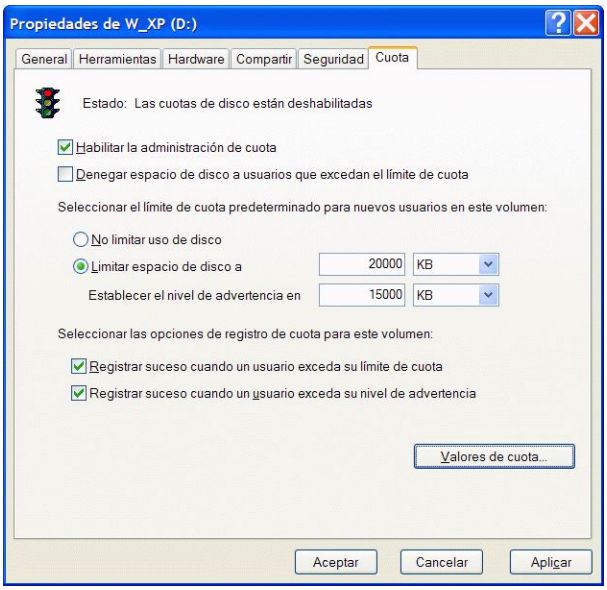
3.6.5 N T F S.

Es algo habitual que un mismo sistema operativo contemple la utilización de varios sistemas de archivos distintos. Windows 2000 y XP, por ejemplo, es capaz de acceder a unidades que utilizan el sistema FAT tradicional, el más reciente FAT32 y, por supuesto, NTFS que es su sistema de archivos nativo. No obstante, cada uno de ellos tiene un sistema de archivos que podríamos denominar preferente. En el caso de Windows 2000 éste es NTFS, que ha ido evolucionando desde Windows NT 3.1 hasta el momento actual, en el que llega a su quinta versión.

Desde su nacimiento, NTFS ha sido un sistema de archivos netamente superior al heredado FAT. Capacidad para discos más grandes, atributos de seguridad en archivos y carpetas, múltiples flujos de datos por archivo, estructura en árbol que acelera las búsquedas, etc. Todas estas características se han visto complementadas, en unos casos, y mejoradas, en otros, en las sucesivas versiones de NTFS que han ido saliendo.

Vamos a ver en primer lugar, algunas de las ventajas que proporciona NTFS sobre otros sistemas de ficheros como FAT32 y luego entraremos en su estructura interna.

Cuestión de cuotas



Los usuarios que utilizan un servidor Windows 2000 comparten numerosos recursos, entre ellos un espacio de almacenamiento que, como todo, es limitado. En versiones previas de NTFS las operaciones que un usuario podía efectuar sobre los archivos y carpetas estaban controladas o restringidas por los correspondientes atributos de seguridad. Nada impedía, no obstante, que un usuario con posibilidad de escritura en una carpeta utilizase todo el espacio disponible en la unidad de almacenamiento llegando, incluso, a llenarla y bloquear el sistema. En las nuevas versiones de NTFS existe un concepto, denominado cuota de disco, mediante el cual el administrador del sistema puede limitar el espacio de forma genérica, a todos los usuarios, o bien de manera selectiva. Esta limitación puede ser blanda, un simple aviso indicándole que está ocupando más espacio del que debería, o bien dura, negándole más espacio en disco del que tiene asignado. La administración del sistema de cuotas de NTFS 2000 se efectúa desde la página de propiedades del disco correspondiente, como puede apreciarse en la siguiente figura. Inicialmente el control de cuotas está desactivado, pero basta con marcar en Habilitar la administración de cuotas para activarlo.

