UD 02_01 – Sistemas Operativos. Introducción.

CONTENIDO

1 CONCEPTO DE SISTEMA OPERATIVO

Se puede definir un **sistema operativo** desde tres puntos de vista, según la visión del usuario, como máquina extendida y como gestor de recursos.

- Según la **visión del usuario,** un sistema operativo es el programa que actúa como interfaz entre los seres humanos y el ordenador.

  El sistema operativo sería una capa compleja entre el hardware y el usuario, concebible también como una máquina virtual, que facilita al usuario o al programador las herramientas e interfaces adecuadas para realizar sus tareas informáticas, abstrayéndole de los complicados procesos necesarios para llevarlas a cabo. Por ejemplo, un usuario normal simplemente abre los ficheros grabados en un disco, sin preocuparse por la disposición de los bits en el medio físico, los tiempos de espera del motor del disco, la posición de un cabezal, el acceso de otros usuarios, etc.

- Como **máquina extendida**, el sistema operativo debe ocultar la complejidad del hardware proporcionando una abstracción de mayor nivel, es decir, el sistema operativo debe hacer transparente al usuario las características hardware concretas de los dispositivos.

- Como **gestor de recursos**, el sistema operativo establece la política que determina a quién, cuándo, cuánto tiempo y la cantidad de recursos que asigna. El sistema operativo también debe colaborar para que el ordenador se utilice eficientemente, el caso más evidente es la capacidad de que varios procesos se alternen en el uso del procesador, ofreciendo a los usuarios la sensación de ejecución paralela.

En definitiva, el sistema operativo es el que realiza todo el trabajo dentro del equipo. El usuario utiliza el hardware, pero se despreocupa de gestionarlo o administrarlo. Gracias a una interfaz sencilla (medio de comunicación entre usuario y equipo), proporciona al usuario una comunicación directa, sin que éste tenga que preocuparse de la gestión de memoria, del procesador o de cualquier otro recurso o componente hardware. También sirve para que el usuario utilice software de aplicaciones y éste se despreocupe de la posición de memoria en la que se almacena, por ejemplo, el texto que está tecleando, o simplemente de qué forma se reproduce una pista musical de un CD-ROM.

Resumiendo, un sistema operativo es un conjunto de programas de control cuya finalidad es hacer más fácil y eficiente el uso del ordenador en que se ejecuta.

2 EVOLUCIÓN HISTÓRICA

Los primeros sistemas operativos se denominaron monolíticos. La característica fundamental de estos sistemas operativos es que eran un software básico prácticamente imposible de modificar una vez creado e instalado en un sistema informático. Cuando los diseñadores del sistema operativo, o los usuarios por necesidades específicas, querían introducir modificaciones en él, la labor era muy complicada, ya que se tenía que reconfigurar todo el sistema operativo.

Para ver cómo han evolucionado los sistemas operativos a lo largo de la historia, tenemos que tener presente la evolución del hardware sobre el que se instalan. Hardware y sistema operativo evolucionan uno al lado del otro y nunca por separado. Si se diseñan sistemas operativos más potentes es debido a que el hardware sobre el que van a funcionar también lo es, y a la inversa, se diseña hardware más potente y rápido debido a que los sistemas operativos necesitan cada vez mayores prestaciones hardware.

Históricamente se ha hablado de cuatro generaciones de los computadores, que revisamos a continuación.

Primera generación (1945 – 1955)

- Se utilizaban las válvulas de vacío (antiguas resistencias electrónicas).
- Se trataba de máquinas de gran tamaño, elevado consumo y lentas.
- Un grupo reducido de gente diseñaba, construía, programaba y mantenía cada máquina.
- Las instrucciones se codificaban a mano.
- El tipo de operación era serie, es decir, se trataba un trabajo detrás de otro. El programador insertaba su trabajo y esperaba su turno.
- Existía un desaprovechamiento inadecuado del computador ("tiempos muertos de la CPU").
- La introducción de datos se realizaba pinchando clavijas en un panel. En 1950 aparecen las tarjetas perforadas.

Segunda generación (1955 – 1965)

- Aparición de los transistores.
- Las computadoras son máquinas más pequeñas, más baratas y de menor consumo.
- Se especializa el personal (diseñadores, analistas...).

- Se desarrollan los primeros sistemas operativos.
- Se procesan los trabajos por lotes ("batch").
    - Se cargaba un hardware (cinta, tarjeta, etc.) con la información a procesar.
    - Se llevaba el soporte a la computadora para ser procesado.
    - Otro soporte con los resultados se llevaba a otro dispositivo para analizar los resultados.
- El S.O. se entiende como un programa de control que planifica los trabajos.
- A principios de los años 60 aparece la multiprogramación, que permite aprovechar los tiempos muertos de CPU.

Tercera generación (1965-1980)

- Aparecen los circuitos integrados.
- Existen muchas computadoras diferentes con sistemas operativos muy diferentes.
- Se suministran una gran cantidad de programas de utilidad.
- Esta generación está marcada por la aparición del IBM 360.
- Multiprocesamiento (sistemas compuestos de varios procesadores).
- Surgimiento de la Ingeniería del Software.
- Separación entre la venta de hardware y software.
- Inconvenientes:
    - El lenguaje de control es complejo.
    - Se consumen gran cantidad de recursos: espacio de memoria y tiempo de CPU.

Cuarta generación (1980-2000)

- Mayor integración y miniaturización de componentes electrónicos.
- Aparecen las memorias de semiconductores.
- Relanzamiento de los ordenadores personales.
- Crecimiento del software para las computadoras personales.
- Se crean sistemas operativos amigables, con gestores de ventanas.

- Crecimiento de las redes de computadoras que ejecutan sistemas operativos en red y distribuidos.
- Los sistemas de seguridad adquieren gran importancia.

Se podría hablar de una quinta generación (2000-...)

- Aparición de las computadoras de bolsillo (PDA, Personal Digital Assistant) como ampliación de las agendas electrónicas.
- Los sistemas operativos se adaptan a estas nuevas computadoras y teléfonos móviles (mucho más pequeños que una computadora).

3 CLASIFICACIÓN

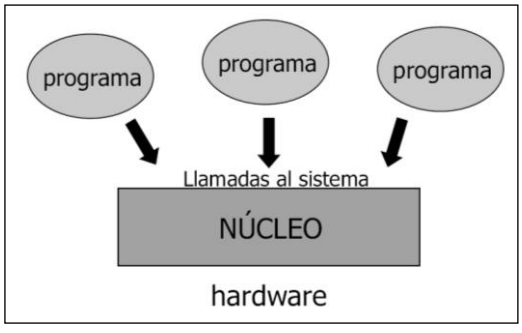
Es posible clasificar los sistemas operativos desde el punto de vista de su **estructura interna** o **arquitectura**, o desde el punto de vista de los **servicios** que ofrecen o la forma en que se pueden **explotar**.

3.1. Según su estructura o arquitectura

A lo largo de su historia, los sistemas operativos, han adoptado principalmente alguna de las siguientes estructuras, bien como opción única o como alguna configuración mixta: estructura monolítica, estructura en capas y micronúcleo cliente/servidor.

3.1.1. Monolítica

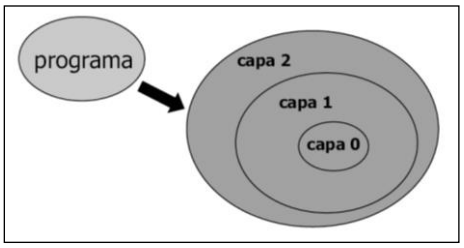
Es la arquitectura más simple para un sistema operativo. Un núcleo monolítico es aquél que incluye prácticamente toda la funcionalidad del sistema operativo en un gran bloque de código que ejecuta como un único proceso. Todos los componentes funcionales del núcleo tienen acceso a todas las estructuras internas de datos y rutinas.



Estos sistemas tienen la ventaja de ser muy rápidos en su ejecución, sólo hay que ejecutar un programa, pero cuentan con el inconveniente de carecer de la flexibilidad suficiente para soportar diferentes ambientes de trabajo o tipos de aplicaciones. Es por esto que estos sistemas operativos suelen ser hechos a medida, para solucionar un problema en concreto.

3.1.2. Por capas o jerárquico

El sistema operativo se construye en niveles jerárquicos, cada uno de los cuales aprovecha los servicios del nivel inferior. El diseño es más modular y escalable que el monolítico. La modularidad permite que la depuración y el mantenimiento del sistema sean más simples.



Se trata de la idea de construir sobre el hardware niveles virtuales superpuestos hasta llegar al nivel del usuario final. La organización más común de los sistemas actuales responde a la siguiente jerarquía:

- Capa 0: gestión del hardware del sistema informático.
- Capa 1: planificación del uso de la CPU.
- Capa 2: administración de la memoria.
- Capa 3: gestión de la E/S y de archivos.
- Capa 4: interfaz de llamadas al sistema.
- Capa 5: programas de usuario.

El concepto de **capa** o **anillo** está totalmente relacionado con el de privilegio de ejecución. Cuanto más cerca del nivel 0 se encuentre una capa, mayor privilegio asociado tendrán los procesos que en ella se ejecutan, así el máximo nivel corresponde a la capa 0, que teóricamente debería de ser la única capaz de dialogar directamente con el hardware del sistema.

3.1.3. Cliente-Servidor

Según este modelo, el sistema operativo se organiza como un conjunto de módulos autónomos, cada uno de los cuales pone a disposición del resto una serie de servicios. Cada módulo actúa como un servidor de ciertas funcionalidades, que atiende las peticiones de otros módulos y que su vez puede ser cliente de otros módulos.

Podemos extender el modelo cliente-servidor hasta el infinito, si consideramos cada módulo del sistema como un conjunto de módulos con relaciones cliente-servidor. El modelo jerárquico no es más que un caso particular del modelo cliente-servidor. Está indicado para sistemas distribuidos.

3.2. Según la forma de explotación

Esta clasificación es la más usada y conocida desde el punto de vista del usuario final. Es posible clasificar las distintas formas en que se puede explotar un sistema a partir del número de usuarios que pueden utilizarlo simultáneamente, de la cantidad de procesos que se pretende ejecutar en un mismo instante y del número de microprocesadores que se pueden utilizar de forma conjunta.

Clasificación de los sistemas operativos	
Número de usuarios	Monousuario
	Multiusuario
Número de tareas	Monoprogramación o monotarea
	Multiprogramación o multitarea
Número de procesadores	Monoprocesador
	Multiprocesador
Estructura	Monolítica
	Por capas
	Cliente/Servidor

3.2.1. Según el número de usuarios

Dependiendo del número de usuarios que pueden explotar simultáneamente un sistema, éste puede ser:

3.2.1.1 Monousuario

Los sistemas operativos monousuario son aquellos que soportan a un usuario a la vez, sin importar el número de procesadores que tenga la computadora o el número de procesos o tareas que el usuario pueda ejecutar en un mismo instante de tiempo. Todos los recursos del sistema estarán disponibles para ese usuario en exclusiva. Las computadoras personales típicamente se han clasificado en esta sección.

3.2.1.2 Multiusuario

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

Actualmente este tipo de sistemas se emplean especialmente en redes, pero los primeros ejemplos de sistemas multiusuario fueron sistemas centralizados que se compartían a través del uso de múltiples dispositivos de interfaz humana (por ejemplo, una unidad central y múltiples pantallas y teclados (terminales tontos)).

Un sistema operativo sólo es multiusuario, en sentido estricto, si permite que los procesos de varios usuarios accedan concurrentemente a la misma CPU compartiendo su espacio de direccionamiento en memoria. En sentido habitual se suele entender por sistema operativo multiusuario aquel que permite el acceso de diferentes usuarios a ciertos recursos, como el disco o las impresoras, conectados a un mismo sistema.

En la familia de los sistemas operativos Microsoft Windows, desde Windows 95 se provee soporte para ambientes personalizados por usuario, pero no admiten la ejecución de múltiples sesiones de usuario al mismo tiempo, haciendo necesaria la finalización de una sesión para poder iniciar otra distinta (y por lo tanto son, en esencia, sistemas operativos monousuario).

3.2.2. Según el número de tareas

Dependiendo del número de procesos o tareas que se pueden ejecutar de forma simultánea en el sistema, se distinguen los siguientes tipos:

3.2.2.1 Monotarea

Los sistemas monotarea son aquellos que sólo permiten una tarea a la vez por usuario. Es decir, sólo se puede ejecutar un proceso y hasta que éste no finalice no puede inicializarse el siguiente. Hoy en día prácticamente no quedan sistemas operativos de este tipo.

