TEMA 1: INTRODUCCIÓN

CONCEPTO DE SISTEMA OPERATIVO

El **sistema operativo como una maquina extendida o virtual** es un programa que oculta el funcionamiento del hardware ala programador (manejo de interrupciones, relojes, control de memoria y otras características de bajo nivel) y presenta una interfaz sencilla. Además, añade el almacenamiento y organización de la información por medio de un sistema de ficheros, la capacidad de ejecutar varios programas a la misma vez, etc…

El **sistema operativo como un controlador de recursos** es un programa que controla todas las piezas de un sistema complejo y proporciona una asignación ordenada y controlada de los procesadores, memoria, dispositivos E/S, etc…, entre los distintos programas que compiten por ellos.

Sistemas operativos hay muchos, cada uno con sus ventajas e inconvenientes en función de las tareas a realizar, entorno de ejecución, precio, tiempo, etc… Una idea que debe quedar clara es que las características de un sistema operativo dependen muchas veces de las propiedades del hardware sobre el que se ejecuta.

PROTECCIÓN DEL SISTEMA OPERATIVO

La protección del S.O. solo se consigue con la ayuda del hardware y de tres mecanismos básicos:

- **Modos de ejecución del procesador**: el procesador en *modo núcleo* es capaz de ejecutar cualquier instrucción disponible en el procesador. En cambio, el procesador en *modo usuario* no puede ejecutar ciertas instrucciones de la CPU; si lo hace, provocara una excepción que hará que el programa termine su ejecución.
- **Protección de la memoria**: el hardware debe tener algún mecanismo que impida que un problema acceda a zonas de memoria RAM que no le pertenecen. De esta forma se evita que un programa pueda leer o modificar el código, los datos, la pila o el sistema operativo.
- **Interrupciones periódicas**: para que un programa no se haga con el control de la CPU, es necesario que el hardware disponga de algún reloj que produzca interrupciones periódicas que permitan al sistema operativo hacerse con el control de la máquina.

Un aspecto importante es que la protección solo funcionará si los tres mecanismos mencionados existen y se usan de forma conjunta.

HISTORIA Y EVOLUCIÓN

PRIMERA GENERACIÓN (1945-1955): VÁLVULAS Y CONEXIONES

Las grandes máquinas que ocupaban habitaciones enteras estaban construidas mediante miles de válvulas de vacío y cables conectados a mano. Se desconocían los lenguajes de programación y, por supuesto, los sistemas operativos. La programación se realizaba totalmente en lenguaje máquina y con frecuencia se utilizaban conexiones para controlar las funciones básicas del sistema.

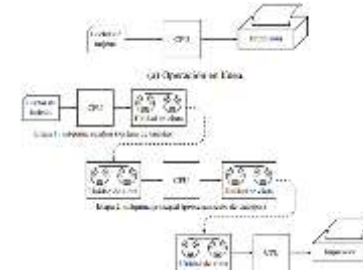
SEGUNDA GENERACIÓN (1955-1965): TRANSISTORES Y SISTEMAS DE PROCESAMIENTO POR LOTES

Lectores de tarjetas, impresoras de líneas, cintas magnéticas, ensambladores, cargadores y enlazadores facilitaron la programación. La introducción del transistor fue fundamental, ya que hizo que los ordenadores se volvieran fiables.

Surgieron las primeras bibliotecas de funciones de E/S, que facilitaban la labor al programador ya que se seguía permitiendo la interacción del programador con la máquina; pero, si el programador no tenía experiencia podía producir importantes pérdidas de tiempo. Para evitar el problema contratan a un operador profesional.

Otra medida fue planificar los trabajos para evitar las pérdidas de tiempo por cambios de cinta. Pero cuando se detenía un trabajo el operador tenía que percatarse de la situación observando la consola. Durante la transición de cargar el lector de tarjetas o de cinta para el siguiente trabajo, la CPU permanecía inactiva, algo inaceptable. Para superar esa inactividad se desarrolló el sistema de procesamiento por lotes, lo que dio lugar a la creación de los primeros sistemas operativos rudimentarios (como el monitor residente).

El *monitor residente* ocupaba de forma permanente una parte de la memoria principal, dejando el resto libre para la ejecución de programas. Este sistema operativo se componía de un intérprete de tarjetas de control, un cargador de programas y aplicaciones y manejadores de dispositivos de E/S.



Los dispositivos de E/S eran lentos en comparación con la velocidad del procesamiento de la CPU. Una solución fue pasar de ***operaciones en línea (on-line)***, donde la CPU interactuaba directamente con los dispositivos de E/S lentos, a ***operaciones fuera de línea (off-line)***, donde la CPU interactúa con dispositivos de E/S. En esta ultima podemos distinguir tres etapas:

1ª etapa: los datos se leen de las tarjetas y se guardan en una cinta (esta se debe llenar).

2ª etapa: los datos de la cinta se procesan y los resultados se guardan en otra cinta.

3ª etapa: los datos de la cinta de salida se imprimen.

La operación fuera de línea dio lugar a la idea de independencia de dispositivos de E/S. El acceso a los dispositivos lógicos siempre se hace de la misma manera, independientemente del dispositivo físico que haya por debajo. Se consiguió que la E/S y la CPU funcionasen en paralelo. El coste del paralelismo era bastante alto, y la solución la dieron los buffers y los spoolers.

Un *buffer* es una zona de memoria principal de un cierto tamaño que se utiliza para el almacenamiento temporal de datos. La CPU no puede intervenir en la E/S del dispositivo; como mucho la CPU puede transferir los datos, aunque lo ideal es que no intervenga. El mecanismo de *spooler* permite la superposición de la E/S de unos programas con el cómputo de otro. Con esto conseguimos que la CPU y los dispositivos de E/S estén más ocupados. Además, la posibilidad de tener varios trabajos distintos en disco hace que sea posible cambiar el orden del procesamiento de estos.

TERCERA GENERACIÓN (1965-1980): CIRCUITOS INTEGRADOS Y MULTIPROGRAMACIÓN

La aparición de los circuitos integrados hace que los ordenadores se vuelvan más pequeños y fiables. La *multiprogramación* consiste en tener varios programas en memoria. Cuando el programa que se usa la CPU no pueda continuar su ejecución porque tiene que esperar, se pasara a otro programa que esté listo para ejecutarse.

El *tiempo compartido* o *multitarea* es una variante de la multiprogramación. Consiste en realizar un cambio rápido entre tareas, lo que permite que cada usuario pueda interactuar con el programa que está ejecutando y tener la sensación que es el único usuario de la máquina.

CUARTA GENERACIÓN (1980-1995): ORDENADORES PERSONALES

La mejora de los circuitos integrados los vuelve más pequeños, dando lugar a la aparición de circuitos LSI y VLSI. Ahora en un chip relativamente pequeño, es posible integrar millones de transistores y otros componentes; con un menor precio, haciéndolos más accesibles. Por lo que, aparece el ordenador personal o PC.

También aparecieron las estaciones de trabajo, que son ordenadores muy potentes conectados por medio de una red Ethernet o similar. Los sistemas operativos que dominan son MS-DOS y Unix; pero a mediados de los 80 empiezan a proliferar la red de ordenadores personales, lo que da lugar a la aparición de nuevos tipos de sistemas operativos.

QUINTA GENERACIÓN (1995-ACTUALIDAD): INTERNET Y DISPOSITIVOS MÓVILES

Si algo caracteriza a esta generación es la explosión del uso de Internet y la proliferación de los teléfonos móviles. En lo que a sistemas operativos se refiere, aparecen Windows 2000 (destinado a los servidores empresariales) y Windows XP (destinado a usuarios). Otros sistemas operativos que alcanzan gran popularidad en esta generación son Linux, Apple Mac OS X, Android e iOS.

TIPOS DE SISTEMAS OPERATIVOS

SISTEMAS OPERATIVOS DE PROPÓSITO GENERAL

Los sistemas operativos diseñados para realizar una gran variedad de tareas computacionales en entornos de diferente naturaleza. Estos sistemas se caracterizan por su flexibilidad y su capacidad para adaptarse tanto al hardware como a los trabajos, Ejemplos: Unix, Microsoft Windows y Apple macOS.

En los supercomputadores, donde el sistema operativo más usado es Unix, son sistemas formados por un gran número de elementos conectados entre si por mediante una red, que son capaces de procesar grandes cantidades de datos.

Unix también es el sistema operativo más común en los mainframes, estos sistemas son capaces de procesar a la vez muchos trabajos que requieren enormes cantidades de E/S,

Cuando pasamos a sistemas de cómputo menos potentes como los servidores podemos encontrar tanto Unix, como Windows o macOS; al igual que en los ordenadores personales. Y finalmente encontramos versiones especiales de estas tres adaptados a los teléfonos inteligentes, tabletas y diferentes sistemas integrados.

SISTEMAS OPERATIVOS DE RED

Unix, Windows y macOS son también sistemas operativos de red, aunque, rara vez se denominan así, E n los sistemas operativos de red, los usuarios son conscientes de la existencia de varios ordenadores conectados mediante una red. Cada máquina es independiente de las demás, ya que ejecuta su propio sistema operativo local y tiene sus propios usuarios. Sin embargo, gracias a la red, puede interactuar con el resto de las máquinas siempre que se hayan concedido las autorizaciones oportunas entre las distintas máquinas que participan.

SISTEMAS OPERATIVOS DISTRIBUIDOS

Son un conjunto de computadores conectados entre sí mediante una red, que son vistos por los usuarios como un sistema tradicional, con un único computador y sistema operativo (*imagen única del sistema*). Estos sistemas son capaces de manejar varios procesadores y en ellos el usuario no es consciente del lugar donde se ejecutan sus programas ni dónde se encuentran sus ficheros (*transparencia de localización*). Ejemplos: Amoeba, Plan 9 e Inferno

Se puede utilizar una impresora o cualquier otro recurso conectado en otra máquina (compartición de recursos); al disponer de varios procesadores se puede hacer que un trabajo use varios y así terminar antes; y confiable ya que, si una máquina falla, el resto seguirá realizando todo el trabajo.

SISTEMAS OPERATIVOS DE TIEMPO REAL

Ejemplos: VxWorks y QNX. El parámetro clave son las restricciones temporales, las podemos encontrar de dos tipos: rigurosos y no rigurosos. En los primeros, una acción se debe realizar necesariamente en cierto momento o intervalo, sino se puede poner el sistema en peligro. En los segundos, es aceptable no cumplir de vez en cuando un plazo, siempre y cuando el comportamiento general del sistema se ajuste a unos ciertos parámetros.

SISTEMAS OPERATIVOS PARA TARJETAS INTELIGENTES

Por las propias características de estas tarjetas, que tienen grandes limitaciones de potencia y memoria, estos sistemas operativos solo disponen de una máquina virtual de Java (JVM) que se encarga de ejecutar los applets que se cargan en la tarjeta.

COMPONENTES Y SERVICIOS DE LOS SITEMAS OPERATIVOS

Un **sistema operativo** proporciona un entorno dentro del cual se ejecutan los programas. Para construir este entorno, se hace necesario dividir lógicamente el sistema operativo en pequeños módulos y crear una interfaz bien definida.

COMPONENTES DEL SISTEMA

Solo es posible crear un sistema operativo dividiéndolo en fragmentos donde cada uno de estos debe ser una porción bien definida del sistema que tenga sus funciones, entradas y salidas cuidadosamente establecidas.

ADMINISTRACIÓN DE PROCESOS

Un *proceso* es un programa en ejecución, una entidad activa, ya que, tiene un contador de programa que dice cuál es la siguiente instrucción por ejecutar. En un *sistema* habrá varios procesos en ejecución, algunos procesos del sistema operativo y otros procesos de usuario.

El **administrador de procesos** del sistema operativo es responsable, entre otras, de crear y eliminar procesos de usuarios y de sistema, repartir el uso de la CPU entre los distintos procesos, suspender y reanudar la ejecución de procesos; y proporcionar mecanismos para la sincronización y comunicación entre procesos.

ADMINISTRACIÓN DE LA MEMORIA PRINCIPAL

La memoria principal (RAM) guarda las instrucciones que ejecuta la CPU y los datos que se leen o escriben durante su ejecución. Para ejecutar un programa necesitamos que se encuentre total o parcialmente en la memoria principal. Además, para mejorar la utilización y la velocidad se deben mantener varios programas en memoria a la vez. Existen varios esquemas para la administración de memoria que dependen del hardware.

La **administración de memoria** del sistema operativo lleva un control de qué zonas de memoria se están usando y qué procesos las usan, decide qué procesos se cargarán en memoria cuando haya suficiente espacio disponible; y asigna y recupera el espacio de memoria según se requiera.

ADMINISTRACIÓN DEL SISTEMA DE E/S

Uno de los objetivos del sistema operativo es ocultar a los usuarios las particularidades de los dispositivos de E/S. Se ocultan al resto una cache en memoria principal construida mediante buffers para acelerar las operaciones de lectura y escritura de disco, manejadores específicos para los dispositivos hardware; y una interfaz general con los manejadores de dispositivo que facilite la implementación de estos. Entre los dispositivos E/S más importantes se encuentran los que sirven de almacenamiento secundario, constituido por discos de diferentes tipos.

ADMINISTRACIÓN DE FICHEROS

Para usar cómodamente un sistema de computación, el sistema operativo ofrece una perspectiva lógica uniforme del almacenamiento de información, que es el fichero. Puesto que el número de ficheros que hay en un dispositivo puede ser grande, estos se organizan en directorios para facilitar el uso. La **administración de ficheros** del sistema operativo crea y elimina ficheros y directorios, cambia el nombre de un fichero o directorio y modificación de los atributos asociados, soporta las operaciones para manipular ficheros y directorios; y se encarga de la correspondencia entre ficheros y almacenamiento secundario.