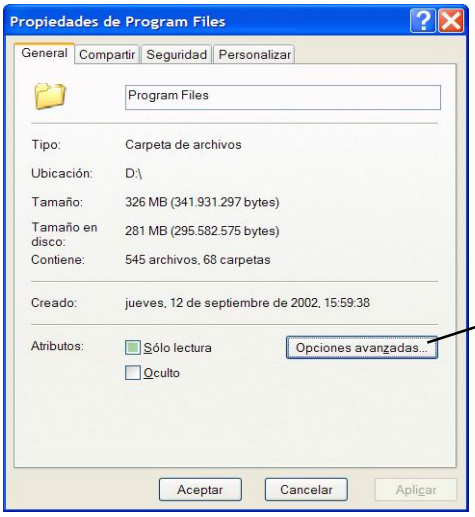
Enlaces blandos y duros.

En los sistemas de archivos Unix existe el concepto de enlace de un fichero. Un enlace no es más que una entrada en el directorio del sistema de archivos. La diferencia es que en sistemas como FAT sólo puede existir una entrada de ese tipo para cada archivo, mientras que en otros sistemas se pueden crear tantos enlaces como se necesiten. De esta forma, un mismo archivo puede aparecer en distintas carpetas o directorios incluso con un nombre distinto.

Con la aparición de Windows 95 se incorporó como novedad al sistema operativo de Microsoft el concepto de acceso directo (enlace blando). A diferencia de un enlace duro, un acceso directo es un archivo y no sólo una entrada de directorio. Ese archivo, no obstante, contiene la información necesaria para apuntar a otro, independientemente de dónde se encuentre, su nombre o su tipo. Un acceso directo puede, además, apuntar a otro tipo de recursos, y no sólo a archivos o carpetas como los enlaces clásicos. En NTFS también pueden crearse enlaces al estilo de Unix, ya que este sistema de archivos se ajusta al estándar POSIX. Windows 2000 o el sistema XP incluyen una utilidad para crear enlaces duros (Fsutil) que además, permite realizar una gran cantidad de acciones administrativas avanzadas sobre los sistemas de ficheros.

Compresión y encriptación de archivos

Un archivo en una unidad NTFS puede albergar varios flujos de datos separados, algo que no es posible en otros sistemas de archivos como FAT. Esos flujos de datos pueden almacenarse en la unidad tal cual, pero también existe la alternativa de comprimir o cifrar dichos flujos de datos. Dichas



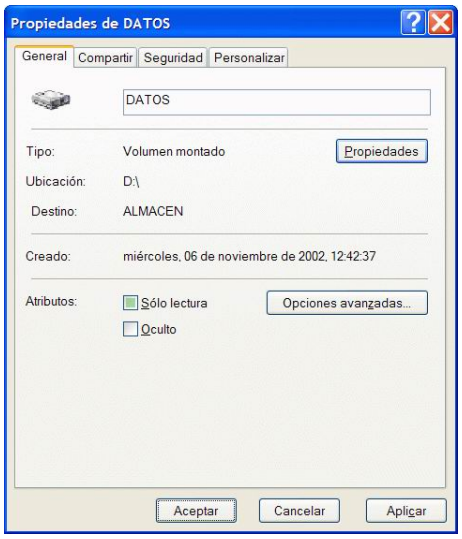
opciones son excluyentes entre sí, es decir, podemos comprimir o cifrar, pero no ambas cosas al mismo tiempo.

El mecanismo de encriptación de archivos de NTFS se basa en el uso de una clave pública de usuario y da lugar al sistema de archivos conocido como EFS (Encrypted File System). En la figura anterior se puede ver la opción que permite cifrar un archivo o todo el contenido de una carpeta.

Los datos cifrados por un usuario no son visibles para los demás a menos, claro está, que dispongan de la clave apropiada. Esto permite un nivel de seguridad en los datos almacenados en disco desconocido hasta ahora, al tiempo que todo el proceso de cifrado y descifrado se efectúa de manera transparente a medida que se leen o escriben los flujos de datos en los archivos. La única indicación de que un archivo está cifrado, la encontrará entre sus atributos.