3.2.2.2 Multitarea

La multitarea se da cuando se pueden ejecutar varios procesos simultáneamente. Estos procesos compartirán el tiempo de uso del procesador hasta la finalización de cada uno de ellos. Usualmente la concurrencia es solo aparente y en un momento dado sólo un proceso ocupará la CPU, para que pueda existir concurrencia real es preciso que se dé el multiproceso. Prácticamente todos los sistemas operativos que se distribuyen en la actualidad son multitarea.

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

Tipos de multitarea

Cooperativa

Los procesos de usuario son quienes ceden la CPU al sistema operativo a intervalos regulares. Muy problemática, puesto que, si el proceso de usuario se interrumpe y no cede la CPU al sistema operativo, el sistema no podrá hacer nada. Da lugar también a latencias muy irregulares, y la imposibilidad de tener en cuenta este esquema en sistemas operativos de tiempo real. Un ejemplo sería Windows hasta la versión 95.

Preferente

El sistema operativo es el encargado de administrar el/los procesador(es), repartiendo el tiempo de uso de este entre los procesos que estén esperando para utilizarlo. Cada proceso utiliza el procesador durante cortos períodos de tiempo, pero el resultado final es prácticamente igual que si estuviesen ejecutándose al mismo tiempo. Ejemplos de sistemas de este tipo serían Unix y sus clones (FreeBSD, Linux...), VMS y derivados y Windows Server 2016.

Real

Sólo se da en sistemas multiprocesador. Es aquella en la que varios procesos se ejecutan realmente al mismo tiempo, en distintos microprocesadores. Suele ser también preferente. Ejemplos de sistemas operativos con esa capacidad: variantes Unix, Linux, Windows actuales y Mac OS.

3.2.3. Según el número de procesadores

Dependiendo del número de microprocesadores existentes en un mismo ordenador y que el sistema puede utilizar simultáneamente, éste será:

3.2.3.1 Monoprocesador

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

3.2.3.2 Multiprocesador

Un sistema operativo multiprocesador es capaz de manejar más de un procesador en el sistema, distribuyendo la carga de trabajo entre todos los procesadores que existan en el sistema.

Generalmente estos sistemas trabajan de dos formas: simétrica o asimétricamente. Cuando se trabaja de manera asimétrica, el sistema operativo selecciona a uno de los procesadores que jugará el papel de procesador maestro y servirá como pivote para distribuir la carga a los demás procesadores, que reciben el nombre de esclavos.

Cuando se trabaja de manera simétrica, los procesos o partes de ellos (*threads*, hebras o hilos) son enviados indistintamente a cualquiera de los procesadores disponibles, teniendo, teóricamente, una mejor distribución y equilibrio en la carga de trabajo bajo este esquema. Se dice que un *thread* es la parte activa en memoria y corriendo de un proceso, lo cual puede consistir de un área de memoria, un conjunto de registros con valores específicos, la pila y otros valores de contexto.

Un aspecto importante a considerar en estos sistemas es la forma de crear aplicaciones para aprovechar varios procesadores. Existen aplicaciones que fueron hechas para correr en sistemas uniproceso que no aprovechan ninguna ventaja a menos que el sistema operativo o el compilador detecte secciones de código paralelizable, los cuales son ejecutados al mismo tiempo en procesadores diferentes. Por otro lado, el programador puede modificar sus algoritmos y aprovechar por sí mismo esta facilidad.

En la siguiente tabla encontramos ejemplos de algunos sistemas operativos y sus funcionalidades según las clasificaciones anteriores:

Sistema Operativo	nº de usuarios	nº de procesos	nº de procesadores
DOS	Monousuario	Monotarea	Monoproceso
Windows 95,98, Me	Monousuario	*Pseudomultitarea* Monotarea	Monoproceso
Windows NT 4.0 Workstation	Monousuario	Multitarea	Multiproceso
Windows NT 4.0 Server	Multiusuario	Multitarea	Multiproceso
Windows 2000 Professional	Monousuario	Multitarea	Multiproceso
Windows Servers	Multiusuario	Multitarea	Multiproceso
Windows 10	Multiusuario	Multitarea	Multiproceso
UNIX / Linux	Multiusuario	Multitarea	Multiproceso

4 DEFINICIÓN DE PROCESO

Un proceso puede entenderse informalmente como un programa en ejecución. Formalmente un proceso es "una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados"[1].

Para entender mejor lo que es un proceso y la diferencia entre un programa y un proceso, A. S. Tanenbaum propone la analogía "Un científico computacional con mente culinaria hornea un pastel de cumpleaños para su hija; tiene la receta para un pastel de cumpleaños y una cocina bien equipada con todos los ingredientes necesarios, harina, huevo, azúcar, leche, etcétera." Situando cada parte de la analogía se puede decir que la receta representa el programa (el algoritmo), el científico computacional es el procesador y los ingredientes son las entradas del programa. El proceso es la actividad que consiste en que el científico computacional vaya leyendo la receta, obteniendo los ingredientes y horneando el pastel.

Cada proceso tiene su contador de programa, registros y variables, aislados de otros procesos, incluso teniendo el mismo programa en ejecución dos veces. Cuando este caso sucede, el sistema operativo usa la misma región de memoria de código, debido a que dicho código no cambiará, a menos que se ejecute una versión distinta del programa. (Fuente: https://es.wikipedia.org/wiki/Proceso_(inform%C3%A1tica))

Recapitulando, un proceso es un concepto manejado por el sistema operativo que consiste en el conjunto formado por:

- Las **instrucciones** de un programa destinadas a ser ejecutadas por el microprocesador.
- Su **estado de ejecución** en un momento dado, esto es, los valores de los registros de la CPU para dicho programa.
- Su **memoria de trabajo**, es decir, la memoria que ha reservado y sus contenidos.
- **Otra información** que permite al sistema operativo su planificación.

[1] Stallings, William (2005). Sistemas operativos: aspectos internos y principios de diseño (5ª edición). Pearson Prentice Hall. p. 109

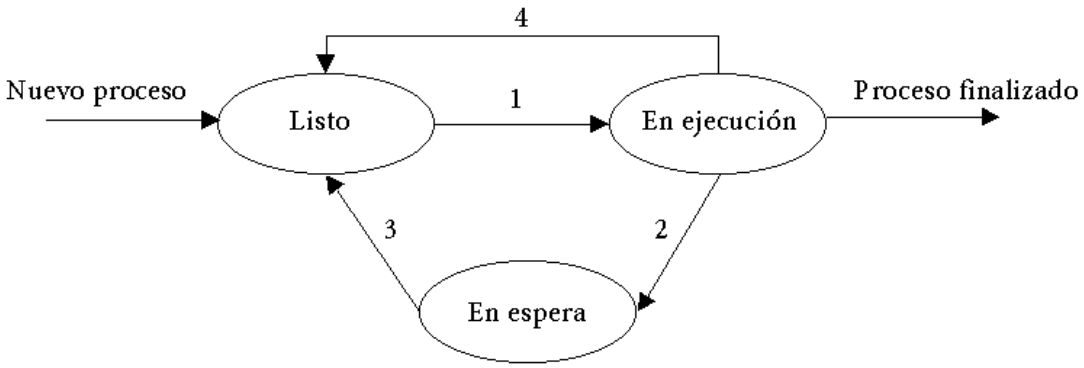
Esta definición varía ligeramente en el caso de sistemas operativos multihilo[2], donde un proceso consta de uno o más hilos, la memoria de trabajo (compartida por todos los hilos) y la información de planificación. Cada hilo consta de instrucciones y estado de ejecución.

Los procesos son creados y destruidos por el sistema operativo, así como también éste se debe hacer cargo de la comunicación entre procesos, pero lo hace a petición de otros procesos.

4.1. Estados de un proceso

Los procesos van a interactuar unos con otros, manteniendo su característica de entidad independiente, su propio contador de programa y su estado interno propio. Los estados en los que se puede encontrar un proceso son:

- **En ejecución**: el proceso está usando el procesador en ese instante.
- **Bloqueado** o en **espera**: el proceso no puede hacer nada hasta que ocurra un evento externo. Por ejemplo, esperando datos de entrada todavía no disponibles.
- **Listo**: el proceso está detenido temporalmente para que pueda ejecutarse otro proceso, pero podría estar en ejecución.



[2] Un hilo de ejecución, en sistemas operativos, es una característica que permite a una aplicación realizar varias tareas concurrentemente. Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

5 NIVELES DEL SISTEMA OPERATIVO

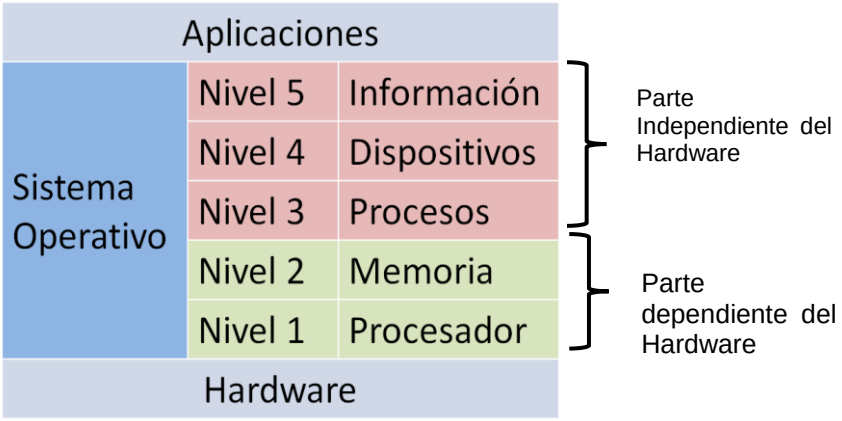
Actualmente los sistemas operativos se organizan en **capas**. Si se tiene en cuenta las funciones que un sistema operativo puede realizar, éstas son:

- Control y ejecución de programas.
- Control, gestión y administración de periféricos.
- Control, gestión y administración de usuarios.
- Control de procesos.
- Control de errores de sistema de sistema y de aplicaciones.
- Control y gestión de seguridad frente a intrusos y virus.

En concordancia con estas funciones principales, es posible analizar la estructura de un sistema operativo en cinco niveles. Los primeros dos niveles entrarían dentro de la parte del sistema operativo dependiente del hardware; el resto de los niveles pertenece a su parte portable.

Nivel 1 – Gestión del procesador. En este nivel se encuentra la parte del sistema operativo encargada de la gestión de la CPU. En los sistemas operativos multitarea (es decir, que pueden ejecutar varios procesos a la vez), este nivel se encarga de compartir la CPU entre los distintos procesos realizando funciones de sincronización, conmutación de la CPU y gestión de interrupciones.

Nivel 2 – Gestión de memoria. Este nivel es el encargado de repartir la memoria libre entre los procesos. Se realizan funciones de asignación y liberación de memoria, y el control de violación de acceso a zonas de memoria no permitidas.



Nivel 3 – Gestión de procesos. Este nivel es el encargado de la creación y destrucción de procesos, intercambio de mensajes y su detección y arranque.

Nivel 4 – Gestión de dispositivos. En este nivel se realiza la gestión de las entradas/salidas (E/S) en función de los dispositivos existentes. Entre otras, se encarga de las funciones de creación de procesos de E/S, asignación y liberación de E/S y planificación de E/S.

Nivel 5 – Gestión de la información. El objetivo de este nivel es el de gestionar el espacio de nombres lógicos y la protección de la información realizando funciones de creación y destrucción de ficheros y directorios, apertura y cierre de ficheros, lectura y escritura de ficheros y protección de acceso.

6 FUNCIONES DE LOS SISTEMAS OPERATIVOS

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

- **Gestión de procesos**. Hay que diferenciar entre los conceptos programa y proceso. Un programa es un ente pasivo, que cuando se carga en memoria y comienza a ejecutarse, puede originar una gran cantidad de procesos. El sistema operativo ha de cargar los distintos procesos, iniciarlos, supervisar su ejecución llevando a cabo los cambios de contexto necesarios y detectar su terminación normal o anormal.

- **Gestión de la memoria**. El sistema operativo debe administrar la memoria, es preciso separar adecuadamente las áreas a las que tiene acceso cada proceso, y repartir correctamente el siempre escaso recurso de la memoria principal. La gestión de memoria suele ir asociada a la gestión de procesos. Para ejecutar un proceso es necesario asignarle unas direcciones de memoria exclusivas para él y cargarlo en ellas, cuando el proceso finalice su ejecución es necesario liberar las direcciones de memoria que estaba usando.