SISTEMA DE PROTECCIÓN

Asegura que los ficheros, segmentos de memoria, la CPU y otros recursos pueden ser usados únicamente por aquellos procesos que han recibido la correspondiente autorización del sistema operativo.

SERVICIOS DEL SISTEMA OPERATIVO

Un sistema operativo ofrece diferentes servicios para proporcionar en un entorno amigable donde desarrollar y ejecutar programas.

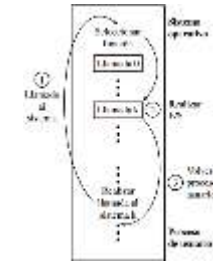
- **Ejecución de programas**: el sistema debe ser capaz de cargar en memoria un programa, ejecutarlo y terminarlo.
- **Operaciones de E/S**: un programa en ejecución puede requerir E/S. Dado que un programa de usuario no puede ejecutar directamente operaciones de E/S, el sistema operativo debe ofrecer alguna forma de llevarlas a cabo.
- **Manipulación del sistema de ficheros**: los programas necesitan leer y escribir ficheros, así como crearlos y eliminarlos basándose en su nombre.
- **Comunicaciones**: un proceso puede necesitar intercambiar información con otro. Para esto existen la memoria compartida y las tuberías.

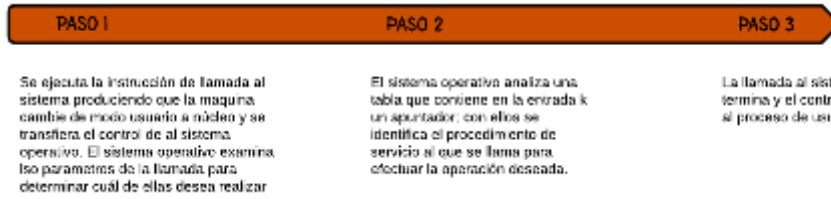
Asegura la eficiencia del sistema

- **Detención de errores**: durante la ejecución de programas pueden producirse errores. El sistema operativo debe estar preparado para detectar y tratar todos los posibles errores, y emprender la acción adecuada para asegurar un funcionamiento correcto y consistente.
- **Asignación de recursos**: cuando varios procesos se ejecutan al mismo tiempo, se deben asignar recursos a cada uno de ellos.
- **Contabilidad**: cuando deseamos llevar un control del uso de los recursos del ordenador por parte de los usuarios con fines contables o para recopilar estadísticas.
- **Protección**: implica revisar la validez de todos los parámetros que se pasan en las llamadas al sistema y asegurar que todo acceso a los recursos del sistema esté controlado. También es importante la seguridad del sistema respecto a personas ajenas, comienza con las contraseñas y se extiende a las conexiones remotas. Para que un sistema esté bien protegido y seguro, hay que establecer controles en todas partes.

LLAMADAS AL SISTEMA

Las **llamadas al sistema** definen la interfaz entre el sistema operativo y un programa en ejecución, son instrucciones en ensamblador que hacen que la ejecución pase de modo usuario a modo núcleo y se salte a una porción de código concreta del sistema operativo para indicar el procesamiento de los servicios solicitados. No existe una llamada al sistema para cada tipo de solicitud, el usuario indica el número de servicio a realizar.





Las llamadas al sistema se pueden clasificar en cinco grandes categorías:

- **Manipulación de ficheros**: estas llamadas crean, eliminan, copian, renombran, imprimen, vuelcan, muestran, y en general manipulan ficheros y directorios.
- **Información de estado o mantenimiento de la información**: algunos programas solicitan al sistema información sobre la fecha, hora, espacio de memoria o disco disponible, número de usuario, etc…
- **Comunicaciones**: estas llamadas proporcionan mecanismos para crear conexiones virtuales, que permiten a los usuarios enviar mensajes a las pantallas de los demás de mayor tamaño o transferir ficheros de una máquina a otra, incluso permiten conectarse de forma remota a otros computadores.
- **Control de procesos**: estas llamadas se encargan de todo lo relacionado con los procesos como cargarlos, crearlos, finalizarlos, etc…
- **Manipulación de dispositivos**: estas llamadas manipulan los dispositivos para solicitar, liberar, leer, escribir, reposicionar y obtener la información del dispositivo.

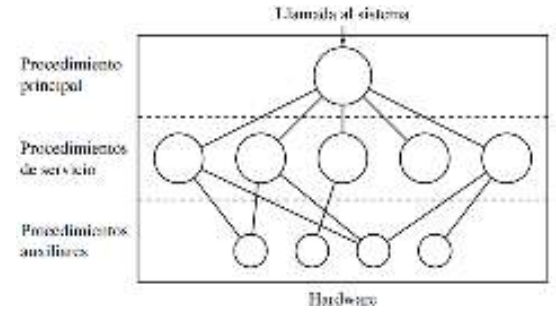
Cuando hablan de programas de sistema son programas que ejecutan llamadas al sistema. Posiblemente el programa de sistema más importante para un sistema operativo es el intérprete de mandatos u órdenes (Shell o Consola).

El diseño de una interfaz útil y amigable depende de los programas del sistema que acompañan al sistema operativo, y no es una función directa de este.

ESTRUCTURA DE LOS SISTEEMAS OPERATIVOS

SISTEMAS MONOLÍTICOS

Este tipo de organización es la más común (Unix y Windows). Se caracteriza por estar formada por una colección de *procedimientos* que se llamas unos a otro, tener cada procedimiento definido en una interfaz muy clara en cuanto a lo que hace el procedimiento, los parámetros que recibe y los resultados que produces; además no existe la ocultación, ya que todo procedimiento es visible a los demás. Y tampoco tiene una estructura bien definida de los procedimientos existentes, aunque algo de estructura tiene que haber. Su estructura es:



- **<u>Procedimiento principal</u>**: recibe las peticiones de llamadas al sistema y tras analizarlas llama a los procedimientos de servicio correspondientes.
- **<u>Procedimientos de servicio</u>**: llevan a cabo las llamadas al sistema.
- **<u>Procedimientos auxiliares</u>**: ayuda a los procedimientos de servicio a realizar su trabajo.

SISTEMAS CON CAPAS

Los avances han permitido realizar un diseño más modular. En un **diseño modular por capas** el sistema operativo se divide en capas, cada una construida sobre la anterior. La capa más baja *(0)* corresponde al hardware y la capa más alta es la interfaz con el usuario.

Cada capa solo utiliza las funciones y servicios de la capa inmediatamente inferior, una vez se ha implementado, verificado y depurado una capa se asciende a la siguiente, haciéndolo mucho más fácil. Y cada capa oculta los detalles de bajo nivel a las capas superiores. Esta permite modificar una capa en cualquier momento sin tener que hacer cambios en otras, excepto que se altere la interfaz de la capa.

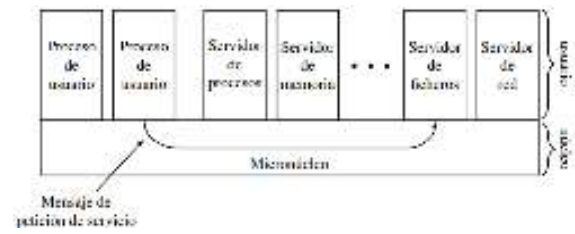
La mayor dificultad de este enfoque es el propio diseño de capas ya que solo puede utilizar los servicios de las capas inferiores; esto hace que surjan problemas de dependencia entre capas y que sea difícil saber dónde colocar una determinada función.

MODELO CLIENTE-SERVIDOR

Explotan la idea de mover parte del sistema operativo a capas superiores y mantener un núcleo mínimo (*micronúcleo* o *microkernel*, controlan las comunicaciones cliente-servidor). Lo usual es implantar la mayoría de las funciones del sistema operativo en los procesos usuario, por lo que se distinguen dos tipos de procesos:

- ***<u>Cliente</u>***: solicitan servicios.
- ***<u>Servicios</u>***: realizan los trabajos solicitados por los procesos clientes y les devuelven las respuestas.

Los servidores se ejecutan en modo usuario, por lo que no hay acceso directo al hardware. Otra ventaja del modelo cliente-servidor es su capacidad de adaptación a los sistemas distribuidos. Un cliente se comunica con el servidor por medio de mensajes (se envía una solicitud y se recibe una respuesta).



Ya que los servidores se ejecutan en modo usuario, hay funciones que no pueden realizar directamente ellos mismos. Para solucionar este problema, existen dos opciones:

- **Incluir el servidor en el núcleo**, esto hace que para el servidor incluido desaparezca la ventaja de protección frente a los fallos en el propio servidor.
- **Existencia de mensajes especiales** enviados por los servidores que captura el micronúcleo para procesarlos el mismo. En este caso, el micronúcleo también verificará si el servidor que le envía una solicitud tiene permiso para hacerlo (la más preferible).

TEMA 2: GESTIÓN DE PROCESOS

INTRODUCCIÓN

En un sistema de computación hay muchos recursos que administrar. Entre ellos, el procesador o la CPU, ya que ningún programa se puede ejecutar si no se le concede el uso de este.

Dado que suele haber bastantes más procesos que procesadores, la forma en la que se reparte el uso del procesador entre los distintos procesos es fundamental, se llama ***planificación de la CPU***.

Mantener un registro de distintas actividades paralelas es una tarea difícil. Para facilitar el uso del paralelismo han desarrollado el ***modelo de procesos***, en el cual todo el software ejecutable del ordenador, incluido el propio sistema operativo, se organiza en torno al concepto de proceso.

CONCEPTO DE PROCESO

Un proceso es básicamente un programa en ejecución. Un programa, en cambio es algo estático, es el contenido de un fichero en disco. Un proceso, en cambio, es algo dinámico que en cada instante tiene un estado concreto, el cual viene determinado por el contador de programa y los registros de la CPU. Un proceso necesita disponer de memoria para almacenar el código y los datos del programa, memoria también para su pila, CPU para poder ejecutarse, espacio en disco para leer y escribir, etc…

Al igual que un proceso puede constar de varios programas, un mismo programa puede dar lugar a varios procesos.

Cuando solo hay una CPU y esta se pasa rápidamente de un proceso a otro, tenemos la sensación de que los procesos se ejecutan al mismo tiempo (*pseudoparalelismo*), cuando no es así, pues en un determinado instante la CPU solo puede estar ejecutando el código de un proceso. El ***paralelismo real***, en cambio, se da cuando hay varias CPU o cuando una CPU dispone de varios núcleos.

Permitir la ejecución concurrente de varios procesos es una tarea compleja. Sin embargo, existen muchas razones que hacen que esto merezca la pena:

- **Compartir recursos físicos**, pues muchas veces los recursos hardware son caros o su disponibilidad en el sistema está limitada, por lo que nos podemos ver obligados a compartirlos en un entorno multiusuario.
- **Compartir recursos lógicos**, como una base de datos o cualquier otro elemento de información.
- **Acelerar los cálculos**, ya que, si queremos acelerar una tarea, podemos dividirla en subtareas para que todas ellas se ejecuten en paralelo.
- **Modularidad**, ya que un sistema se construye con más facilidad si se diseña como un conjunto de procesos separados.
- **Comodidad** para el usuario.

Los procesos nunca deben programarse con hipótesis implícitas de tiempo, al existir varios procesos ejecutándose, no se sabe con certeza cuándo se ejecutará cada proceso ni cuándo se atenderán sus solicitudes. Cuando un proceso deja la CPU hay que decidir qué proceso, pasará mediante un **algoritmo de planificación**.

CAMBIO DE PROCESO, DE CONTEXTO Y DE MODO

El hecho de que la CPU deje de ejecutar un proceso y pase a ejecutar otro se denomina ***cambio de proceso***.

Para cambiar de proceso, el control de la CPU debe pasar al sistema operativo, que guardará el estado o contexto del proceso que estaba usando la CPU, seleccionará un nuevo proceso y restaurará su estado o contexto para que este otro proceso haga uso de la CPU. A estos saltos los llamaremos

cambio de contexto. Un cambio de proceso siempre conlleva uno o más cambios de contexto, pero no al contrario.

Un ***cambio de modo*** es cuando pasa de modo usuario a modo núcleo y viceversa. Un cambio de modo supone siempre un cambio de contexto, pero no un cambio de proceso. En cambio, un cambio de contexto no implica siempre un cambio de modo.

CREACIÓN Y DESTRUCCIÓN DE PROCESOS. JERARQUIA DE PROCESOS

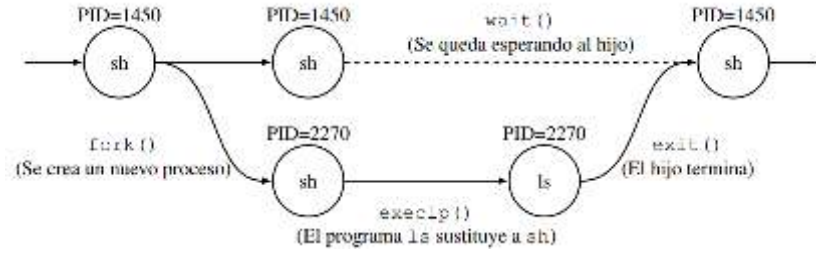
Si es un sistema operativo soporta el concepto de proceso, debe proporcionar llamadas al sistema para crear y destruir proceso. En Unix, existe fork(), que crea una copia idéntica del proceso padre que hace la llamada. Podemos decir qur el proceso hijo es un clon del proceso padre. La única diferencia entre uno y otro es el valor devuelto por la función fork(), que en el hijo es 0 y en el padre es un número que se le llama identificador de proceso (PID) que identifica el proceso hijo.