Puntos de montaje

Los usuarios de DOS y Windows nunca han tenido que preocuparse de montar las unidades de almacenamiento que iban a utilizar conectándolas al sistema de archivos, ya que de esto se ocupaba el propio sistema operativo. Montar y desmontar unidades es algo habitual, sin embargo, en otros sistemas operativos. En Windows 2000 es el propio sistema el que se ocupa de montar automáticamente las unidades de almacenamiento, pero también existe la posibilidad de que el usuario o las aplicaciones, monten y desmonten las unidades, según su conveniencia. Por defecto, Windows 2000 identifica las unidades asignándoles una letra a cada una, (A, B, C).



Si observamos la siguiente figura vemos la página de propiedades de una carpeta llamada DATOS que se encuentra en la unidad D:\ , nos daremos cuenta de que esa carpeta aparece con un icono distinto, indicando así que no se trata de una carpeta corriente sino de un punto de montaje. En la página de propiedades se indica que datos es un Volumen montado, es decir, su contenido no está físicamente en la unidad donde aparece la carpeta (D), sino en otro volumen. (En este caso, el marcado como destino, un disco duro con nombre ALMACEN).

Estos puntos de montajes se pueden realizar, bien mediante funciones de programación o mediante el administrador de discos de Windows 2000 o XP (Inicio – Ejecutar - diskmgmt.msc).

Seguimiento de enlaces y diario de cambios

Desde hace tiempo, principalmente desde la aparición de Windows 95, en Windows es habitual que determinadas aplicaciones mantengan enlaces a datos de otras, o bien que unos archivos actúen como accesos directos a otros. En una hoja de cálculo Excel, por ejemplo, es posible introducir datos enlazados de una base de datos. ¿Qué ocurre, sin embargo, si el destino de un acceso directo o los datos enlazados desde una aplicación se mueven a otro lugar? Generalmente el resultado es que se pierde el enlace.

En Windows 2000, sin embargo, existe un nuevo servicio de seguimiento distribuido de enlaces, un servicio que garantiza que tanto los accesos directos como los enlaces que las aplicaciones tengan a otros archivos mediante OLE, se mantendrán sin problemas, a pesar de que los destinos cambien de nombre, Puede realizar pruebas sobre una unidad NTFS 2000, creando un acceso directo a un cierto archivo y, posteriormente, cambiando el nombre de ese archivo, moviéndolo a otra carpeta u otra unidad. La única limitación, e inconveniente, es que el seguimiento se mantiene siempre que las unidades sean NTFS 5, es decir, en el momento en que movamos el archivo a una unidad FAT o de NT 4.0, por ejemplo, el seguimiento llega a su fin.

Otra de las novedades de la nueva versión de NTFS es el diario de cambios (USN), un archivo de registro o Log en el que van anotándose todos los cambios que se efectúan en los archivos y carpetas de cada unidad. Este diario de cambios garantiza una fácil y ágil recuperación ante fallos del sistema o las unidades, manteniendo la integridad de la información. Este diario de cambios no debe ser tocado directamente, sino que debe ser usado mediante herramientas especiales de recuperación.

NTFS también permite integrar funciones de seguridad sobre ficheros y carpetas, permitiendo llevar un control de quien es el usuario que ha creado el archivo, quien tiene permiso para leerlo, para escribirlo, para borrarlo, etc. Estas funciones se pueden realizar gracias al uso de NTFS, un sistema Windows 2000 o XP montado sobre una partición FAT32 no puede utilizar estas características de seguridad. Estas funciones de seguridad las veremos de forma exhaustiva más adelante.

Estructura de NTFS

Cada fichero en un volumen NTFS está representado por un registro en un fichero especial llamado tabla de fichero maestro (MFT: Master File Table). NTFS reserva los 16 primeros registros de la tabla para información especial. El primer registro describe la propia MFT, seguido por un registro espejo (otras copia de la MFT). Si el primer registro MFT es erróneo, NTFS lee el segundo registro para encontrar el fichero espejo, cuyo contenido es idéntico al del primer registro. Las localizaciones de los segmentos de datos para ambos (MFT y MFT espejo) están grabadas en el sector de arranque (Boot). En el centro lógico del disco está almacenado un duplicado del Boot.