- **Gestión de archivos**. El sistema de archivos, compuesto usualmente por directorios, subdirectorios y ficheros, es un componente principal de cualquier sistema informático. Un fichero es una abstracción para definir una colección de información no volátil. Su objetivo es proporcionar un modelo de trabajo sencillo con la información almacenada en los dispositivos de almacenamiento. Estos ficheros deben tener espacio asignado en los dispositivos, deben estar protegidos entre ellos, deben organizarse según unos determinados esquemas, se debe realizar la gestión del espacio libre y el espacio ocupado... De todo esto se ocupa la gestión de ficheros.

- **Gestión de los dispositivos de E/S**. El acceso a los canales de E/S, y por extensión a los periféricos a ellos conectados, es una función muy importante que ha de llevar a cabo un sistema operativo. La gestión de la E/S tiene como objetivo proporcionar una interfaz de alto nivel sencilla de utilizar. En algunos sistemas operativos esta interfaz es semejante a la gestión de archivos (como por ejemplo en los sistemas Linux).

Hay todo un conjunto de funciones adicionales, como la gestión general de errores del sistema, la gestión de la red, mecanismos de protección y seguridad, etc. La lista de tareas de las que pueden hacerse cargo los sistemas operativos va incrementándose conforme pasa el tiempo y evoluciona la tecnología.

6.1. Gestión de procesos

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

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

6.1.1. Niveles de planificación

Se pueden definir tres niveles de planificación:

- Planificación **de admisión** o alta: cuyo fin es determinar que trabajos deben admitirse en el sistema.
- Planificación **intermedia**: su propósito es determinar que procesos pueden competir por la CPU.
- Planificación **de bajo nivel**: se determina a que procesos en estado listo se le va asignar la CPU convirtiéndose en proceso en ejecución. Esta planificación corre a cargo del **planificador** (scheduler) que utiliza para ello un algoritmo de planificación, siendo ejecutada, en lo que se refiere a cambios de contexto por el **distribuidor** (dispatcher).

Una buena estrategia de planificación debe buscar que los procesos obtengan sus turnos de ejecución adecuadamente, conjuntamente con un buen rendimiento y minimización de la sobrecarga del planificador y del distribuidor. En general, se busca que el sistema finalice el mayor número de procesos por unidad de tiempo y que los procesos terminen en un plazo finito. El algoritmo de planificación deberá repartir de la mejor forma posible el uso de la CPU entre diversos **tipos de procesos**:

- **Procesos de tiempo real:** son aquellos cuyo tiempo de respuesta tiene que estar acotado para que no exceda un umbral máximo, si no se produce un error.
- **Procesos de segundo plano:** no tienen interacción con el usuario y su tiempo de respuesta no es crítico.
- **Procesos interactivos:** no suelen usar mucho tiempo de cómputo y la respuesta debe estar dentro de un rango de tiempo adecuado.

6.1.2. Criterios de planificación

Algunos de los criterios de planificación son:

- **Utilización de la CPU:** es el porcentaje de uso de la CPU, cuánto más mejor.
- **Rendimiento (Throughput):** es el número de procesos ejecutados por unidad de tiempo. Cuánto más mejor.

  Rendimiento = (número de procesos finalizados) / (tiempo total)
- **Tiempo de espera:** tiempo que un proceso está en la cola de procesos listos. Cuanto menor sea, mejor.
- **Tiempo de retorno:** es el tiempo que tarda un proceso en ejecutarse. Cuanto menor sea, mejor.

  Tiempo de retorno = (tiempo de espera) + (tiempo de ejecución)

Para asegurar que ningún proceso monopolizará el procesador durante mucho tiempo se utiliza una **planificación expulsiva**, es aquella en que, con cada interrupción de reloj, el planificador toma el control y decide si el mismo proceso seguirá ejecutándose o si suspende su ejecución y cede el uso del procesador a otro proceso.

6.1.3. Objetivos de la planificación

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

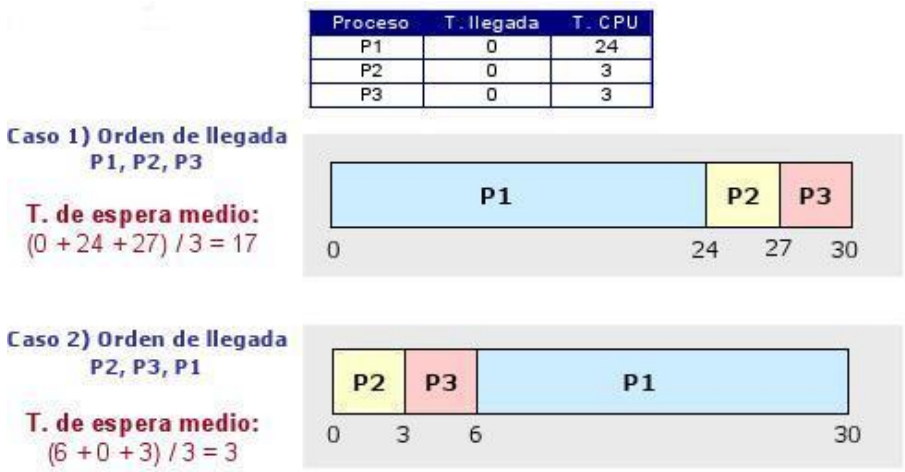
- **Justicia o Imparcialidad**: Todos los procesos son tratados de la misma forma, y en algún momento obtienen su turno de ejecución o intervalos de tiempo de ejecución hasta que finalizan con éxito.
- **Maximizar la Producción**: El sistema debe finalizar el mayor número de procesos por unidad de tiempo.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

6.1.4. Algoritmos de asignación del turno de ejecución

Algunos de los **algoritmos de bajo nivel** para asignar el turno de ejecución de cada proceso se describen a continuación:

- **FCFS (First Come, First Served):** primero en llegar primero en ejecutarse, es un algoritmo no expulsivo. El primer proceso que accede a la CPU, procedente de la cola de listos, se mantiene en ejecución hasta que se bloquea o termina.

  Veamos el siguiente ejemplo con tres procesos.

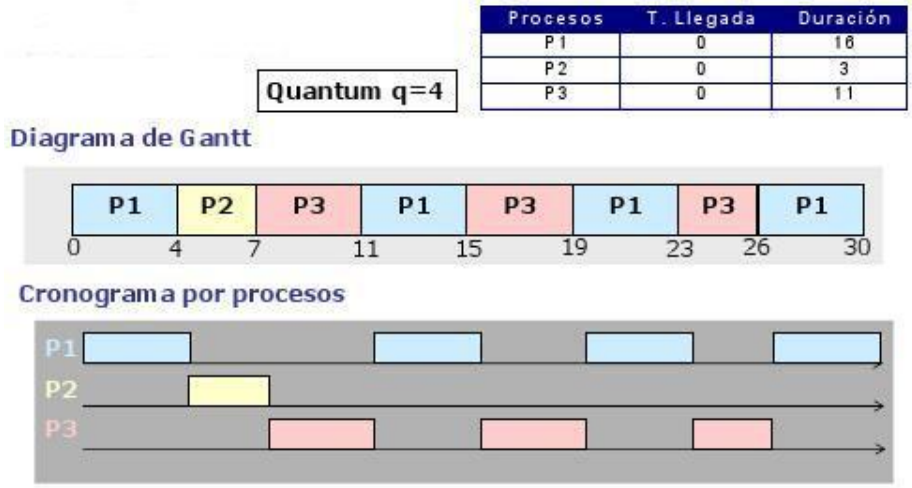


  - Los tres procesos llegan en T. llegada = 0.
  - Alguno tiene que entrar antes o de alguna forma tiene que ser atendido antes. Supongamos que el orden fuera P1, P2, P3. El proceso P1 es el más largo con diferencia.
  - Primero se atendería P1, que ocuparía la CPU durante 24 ciclos, después P2 y finalmente P3.
  - El tiempo de espera medio es de 17 ciclos.
  - Si el primero que ocupara la CPU fuera P2, luego P3 y luego P1, la situación cambiaría, ya que el tiempo de espera medio en este caso es de 3 ciclos.

- **RR (Round Robin)**: la planificación por turno rotatorio es una de las más sencillas y utilizadas, los procesos entran en la CPU siguiendo un esquema FCFS pero con una cantidad de tiempo de ejecución limitada, denominada **quantum**.

  Si el proceso activo sigue en ejecución al final de su quantum, es extraído de la CPU por el planificador y ubicado al final de la cola de procesos listos, también se asigna la CPU al siguiente proceso si el proceso activo termina o se bloquea antes del fin de su quantum.

Lo fundamental de este algoritmo es el **tamaño del quantum**, si éste es muy grande la planificación degenera en una FCFS y si es muy pequeño se sobrecarga el sistema operativo debido a que se produce una gran alternancia de contextos.



Veamos la planificación por turno rotatorio con el ejemplo de la figura de arriba (ve siguiendo al mismo tiempo el cronograma de la figura):

- De nuevo tenemos tres procesos, P1, P2 y P3, que según la tabla llegan en el momento 0. De todas formas, se atenderán siguiendo el orden de su numeración.
- En la tabla tenemos también indicada la duración de cada uno de los procesos.
- El quantum = 4
- El proceso P1 dura 16 ciclos, pero como el quantum es de 4, ocupa la CPU durante 4 ciclos y luego sale a falta de completar todavía 12 ciclos.
- El proceso P2 dura 3 ciclos. Ocupa la CPU y termina. Aunque el quantum es de 4, como el proceso necesita menos ciclos, inmediatamente se libera la CPU y se pasa al siguiente proceso.
- El siguiente proceso que está en cola para ser atendido es P3, que necesita 11 ciclos de CPU. Recordemos que, como el quantum es 4, ocupa la CPU durante 4 ciclos y sale. Le faltan aún 7 ciclos.
- Ahora le toca el turno de nuevo a P1, que tiene pendientes 12 ciclos de CPU. Realiza 4, le quedan pendientes 8, y sale.
- Ahora es el turno de P3, con 7 ciclos pendientes. Ocupa la CPU durante 4 ciclos, le quedan pendientes 3 y sale.

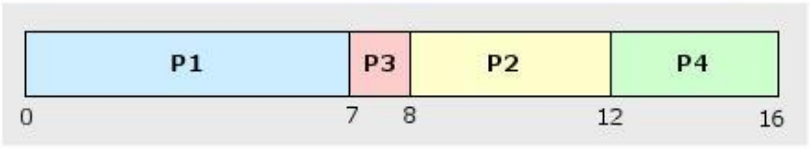
- Turno de P1, con 8 ciclos pendientes. Ocupa la CPU durante 4 ciclos, le quedan pendientes 4 y sale.
- Turno de P3, con 3 ciclos pendientes. Ocupa la CPU durante 3 ciclos, acaba y sale.
- Turno de P1, con 4 ciclos pendientes. Ocupa la CPU durante 4 ciclos, acaba y sale.

- **SJF (Shortest Job First)**: es un sistema no expulsivo según el cual se ejecuta primero el proceso que tiene el menor tiempo estimado de ejecución, lo que hace que el tiempo promedio de espera para cada proceso sea mínimo.

  La implementación de este algoritmo requiere del conocimiento previo del tiempo de ejecución de cada proceso y tiene el riesgo de que los procesos más largos lleguen a la muerte por inanición.

SJF (sin expulsión)

Procesos	Instante de llegada	Duración
P1	0	7
P2	2	4
P3	4	1
P4	5	4



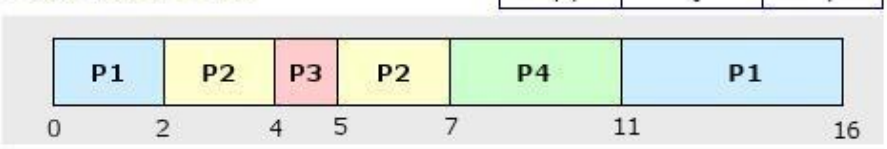
Tiempo de espera medio: (0 + 3 + 6 + 7) / 4 = 4

¿Eres capaz de seguir el ejemplo habiendo visto cómo funcionan los otros algoritmos y teniendo en cuenta las peculiaridades de éste?