Después del fork(), el padre e hijo continúan siu ejecución en paralelo de forma totalmente independiente. Dado que el padre y el hijo son idénticos, ambos ejecutarán el mismo código. La forma de conseguir que ambos realicen tareas distintas, a través de otra función como execve() o exec(), que no crea un nuevo proceso, sino que simplemente cambia el código (mutación) y los datos por otros dentro del mismo proceso. A pesar de este cambio, hay elementos que conservan como los ficheros abiertos, el tratamiento de las señales y el PID y otras propiedades del proceso.

En los sistemas Windows, la llamada al sistema para crear procesos y para que el nuevo proceso hijo ejecute un nuevo programa es CreateProcess().

Además, en Windows, un proceso hijo puede pasar a ser hijos de otro proceso que no sea su padre original, a través de transferir la propiedad. Esto hace que en Windows el concepto de jerarquía desaparezca.

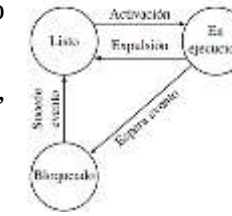
Un proceso puede terminar de forma **voluntaria** o **involuntaria**. La ***finalización voluntaria*** es el propio proceso el que decide finalizar su ejecución, bien porque ha terminado su tarea o bien porque ha encontrado algún error que le impide continuar su ejecución. Para esta finalización voluntaria, en Unix existe exit() y en Windows ExitProcess(). En una ***terminación involuntaria***, un proceso es finalizado por el propio sistema operativo o por otro proceso, siempre que tenga autorización para ello. Para esto en Unix existe kill() y en Windows TerminateProcess().



ESTADOS DE UN PROCESO

Un proceso puede estar en 3 estados posibles:

- En ***ejecución***, el proceso está utilizando la CPU (solo puede haber uno a la vez).
- En ***listo***, el proceso es ejecutable, pero está a la espera de que llegue su turno de CPU.
- En ***bloqueado***, el proceso no se puede ejecutar, aunque se le asigne la CPU, porque se encuentra a la espera de que ocurra algún evento externo.



Lo ideal es un proceso en ejecución y varios en listo y varios en bloqueados.

Entre estos estados, existen 3 posibles transiciones:

- *en ejecución → bloqueado*: cuando un proceso no puede continuar su ejecución por algún motivo debe bloquearse.
- *en ejecución → listo* o *listo → en ejecución*: un proceso pasa de estar en ejecución a estar listo cuando es expulsado de la CPU por haber excedido su tiempo de uso, o cuando se debe ejecutar un proceso más importante. Un proceso listo pasa a ejecutarse cuando la CPU queda libre y, según el planificador, al que le corresponde usar la CPU; también si es más importante.
- *bloqueado → listo*: el proceso ya dsipone de lo que necesitaba. Si la CPU está desocupada, pasará a ejecutarse de forma inmediata.

El diagrama de estados anterior es demasiado sencillo. Cuando un proceso se crea, hace que aparezca el estado *nuevo* en el que un proceso recién creado permanecerá hasta disponer de todos los recursos necesarios.

Y desde que un proceso termina su ejecución hasta que desaparece completamente puede transcurrir un cierto tiempo. Esto da lugar al estado *saliente*.

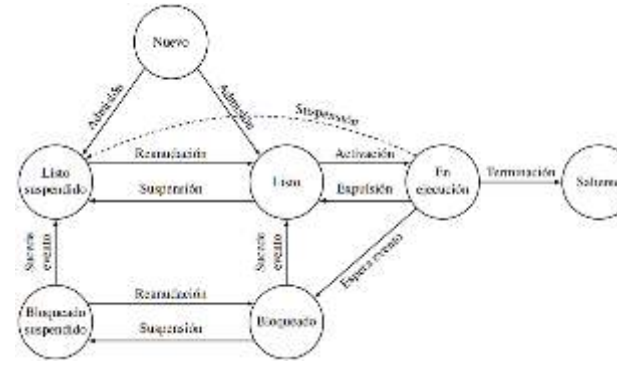
Existe más situaciones que pueden dar lugar a otros estados. Así, un proceso puede ser suspendido durante un tiempo y después ser reanudado por otro proceso. Estas operaciones son útiles por:

- Se pueden suspender procesos activos hasta corregir el problema y luego reanudarlos.
- Si se piensa que los resultados de un proceso son incorrectos, se puede suspender dicho proceso. Si después se comprueba que los resultados son correctos, se puede reanudar.
- Se puede reducir la carga suspendiendo procesos para así poder dar servicio a los procesos de mayor prioridad.

Los procesos suspendidos generalmente se guardan en disco para liberar la memoria principal que ocupan, la cual quedará disponible para otros procesos.

Para añadir estos dos nuevos estados hay 8 posibles transiciones más:

- *bloqueado → bloqueado suspendido*: cuando interesa para liberar memoria y que otros procesos listos pueden usarla y ejecutarse.
- *bloqueado suspendido → listo suspendido*: cuando el evento por el que se bloqueó el proceso se produce mientras este está suspendido.
- *listo suspendido → listo*: cuando no queden procesos listos, que haya quedado memoria libre, o que un proceso listo suspendido deba ejecutarse por tener más propiedad que los procesos que haya en memoria.
- *listo → listo suspendido*: se suspenden ya que están consumiendo memoria.
- *nuevo → listo suspendido*: cuando se crea un proceso, no hay memoria principal disponible y el nuevo proceso no tiene prioridad suficiente como para expulsar a otro de memoria principal.
- *bloqueado suspendido → bloqueado*: si un proceso bloqueado suspendido tiene una prioridad alta, y el sistema operativo sospecha que el evento que espera sucederá en breve, entonces puede tener sentido mover el proceso bloqueado de disco a memoria.
- *En ejecución → listo suspendido*: cuando un proceso de prioridad alta despierta de *bloqueado suspendido* y no hay memoria suficiente si no se pasa inmediatamente a disco.
- *cualquier estado → saliente*: en cualquier momento, un proceso puede finalizar.



IMPLEMENTACIÓN DE PROCESOS

El sistema operativo debe tener conocimiento de qué procesos existen y en qué estado se encuentra cada uno. Para ello, existe una tabla de procesos con una entrada por proceso llamados **PCB** (*Process Control Block*) o **BCP** (*Bloque de Control de Proceso*), y en ella se guarda toda la información relacionada con el proceso correspondiente.

En el PCB se debe guardar, todo aquello que sea necesario para que, después de perder la CPU por cualquier motivo y recuperarla más tarde, el proceso pueda continuar su ejecución.

CREACIÓN DE PROCESOS

Cuando se crea un proceso, al menos, se debe:

1. Dar nombre al proceso (generalmente es el PID).
2. Insertarlo en la tabla de procesos, creando su PCB.
3. Determinar su prioridad inicial.
4. Asignarle recursos iniciales.

Si la creación de procesos se realiza mediante la llamada al sistema `fork()`:

1. Lo que produce el salto al núcleo del sistema operativo.
2. El núcleo busca una entrada libre en la tabla de procesos para el proceso hijo y le asigna un PID.
3. Toda la información del padre se copia a la entrada del hijo con algunas excepciones.
4. Se asigna memoria para los segmentos de datos y de pila del hijo.
5. Se incrementan los contadores asociades a cualquier fichero abierto en un nuevo proceso,
6. Se le asigna al proceso hijo el estado *listo*. Se devuelve el PID del hijo al padre y el valor 0 al hijo.

ESTRUCTURA DE UN PROCESO

Cuando un proceso realiza una llamada al sistema, se considera que ese mismo proceso pasa de modo usuario a modo núcleo y ejecuta el código del sistema operativo que atiende dicha llamada. Por lo tanto, cada proceso tiene dos partes: la que se ejecuta en espacio de usuario, y la que se ejecuta en espacio del núcleo.

Dado que varios procesos pueden estar haciendo llamadas al sistema al mismo tiempo, es posible que varios procesos se encuentren a la vez ejecutando su parte del núcleo, es decir, ejecutando la misma copia de código.

Es posible que una llamada al sistema haga que un proceso se bloquee dentro del núcleo. Cuando se desbloquee, continuará su ejecución dentro del núcleo hasta finalizar y regresar a su parte de usuario.

Otro enfoque, donde el sistema operativo es una colección de procesos de sistema, distintos de los procesos de usuario se corresponde con la estructura de sistemas operativos que sigue el *modelo cliente-servidor*.

CAMBIO DE PROCESO

Un aspecto clave es que el cambio tiene que ser automático y transparente de un proceso a otro. Por seguridad, este cambio solo lo puede hacer el sistema operativo cuando se produce una interrupción (por un error o por un evento de E/S), una llamada al sistema solicitando algún tipo de servicio, o una excepción (un fallo de página pasaría a estado ***bloqueado*** y una división por cero pasaría a estado ***saliente*** porque lo mata).

Los pasos que se dan para cambiar de un proceso a otro son:

1. Al producirse la llamada al sistema, el hardware almacena el contador de programa del proceso que la realiza en la pila. Esta pila es la que utiliza el propio proceso cuando se ejecuta en modo usuario.
2. El hardware pasa a modo núcleo y carga el nuevo contador de programa que debe contener la dirección de inicio de alguna rutina.
3. Se guarda el resto de los registros y otros datos en el PCB del proceso activo que es el que realiza la llamada al sistema. También es posible que se actualice otra información del PCB (estado del proceso, tiempo de uso de CPU.
4. Un procedimiento en lenguaje ensamblador configura la nueva pila que va a utilizar el núcleo mientras se atiende la llamada al sistema. Como puede haber varios procesos, cada uno tendrá una pila para cuando ejecute código en modo núcleo. Suele ser diferente a la que se emplea en modo usuario. Antes de avanzar al paso siguiente, es posible que el procedimiento en ensamblador deba realizar algún que otro trabajo dependiente de la arquitectura.
5. Tras comprobar que la llamada al sistema existe, el procedimiento en ensamblador llama al procedimiento en C que la implementa. Durante la ejecución el proceso se puede bloquear.
6. Tras la ejecución, se vuelve al procedimiento en lenguaje ensamblador. Esta consulta si hay que ejecutar el planificador de procesos y, en caso afirmativo, lo ejecuta. Si el proceso se ha bloqueado es posible que el planificador ya haya sido invocado, por lo que no sea necesario llamarlo de nuevo.
7. El procedimiento en ensamblador carga en los registros del procesador los valores que había en el momento en el que se le quitó la CPU. Entre otras cosas, prepara la pila que el nuevo proceso usará. Si la CPU cambia de proceso, entonces tanto el estado del proceso que sale como el estado del proceso que entra deben cambiar en consonancia. A esta parte del sistema operativo que entrega el control de la CPU al proceso seleccionado por el planificador se le denomina despachador.
8. Se cambia a modo usuario y se regresa de la llamada al sistema, lo que hace que se desapile la dirección de la siguiente instrucción a ejecutar.

Cuando finaliza el tratamiento de la llamada al sistema y se invoca al planificador, este puede decidir darle la CPU al mismo proceso que la tenía. En este caso, no se habrá producido un cambio de proceso sino varios cambios de contexto y de modo.

OPERACIONES CON PROCESOS

Un **sistema operativo** debe poder crear, destruir, suspender, reanudar, bloquear, ejecutar un proceso; además, debe poder cambiar su prioridad, cambiarlo de estado y permitirle que se comunique con otro proceso.

HILOS

Nos referimos a la unidad de planificación y ejecución como ***hilo*** (es una traza de ejecución a través de uno o más programas, un proceso tiene un estado, una prioridad y un contador de programa, y es la entidad que es planificada y ejecutada por el sistema operativo).

El uso más importante del concepto de hilo se da en aquellos casos en los que varios hilos pueden existir dentro de un mismo proceso. Por contra, los procesos tradicionales son procesos con un único hilo.

ELEMENTOS POR HILO Y POR PROCESO

Los hilos tienen cada uno:

- Zona de memoria o espacio de direcciones que contiene la imagen del proceso.
- Acceso protegido al procesador, otros procesos, ficheros y recursos de E/S.
- Variables globales, procesos hijos, alarmas y señales.
- Información contable.
- Un estado.
- Un contexto del procesador con al menos el registro contador de programa y el resto de los registros de la CPU.
- Una pila en ejecución.
- Algún almacenamiento por hilo para las variables locales, puede ser la pila del hilo u otra zona de memoria reservada.
- Acceso a la memoria y recursos del proceso al que pertenece el hilo, compartiendo con todos los restantes hilos del proceso.

Los distintos hilos de un proceso no son tan independientes como procesos diferentes. Todos los hilos tienen el mismo espacio de direcciones de memoria, lo que quiere decir que comparten también las mismas variables globales. Un hilo puede leer, escribir o limpiar de manera completa la pila de otro hilo; no existe protección entre los hilos.

APLICACIONES DE LOS HILOS

Puesto que los hilos comparten un mismo espacio de direcciones, se pueden comunicar entre sí sin intervención del núcleo; es más rápido que entre procesos.

Los hilos se pueden bloquear a la espera de que se termine una llamada al sistema. Esto permite solapar la E/S del proceso con su propio cómputo, ya que unos hilos del proceso pueden estar bloqueados en E/S mientras otros están haciendo uso de la CPU.

Es más rápido crear un nuevo hilo en un proceso existente que crear un nuevo proceso al que hay que asignarle nuevos recursos. También lleva menos tiempo finalizar un hilo, pues no hay que liberar recursos.