El tercer registro del MFT es el fichero de Log, usado para recuperación de ficheros. A partir del registro 16 de la MFT, los registros son para cada fichero y directorio del volumen. La siguiente figura proporciona una visión simplificada de la estructura del MFT:

Estructura de la MFT.

La MFT reserva una cierta cantidad de espacio para cada registro de fichero. Los atributos de cada fichero son escritos en ese espacio dentro de la MFT. Los ficheros y directorios pequeños (normalmente menos de 1500 bytes), como el fichero de la siguiente figura, pueden ser colocados directamente dentro de la MFT.

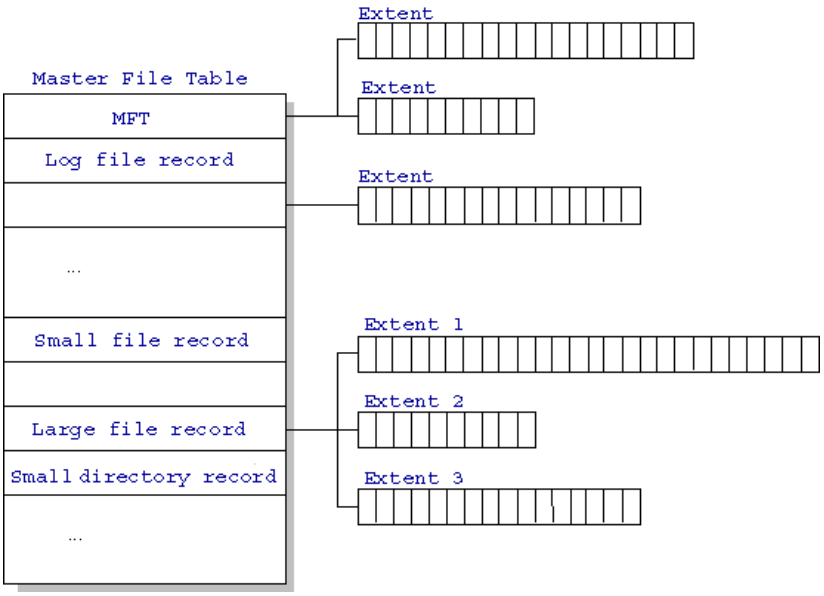
Registro de un directorio o fichero pequeño.

Este diseño hace que los accesos al fichero sean muy rápidos. Vamos a compararlo con un volumen FAT, que usa una tabla de localización de ficheros para listar el nombre y dirección de cada fichero. Las entradas de directorio contienen un índice dentro la FAT. Cuando se quiere ver un fichero, primero se lee la tabla de localización de fichero y se asegura que existe, después recobra el fichero buscando la cadena de clusters asignada al fichero. Con NTFS, tan pronto como se mira el fichero, éste está listo para usar.

Los registros de directorio están alojados dentro la MFT, al igual que los registros de ficheros. En lugar de datos, los directorios contienen información de índice. Los registros de directorio pequeños residen enteramente dentro la estructura de la MFT, pero los directorios grandes están organizados en árboles binarios, teniendo registros con punteros a los clusters externos, los cuales contienen las entradas de directorio que no podían estar dentro la estructura MFT.