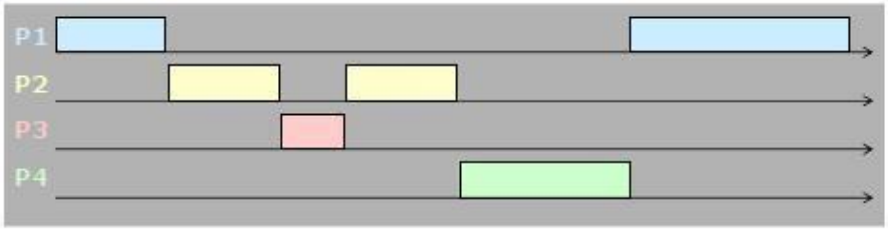
- **SRTF (Shortest Remaining Time First):** es la versión expulsiva del SJF. Cuando un proceso entra a la cola de procesos listos, si le resta menos tiempo

Procesos	T. Llegada	Duración
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Diagrama de Gantt



Cronograma por procesos



Media del tiempo de espera: (9 + 1 + 0 + 2) / 4 = 3

de CPU para finalizar que al proceso que está en ejecución, ocupa el lugar de éste en el procesador. Esta planificación sigue presentando el riesgo de inanición para procesos largos.

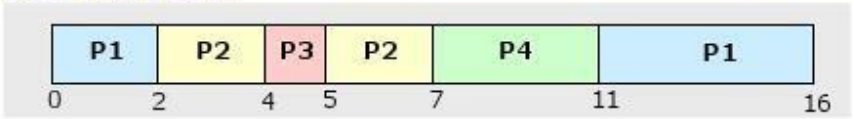
- **Por prioridades:** en estos algoritmos a cada proceso se le asigna un número entero siguiendo algún criterio, llamado **prioridad**. Los procesos de mayor prioridad (normalmente menor número) se ejecutan primero.

  Si existen varios procesos con la misma prioridad, pueden ejecutarse estos de acuerdo a otro algoritmo como FCFS o por Round Robin.

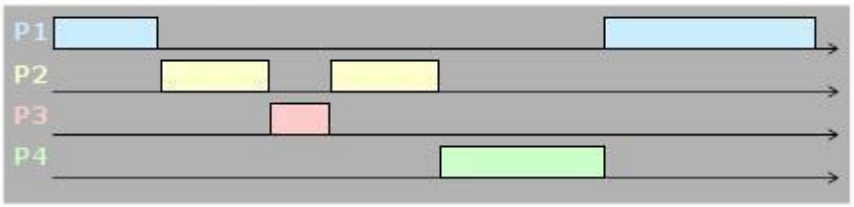
  Estos algoritmos pueden ser **expulsivos** o **no expulsivos** y con **prioridades estáticas** o **dinámicas,** en este último caso las prioridades de los procesos pueden cambiar con el tiempo evitando así el problema de la inanición.

Procesos	T. Llegada	Duración	Prioridad
P1	0	7	15
P2	2	4	10
P3	4	1	5
P4	5	4	10

Diagrama de Gantt



Cronograma por procesos



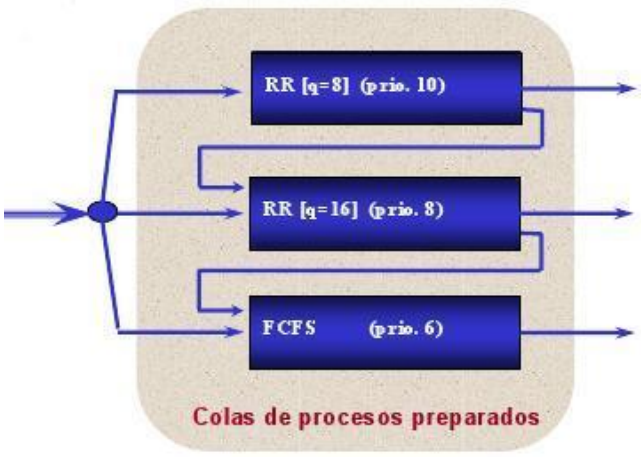
  - En el ejemplo de la figura anterior, en primer lugar, llega el proceso P1 en el instante 0. Como es el único que hay en ese momento, ocupa la CPU, no hay que aplicar ningún algoritmo para decidir.
  - En el instante 2 llega el proceso P2 con prioridad 10. Como es de mayor prioridad, P1 sale, quedándole pendientes 5 ciclos, y es P2 el que ocupa la CPU.
  - En el instante 4 llega P3, con prioridad 5 (mayor prioridad que P2), así que P2 libera la CPU (con 2 ciclos pendientes) y se atiende el proceso P3, de 1 ciclo.
  - P3 acaba en el instante 5. En ese instante también llega P4 con prioridad 10. Pero P2 todavía estaba pendiente con 2 ciclos y prioridad 10. En este momento hay que decidir de otra manera, ya que tenemos dos procesos con la misma prioridad. Se podría atender primero el

proceso más corto o el que ha llegado antes. En este caso, da igual el criterio que se escoja, ya que el proceso que primero llegó y el que tiene menos ciclos pendientes es el P2.

- Se atienden los 2 ciclos pendientes de P2 y libera la CPU.
- Los procesos que quedan pendientes ahora son el P4, con prioridad 10 y 4 ciclos, y el P1, que fue el primero en llegar, pero como tenía prioridad 15 se ha ido postergando su ejecución. Como P4 tiene mayor prioridad, entra en la CPU, se ejecutan sus cuatro ciclos y sale.
- El único proceso que queda pendiente es P1, del que todavía quedaban 5 ciclos por completar. Ocupa la CPU, se ejecutan sus ciclos y sale.

- **Planificación con múltiples colas:** los procesos se clasifican en distintas colas, y debe existir un algoritmo de planificación para cada cola y un algoritmo de planificación entre colas.

También existen planificaciones con **colas retroalimentadas** en las que cada cola tiene una prioridad diferente y los procesos pueden cambiar de colas si cumplen o no cumplen alguna propiedad.



- **Planificación garantizada:** consiste en llegar a un compromiso con el usuario sobre el rendimiento del sistema que se va a poner a su disposición. Por ejemplo, si hay n usuarios accediendo simultáneamente al sistema, se asignará cerca de 1/n del tiempo de proceso de la CPU a cada usuario.

6.1.5. Problemas de Concurrencia

En los sistemas de tiempo compartido (aquellos con varios usuarios, procesos, tareas, trabajos que reparten el uso de CPU entre estos) se presentan muchos problemas debido a que los procesos compiten por los recursos del sistema. Imagina que un proceso está escribiendo en la unidad de cinta y se le termina su turno de ejecución e inmediatamente después el proceso elegido para ejecutarse comienza a escribir sobre la misma cinta. El resultado es una cinta cuyo contenido es un desastre de datos mezclados. Así como la cinta, existen una multitud de recursos cuyo acceso debe ser controlado para evitar los problemas de la concurrencia.

Se denomina **sección crítica** a aquella parte del programa en la cual se accede al recurso compartido. De este modo, si se puede evitar que dos procesos entren simultáneamente en una sección crítica se evitarán condiciones de competencia. Además, para que los procesos paralelos cooperen y usen eficazmente los datos compartidos se necesitan otras condiciones:

- Dos o más procesos no pueden solaparse en sus secciones críticas.
- Ningún proceso en ejecución fuera de su sección crítica puede bloquear a otros procesos.
- Ningún proceso debe esperar eternamente para entrar en su sección crítica.

El sistema operativo debe ofrecer mecanismos para sincronizar la ejecución de procesos: semáforos, envío de mensajes, 'pipes', etc. Los semáforos son rutinas de software (que en su nivel más interno se auxilian del hardware) para lograr **exclusión mutua**[3] en el uso de recursos. Para entender este y otros mecanismos es importante entender los problemas generales de concurrencia, los cuales se describen a continuación:

- **Condiciones de Carrera o Competencia**: La condición de carrera (race condition) ocurre cuando dos o más procesos acceden a un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada.

  Supongamos, por ejemplo, que dos personas realizan una operación en un banco en la misma cuenta corriente, uno A por el cajero, y uno B mediante Internet desde su casa.

  - El usuario A quiere hacer un depósito. El B un retiro.
  - El usuario A comienza la transacción y lee su saldo que es 1000.

[3] La exclusión mutua se utiliza para evitar que fragmentos de código conocidos como 'secciones críticas' accedan al mismo tiempo a recursos que no deben ser compartidos.

    - En ese momento pierde su turno de ejecución por parte del ordenador del banco (y su saldo queda igual a 1000, se guarda este contexto para el proceso del usuario A)
    - y el usuario B inicia el retiro: lee el saldo que es 1000, retira 200 y almacena el nuevo saldo 1000-200=800 y termina.
  - El turno de ejecución regresa al usuario A el cual hace su depósito de 100, quedando el saldo igual al saldo que había leído inicialmente, 1000, más lo que deposita: 1000 + 100 = 1100.

  Como se ve, el retiro se perdió y eso será magnifico para los usuarios A y B, pero al banquero no le hará demasiada gracia. El error pudo ser al revés, quedando el saldo final en 800.

- **Postergación o Aplazamiento Indefinido(a)**: Consiste en el hecho de que uno o varios procesos nunca reciban el suficiente tiempo de ejecución para terminar su tarea.

  Por ejemplo, que un proceso ocupe un recurso y lo marque como 'ocupado' y que termine sin marcarlo como 'desocupado'. Si algún otro proceso pide ese recurso, lo verá 'ocupado' y esperará indefinidamente a que se 'desocupe'.

- **Condición de Espera Circular**: Esto ocurre cuando dos o más procesos forman una cadena de espera que los involucra a todos.

  Por ejemplo, supón que el proceso A tiene asignado el recurso 'cinta' y el proceso B tiene asignado el recurso 'disco'. En ese momento al proceso A se le ocurre pedir el recurso 'disco' y al proceso B el recurso 'cinta'. Ahí se forma una espera circular entre esos dos procesos que se puede evitar quitándole a la fuerza un recurso a cualquiera de los dos procesos.

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

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

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

- **Algoritmo del banquero**: Este algoritmo usa una tabla de recursos para saber cuántos recursos tiene de todo tipo. También requiere que los procesos informen del máximo de recursos que va a usar de cada tipo.

  Cuando un proceso pide un recurso, el algoritmo verifica si asignándole ese recurso todavía le quedan otros del mismo tipo para que alguno de los procesos en el sistema todavía se le pueda dar hasta su máximo.

  - Si la respuesta es afirmativa, el sistema se dice que está en 'estado seguro' y se otorga el recurso.
  - Si la respuesta es negativa, se dice que el sistema está en estado inseguro y se hace esperar a ese proceso.

6.1.6. Threads

En los sistemas operativos multitarea es posible tener muchos hilos de ejecución o threads de control dentro de un mismo proceso, otro nombre que reciben es el de proceso ligero.

Los hilos son en realidad subprocesos, cada hilo tiene su propio contador de programa y una pila para llevar un registro de posición, los hilos se alternan en la CPU de la misma forma que lo hacen los procesos.

El multihilado o multithreading, provee una manera para tener más de un hilo ejecutándose en el mismo proceso y compartiendo el mismo espacio de memoria, lo que permite una comunicación muy rápida entre hilos. Además, los hilos son mucho más rápidos de crear ya que no requieren un espacio de direcciones de memoria propias.

6.2. Gestión de memoria

La parte del sistema operativo que se encarga de la memoria se llama "*administrador de memoria*" y se encarga de:

- Llevar el **control** de qué **partes** de la memoria están **en uso** y cuáles no.
- **Asignar memoria a** los **procesos** cuando la necesiten y **retirársela** cuando terminen.
- **Administrar** el **intercambio** entre la **memoria central** y el **disco** cuando la memoria central no sea suficiente para contener todos los procesos.

En un sistema **monotarea** o monoprogramado, al contrario que en uno multitarea, en la memoria del ordenador sólo hay un único programa, acompañado de sus datos y del sistema operativo. Esto hace que el uso de la memoria, y la asignación de la misma sea muy simple.