Si se dispone de varias CPU, se puede conseguir paralelismo real dentro de un mismo proceso.

Los hilos facilitan la construcción de programas, ya que, si un programa realiza varias funciones diferentes, cada función puede ser desempeñada por un hilo.

HILOS EN MODO USUARIO E HILOS EN MODO NÚCLEO

Hilos implementados en espacio de usuario:

Ventajas

- El núcleo del sistema operativo no sabe que existen. Existe una tabla de hilos privada en cada proceso para realizar los cambios de contexto entre sus hilos.
- Los cambios de contexto entre hilos son mucho más rápidos, ya que no es necesario pasar al núcleo.
- Cada proceso puede tener su propio algoritmo de planificación para los hilos.

Inconvenientes

- Cualquier llamada al sistema bloqueante realizada por uno de los hilos bloqueará a todo el proceso y sus hilos.
- De forma similar, un fallo de página bloquea a todo el proceso y a sus hilos.
- La CPU asignada al proceso tiene que repartirse entre todos sus hilos.
- Si hay varias CPU, no es posible obtener paralelismo.

Hilos implementados en espacio núcleo:

Ventajas
- El núcleo mantiene la tabla de hilos y reparte la CPU entre todos ellos. Si hay varias CPU, varios hilos de un mismo proceso se pueden ejecutar a la vez, consiguiendo paralelismo real.
- Las llamadas al sistema bloqueantes no suponen ningún problema. Si un hilo se bloquea, el núcleo puede dar la CPU a otro hilo del mismo proceso o de otro proceso. Igual que para los fallos de página.

Inconvenientes
- Las funciones usadas para la sincronización entre hilos son llamadas al sistema y son más costosas. Una forma de aliviar este problema es implementar las funciones de biblioteca para la sincronización, y que solo se realice una llamada al sistema cuando sea estrictamente necesario.
- La creación y destrucción de hilos dentro de un proceso es también más costosa. Una posible solución sería la reutilización de estos o de sus estructuras de datos.

La suspensión de un proceso implica intercambiar de memoria a disco el espacio de direcciones del proceso, todos los hilos deben pasar al estado *suspendido*. De igual modo la terminación de un proceso provoca la terminación de un proceso provoca la terminación de todos los hilos de ese proceso.

PLANIFICACIÓN DE PROCESO

La parte del sistema operativo que decide cuál ejecutar se llama *planificador* y el algoritmo que utiliza para tomar la decisión se llama *algoritmo de planificación*.

METAS DE LA PLANIFICACIÓN

El orden en el que un planificador selecciona los procesos listos viene determinado muchas veces por la meta o metas que se pretenden conseguir.

1. **Equidad**: cada proceso obtiene su proporción justa de CPU; no debemos confundir la equidad con la igualdad.
2. **Eficiencia**: mantener la CPU ocupada al 100% o lo más cerca posible.
$$E = \frac{Tiempo\ útil}{Tiempo\ total} \times 100 = \frac{Tiempo\ útil}{Tiempo\ útil + Tiempo\ gestión + Tiempo\ ociosa}$$
3. **Tiempo de espera**: minimizar el tiempo que pasa un proceso en la cola de listos.
4. **Tiempo de respuesta**: este es el tiempo que transcurre desde que se solicita la ejecución de una acción, hasta que se obtienen los primeros resultados de la acción solicitada.
5. **Tiempo de regreso o de retorno**: es el tiempo que transcurre desde que se entrega un trabajo para que sea procesado hasta que se obtienen sus resultados.
6. **Rendimiento o productividad**: maximizar el nº de tareas procesadas por unidad de tiempo.

PLANIFICACIÓN APROPIATIVA Y NO APROPIATIVA

Un planificador puede adoptar dos estrategias de funcionamiento. La ***planificación apropiativa*** consiste, en poder expulsar a procesos ejecutables de la CPU para ejecutar otros procesos (se usa para procesos interactivos). Y la ***planificación no apropiativa***, consiste en permitir a un proceso ejecutarse hasta terminar o hasta bloquearse (se usa para lotes de trabajos).

CICLOS DE RÁFAGAS DE CPU Y E/S

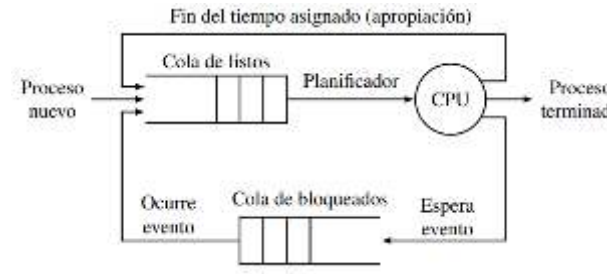
El éxito de la planificación de la CPU depende de la ***ráfaga de CPU*** y la ***ráfaga de E/S***. Los procesos alternan entre estas ráfagas, comenzando y terminando con una ráfaga de CPU.

La duración de las ráfagas de un proceso no se conoce, aunque se pueden hacer estimaciones a partir de ejecuciones anteriores.

Esta alternancia entre ráfagas de CPU y de E/S de los procesos es la que hace posible la existencia de la multiprogramación.

Un proceso limitado por E/S es aquel que pasa la mayor parte de su tiempo en espera de una operación de E/S. Un proceso de este tipo tendrá muchas ráfagas de CPU, aunque breves.

Un proceso limitado por la CPU es aquel que necesita usar la CPU la mayor parte de su tiempo. Un proceso así tendrá pocas ráfagas de CPU de muy larga duración.



ALGORITMOS DE PLANIFICACIÓN

PLANIFICACIÓN PRIEMRO EN LLEGAR, PRIMERO EM SER SERVIDO (FCFS)

El proceso que primero solicita la CPU es el primero al que se le asigna. El algoritmo **FCFS** es no apropiativo, algoritmo es fácil de implementar con una cola FIFO.

En esta política, el tiempo promedio de respuesta puede ser bastante largo/alto.

Un problema de la planificación FCFS es que puede producir el denominado *efecto convoy*. Lo que pasa es que se desaprovechan los recursos. También podemos ver que el nombre *efecto convoy* viene de hecho que hay un proceso grande y pesado (el limitado por CPU), que actúa de locomotora, seguido continuamente por un montón de procesos pequeños y ligeros (los limitados por E/S).

PLANIFICACIÓN PRIMERO EL TRABAJO MÁS CORTO (SFJ)

SFJ es un algoritmo de planificación no apropiativo donde los tiempos de ejecución se aproximan. El algoritmo consiste en coger, de entre todos los trabajos listos, aquel que menos tiempo tarda en ejecutarse en total.

Como no se conoce el tiempo exacto de ejecución, debemos hacer *estimaciones* basándonos en el comportamiento pasado.

$$E_t = a \cdot E_{t-1} + (1 - a) \cdot T_{t-1}$$

donde a es un parámetro llamado coeficiente de credibilidad que oscila entre 0 y 1.

A esta técnica de estimación se conoce como ***maduración***. Un valor de a pequeño hace que se olviden con rapidez los tiempos de las ejecuciones anteriores y un valor de a grande hace que se recuerden durante largo tiempo.

PLANIFICACIÓN PRIMERO EL QUE TENGA EL MENOR TIEMPO RESTANTE (SRTF)

El algoritmo SFJ se puede hacer apropiativo cuando se quita la CPU a un proceso para dársela a otro proceso listo con tiempo total de ejecución o ráfaga de CPU menor que lo que resta del proceso que se está ejecutando.

PLANIFICACIÓN ROUND ROBIN (RR) O CIRCULAR

En esta planificación los procesos se atienden en el orden de llegada a la cola de procesos listos, aunque se asigna a cada proceso un intervalo de tiempo de ejecución llamado quantum. La **planificación round robin** es apropiativa Se cambia de un proceso a otro cuando se consume su *quantum*, termina antes o se queda bloqueado.

Es importante decidir la *longitud del quantum* ya que si el quantum es pequeño puede pasar más tiempo realizando cambios de proceso que ejecutando procesos. Pero si el quantum es grande los últimos procesos de la lista tardan mucho en ser atendidos. Esto puede suponer tiempos de respuesta muy pobres y, por tanto, una mala experiencia de usuario.

PLANIFICACIÓN POR PRIORIDAD

Lo habitual no es que los procesos tengan la misma importancia por eso en la **planificación por prioridad** cada proceso tiene asociada una prioridad y el proceso ejecutable de mayor prioridad es el que toma la CPU.

La asignación de prioridades puede ser estática o dinámica. También hay que decidir si la planificación por prioridad es apropiativa o no, dependiendo de si al llegar un nuevo proceso listo de prioridad mayor, al nuevo proceso se le da la CPU o no.

Un problema serio de los algoritmos de planificación por prioridades es el del *bloqueo indefinido*, que surge cuando los procesos de baja prioridad nunca consiguen el control de la CPU. Para evitar que los procesos de alta prioridad se ejecuten de forma indefinida, el planificador puede disminuir poco a poco la prioridad del proceso en ejecución. Otra posibilidad, que soluciona este problema, es aumentar periódicamente la prioridad de los procesos listos de baja prioridad.

PLANIFICACIÓN DE MÚLTIPLES COLAS CON REALIMENTACIÓN

Es la planificación más general de todas, pero también la más compleja. La idea es tener varias colas de procesos listos. Un proceso irá a una cola u otra en función de ciertos criterios.

A la hora de asignar la CPU a un proceso, hay que decidir de qué cola se selecciona el proceso y qué proceso, dentro de esa cola, hay que coger. Por lo tanto, debe haber una planificación entre colas y también una planificación dentro de cada cola. Debemos permitir que los procesos puedan cambiar de una cola a otra, en este caso habrá que estableces también los criterios para cambiar a un proceso de una cola a otra.

Para está planificación debes tener el numero de colas, el algoritmo de planificación dentro de cada cola o entre estas y el criterio de ascenso y descenso de procesos entre colas.

PLANIFICACIÓN A CORTO, MEDIO Y LARGO PLAZO

Si no se dispone de suficiente memoria, será necesario que algunos de los procesos ejecutables se mantengan en disco. El problema ahora es el coste de darle la CPU a un proceso que está en disco ya que es mayor que el dársela a un proceso que se encuentra en memoria.

Una forma práctica de trabajar con esto es por medio de un planificador de dos niveles: un ***planificador a corto plazo*** (***PCP***), que es el que planifica los procesos que se encuentran en memoria y un ***planificador a medio plazo*** (***PMP***), que es el que planifica el intercambio de procesos entre la memoria principal y el disco.

Periódicamente se llama al PMP para eliminar de memoria procesos que lleven mucho tiempo y para cargar en memoria los procesos que hayan estado en disco demasiado tiempo. Tras esto actúa el PCP hasta que se llame de nuevo al PMP.

Algunos de los criterios que puede utilizar el PMP para tomar decisiones sobre un proceso son:

- Los que llevan tiempo en memoria o en disco pueden ser candidatos para ser intercambiados.
- Puede interesar suspender a disco procesos que consuman mucha CPU. Si el sistema está poco cargado y hay procesos que consumen poca CPU, estos procesos pueden ser suspendidos a disco para poder traer a memoria otros procesos que puedan hacer un mayor uso de la CPU.
- Los procesos pequeños podrían no ser beneficioso el suspenderlos, pero si un proceso pequeñp consume mucha CPU, podría ser necesario suspenderlo.
- Los procesos de mayor prioridad deberían permanecer en memoria, mientras que los procesos de baja prioridad podrían intercambiarse entre disco y memoria.

También es posible un ***planificador a largo plazo*** (***PLP***) que decida, de entre los trabajos por lotes preparados, cuál entra al sistema para su ejecución.

TEMA 3: SEGURIDAD Y PROPTECCIÓN

INTRODUCCIÓN

La ***protección*** tiene carácter interno, ya que es una tarea encargada al sistema operstivo. Y la ***seguridad*** tiene un carácter más general, en el que se incluyen, ademas de la protección, otros aspectos como la politica de copias de seguridad, las autorizaciones de acceso.

SEGURIDAD

REQUISITOS DE SEGURIDAD

En el caso de la seguridad de ordenadores y redes, se definen los siguientes requisitos de seguridad:

- **Confidencialidad**: exige que la información pueda ser leída solo por usuarios autorizados.
- **Integridad**: exige que los elementos puedan ser modificados solo por usuarios autorizados.
- **Disponibilidad**: exige que los elementos estén disponibles solo para usuarios autorizados.

TIPOS DE AMENAZAS

Las amenazas se caracterizan mejor contemplando la función que del sistema como suministrador de información. Estas son cuatro categorías generales de amenazas:

- **Interrupción**: se destruye un elemento del sistema, o se hace inaccesible o inútil (amenaza a la disponibilidad).
- **Intercepción**: una parte no autorizada consigue leer (amenaza a la confidencialidad).
- **Modificación**: una parte no autorizada no solo consigue acceder, sino que también es capaz de falsificar un elemento (amenaza a la integridad).
- **Invención**: una parte no autorizada inserta objetos falsos (amenaza a la integridad).

AMENAZAS SEGÚN LOS ELEMENTOS DE UN SISTEMA DE COMPUTACIÓN

Los elementos de un sistema de computación pueden clasificarse en cuatro categorías: hardware, software, datos y líneas de comunicaciones (redes).

ATAQUES GÉNERICOS A LA SEGURIDAD

La forma más habitual de probar la seguridad de un sistema es contratar a un grupo de expertos para ver si son capaces de penetrar en él de alguna forma. A este grupo se le conoce como *equipo tigre* o *equipo de penetración*.