Atributos de fichero

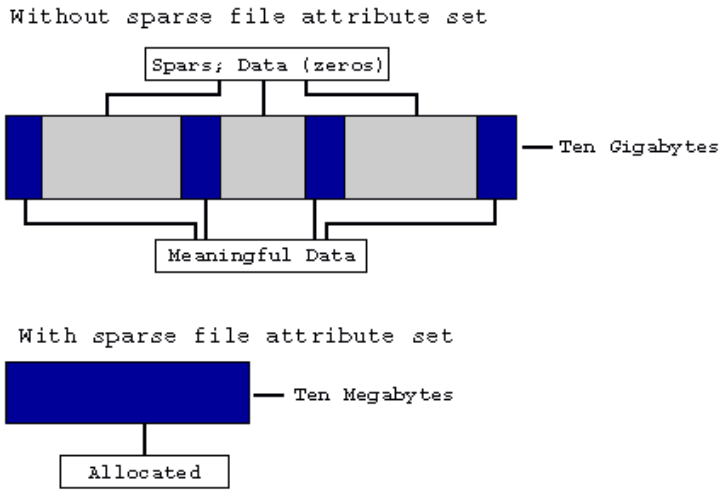
NTFS ve a cada fichero (o directorio) como un conjunto de atributos. Elementos tales como el nombre de fichero, su información de seguridad y sus datos, son todos atributos de fichero. Cada atributo es identificado por un código de tipo de atributo y, opcionalmente, un nombre de atributo. Cuando los atributos de fichero pueden ser escritos dentro de un registro de fichero de la MFT, son atributos denominados residentes. Por ejemplo, la información que especifica que hay que guardar el nombre y el sello de tiempo (time stamp) siempre se incluye en el registro de fichero de la MFT. Cuando un fichero es demasiado grande para colocar todos sus atributos dentro de la MFT, algunos de sus atributos pasan a ser no residentes, y éstos son colocados en uno o más espacios de disco contiguos, en otra parte en el volumen.



En general, todos los atributos pueden ser referenciados como un flujo de bytes sean residentes o no. (Un flujo de datos es una colección de información asociada al fichero. En FAT tenemos que cada fichero es un flujo de dato formada por el propio fichero, en NT tenemos que por cada fichero podemos escribir varios flujos de datos, así un flujo puede ser el propio fichero, otro flujo puede ser una clave para codificación, otro flujo puede ser información de seguridad, etc.).

Por ejemplo, un flujo de datos asociado a un fichero en NTFS es el flujo de la ACL (Lista de Control de Acceso). En este flujo se apuntan todos los identificadores de usuarios y grupos de usuarios que tienen permiso para usar dicho fichero, así como que tipo de permiso tienen. Esta ACL puede tener un tamaño muy elevado si tenemos muchos usuarios en el sistema, siendo muy cómodo almacenarlo como un flujo de datos asociado al fichero.

Tambien cuenta NTFS con el concepto de “sparse”, que permite disminuir el tamaño de grandes archivos, lo que lo hace muy recomendable para operar con estos.



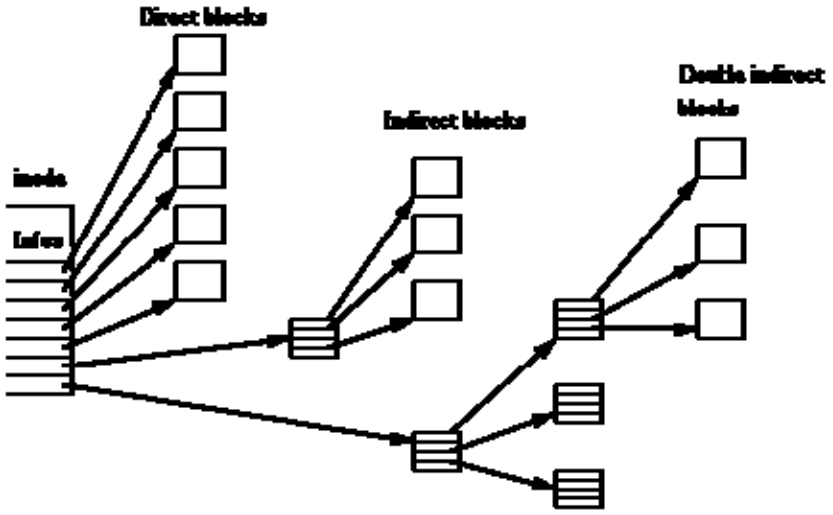
Como inconvenientes de NTFS podemos citar que todas estas estructuras ocupan bastante sitio, lo que no lo hace recomendable para particiones inferiores a 1 GB, y que se puede experimentar una disminución de la velocidad del sistema informático si usamos procesadores o discos dirás antiguos.