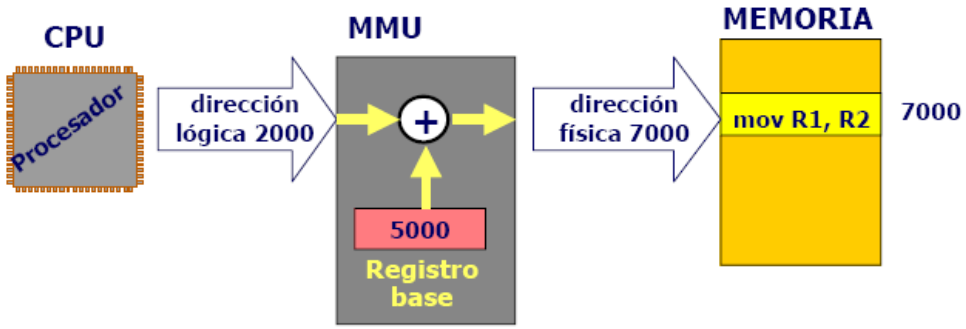
6.2.1. Multiprogramación con particiones fijas

Consiste en **dividir la memoria en particiones de tamaño fijo**, cuando llega un trabajo se puede colocar en la cola de espera de entrada de la menor partición lo suficientemente grande para contenerlo. Al tratarse de particiones fijas se produce **fragmentación interna**, es decir, cualquier espacio que no utilice un trabajo en una partición se pierde. En estos sistemas se pueden encontrar en memoria varios programas a la vez, lo que conlleva dos **problemas** fundamentales, la **protección** y la **relocalización**.

El **problema de la relocalización** consiste en que los programas que necesitan cargarse a memoria real ya están compilados y montados, de manera que internamente contienen una serie de referencias a direcciones de instrucciones, rutinas y procedimientos que ya no son válidas en el espacio de direcciones de memoria real en la que se carga el programa. Esto es, cuando se compiló el programa se definieron o resolvieron las direcciones de memoria de acuerdo a la sección de ese momento, pero si el programa se carga en otro sistema o en otro instante, las direcciones reales ya no coinciden.

La **solución** está en utilizar un componente hardware específico denominado **MMU** (Unidad de Manejo de Memoria) que se encarga de traducir las direcciones lógicas o virtuales en direcciones físicas o reales. En la MMU existe un registro que guarda la dirección base de la sección que va a contener el programa. Cada vez que el

programa haga una referencia a una dirección de memoria, se le suma el **registro base** para encontrar la dirección real.

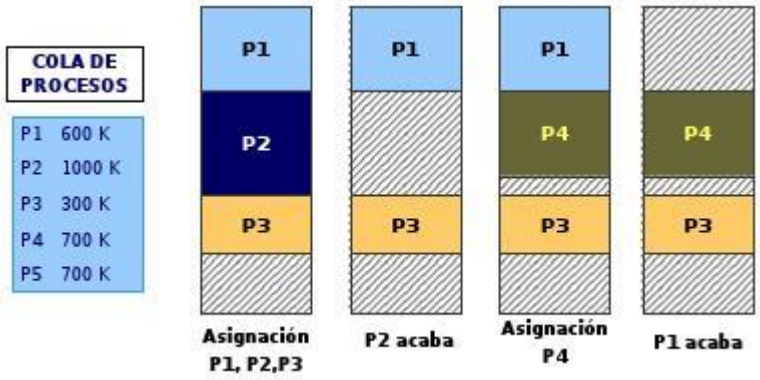


El **problema de la protección** se refiere a que, una vez que un programa ha sido cargado en memoria en algún segmento en particular, nada le impide al programador que intente direccionar, por error o deliberadamente, direcciones de memoria fuera de su espacio de direcciones.

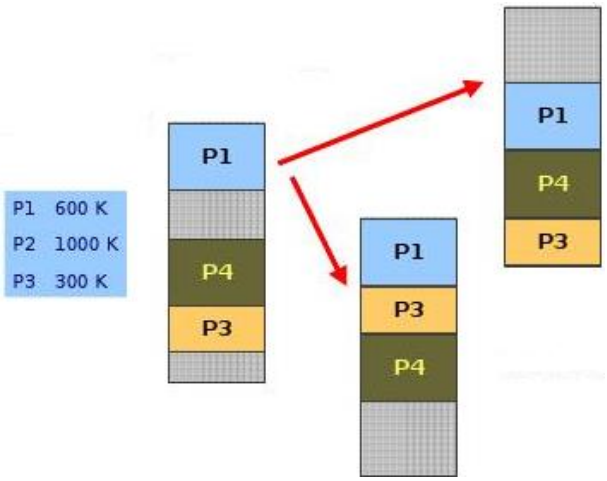
La **solución** a este problema también puede ser el uso de un **registro base** y un **registro límite**. El registro **base** contiene la dirección del comienzo de la sección que contiene el programa, mientras que el **límite** contiene la dirección donde termina. Cada vez que el programa hace una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

6.2.2. Multiprogramación con particiones variables

Con las particiones variables el **número y tamaño de los procesos en la memoria varían de forma dinámica en tiempo de ejecución**. Al no estar condicionado por un número fijo de particiones mejora la utilización de la memoria, pero complica la asignación de la memoria y su control.



Uno de los problemas de utilizar este método de asignación de memoria es que se produce **fragmentación externa**, quedando huecos de memoria entre las particiones variables que no se pueden utilizar. La **solución** es combinar todos los huecos en uno grande pasando todos los procesos lo más abajo o lo más arriba que sea posible dentro del espacio de direccionamiento. Esta técnica se conoce como **compactación de memoria** y solo se usa en caso extremos por el coste temporal que supone.



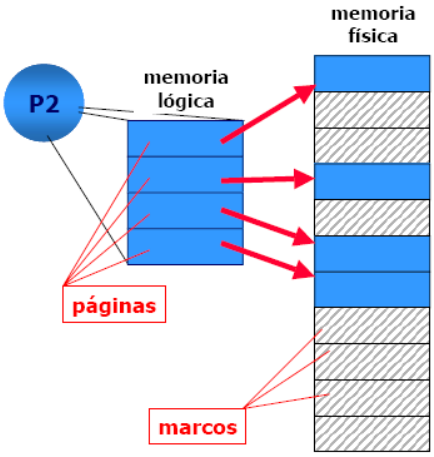
Para gestionar los huecos libres que hay en memoria se puede utilizar algunos de los siguientes algoritmos:

- **Primer ajuste:** se asigna al programa el primer hueco lo suficientemente grande, después el hueco se divide en dos partes una para el proceso y otra para memoria libre. Es un algoritmo veloz.
- **Siguiente ajuste:** funciona como el primer ajuste, pero lleva el control de dónde se encuentra cuando localiza un hueco adecuado. La siguiente vez que se llama empieza a buscar por donde se quedó.
- **Mejor ajuste:** busca el hueco más pequeño posible, es más lento que el primer ajuste y tiende a llenar la memoria con huecos libres muy pequeños que ya no son útiles.
- **Peor ajuste:** asigna el hueco libre más grande de manera que tras la división sea lo suficientemente grande para ser útil. Es poco funcional.
- **Optimización adicional:** los algoritmos anteriores se pueden acelerar conservando listas distintas de huecos libres y de procesos, pero se ralentiza el trabajo cuando se hace una liberación de memoria ya que el segmento liberado tiene que pasar de la lista de procesos a la de huecos.
- **Ajuste rápido:** Conserva listas separadas de algunos de los tamaños que más se utilizan. De esta forma la obtención de un hueco del tamaño que se requiere es rápida, pero se tiene que realizar una clasificación por tamaño de hueco.

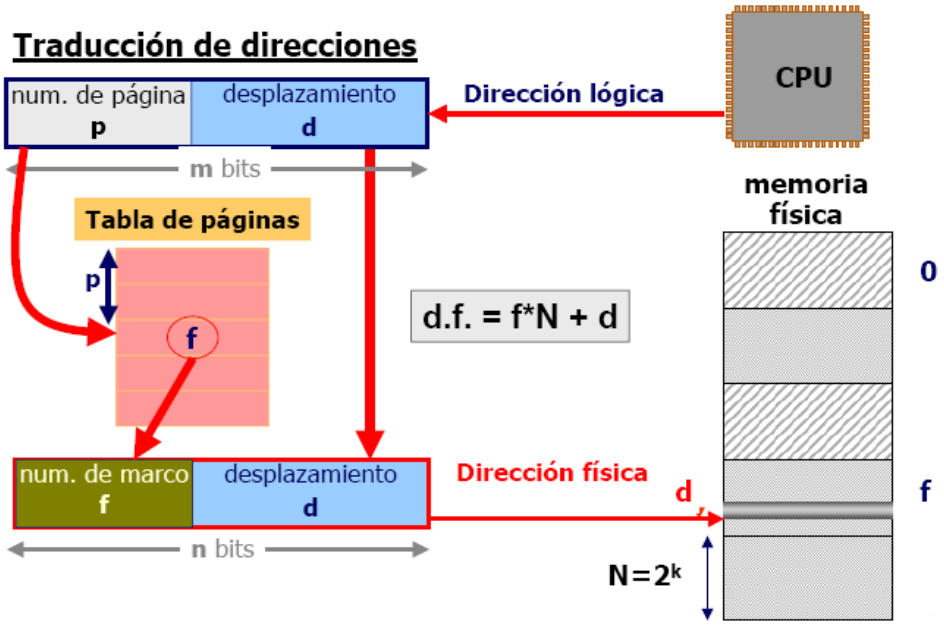
6.2.3. Paginación

La paginación surge para paliar los defectos de la asignación contigua de memoria, sofisticando el hardware de gestión de memoria del procesador y aumentando la cantidad de información de traducción que se almacena por cada proceso.

El mapa de memoria de cada proceso (memoria lógica) se considera dividido en **páginas**, y la memoria principal del sistema se considera dividida en bloques del mismo tamaño que se denominan **marcos de página** (frames). Un marco de página contendrá en un determinado instante una página de memoria de un proceso.



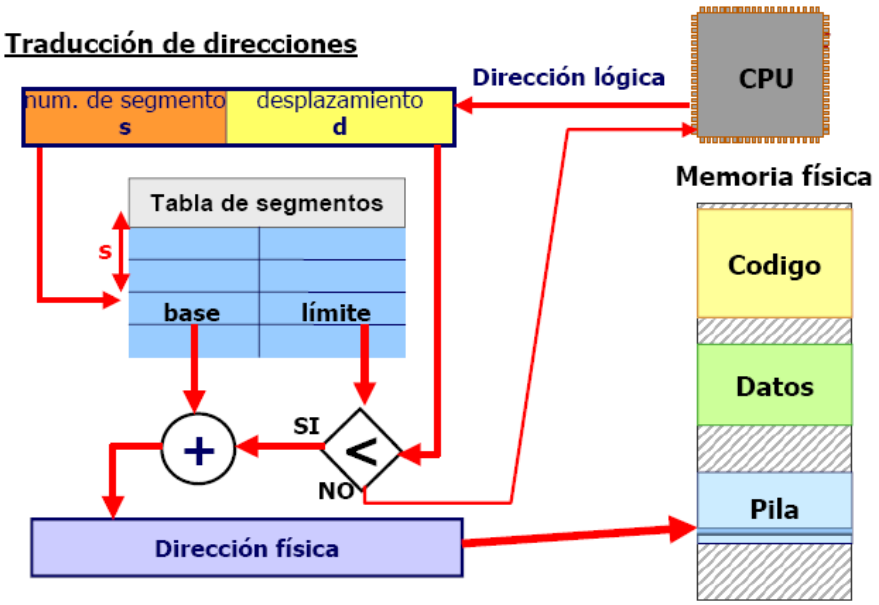
La estructura de datos que relaciona cada página con el marco donde está almacenada es la **tabla de páginas**. Se utiliza para traducir todas las direcciones lógicas (número de página y desplazamiento) a direcciones físicas, es decir, detecta a qué página del mapa corresponde una dirección lógica y accede a la tabla de páginas para obtener el número del marco donde está almacenada dicha página. La dirección física tendrá un desplazamiento con respecto al principio del marco igual que el de la dirección lógica.



Para determinar el tamaño de las páginas hay que tener en cuenta que cuanto más pequeñas sean las páginas más se reducirá la **fragmentación interna** pero más grandes serán las tablas de página de cada proceso.

6.2.4. Segmentación

La segmentación consiste en dividir el espacio de direcciones en fragmentos de **longitud variable**. Se utiliza un registro base y un registro límite por cada segmento y la MMU maneja una **tabla de segmentos**. Cada entrada de esta tabla tiene información de protección y los registros límites. La traducción de dirección lógica a física consiste en acceder al número de segmento correspondiente y usar los registros límites almacenados en dicha entrada para detectar si el acceso es correcto.