1. Reserve espacio de memoria, disco o cinta, solo léalo.
2. Intente llamadas al sistema inválidas, o bien llamadas válidas con parámetros inválidos.
3. En determinados sistemas, oprimir DEL, CTRL+C o CTRL+PAUSA, el programa de verificación de contraseñas quedará invalido y el acceso se considerará exitoso.
4. Engañe al usuario escribiendo un programa que muestre el mensaje login.
5. Busque manuales que digan «no lleve a cabo X» e intente tantas variaciones de X como sea posible.
6. Convenza al administrador del sistema para que evite ciertas verificaciones.
7. Encontrar al personal de administración del centro de cálculo y engañarlo o sobornarlo.

ATAQUES ESPECÍFICOS A LA SEGURIDAD

- **Bombas lógicas**: "*estallan*"; en muchas ocasiones, son creadas por el propio desarrollador para protegerse de un posible despido.
- **Puertas traseras (*backdoors*)**: son programas que muchas veces realizan una tarea correcta, pero a través de los cuales es posible el acceso al sistema.
- **Desbordamiento de buffer**: consiste en aprovechar fallos de programación sobrescribiendo la pila de una función para que, cuando la función termine, no regrese al punto desde el que se la invocó, sino a un código cuidadosamente preparado.
- **Caballos de Troya**: sustituyen una orden interna por otra con el mismo nombre, pero que realiza labores ilegales.
- **Virus y gusanos**: programas cuya principal característica es su habilidad para extenderse dentro de un mismo ordenador o entre diferentes ordenadores por medio de sistemas de almacenamiento.
- **Spyware**: programas que se instalan sin que el usuario sea consciente de ello ni lo autorice; se ejecutan en segundo plano para recopilar información.
- **Rootkits**: son programas que han corrompido el sistema de tal forma que no pueden ser fácilmente detectados y permiten a un atacante externo acceder con privilegios.

PRINCIPIOS DE DISEÑO PARA LA SEGURIDAD

Saltzer y Schroeder (1975) crearon una guía para el diseño de sistemas seguros:

1. El diseño de un sistema debe ser público.
2. El estado predefinido debe ser el de «no acceso».
3. Se debe verificar la autorización actual.
4. Los procesos deben tener el mínimo privilegio posible que les permita seguir realizando su trabajo.
5. El mecanismo de protección debe ser simple, uniforme e integrado hasta las capas más bajas del sistema.
6. El esquema elegido debe ser psicológicamente aceptable. Si los usuarios sienten que la protección de sus ficheros implica trabajo o es molesta, simplemente no los protegerán.

PROTECCIÓN

El **hardware** proporciona tres mecanismos que ayudan al sistema operativo a proteger:

- **Hardware de direccionamiento de memoria**: asegura que un proceso únicamente se puede ejecutar dentro de su espacio de direcciones.
- **Cronómetro**: garantiza que ningún proceso podrá obtener el control de la CPU sin renunciar al mismo en algún momento.
- **Modo dual**: asegura que solo el sistema operativo puede realizar las operaciones privilegiadas (E/S).

POLÍTICA Y MECANISMO

La política indica qué operaciones será posible realizar sobre qué elementos y el mecanismo especifica cómo el sistema hará cumplir la política.

Las políticas determinan qué se va a proteger. Unas políticas vendrán determinadas por el propio diseño del sistema, otras serán especificadas por los administradores, mientras que algunas serán definidas por los usuarios.

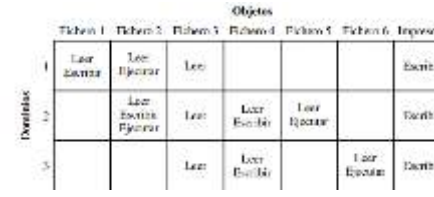
DOMINIOS DE PROTECCIÓN

Teniendo en cuenta los objetos existentes en un sistema y los procesos que pueden acceder a ellos, es necesario disponer de un mecanismo que impida que los procesos accedan a aquellos objetos sobre los que no tienen ningún permiso. Este mecanismo también debe posibilitar que los procesos puedan acceder a los objetos sobre los que tienen permitido el acceso, pero solo para realizar aquel subconjunto de operaciones que les hayan sido autorizadas. Este mecanismo se llama ***dominio de protección***. Un dominio es un *conjunto de parejas* (objeto, derechos).

Cada proceso existente en el sistema se tiene que ejecutar en, al menos, uno de los dominios de protección que haya definidos.

MATRIZ DE ACCESO

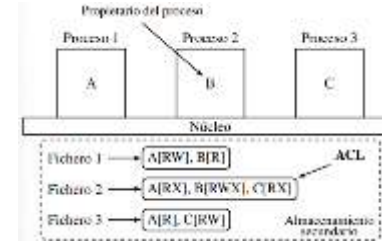
Un aspecto importante a tener en cuenta es la forma en la que el sistema lleva un registro de las parejas (objeto, derechos) que pertenecen a un dominio dado. Teóricamente es una enorme matriz en la que las filas corresponden a los dominios y las columnas a los objetos (matriz de protección o matriz de acceso). Cada celda contendrá los derechos correspondientes a un objeto en un dominio.



En la práctica, rara vez se almacena tal cual la matriz de protección que acabamos de describir; como la mayor parte de los dominios solo tiene acceso a un número reducido de objetos, la matriz estará casi vacía en relación con su tamaño. No obstante, existen dos métodos que guardan solo los elementos no vacíos de la matriz. Estos métodos son las listas de control de acceso y las listas de posibilidades.

LISTAS DE CONTROL DE ACCESO

En las listas de control de acceso, la matriz de protección se almacena por columnas; se asocia a cada objeto una lista con todos los dominios que pueden acceder al objeto y los derechos concretos de cada dominio para dicho acceso. Además de los típicos permisos sobre ficheros, un sistema puede definir permisos adicionales para realizar otro tipo de operaciones (copiar, borrar, ordenar, etc).



Lo más razonable es que cada ACL se almacene en disco junto con su fichero. Sin embargo, las ACL están protegidas por el núcleo del sistema operativo, por lo que no pueden ser directamente manipuladas por los procesos de usuario.

Si los usuarios se pueden agrupar, las listas de control de acceso también se pueden aplicar a grupos. Y un aspecto a tener en cuenta es que si la ACL de un fichero solo se comprueba cuando se abre este.

POSIBILIDADES O CAPACIDADES

Las ***listas de posibilidades*** o ***listas de capacidades*** almacenan una matriz de protección por filas, se trataría de asociar a cada dominio una lista de los objetos a los cuales puede acceder junto con una indicación de las operaciones permitidas sobre cada uno. Ya que son los procesos, los que realmente acceden a los objetos, cada proceso tendrá su propia lista, en la que cada elemento recibe el nombre de capacidad o posibilidad.

Generalmente, cada *posibilidad* (entrada de una lista) contiene el tipo de objeto al que hace referencia la entrada, los derechos y un apuntador al propio objeto.

Aunque las listas de posibilidades se asocian a procesos, deben ser protegidas. Una forma de conseguir esto es manteniendo la lista de posibilidades en el núcleo del sistema operativo.

Además de los derechos dependientes del objeto, las posibilidades suelen tener también derechos genéricos, como:

- **Copiar posibilidad**: crea una nueva posibilidad para el mismo objeto.
- **Eliminar posibilidad**: elimina un dato de la lista.
- **Destruir objeto**: elimina de forma permanente un objeto y una posibilidad.

La primera vez que un proceso accede a un fichero, el sistema operativo consulta la ACL asociada al fichero para saber si el proceso tiene permisos para realizar la operación que desea. Si no tiene permiso, se produce un error. En caso contrario, se crea una posibilidad para dicho fichero en la lista de posibilidades y se devuelve la posición ocupada por la posibilidad.

COMPARACIÓN

Para la protección de un equipo se necesita comparar si los usuarios o procesos tienen acceso a la información o sistema operativo o al proceso, etc…

CANCELACIÓN Y REVOCACIÓN

En un sistema de protección dinámico es necesario a veces cancelar derechos de acceso a objetos que son compartidos por usuarios. ***La cancelación*** o ***la revocación*** se demora, no tiene efecto inmediato.

Con las listas de control de acceso, la *cancelación* es bastante sencilla. Dado un objeto, se busca en su ACL los derechos que serán cancelados y simplemente se eliminan. La cancelación es inmediata y puede ser general o selectiva, total o parcial, o permanente o temporal.

Las posibilidades, sin embargo, presentan un problema de cancelación más difícil. Dado que las posibilidades de un objeto están distribuidas por el sistema, es necesario localizarlas antes de poder cancelarlas. Al sistema le lleva tiempo, ya que pueden estar almacenadas en listas de posibilidades de muchos procesos. Aunque existen varias técnicas que permiten implementar la cancelación de posibilidades.

Una primera técnica consiste en hacer que cada posibilidad apunte hacia un objeto indirecto. Si el objeto indirecto apunta hacia el objeto real, el sistema siempre puede romper esta conexión, invalidando así todas las posibilidades.

Otra técnica es asignar a cada objeto un número aleatorio y almacenarlo también en la posibilidad. De esta forma, cuando se intenta acceder a un objeto a través de su posibilidad, se comparan los dos números y se permite la operación si ambos números coinciden.

Como vemos, ninguna de estas dos técnicas permite ni una revocación selectiva ni parcial.

AUTENTIFICAIÓN DE USUSARIOS

La protección, tal como la hemos descrito, es estrictamente un problema interno del sistema operativo. La seguridad, por otra parte, no solo requiere un adecuado sistema de protección, sino también tener en cuenta el entorno externo donde opera el sistema. Los problemas de seguridad son esencialmente problemas administrativos, no del sistema operativo. El principal problema de seguridad para los sistemas operativos es el de la validación. El problema de la identificación de los usuarios cuando se conectan al sistema se conoce como autenticación de usuarios.

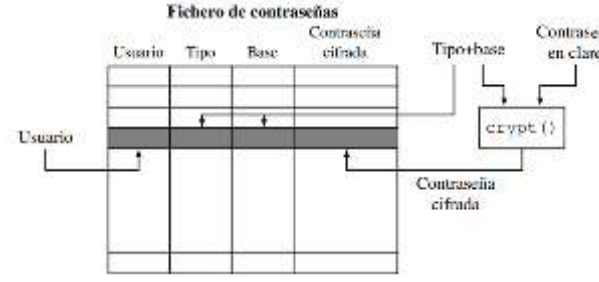
El método más general para realizar la autenticación de usuarios consiste en usar un identificador (ID o usuario) más una contraseña. La contraseña permite autenticar el ID del individuo que se conecta

al sistema. Por su parte, el ID introduce seguridad en dos sentidos; si el usuario está autorizado para acceder al sistema y los privilegios acordados con el usuario.

CONTRASEÑAS EN LINUX

A la hora de almacenar una nueva contraseña para un usuario, se pasan como parámetros a una rutina de cifrado conocida como crypt() la propia la contraseña, un valor «base» y el algoritmo de cifrado a utilizar. Esta rutina devuelve la contraseña cifrada, la cual se almacena en el fichero de contraseñas en la línea asociada al ID del usuario. En concreto, el formato del campo que guarda la contraseña es: $tipo$base$contraseña_cifrada, donde el tipo indica el algoritmo empleado. El valor de la base normalmente está relacionado con el momento en que se asigna la contraseña a un usuario y sirve para impedir que contraseñas iguales se cifren de la misma manera.

Cuando un usuario intenta conectarse a un sistema Unix, debe proporcionar un ID y una contraseña. El sistema operativo utiliza el ID como índice en el fichero de contraseñas para recuperar el tipo, la base y la contraseña cifrada. El tipo y la base, junto con la contraseña introducida por el usuario, se pasan como parámetros a la función crypt(), cuyo resultado se compara con la contraseña cifrada almacenada en el fichero de contraseñas. Si el resultado obtenido es igual al valor almacenado, la contraseña es aceptada. En caso contrario, se produce un error de autenticación.



ESTRATEGIAS DE ELECCIÓN DE CONTRASEÑAS

Las funciones ***hash*** utilizadas para el cifrado de contraseñas están diseñadas de tal forma que no existe la correspondiente función inversa. Además, el diseño de estas funciones evita los ataques por adivinación.

Por desgracia, algunos usuarios, cuando pueden elegir su propia contraseña, eligen una fácilmente adivinable. La solución a este problema es hacer que el sistema rechace cualquier contraseña que no cumpla unos criterios como la longitud que contengan unos mínimos números, símbolos, etc…

Existen varias técnicas básicas para elegir contraseñas:

En la ***instrucción del usuario***, se explica a los usuarios la importancia de seleccionar contraseñas difíciles de adivinar y se les enseña cómo escoger contraseñas seguras. También se recomiendan algunos buenos hábitos. En la actualidad, una buena contraseña está formada por letras mayúsculas y minúsculas, números y signos de puntuación.

En la ***inspección proactiva de contraseñas***, un usuario puede elegir su propia contraseña, pero, si en el momento de la selección el sistema estima que la contraseña es fácilmente adivinable, la rechaza.

Las dos técnicas no son incompatibles, es más, se complementan perfectamente.

PROTECCIÓN EN UNIX

DOMINIOS EN UNIX

En Unix, cada usuario se identifica mediante un número llamado ***identificador de usuario*** o ***UID***. Este número se asocia al login del usuario a través de la entrada correspondiente del fichero /etc/passwd. Además, existen ***grupos de usuarios***, también se identifican mediante un número llamado ***identificador de grupo*** o ***GID***. El fichero /etc/group es el que asocia el nombre del grupo con su número; también almacena la lista de usuarios pertenecientes a cada grupo.