3.6.6 Sistemas de Ficheros para Linux.

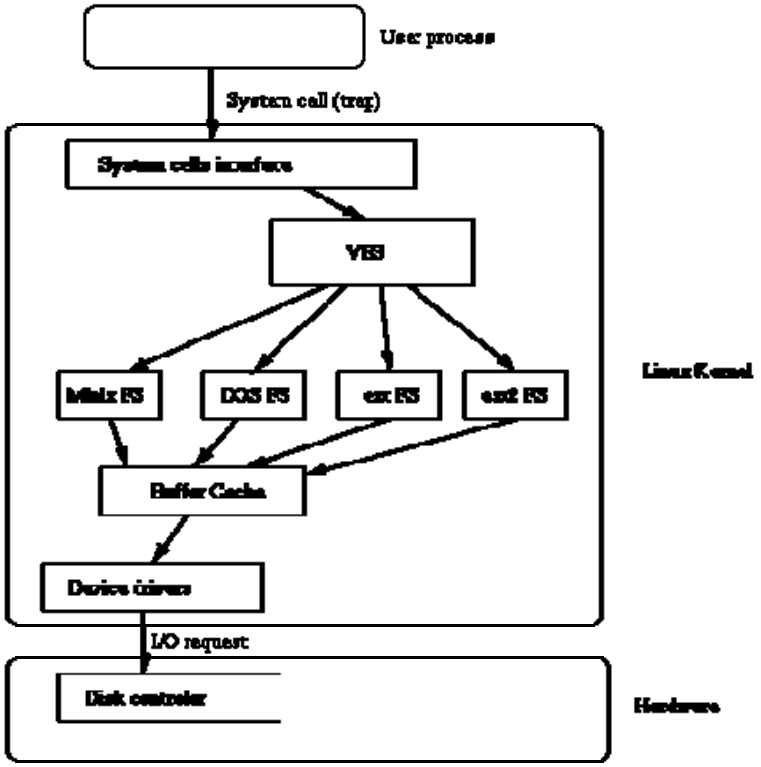
Ext3.

Al principio, el sistema operativo Linux usaba el sistema de ficheros de Minix, sistema en el que se basó Linux. Sin embargo, este sistema de ficheros estaba muy restringido, asi que se empezó a trabajar para implementar un nuevo sistema de ficheros en Linux. En 1992 se creó un nuevo sistema de ficheros llamado Extended File System (Ext), que permitía particiones de 2 GB y tenia nombres de ficheros de 255 caracteres, aunque presentaba diversos problemas. Un año mas tarde apareció un Second Extended File System (Ext2). Es un sistema de ficheros indexado, basado en inodos (índices).

En este sistema, cada fichero es representado por una estructura, denominada inodo (inode). Cada inodo contiene la descripción del fichero, el tipo, derechos de acceso, propietario, fecha y hora, tamaño y punteros a los clusters del fichero. Ext2 implementa el concepto de enlace, en que varios nombres de ficheros pueden ser asociados al mismo inodo.



El núcleo (kernel) de Linux cuenta con un Sistema de Ficheros Virtual (Virtual File System, VFS) que se usa siempre que se llama al Sistema de Ficheros. Esto permite que en Linux se puedan usar distintos sistemas de ficheros fácilmente.



Este inodo de Ext2 tiene un tamaño fijo entre 1 y 4 K, y no usa una FAT, sino una tabla de inodos distribuidos en un número determinable de grupos a través de la superficie, tiene un limite máximo de 4 GB por archivo, 4 TB por partición y detección de desmontaje incorrecto. Sin embargo su gran handicap es que no permite usar "journaling" (registro por diario). El journaling se basa en llevar un registro de todas las transacciones de lectura/escritura que sufre un sistema de archivos, que básicamente funciona de la siguiente manera:

El procedimiento es básicamente el siguiente:

1. Se bloquean las estructuras de datos afectadas por la transacción para que ningún otro proceso pueda modificarlas mientras dura la transacción
2. Se reserva un recurso para almacenar el journal. Por lo general suelen ser unos bloques de disco, de modo que si el sistema se para de forma abrupta (corte eléctrico, avería, fallo del sistema operativo...) el journal siga disponible una vez reiniciado el sistema.
3. Se efectúan una a una las modificaciones en la estructura de datos. Para cada una:
    a. Se apunta en el journal como deshacer la modificación y se asegura de que esta información se escribe físicamente en el disco.
    b. Se realiza la modificación.
    c. Si en cualquier momento se quiere cancelar la transacción se deshacen los cambios uno a uno leyéndolos y borrándolos del journal.
    d. Si todo ha ido bien, se borra el journal y se desbloquean las estructuras de datos afectadas.

En el caso concreto de los sistemas de archivos, el journaling se suele limitar a las operaciones que afectan a las estructuras que mantienen información sobre:

- Estructuras de directorio.
- Bloques libres de disco.
- Descriptores de archivo (tamaño, fecha de modificación...)

El hecho de que no se suela implementar el journaling de los datos concretos de un archivo suele carecer de importancia, puesto que lo que persigue el journaling de sistemas de archivos es evitar los engorrosos y largos chequeos de disco que efectúan los sistemas al apagarse bruscamente, ya que el sistema al arrancar solo deberá deshacer el journal para tener un sistema coherente de nuevo.

Por la imposibilidad de usar journaling en Ext2, se ha creado el sistema de ficheros Ext3, que es una implementación de Ext2 con journaling.

Aparte de ext2 y ext3, gracias al VFS de Linux podemos usar diversos sistemas de ficheros, como ReiserFS, Reiser4, UFS, XFS, JFS, FAT, UFS, UMSDOS, ZFS, etc.

ReiserFS.

ReiserFS es un sistema de ficheros que ofrece journaling, reparticionamiento con el sistema de ficheros montado y desmontado y un esquema para reducir la fragmentación interna.

Comparado con EXT2 y EXT3 en el uso de archivos menores de 4k, ReiserFS es normalmente más rápido en un factor de 10–15.

ReiserFS tiene varias desventajas, como:

- Necesidad de formatear todos los datos para implementar ReiserFS.
- Puede llegar a corromper el sistema de archivos cuando el árbol es reconstruido al realizar un chequeo.
- ReiserFS en versiones del kernel anteriores a la 2.4.10 se considera inestable y no se recomienda su uso, especialmente en conjunción con NFS
- No se conoce una forma de desfragmentar un sistema de archivos ReiserFS, aparte de un volcado completo y su restauración.

La compañía que desarrollaba ReiserFS, ha abandonado este proyecto para centrarse en un nuevo sistema de ficheros, Reiser4.

XFS

XFS es un sistema de archivos con journaling de alto rendimiento creado por SGI para su implementación en UNIX. En mayo del 2000, SGI liberó XFS bajo una licencia de código abierto.

Sus características más destacables son:

- Journaling muy cuidado y optimizado.
- Implementación paralelizada, que escala con el número de CPU's.
- Direccionamiento de 64 bits.

Todo esto hace de XFS un sistema de archivos altamente escalable y fiable. Su principal problema se encuentra en que está mucho menos probado que otros sistemas de ficheros, y que se encuentra al inicio de su desarrollo.

Viene incorporado en las ramas 2.5 y 2.6 del kernel Linux y existen también proyectos para incorporar XFS en FreeBSD.

Se recomienda usar normalmente Ext3 dado que es el más común, y ha demostrado ser un sistema de ficheros bastante estable. Los demás se hallan en mayor o en menor grado en desarrollo, lo que hace que se puedan producir errores importantes, que aunque improbables son muy peligrosos.

Se pueden encontrar descripciones de otros sistemas de ficheros para Linux en la wikipedia:

http://es.wikipedia.org/wiki/Categoría:Sistemas_de_archivos