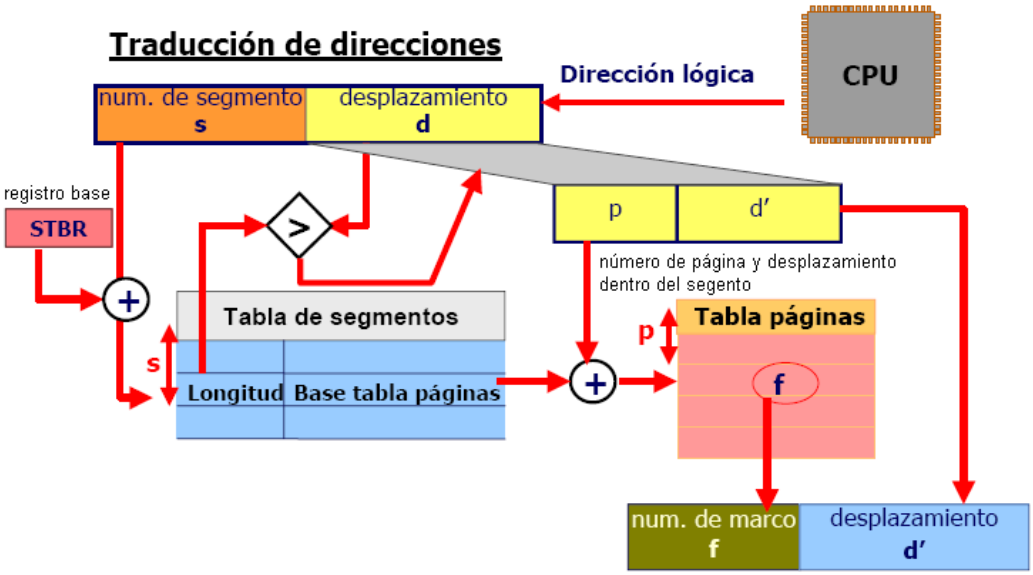
El sistema operativo mantendrá una tabla de segmentos por cada proceso y en cada cambio de proceso irá informando a la MMU de qué tabla debe usar.

Entre las **ventajas** que presenta este sistema destacamos que simplifica el manejo de estructuras de datos dinámicas, no aparece fragmentación interna, facilita la reubicación y proporciona protección.

Como **desventaja** cabe destacar que cuando los procesos crecen aumenta la fragmentación externa y aumenta la dificultad de encontrar huecos libres para ubicar los procesos.

6.2.5. Segmentación paginada

Esta técnica consiste en paginar los segmentos, es decir, la memoria se divide en bloques de tamaño variable (segmentos), los que a su vez se dividen internamente en otros bloques de tamaño fijo (páginas). La MMU utiliza una tabla de segmentos en la que cada entrada de la tabla apunta a una tabla de páginas.



Con esta técnica se consiguen las ventajas de la paginación y de la segmentación.

- Cada proceso tiene una tabla de segmentos que crea un espacio lógico independiente para el mismo.
- La tabla de segmentos de un proceso restringe qué parte de la memoria puede ser accedida por el mismo.
- Bajo la supervisión del sistema operativo, dos o más procesos pueden tener un segmento asociado a la misma zona de memoria.
- La paginación proporciona un buen aprovechamiento de la memoria, aunque se sigue produciendo fragmentación interna, y puede servir como base para construir un esquema de memoria virtual (ver apartado siguiente).

Para espacios de direcciones muy grandes, la tabla de páginas puede ser excesivamente grande y la solución consiste en paginar la propia tabla de páginas creando así **paginación multinivel**.

6.2.6. Memoria virtual

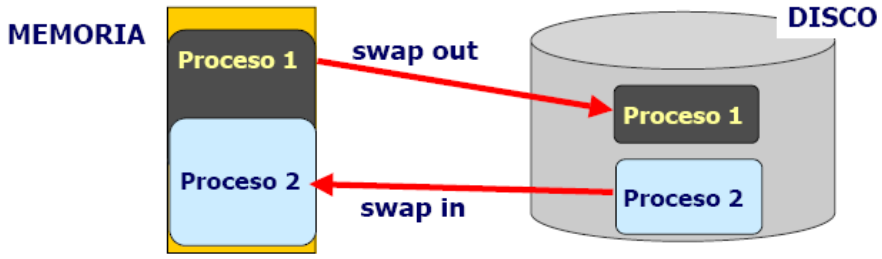
Una vez que surgió la multiprogramación, los usuarios comenzaron a explorar la forma de ejecutar grandes cantidades de código en áreas de memoria muy pequeñas, auxiliados por algunas llamadas al sistema operativo. Es así como nacen los ***overlays***.

Esta técnica consiste en que el programador divide lógicamente un programa muy grande en secciones que puedan almacenarse en las particiones de RAM. Al final de cada sección del programa (o en otros lugares necesarios) el programador insertaba una o varias llamadas al sistema con el fin de descargar la sección presente de RAM

y cargar otra, que en ese momento residía en disco duro u otro medio de almacenamiento secundario.

La necesidad de ejecutar programas cada vez más grandes y el crecimiento en potencia de las CPUs empujaron a los diseñadores de los sistemas operativos a implantar un mecanismo para ejecutar automáticamente programas más grandes que la memoria real disponible, esto es, ofrecer **memoria virtual**[4] [5].

La memoria virtual se llama así porque el programador ve una cantidad de memoria mucho mayor que la real, y en realidad se trata de la suma de la memoria de almacenamiento primario (RAM) y una cantidad determinada de almacenamiento secundario. El sistema operativo, en su unidad de manejo de memoria (MMU), se encarga de intercambiar programas enteros, segmentos o páginas entre la memoria real y el dispositivo de almacenamiento secundario. A este proceso de intercambiar páginas, segmentos o programas completos entre RAM y disco se le conoce como intercambio o **swapping**.



Hay que comentar que algunos sistemas de memoria virtual usan la técnica de la **prepaginación**. En un fallo de página no sólo se traen la página en cuestión, sino también las páginas adyacentes, ya que es posible que el proceso las necesite en un corto plazo de tiempo. La efectividad de esta técnica va a depender de si hay acierto en la predicción o no.

Cuando ocurre un **fallo de página**, el sistema operativo debe elegir una página para retirarla de memoria para dejar espacio para la página que ha de cargarse en el marco de página que se ha quedado libre.

Las **políticas de reemplazo** determinan qué página debe ser desplazada de la memoria principal para dejar sitio a la página entrante. El objetivo básico de cualquier

[4] Necesidad de swap (memoria de intercambio o memoria virtual) en Linux. Además, se explica qué es la memoria virtual: https://geekytheory.com/es-necesaria-una-particion-swap-en-linux

[5] En la actualidad, en equipos en que se dispone de mucha memoria RAM, por ejemplo, 16GB, la memoria virtual puede no ser tan necesaria. Algunos artículos relacionados: https://www.nireleku.com/2013/09/como-liberar-espacio-en-disco-eliminando-pagefile-sys-hiberfil-sys-windows-7/

http://linuxmanr4.com/2014/04/03/desactivar-la-particion-de-swap/

algoritmo de reemplazo es minimizar la tasa de fallos de página. Algunos de estos algoritmos son:

- **Algoritmo de reemplazo óptimo:** se reemplaza la página que tardará más tiempo en volverse a utilizar. Es el mejor algoritmo, pero imposible de implementar.

- **Sustitución de página no usada recientemente (NRU):** se asocian dos bits a cada página, el **bit R** (referencia), que se pone a 1 cuando se lee o se escribe en la página y el **bit M** (modificación), se pone a 1 cuando se escribe en la página. Cada cierto tiempo se pone a 0 el bit R. El algoritmo NRU pasa al disco una página seleccionada al azar del primer grupo en el siguiente orden:

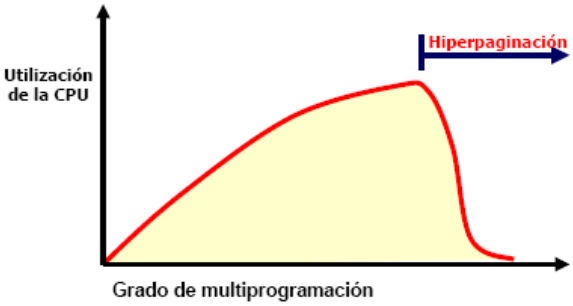
  Grupo 0: R=0; M=0
  Grupo 1: R=0; M=1
  Grupo 2: R=1; M=0
  Grupo 3: R=1; M=1

- **Sustitución primera en entrar primera en salir (FIFO):** se reemplaza la página que lleva más tiempo en memoria. El sistema operativo debe mantener una lista de las páginas que están en memoria, ordenada por el tiempo que llevan cargadas.

- **Algoritmo de la segunda oportunidad o del reloj:** es una modificación del FIFO. Cuando se necesita reemplazar una página se examina el bit de referencia (bit R) de la página más antigua. Si está a 0, se usa esta página, en caso contrario, se le da una segunda oportunidad a la página poniéndola al final de la lista y desactivando su bit de referencia. La búsqueda continuará hasta que se encuentre una página con su bit de referencia desactivado.

- **Sustitución de página menos recientemente usada (LRU):** se sustituye la página que ha estado sin uso por el periodo de tiempo más largo. Esta estrategia se basa en el principio de localidad, lo que acaba de ocurrir es un buen indicador de lo que está a punto de suceder.

Si el número de marcos de página asignados a un proceso no es suficiente para almacenar las páginas se producirá un número elevado de fallos de página. A esta situación se le denomina **hiperpaginación (thrashing)**. Cuando se produce esta situación el proceso pasa más tiempo en la cola de espera que en ejecución.



6.3. Gestión de entrada/salida.

El concepto de entrada y salida hace referencia a toda comunicación o **intercambio** de información entre la CPU o la memoria principal con el exterior. La parte del ordenador que permite esta comunicación es la **unidad de entrada/salida**.

En el sistema de E/S se encuentran **dos partes** fundamentales: los periféricos y la interfaz. Los **periféricos** son los dispositivos electromecánicos y la **interfaz** es el sistema hardware/software que permite la comunicación entre el periférico y la CPU o la memoria principal.

Para manejar los diferentes periféricos existentes se deben enviar **comandos** a los dispositivos, detectar **interrupciones** y controlar **errores**. Para que se pueda llevar a cabo el intercambio de información se deben realizar las siguientes **tareas**:

- **Direccionamiento**: es la selección del dispositivo de E/S implicado en una transferencia determinada.
- **Transferencia**: de los datos desde o hacia el dispositivo seleccionado.
- **Sincronización**: entre los periféricos y la CPU.
- **Gestión de prioridades**: para peticiones simultáneas.

Los dispositivos de entrada/salida se dividen, en general, en dos **tipos**:

- Los **dispositivos orientados a bloques,** que trabajan con bloques de tamaño fijo y además tienen la propiedad de que se pueden direccionar, es decir, es posible escribir o leer cualquier bloque independientemente ya que cada uno tiene una dirección concreta. Entre ellos se encuentran los discos duros, la memoria y los discos compactos.
- Los **dispositivos orientados a carácter** trabajan con secuencias de bytes sin importar su longitud ni ninguna agrupación especial. No son dispositivos direccionables. Entre ellos se encuentran el teclado, el ratón y la pantalla.

6.3.1. Direccionamiento de dispositivos

Generalmente un ordenador tiene conectado más de un dispositivo de E/S, por lo que es necesario disponer de algún mecanismo para seleccionar uno de ellos para que realice una operación de E/S.

El direccionamiento de dispositivos de E/S puede ser de dos tipos, mediante **buses separados** para la memoria y la entrada/salida o **mapeando** los dispositivos en la memoria.

- En la estructura con **buses separados para memoria y entrada salida**, todos los dispositivos se conectan al bus de E/S. Este bus consta de tres conjuntos

de líneas: de direcciones, de datos y señales de control. Si el bus de direcciones consta de n líneas se podrán especificar 2^n direcciones distintas, cada una especificará lo que se conoce como puerta de E/S. Cada periférico empleará una o varias puertas para comunicarse con la CPU.

- En las máquinas de **bus único**, que es el caso de la mayoría de los ordenadores, el mismo bus sirve tanto de bus de memoria como bus de E/S. En este caso se suele tratar las puertas de E/S como si fueran direcciones de memoria, a esto se le conoce como E/S por mapa de memoria. Este tipo de direccionamiento permite disponer de un mapa de E/S más grande, aunque se pierden direcciones de memoria principal.