Un aspecto a tener en cuenta es que puede haber varios usuarios definidos con el mismo UID, o grupos con el mismo GID. Por tanto, para el sistema operativo se tratará siempre del mismo usuario o del mismo grupo.

Dos procesos con el mismo par (UID, GID) tendrán acceso al mismo conjunto de objetos por pertenecer al mismo dominio, mientras que procesos con diferentes valores (UID, GID) tendrán acceso a un conjunto distinto de objetos.

Entre todos los UID, el 0 tiene todos los privilegios y está asociado al usuario con nombre root.

LISTAS DE CONTROL DE ACCESO RESTRINGIDAS

Cada fichero y cada dispositivo pertenece a un usuario y a un grupo. Estos ficheros tienen asociadas listas de control de acceso que se califican como restringidas por estar limitadas a tres conjuntos de usuarios (propietario, grupo y otros).

Cada conjunto de usuarios tiene asociados tres posibles permisos: lectura («r»), escritura («w») y ejecución («x»). Estos permisos se suelen mostrar mediante una cadena de 9 letras, 3 para cada conjunto.

Además de estas ***ACL restringidas*** también implementan listas de control de acceso completas para aquellos sistemas de ficheros que las admiten. Gracias a estas listas, es posible especificar con gran detalle los usuarios y grupos concretos que podrán acceder a un fichero y con qué permisos.

Todo lo que un proceso puede hacer sobre un determinado fichero depende del UID y GID a los que pertenece el proceso del UID y GID a los que pertenece el fichero, y de la ACL restringida del fichero. A la hora de comprobar el acceso, se llevan a cabo los siguientes pasos:

1. Si el UID del proceso coincide con el UID del fichero, se consultan los permisos del fichero asociados al usuario propietario del fichero.
2. Si lo anterior no se cumple, entonces se comprueba si el GID del proceso coincide con el GID del fichero. Si coinciden, entonces se consultan los permisos del fichero asociados al grupo propietario del fichero.
3. Si lo anterior tampoco se cumple, entonces el proceso podrá hacer lo que indiquen los permisos asociados al resto de usuarios.

DOMINIO DE UN PROCESO Y CAMBIO DE DOMINIO

Por omisión, cada proceso se ejecuta siempre con el **dominio (UID, GID)** del usuario que lo crea. No obstante, de forma excepcional, los ejecutables pueden poseer en sus atributos los bits ***SETUID*** y ***SETGID***, que pueden cambiar el dominio final al que pertenece un proceso. En Unix, cuando se crea un proceso con `fork()`, el nuevo proceso hereda del padre su dominio. Sin embargo, si un proceso cualquiera realiza la llamada al sistema `execve()` y el ejecutable que se pasa como parámetro tiene activo el bit SETUID, entonces el proceso cambiará su UID cambiando así de dominio de forma efectiva. De igual forma le ocurre al bit SETGID.

Un posible cambio de dominio producido por los bits SETUID y SETGID hace que, en realidad, un proceso tenga dos parejas de identificadores: el usuario/grupo real (UID, GID) y el usuario/grupo efectivo (EUID, EGID).

La orden `passwd` permite a un usuario cambiar su contraseña, pero no la de los demás usuarios. Para saber qué tipo de usuario está cambiando una contraseña, `passwd` utiliza la función `getuid()` que devuelve el UID real. De esta manera, sabe qué contraseñas puede cambiar. Si la función devuelve 0, entonces el usuario root puede cambiar la contraseña de cualquier usuario. En cambio, si la función devuelve un valor distinto de 0, entonces el programa lo está ejecutando un usuario normal que solo podrá cambiar su contraseña.

COMBINACIÓN DE LISTAS DE CONTROL DE ACCESO Y DE POSIBILIDADES

Unix implementa un ***sistema híbrido*** entre *listas de control de acceso* y *listas de posibilidades*. Los ficheros tienen asociada una ACL restringida. Cuando un proceso quiere abrir un fichero, se comprueba la ACL para ver si tiene acceso. En caso afirmativo, se crea una entrada en la tabla de ficheros abiertos y se devuelve un índice a la misma denominado descriptor del fichero. La entrada contiene los permisos para ese proceso sobre ese fichero. De este modo, cuando el proceso quiera realizar una operación sobre el fichero, solo tendrá que pasar el descriptor del fichero como parámetro para tener acceso al mismo. Cuando se cierra el fichero, se elimina la entrada de la tabla.

Aún debe verificarse el derecho en cada acceso, y que la entrada de la tabla de ficheros tiene una posibilidad solo para las operaciones permitidas. Un problema de esta combinación de listas de control de acceso y listas de posibilidades es que la ACL de un fichero solo se comprueba al abrirlo. Esto incumple el principio de diseño que establece que la autorización se tiene que comprobar continuamente. No obstante, este tipo de seguridad un poco más laxa permite a cambio mejorar el rendimiento al trabajar con ficheros.

TEMA 4: SISTEMAS DE FICHEROS

INTRODUCCIÓN

Los sistemas de computación actuales necesitan algún tipo de almacenamiento secundario (discos) para guardar información que complemente al almacenamiento primario (RAM). Se usa para almacenar gran cantidad de información, preservar la información para que esta no desaparezca y permitir que cierta información sea fácilmente compartida y accedida.

El sistema operativo tiene que crear abstracciones que faciliten a los usuarios y a los programadores el uso de este tipo de almacenamiento. La solución es almacenar la información en unidades llamadas ficheros, los cuales se organizan y agrupan en directorios. La estructura de datos a través de la cual los ficheros y directorios se almacenan en el almacenamiento secundario se llama sistema de ficheros.

FICHEROS

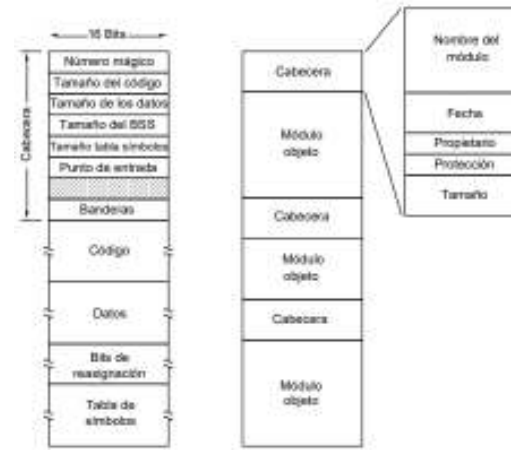
Un **fichero** es la menor unidad lógica de almacenamiento. En Windows, Linux, Mac OS X, etc…, un fichero es una secuencia de bytes cuyo significado está definido por el programa o programas que acceden al mismo. Con esta estructura de fichero se consigue máxima flexibilidad, ya que los programas pueden colocar y organizar información como deseen según una determinada estructura de datos. Cada fichero tiene un nombre que lo identifica. Dependiendo del diseño realizado en el sistema operativo y en el propio sistema de ficheros.

TIPOS DE FICHEROS

Ficheros regulares: contienen información del usuario. Los ***ficheros de texto*** constan de líneas de texto que se pueden ver e imprimir tal cual son y pueden modificarse con un editor de texto. Los ***ficheros binarios*** suelen aparecer como una lista incomprensible de símbolos al ojo humano.

Directorios: son ficheros gestionados por el propio sistema operativo para poder organizar y registrar los ficheros existentes en el sistema de ficheros.

Ficheros especiales de caracteres: están relacionados con la E/S y se utilizan para referenciar y acceder a dispositivos serie de E/S.



Ejemplo Fichero Binario

Ficheros especiales de bloques: son iguales que los anteriores, pero para referenciar y acceder a discos y otros dispositivos de almacenamiento secundario.

Muchas veces el nombre de un fichero contiene una extensión que indica de qué tipo de fichero se trata (texto o binario); hay programas, como un explorador de ficheros, para los que la extensión puede ser importante de cara a determinar qué acción realizar con un fichero. Las extensiones también ayudan a los usuarios a saber, con un simple vistazo, qué tipo de información contiene cada fichero.

ACCESO A UN FICHERO

En el acceso secuencial todos los bytes de un fichero deben leerse en orden, uno detrás de otro, sin posibilidad de saltarse algunos o leerlos en otro orden. En los ficheros de acceso aleatorio, es posible leer los bytes en un orden cualquiera, existiendo llamadas al sistema que permiten el posicionamiento (como lseek).

ATRIBUTOS DE UN FICHERO

Un fichero tiene un nombre y contiene ciertos datos. Además, los sistemas operativos añaden fecha y hora de creación, tamaño actual, tamaño máximo; y a estos elementos adicionales se les llama atributos. Los distintos atributos de un fichero son:

Campo	Significado
Protección	Quién debe tener acceso y de qué forma
Contraseña	Contraseña necesaria para tener acceso al fichero
Creador	Identificador de la persona que creó el fichero
Propietario	Propietario actual
Bandera de solo lectura	0 Lectura/escritura, 1 para lectura exclusivamente
Bandera de ocultación	0 normal, 1 para no exhibirse en listas
Bandera del sistema	0 fichero normal, 1 fichero de sistema
Bandera de biblioteca	0 ya se ha respaldado, 1 necesita respaldo
Bandera texto/binario	0 fichero de texto, 1 fichero binario
Bandera de acceso aleatorio	0 solo acceso secuencial, 1 acceso aleatorio
Bandera temporal	0 normal, 1 eliminar al salir del proceso
Banderas de cerradura	0 no bloqueado, ≠ 0 bloqueado
Tiempo de creación	Fecha y hora de creación del fichero
Tiempo del último acceso	Fecha y hora del último acceso al fichero
Tiempo de la última modificación	Fecha y hora de la última modificación del fichero
Tamaño actual	Número de bytes en el fichero
Tamaño máximo	Tamaño máximo al que puede crecer el fichero

OPERACIONES SOBRE FICHEROS

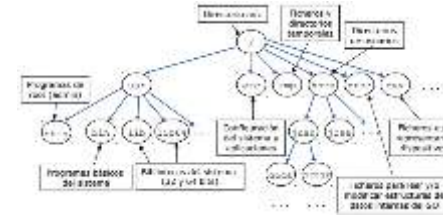
- ***Create***: crear un fichero.
- ***Delete***: borrar un fichero, liberando espacio en disco.
- ***Open***: abre un fichero para su uso.
- ***Close***: cierra un fichero abierto
- ***Read***: lee de un fichero la cantidad de bytes indicada como parámetros.
- ***Write***: escribe en un fichero la información indicada.
- ***Append***: es una forma restringida de write. Solo se puede añadir datos al final del fichero.
- ***Seek***: para los ficheros de acceso aleatorio, nos permite indicar una posición a partir de la cual leer o escribir.
- ***Get attributes***: obtiene los atributos asociados a un fichero.
- ***Set attributes***: nos permite cambiar un atributo con el valor indicado.
- ***Rename***: nos permite cambiar el nombre.
- ***Truncate***: elimina el contenido de un fichero a partir de una posición dada

DIRECTORIOS

Para organizar y llevar un registro de los ficheros, los sistemas operativos utilizan, directorios. Los directorios son ficheros que contienen información sobre otros ficheros; principalmente, contienen el nombre de esos otros ficheros, aunque también pueden incluir información sobre sus atributos. Estos ficheros son gestionados por el propio sistema operativo.

SISTEMAS JERÁRQUICOS DE DIRECTORIOS

Los sistemas operativos actuales utilizan un árbol de directorios para organizar los ficheros. En esta estructura, cada directorio, además de ficheros, puede contener, otros directorios.



NOMBRE DE LA RUTA DE ACCESO

Cada fichero tiene una ***ruta de acceso absoluta***, la cual consta de la ruta de acceso desde el directorio raíz hasta el fichero. Siempre comienzan por " / ".

Cada fichero tiene una ***ruta de acceso relativa*** que se utiliza junto con el concepto de directorio de trabajo o directorio actual. Se construye indicando los directorios que hay que recorrer para llegar desde el directorio actual hasta el fichero o directorio deseado. Además, podemos representar el director actual por ". " y al directorio padre por " .. "

OPERACIONES CON DIRECTORIOS

- ***Create***: crea un directorio vacío. Cuando se refiere a vacío en realidad tiene " . " y " .. ".
- ***Delete***: elimina un directorio vacío.
- ***Opendir***: abre un directorio para ser recorrido.
- ***Closedir***: cierra un directorio abierto.
- ***Readdir***: devuelve la siguiente entrada de un directorio abierto.
- ***Rename***: cambia de nombre.
- ***Link***: permite que un mismo fichero aparezca a la vez con nombres diferentes en un mismo directorio, o en varios directorios con el mismo nombre o nombre distintos.
- ***Unlink***: elimina una entrada del directorio.

Si los ficheros tienen atributos, los directorios también los tendrán, aunque, para algunos atributos, el significado podría cambiar ligeramente.

IMPLEMENTACIÓN DEL SISTEMA DE FICHEROS

En conclusión, el sistema operativo ve los sistemas de ficheros como arrays lineales de bloques y tiene *N* bloques, desde *0* hasta $N-1$, todos del mismo tamaño. Tanto el tamaño como el número total de bloques dependen del dispositivo y su capacidad. Cada bloque tiene también una posición dentro del array que lo llamaremos dirección de disco del bloque.

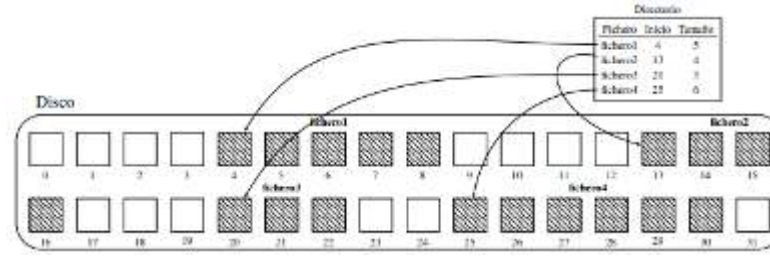
IMPLEMENTACIÓN DE FICHEROS

Todo fichero tiene asociado un conjunto de bloques de disco donde guarda sus datos. Cuando hablamos de implementación de ficheros estamos hablando de una forma de llevar un registro de esos bloques. Puesto que hay varias formas de llevar ese registro, existen varias implementaciones.

ASIGNACIÓN ADYACENTE O CONTIGUA

Es el esquema más sencillo. En esta implementación, cada fichero se almacena como un conjunto de bloques adyacentes en disco. Esta implementación posee dos ventajas. Una es que es de fácil implementación, ya que el registro de la localización se reduce a recordar un solo número. Y la otra ventaja es que esta implementación ofrece un rendimiento excelente, ya que, al estar todos los bloques de un fichero juntos en disco, el recorrido de ese fichero supondrá muy pocos movimientos. Además, es posible leer o escribir varios bloques consecutivos del fichero en una única operación de disco

Sin embargo, presenta algunos inconvenientes. El primero es que no es realizable de manera eficiente en un sistema de ficheros de propósito general si no se sabe qué tamaño máximo va a tener un fichero, no se sabe qué espacio reservarle. Si se reserva un espacio demasiado pequeño puede ocurrir que, si un fichero quiere crecer, no pueda. Otro problema de esta implementación es el de la fragmentación externa, puede ocurrir que el espacio libre esté repartido en huecos pequeños, de tal manera que no se pueda crear un nuevo fichero.

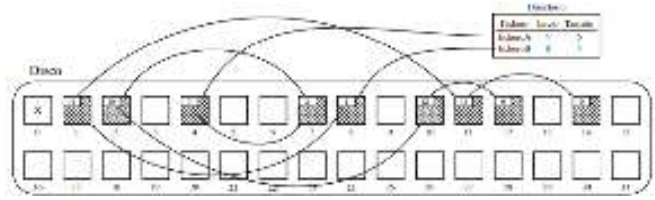


Esta implementación de ficheros es útil en la creación de DVDs o CD ROMs.

ASIGNACIÓN MEDIANTE LISTA LIGADA

En esta implementación cada fichero se mantiene como una lista ligada de bloques en disco. Al principio cada bloque se guarda la dirección de disco del siguiente bloque (un puntero).

Las ventajas de esta implementación son dos. No hay fragmentación externa, ya que se pueden utilizar todos los bloques de disco. Por otro lado, es suficiente que la entrada de directorio correspondiente al fichero guarde solo la dirección en disco del primer bloque. A partir de él se puede acceder a todos los demás.



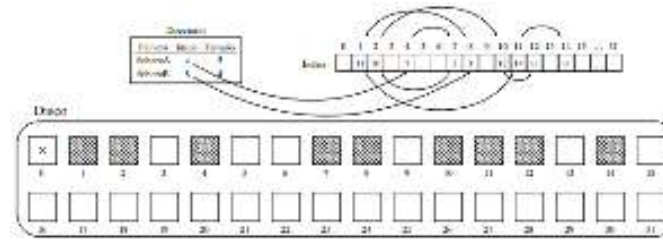
Sin embargo, aunque la lectura secuencial es directa, el acceso aleatorio a un fichero es lento, ya que, tendríamos que leer todos sus bloques. Y el tamaño del espacio para almacenamiento de datos en un bloque ya no es potencia de dos, puesto que el apuntador ocupa algunos bytes.

ASIGNACIÓN MEDIANTE LISTA LIGADA E ÍNDICE

Una forma de eliminar los inconvenientes de la asignación mediante listas ligadas es almacenar los punteros en una tabla o índice en memoria, en esa tabla o índice hay una entrada por cada bloque de disco. La dirección de disco del bloque X de un fichero es D_{X+1} de su bloque $X+1$, se guarda en la entrada D_X de la tabla. La entrada corresponde al último bloque del fichero, almacena un 0, que no se considera un número de bloque válido, por lo que se utiliza para indicar el final de la lista. Con esta técnica ahora todo el bloque está disponible para datos y el acceso aleatorio es mucho más rápido, puesto que, aunque hay que seguir la cadena de punteros, este recorrido se hace en memoria.

La principal desventaja de esta técnica es que toda la tabla debe estar en memoria, lo que puede ser un problema si la tabla es grande. Otro aspecto a tener en cuenta es que la tabla se tiene que guardar en disco para que esté disponible cada vez que se arranca el sistema.

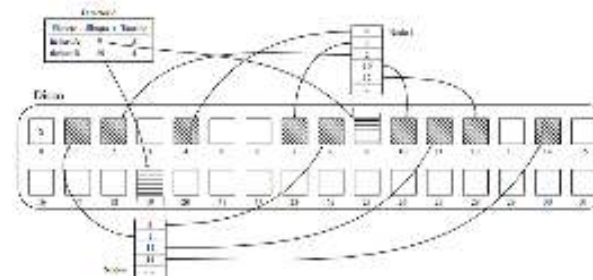
Si hacemos que ciertos números de bloques sean inválidos, entonces podríamos usar la propia tabla y esos valores para indicar qué bloques están libres.



El sistema de ficheros FAT utiliza este método (el sistema de ficheros que usaba MS-DOS y que usa Windows).

NODOS-I

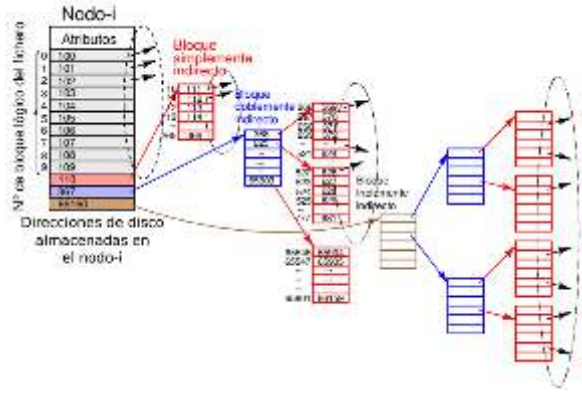
A cada fichero se le asocia una pequeña tabla llamada nodo-i (nodo índice), la cual contiene las direcciones en disco de los bloques del fichero en orden. En la imagen el nido-i de cada fichero se almacena en un bloque de disco; aunque, esta opción es posible no se suele utilizar, lo habitual es utilizar nodos-i pequeños, capaces de almacenar unas pocas direcciones de disco, y guardar varios nodos-i juntos en un mismo bloque de disco.



El problema surge cuando se usa todo un bloque como nodo-i, ¿qué pasa con los ficheros que grandes que tienen más bloques que direcciones caben? La solución es utilizar bloques indirectos, utilizar el nodo-i para almacenar las direcciones de disco de los primeros bloques del fichero correspondiente.

Para ficheros pequeños, las direcciones de todos sus bloques se almacenarán en el nodo-i. Para ficheros un poco más grandes, una de las direcciones en el nodo-i será la dirección de un bloque en el disco, llamado **bloque simplemente indirecto (BSI)**, que contendrá las direcciones en disco de bloques de datos adicionales del fichero. Si lo anterior no es suficiente, otra dirección en el nodo-i será la dirección de un **bloque doblemente indirecto (BDI)**, cuyo contenido serán las direcciones de más bloques simplemente indirectos. Si aún no es suficiente, se puede utilizar un **bloque triplemente indirecto (BTI)**, que almacena bloques doblemente indirectos.

Las ventajas de este método es que se puede calcular que un fichero puede tener hasta 16 843 018 bloques de datos de 1 KiB cada uno, lo que representa un tamaño do poco mas de 16 GiB. Aun así, acceder a cualquier posición del fichero supone, como mucho leer 4 bloques de disco (en el peor de los casos el triplemente indirecto, después el doblemente indirecto, uno simplemente indirecto y por ultimo el bloque de datos). Con el método de la lista ligada con índice necesitaríamos, solo para el fichero, un índice con un tamaño superior a 64 MiB para poder guardar todas las direcciones de bloques de datos. Este índice tendría que estar completamente en memoria.



En los sistemas de ficheros que usan nodos-i de este tipo, todos los nodos-i se guardan juntos en bloques consecutivos de disco a los que se llama tabla de nodos-i. Cada nodo-i recibe un número que es el que se guarda en la entrada de directorio correspondiente a su fichero. Sabiendo el número de nodo-i al que queremos acceder y cuántos nodos-i caben en un bloque, podemos saber qué bloque de la tabla de nodos-i tenemos que leer, y la posición de nuestro nodo-i dentro de ese bloque.

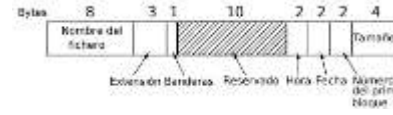
Un nodo-i se lee cuando se abre su fichero correspondiente y solo tiene que estar en memoria mientras el fichero esté abierto. Por lo tanto, la memoria necesaria para almacenar nodos-i es independiente del tamaño del sistema de ficheros y solo depende del número de ficheros abiertos en ese momento.

IMPLEMENTACIÓN DE DIRECTORIOS

La principal función de un directorio es la de asociar el nombre de un fichero con la información necesaria para localizar los datos de dicho fichero (atributos, direcciones de bloques). Toda esta información se almacena en una tabla, dentro de la memoria principal.

DIRECTORIOS MS-DOS

En **MS-DOS**, los directorios son ficheros que almacenan una lista desordenada de entradas de 32 bytes, una por fichero. Cada una de estas entradas se divide en varios campos:



- ***Nombre y extensión del fichero***: los primeros 11 bytes.
- ***Atributos***: un byte donde varios bits se utilizan como banderas.
- Un campo de 10 bytes reservado para ***usos futuros***.
- ***Hora de la última modificación del fichero***. Observa que este campo tiene 16 bits; 5 bits para la hora, 6 bits para los minutos y 5 bits para guardar los segundos pares.
- ***Fecha de la última modificación del fichero***. Este campo tiene un tamaño de 16 bits; 7 bit para almacenar el año (el valor 0 al año 1980), 4 bits que almacena el mes y los últimos 5 bits almacenan el día.
- ***Número del primer bloque***, es decir, dirección de disco donde comienza.
- ***Tamaño del fichero***, en bytes, Al ser este campo de 4 bytes, ningún fichero podrá ser más grande de 4 GiB.

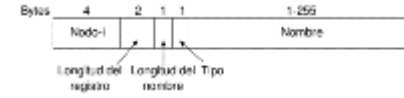
El directorio raíz es una excepción, ya que ocupa unos bloques fijos en disco en lugar de implementarse como un fichero, lo que hace que tenga un tamaño máximo preestablecido.

MS-DOS permite crear un árbol de directorios de tamaño arbitrario (un directorio puede tener subdirectorios). Y un bit del campo de atributos permite distinguir a los ficheros normales de los ficheros que son directorios. Esta estructura es utilizada, por los sistemas de ficheros FAT32 y vFAT.

DIRECTORIOS EN UNIX

En Unix, los directorios son ficheros gestionados por el propio sistema de ficheros. También es posible crear un árbol de directorios, ya que uno de los bits de los atributos almacenados en los nodos-i permite distinguir a un fichero normal de un directorio. En Unix, el directorio raíz no tiene ningún tratamiento especial y es un fichero como cualquier otro.

En las primeras versiones de UNIX, cada entrada tenía un tamaño fijo de 16 bytes, dos de los cuales se utilizaban para guardar el número de nodo-i y los otros 14 bytes almacenaban el nombre. Posteriormente, se han creado sistemas de ficheros para Unix que soportan nombres de hasta 255 caracteres, las entradas ya no tienen un tamaño fijo y se estructuran en varios campos:



- Un ***número de nodo-i***, que ocupa los primeros 4 bytes.
- La ***longitud del registro***, que indica el tamaño de toda la entrada en bytes. Es habitual que esta longitud sea múltiplo de 4.
- La ***longitud del nombre*** un único byte.
- El ***tipo de la entrada*** (directorio, fichero regular o cualquier otra cosa). Este campo es una copia del que hay almacenado en el nodo-i y se utiliza para acelerar.
- El ***nombre del fichero***.

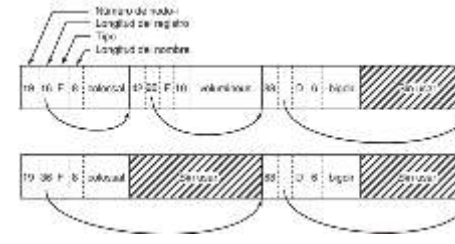
Una entrada fundamentalmente contiene el nombre de un fichero y su número de nodo-i. Toda la información relativa al tipo, tamaño, tiempos, propiedad y bloques en disco del fichero está contenida en su nodo-i.

Necesitamos el campo con la longitud del registro, porque, nos permite tener registros más grandes. Hay varias razones para querer poder hacer esto.

Si renombramos una entrada y usamos un nombre más corto, los bytes que sobren en el campo del nombre seguirán formando parte del registro y no se generará un hueco que no se podrá aprovechar para otro registro por ser excesivamente pequeño. Además, si el tamaño del registro es múltiplo de 4 será frecuente que en el campo del nombre siempre haya algún byte libre. Esto nos permitiría renombrar una entrada a un nombre con uno, dos y hasta tres caracteres más. Es decir, suele ser posible renombrar entradas sin tener que cambiar el tamaño del registro y, por tanto, sin tener que borrar, crear y copiar registros.