6.3.2. Transferencia de datos.

Podemos encontrar las siguientes técnicas para llevar a cabo la entrada/salida:

6.3.2.1 E/S controlada por programa

El procesador envía una orden de E/S, a petición de un proceso, a un módulo de E/S, ejecutando instrucciones de E/S y estableciendo la sincronización adecuada. El proceso realiza una espera activa hasta que se complete la operación antes de continuar. En el caso de periféricos de alta velocidad, la E/S controlada por programa es muy poco eficiente.

6.3.2.2 E/S por acceso directo a memoria

En la entrada/salida por acceso directo a memoria todas las funciones de E/S las realiza un circuito controlador denominado controlador de DMA. Este controlador permite la transferencia de datos directa entre el dispositivo y la memoria principal sin intervención de la CPU.

6.3.3. Sincronización entre los periféricos y la CPU

La sincronización se utiliza para conseguir que los dispositivos periféricos estén permanentemente atendidos en sus demandas y ofrecimientos de información. Los dos mecanismos principales de sincronización son: la sincronización por sondeo y la sincronización por interrupciones.

- En la **sincronización por consulta de estado o sondeo** es la CPU la encargada de la sincronización, realizando periódicamente una encuesta a los dispositivos consultando su situación. Esto se lleva a cabo mediante la lectura de un registro de estado, que contiene la interfaz del dispositivo, que indica si el periférico está preparado para enviar o recibir información.

  Los **inconvenientes** de este método de sincronización es que se pierde tiempo de CPU en consultar dispositivos que no están preparados y que dispositivos preparados deben esperar a ser sondeados.

- En la **sincronización mediante interrupciones** se permite que sean los dispositivos o sus controladores, mediante el uso de interrupciones, los que interrumpan la ejecución de un proceso en la CPU cuando están en disposición de participar en una operación de E/S.

  El procesador emite una orden de E/S a petición de un proceso y continúa ejecutando las instrucciones siguientes, siendo interrumpido por el módulo de E/S cuando éste ha completado su trabajo. Las siguientes instrucciones pueden ser del mismo proceso, en el caso de que ese proceso no necesite esperar hasta que se complete la E/S. En caso contrario, se suspende el proceso en espera de la interrupción y se realiza otro trabajo.

  Esta interrupción se hace a través de una línea especial del bus de control llamada INTR. Cuando la CPU recibe una petición de interrupción se desencadenan una serie de acciones:

  - se termina de ejecutar la instrucción en curso,
  - si la petición es aceptada se inhibe total o parcialmente el sistema de interrupciones,
  - se guarda el estado de la tarea en curso,
  - se trata la interrupción ejecutando las tareas asociadas,
  - se restituye el estado de la tarea interrumpida,
  - se rehabilita el sistema de interrupciones
  - y se continúa con el programa interrumpido.

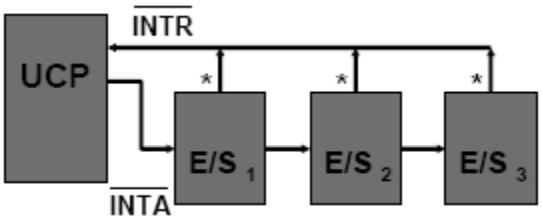
	Sin interrupciones	Con interrupciones
Transferencia de E/S a memoria a través del procesador	E/S programada	E/S dirigida por interrupciones
Transferencia directa de E/S a memoria		DMA

6.3.4. Gestión de prioridades

Es necesario disponer de un mecanismo para dilucidar qué dispositivo será atendido primero, en el caso de existir varios dispositivos preparados, y establecer el ritmo de servicio tanto a éste como a los demás dispositivos.

- En la sincronización por sondeo, la prioridad se implanta de forma inmediata según el orden y frecuencia en que se consultan los periféricos.
- En la sincronización mediante interrupciones existen dos métodos para solucionar el problema de las prioridades: la **gestión centralizada de prioridades** y la **gestión distribuida de prioridades**.
    - En la **gestión distribuida,** la prioridad de cualquier dispositivo viene determinada por la forma en que se conecta a la CPU.

      Por ejemplo, en la conexión en cadena de margarita, la línea de solicitud de interrupción INT es común a todos los dispositivos y la línea de aceptación de interrupción INTA está conectada en forma de cadena, con lo que un dispositivo con mayor prioridad puede bloquear la propagación de INTA y colocar su número de interrupción.

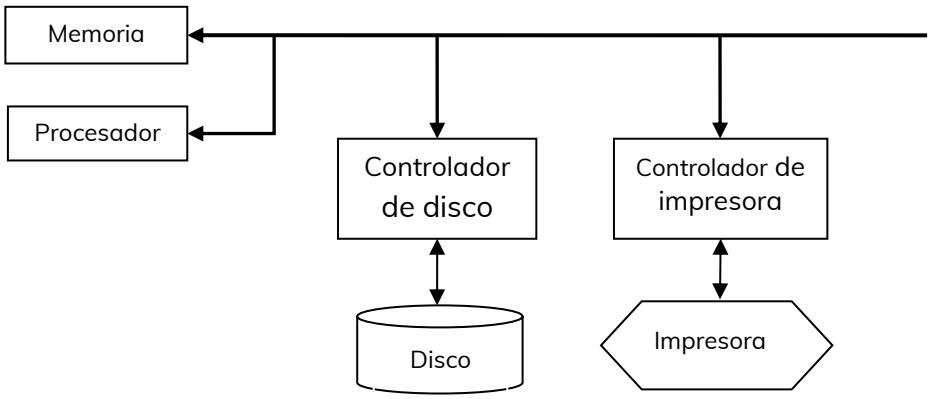
      

    - En la **gestión centralizada** cada dispositivo tiene líneas individualizadas de petición de interrupción. Todas las peticiones son controladas por un órgano común que, de acuerdo a la política establecida, selecciona el dispositivo que debe recibir el servicio.

6.3.5. Elementos que intervienen en la gestión de E/S

6.3.5.1 Controladores de dispositivos

Los dispositivos de E/S pueden dividirse en una parte mecánica y una parte electrónica. Cuando se accede a un dispositivo, **se accede siempre** a su **parte electrónica**, denominada **controlador** de dispositivo. El controlador se encarga de mostrar al sistema operativo una interfaz estándar adaptándose a las peculiaridades del hardware que está por debajo.



6.3.5.2 Acceso Directo a Memoria (DMA)

Muchos controladores, en particular los correspondientes a dispositivos de bloque, permiten DMA (Direct Memory Access). El objetivo del DMA es solapar operaciones de CPU y E/S. La CPU proporciona al controlador información sobre la dirección de memoria para acceder y el número de datos a transferir. Una vez realizada la petición, la CPU se despreocupa momentáneamente de la transferencia y continúa realizando otras tareas.

En el caso de copia de datos desde un disco a memoria, el controlador va leyendo los datos del disco y copiándolos en memoria; una vez realizada la transferencia, el controlador provoca una interrupción que hace que la CPU abandone el trabajo que estaba realizando.

6.3.5.3 Manejadores de dispositivos (drivers)

Una vez comentado el hardware, es conveniente analizar el software de manejo de los dispositivos. Este software está organizado en capas. Las capas inferiores se encargan de ocultar las peculiaridades del hardware y las capas superiores de presentar una interfaz amigable a los usuarios.

Los manejadores de dispositivos (también conocidos como drivers) se encargan de aceptar las solicitudes abstractas que le hace el software independiente del dispositivo y ponerse en contacto con el controlador para realizar esa petición.

Si el dispositivo se encuentra ocupado atendiendo otra petición, el manejador se encargará de gestionar una cola de peticiones para darles paso tan pronto como sea posible.

6.4. Gestión de datos. Sistema de ficheros.

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

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

Un sistema de ficheros es el aspecto más visible de todo sistema operativo. Surge debido a la necesidad del sistema operativo de poder gestionar la información de forma eficiente y estructurada, además de establecer unos parámetros de seguridad y protección en entornos críticos.

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

6.4.1. Objetivos

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

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

Hay otros objetivos secundarios, entre los que destacan:

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

6.4.2. Tipos de ficheros

Normalmente los ficheros se organizan en directorios (también llamados carpetas) para facilitar su uso. Los directorios no son más que ficheros que contienen información sobre otros ficheros: no son más que contenedores de secuencias de registros, cada uno de los cuales posee información acerca de otros ficheros.

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

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

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

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

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

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

6.4.3. Estructuras de directorios

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

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

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

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

Hay varias formas de organizar los directorios sobre un disco:

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

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

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

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

Dentro de la estructura de directorios podemos encontrar los siguientes tipos:

- Se define **directorio padre** de un fichero o subdirectorio como el directorio en el que se encuentra su entrada de referencia Cada fichero o directorio (a excepción del directorio raíz) posee un único directorio padre.

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

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

En este caso, los nombres de ruta de acceso pueden adoptar dos formas alternativas:

- La primera es la del nombre de ruta **absoluto**, la que hemos definido, por la cual se nombra a cada fichero con respecto al directorio raíz.

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

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

6.4.4. Información en una entrada de directorio

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

Pero, ¿qué se almacena realmente en una entrada de directorio? La siguiente tabla muestra algunas de estas informaciones, aunque en un sistema de ficheros concreto esta relación puede variar, encontrándose más o menos que las que se especifican a continuación:

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

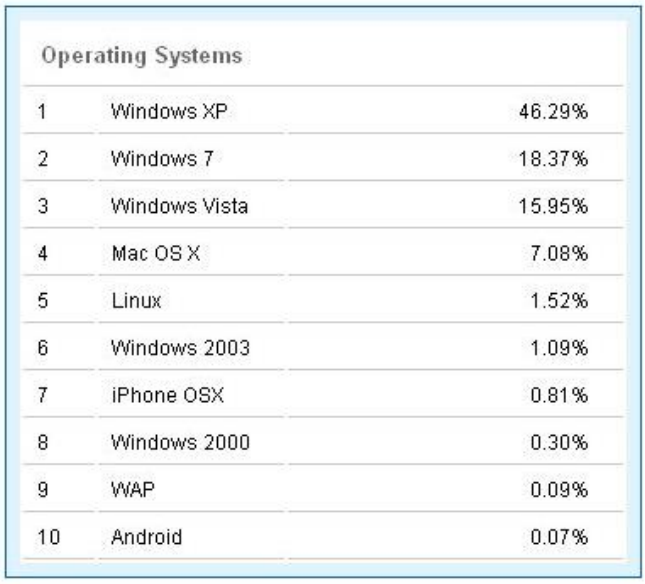
Según el sistema concreto, una entrada de directorio puede ocupar desde una decena hasta varios miles de bytes. Por esta razón, la estructura de directorios suele almacenarse en el propio dispositivo que está organizando, como un fichero especial.

7 SISTEMAS OPERATIVOS ACTUALES

Contar los usuarios de cada sistema operativo es una tarea imposible de realizar. En este apartado vamos a ver, como curiosidad, una aproximación del uso de algunos de los sistemas operativos más habituales.

Los informes periódicos realizados por la web *w3counter* pueden ser un indicador del uso de los sistemas operativos que acceden actualmente a Internet. Vamos a ver los datos de hace siete años y los datos actuales:

This report was generated **08/31/2010** based on the last 15,000 page views to each website tracked by W3Counter. W3Counter's sample currently includes **40,058** websites.



Operating Systems		
1	Windows XP	46.29%
2	Windows 7	18.37%
3	Windows Vista	15.95%
4	Mac OS X	7.08%
5	Linux	1.52%
6	Windows 2003	1.09%
7	iPhone OSX	0.81%
8	Windows 2000	0.30%
9	WAP	0.09%
10	Android	0.07%

www.w3counter.com/globalstats.php

Siete años más tarde:

This report was generated 10/31/2017 based on the past month's traffic to all websites that use W3Counter's free web stats.