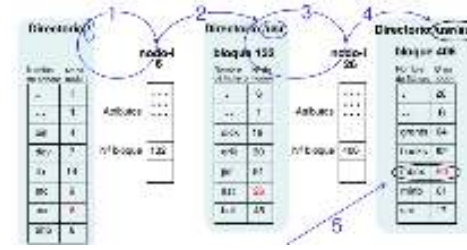
Otra razón es la gestión del espacio libre cuando se crea un directorio, el sistema de ficheros buscará un registro con suficiente espacio libre para crear un nuevo registro, dividiendo el registro existente en dos: uno para la entrada que ya existe y otro para la nueva entrada que, además, se quedará con el resto del espacio disponible. Si no hay ningún registro con espacio libre suficiente, entonces se añade un nuevo bloque de datos al directorio.

Un aspecto a tener en cuenta es que un registro solo puede estar en un bloque, es decir, un registro no puede atravesar la frontera entre bloques. El campo con la longitud del registro también facilita el borrado de entradas. Basta con añadir el espacio que queda libre al registro que hay justo antes. Si no hay ningún registro previo, entonces el registro se marca como libre utilizando un número de nodo-i inválido (habitualmente, 0).



RESOLUCIÓN DE RUTAS **Ejemplo:** /usr/ast/mbox

1. Buscamos en el bloque de datos del directorio raíz la entrada correspondiente a /usr. Una vez localizada la entrada, de ella obtenemos su número de nodo-i.
2. Del nodo-i obtenemos la dirección del bloque de datos del directorio.
3. Ahora, en el bloque de datos de /usr, buscamos la entrada ast, esta entrada nos dice su nodo-i y leemos el nodo-i de disco.
4. Del nodo-i obtenemos la dirección del bloque de datos del directorio /usr/ast.
5. Finalmente, en el bloque de datos de /usr/ast, buscamos la entrada mbox, la cual nos dice cuál es el nodo-i del fichero.



Puesto que las resoluciones de rutas son algo muy frecuente, para acelerar su procesamiento el sistema operativo suele mantener en memoria información de las últimas resoluciones realizadas (*dentry cache* o *caché de entradas de directorio*).

Los nombres relativos se buscan de la misma manera, pero comenzando en el directorio de trabajo. Los directorios «.» y «..», se buscan como cualquier otro nombre, ya que existen entradas reales.

FICHEROS COMPARTIDOS

Cuando un mismo fichero puede aparecer en varios directorios, con el mismo o distinto nombre, o en un mismo directorio con nombres distintos, se dice que es un fichero compartido. La conexión entre un directorio y un fichero compartido se llama enlace.

Los ficheros compartidos se pueden implementar básicamente de dos formas. ***Enlaces físicos*** (*hard link*) los datos relativos a un fichero/atributos se guarden en una estructura de datos y las entradas de los directorios contengan apuntadores a esa estructura. En Unix, varias entradas de un mismo directorio o de directorios distintos guarden un mismo número de nodo-i. Es necesario que exista un contador de enlaces en el nodo-i, si se elimina un enlace borrando un fichero y hay más enlaces, el nodo-i no se debe liberar.

Enlaces simbólicos (*soft links*) es un nuevo tipo de fichero cuyo contenido sea la ruta de acceso del fichero al que se enlaza. Estos ficheros especiales se distinguen del resto de ficheros mediante un bit de bandera. Cuando el fichero original se borra, el uso posterior del fichero a través de un enlace simbólico simplemente fallará, ya que el enlace apuntará a un fichero con el mismo nombre, el enlace simbólico volverá a estar operativo.

El problema es su elevado coste, tanto temporal, como de espacio (hace falta un nodo-i para cada enlace). Sin embargo, son más flexibles que los enlaces físicos. Un problema de todos los enlaces es el riesgo de duplicar datos.

ADMINISTRACIÓN DEL ESPACIO EN DISCO

TAMAÑO DE BLOQUE LÓGICO

Los discos son dispositivos de bloques en los que la unidad mínima de lectura y escritura es el bloque. Estos bloques físicos (sectores), suelen tener tamaños de 512, 1024, 2048 y 4096 bytes. Para facilitar el uso de los discos, los propios dispositivos o el sistema operativo, hacen que estos se vean como un array lineal de bloques físicos.

Es común que el sistema operativo agrupe los sectores para formar bloques lógicos. Si el bloque lógico es grande un fichero constara de pocos bloques, pero puede aparecer una fragmentación interna, lo que puede dar lugar a un gran desperdicio de espacio. Si el bloque lógico es pequeño, se producirá poca fragmentación interna, pero cada fichero ocupará muchos bloques.

El tamaño de bloque lógico también afecta al rendimiento, pero duplicar el tamaño no duplica el tiempo de E/S solo lo aumenta un poco. Por lo tanto, a la hora de elegir un tamaño de bloque lógico, hay que buscar un equilibrio razonable entre la eficiencia en el uso del espacio en disco y la tasa de transferencia en las operaciones de disco.

REGISTRO DE BLOQUES LIBRES

En la lista ligada de bloques, cada bloque de la lista contiene tantos números de bloques libres como pueda, además de un puntero al siguiente bloque. Los bloques de la lista son bloques libres podrán ser usados en el momento en el que dejen de contener información útil.

En el ***mapa de bits***, un disco con *N* bloques necesita un mapa de bits con *N* bits. Los bloques libres se representan con el valor 0 en el mapa y los bloques asignados con el valor 1. El mapa de bits debe guardarse también en bloque de disco, estos bloques nunca se convertirán en bloques libres, pues el mapa de bits siempre tiene el mismo tamaño al depender del tamaño del disco y no del número de bloques libres.

De las dos opciones, la más utilizada en la práctica es el mapa de bits. Linux o Windows usan esta técnica. Un motivo es que el mapa de bits suele ocupar bastante menos espacio. Solo si el disco está casi lleno, el mapa de bits ocupará más que la lista ligada. Otro motivo es que el mapa de bits permite buscar de forma sencilla grupos de bloques libres consecutivos en disco. Basta con buscar una secuencia de bits a 0 lo suficientemente larga. Esto es importante de cara al rendimiento, ya que es más eficiente leer o escribir un fichero que tiene todos sus bloques consecutivos. Para conseguir lo mismo con la lista ligada, los bloques tendrían que estar ordenados por dirección de disco, lo cual puede ser bastante costoso.

CACHE DE DISCO

El acceso a un disco SSD es bastante más lento que el acceso a memoria principal. Para ello, existe la cache de disco, que trata de reducir los accesos a disco necesarios. Esta caché es implementada por el sistema operativo utilizando una parte de la memoria principal y contiene bloques que, pertenecen al disco, pero que se mantienen temporalmente en la memoria principal por razones de rendimiento.

El funcionamiento es similar al de cualquier otra caché. Cuando se lee un bloque, se comprueba si está o no ya en memoria principal. En caso afirmativo, se satisface la solicitud sin acceder al disco; si no, se lee el bloque del disco, se coloca en caché y después se procesa la solicitud de lectura. En este segundo caso la caché puede estar totalmente ocupada, por lo que habrá que eliminar algún bloque (escribiéndolo en disco si ha sido modificado) para hacer hueco.

A la hora de decidir qué bloque expulsar de una caché llena, podemos usar un **algoritmo LRU** con listas ligadas, ya que las referencias a caché son mucho menos frecuentes, pero no es recomendable si queremos mantener la consistencia del sistema de ficheros ante los posibles fallos del sistema.

Si el bloque acaba de ser modificado, se encontrará en la cabeza de la lista LRU, por lo que podrá transcurrir mucho tiempo antes de que sea escrito en disco. Cuanto más tiempo pase sin que un bloque modificado se escriba en disco, mayor será la probabilidad de que se produzcan inconsistencias por caídas del sistema. Además, hay bloques, como los bloques doblemente indirectos, que rara vez tienen dos referencias en un intervalo corto de tiempo, por lo que puede interesar colocarlos en la cola de la lista para que sean los primeros en salir de caché. Las dos consideraciones anteriores dan lugar a un LRU modificado que tiene en cuenta que los bloques recién usados que probablemente no se vuelvan a utilizar pronto pasarán directamente al final de la lista LRU para que sus buffers se reutilicen con rapidez. Si un bloque es esencial para la consistencia del sistema de ficheros y ha sido modificado, debe escribirse en disco lo antes posible, sin importar en qué lugar de la lista LRU se encuentre, se escribe en disco sin esperar a que sea expulsado.

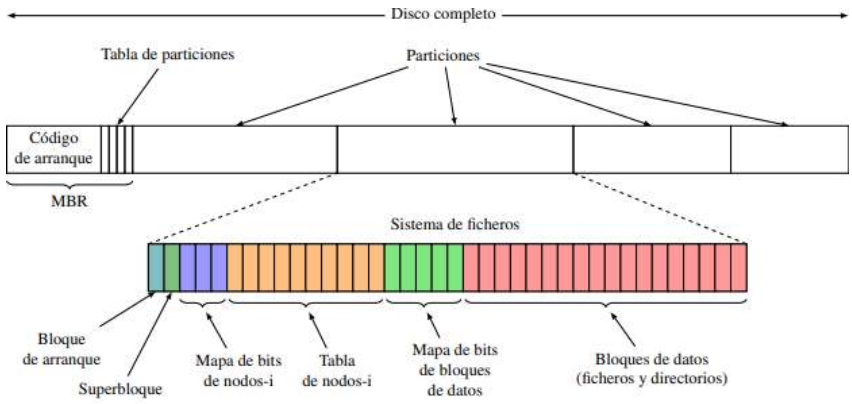
Tampoco es recomendable tener mucho tiempo los bloques de datos en caché, por el riesgo de perder información. En Unix, cualquier bloque de datos modificado se escribe en disco a los 30 segundos o antes. En cambio, los bloques de metadatos se escriben en disco a los 5 segundos o antes.

DISCOS Y SISTEMAS DE FICHEROS

Lo habitual es que un disco contenga varios sistemas de ficheros.

PARTICIONES

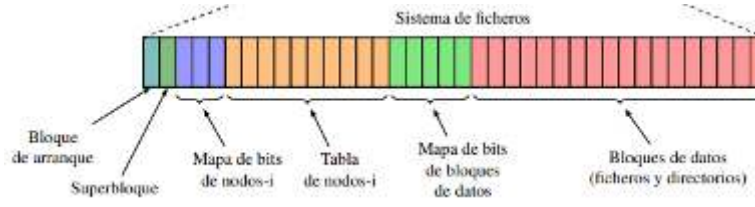
Para facilitar el uso de los discos, los sistemas operativos permiten crear particiones. Una partición es una porción de bloques consecutivos de un disco. Las particiones son manejadas por el sistema operativo que las representa también como un array lineal de bloques. Este array tendrá un bloque 0 y un bloque $N-1$, donde ahora N dependerá del tamaño de la partición en bloques lógicos.



El número y tamaño de las particiones existentes en un disco, junto con los bloques de inicio y fin de cada una, se ha guardado tradicionalmente en la tabla de particiones, la cual, se almacena en el bloque 0 del disco (**MBR**). El MBR también suele contener una pequeña porción de código de arranque que la BIOS carga en memoria y ejecuta para terminar arrancando un sistema operativo.

ESTRUCTURA DE UN SISTEMA DE FICHEROS

El sistema de ficheros usa zonas o grupos de bloques consecutivos de disco para almacenar distintas estructuras de datos. Estas estructuras de datos, su organización en disco y su contenido inicial se crean y escriben en disco cuando se formatea la partición que va a contener el sistema de ficheros.



El bloque de arranque ocupa el bloque 0 de la partición. Este bloque generalmente no se usa, pero puede contener código de arranque del sistema operativo que se encuentra en la partición. Ya que generalmente este bloque no es usado por el sistema de ficheros, se puede usar su dirección (0) como valor nulo o no válido.

El bloque 1 es el ***superbloque***. Contiene información crítica relativa a la organización del sistema de ficheros: número total de bloques lógicos, número total de nodos-i, tamaño de los mapas de bits de bloques y nodos-i, etc. La destrucción del superbloque provocaría que el sistema de ficheros quedara ilegible.

El ***mapa de bits de bloque***, cada bit indica si el bloque correspondiente está libre u ocupado. Su tamaño depende del tamaño de la zona dedicada a ficheros y directorios. Si esta zona tiene B bloques total, de tamaño T_B bytes cada uno, entonces el mapa de bits ocupa $\left\lceil \frac{B}{8 \cdot T_B} \right\rceil$ bloques.

El ***mapa de bits de nodos-i*** se utiliza para saber qué nodos-i hay libres y cuáles están ocupados. Su tamaño depende del total de nodos-i. Si tenemos I nodos-i y un tamaño de bloque de T_B bytes, entonces este mapa necesita $\left\lceil \frac{I}{8 \cdot T_B} \right\rceil$ bloques.

La ***tabla de nodos-i*** contiene los nodos-i usados para ficheros. El tamaño de la tabla de nodos-i depende del tamaño de nodo-i y del total de nodos-i en el sistema. Si tenemos I nodos-i de tamaño T_I bytes cada uno, y bloques de tamaño T_B bytes, esta tabla ocupa $\left\lceil \frac{I \cdot T_I}{T_B} \right\rceil$ bloques.

El ***resto de los bloques*** se usa para almacenar en ellos los bloques de datos de los ficheros regulares, los directorios y los bloques indirectos.