Permanent link to this report: http://www.w3counter.com/globalstats.php?year=2017&month=10

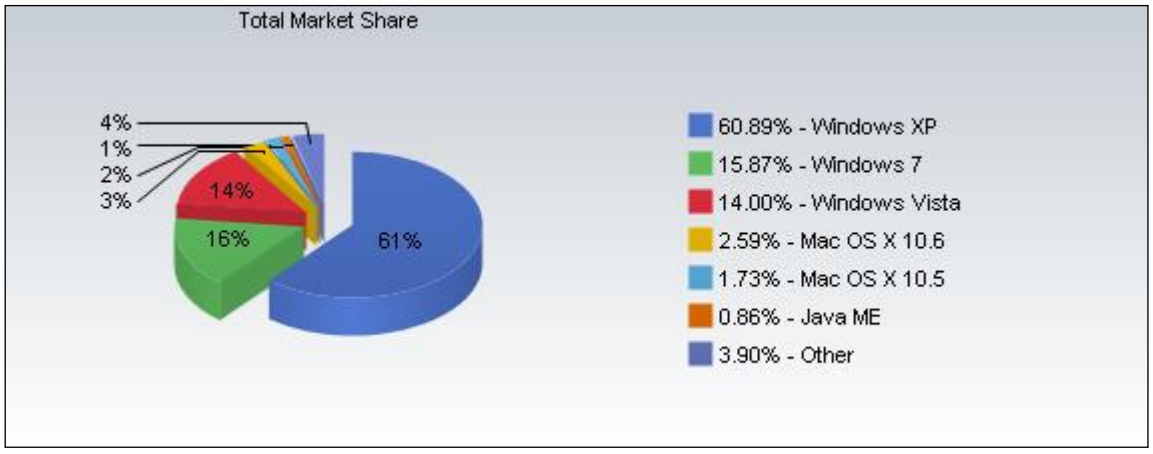
Uso de sistemas operativos en el mundo: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

Top 10 Platforms		
1	Windows 7	19.14%
2	Windows 10	15.47%
3	Android 6	10.98%
4	Android 5	8.53%
5	Android 7	8.02%
6	iOS 10	6.39%
7	iOS 11	6.07%
8	Android 4	5.73%
9	Mac OS X	4.00%
10	Windows 8.1	3.52%

A continuación, se presentan datos similares de MarketShare. Ellos ofrecen por separado los datos de sistemas operativos para sistemas de escritorio y los datos para teléfono móviles, por eso en los gráficos que hay a continuación no figuran los sistemas Android. http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0 .

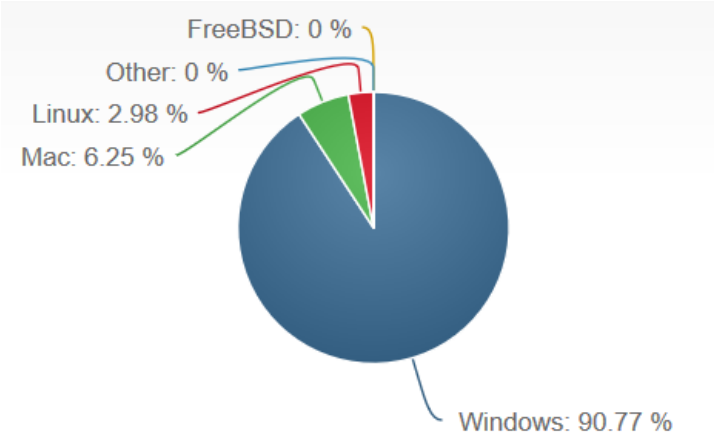
Operating System Market Share

Agosto, 2010

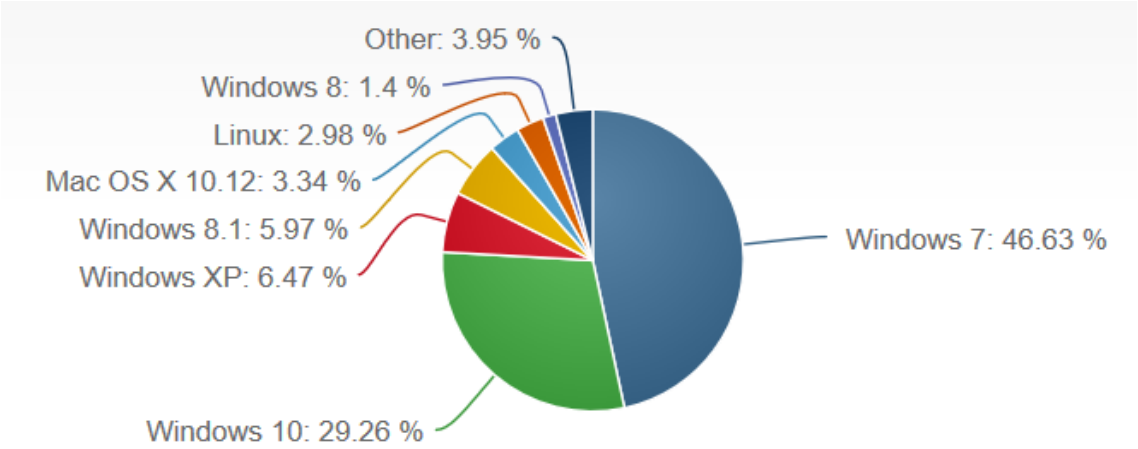


Noviembre de 2017:

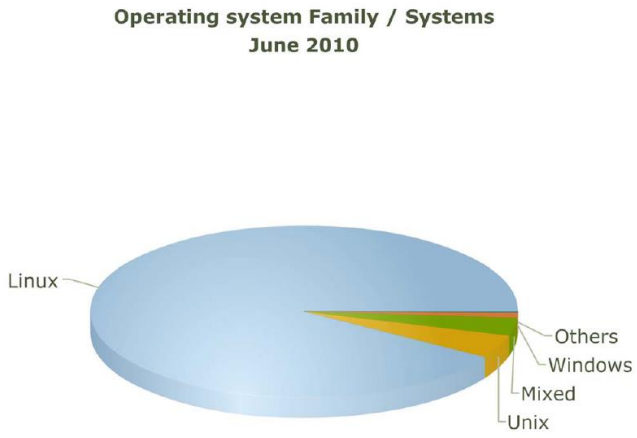
Por familia del sistema operativo:



Por versión del sistema operativo:

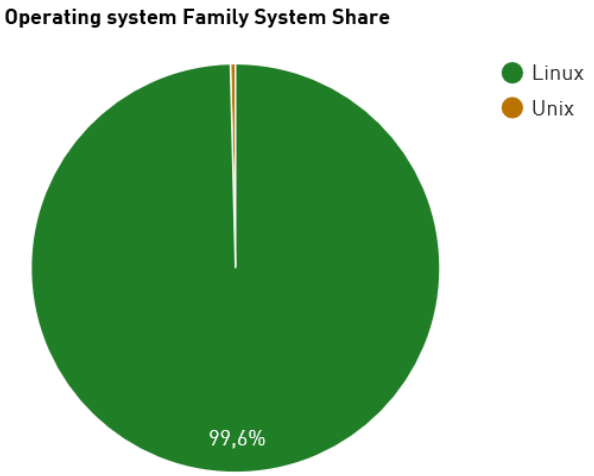


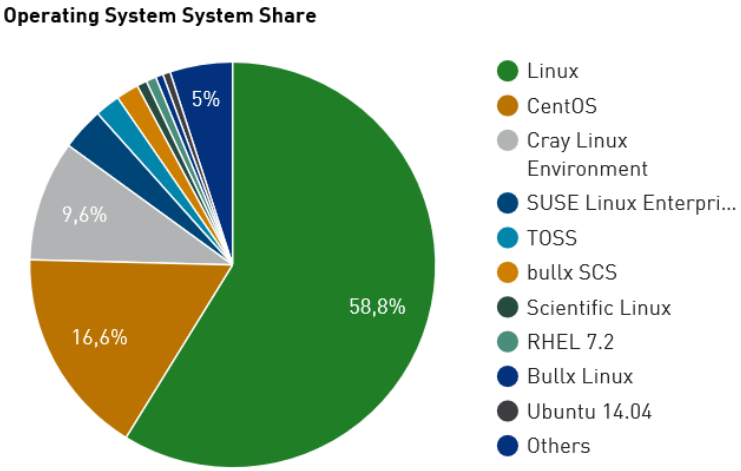
Otro dato curioso es la lista de los 500 ordenadores más grandes del mundo, que se publican en la web www.top500.org . En una tendencia que se mantiene desde hace tiempo, en el año 2010, el 95% estaban basados en derivados de Unix, y el 91% utilizaba Linux.



Operating system Family	Count	Share %
Linux	455	91.00 %
Windows	5	1.00 %
Unix	22	4.40 %
BSD Based	1	0.20 %
Mixed	17	3.40 %
Totals	**500**	**100%**

En el año 2017 la situación es la siguiente (https://www.top500.org/statistics/list/):





8 LICENCIAS DE SOFTWARE

Una **licencia de software** (en inglés *software license*) es la autorización o permiso concedido por el titular del derecho de autor, en cualquier forma contractual, al usuario de un programa informático, para utilizar éste en una forma determinada y de conformidad con unas condiciones convenidas.

La licencia, que puede ser gratuita o de pago, precisa los derechos (de uso, modificación o redistribución) concedidos a la persona autorizada y sus límites. Además, puede señalar el plazo de duración, el territorio de aplicación y todas las demás cláusulas que el titular del derecho de autor establezca.

Los distintos tipos de licencias están asociados a distintos tipos de software, que pasamos a ver a continuación.

8.1. Software libre

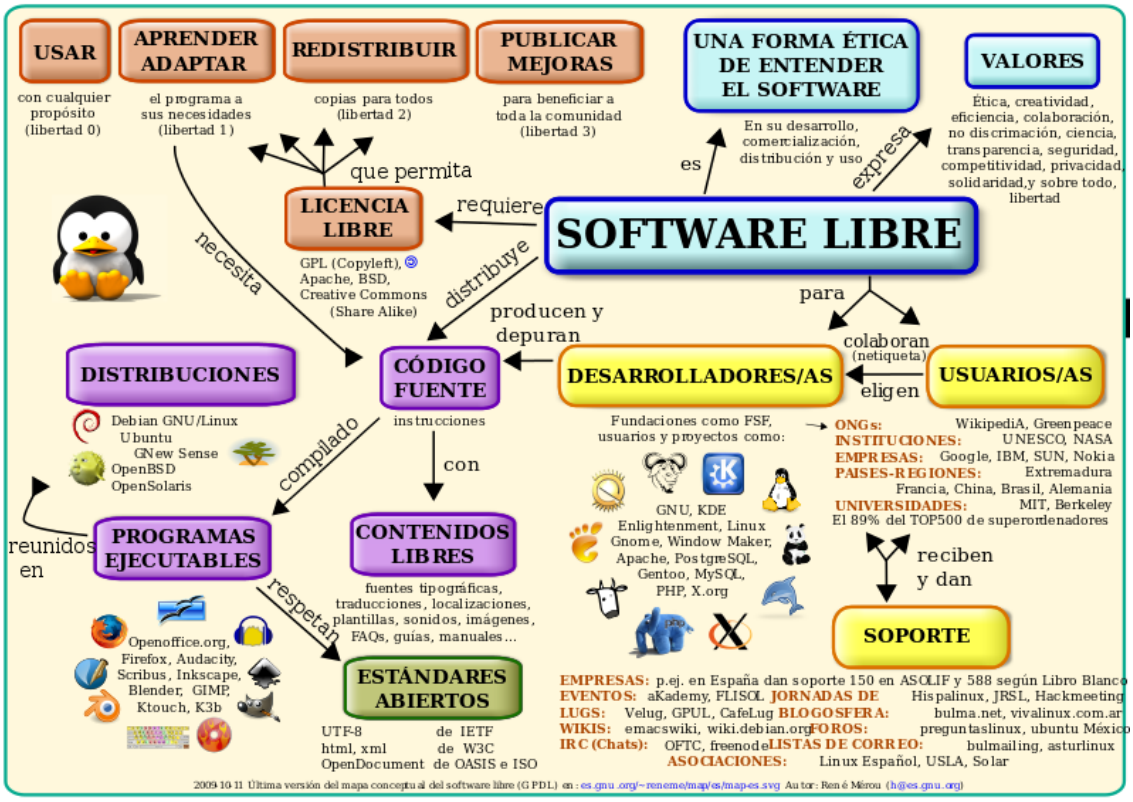
https://es.wikipedia.org/wiki/Software_libre

El término software libre se refiere al conjunto de software (programa informático) que, por elección manifiesta de su autor, puede ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras.

Su definición está asociada al nacimiento del movimiento de software libre, encabezado por Richard Stallman y la consecuente fundación en 1985 de la Free Software Foundation (www.fsf.org), que coloca la libertad del usuario informático como propósito ético fundamental. Proviene del término en inglés free software, que presenta ambigüedad entre los significados «libre» y «gratis» asociados a la palabra free. Por esto que suele ser considerado como software gratuito y no como software que puede ser modificado sin restricciones de licencia. En este sentido es necesario resaltar que la libertad tiene que ver con el uso y no con la gratuidad.

Un programa informático es software libre si otorga a los usuarios todas estas libertades de manera adecuada. De lo contrario no es libre.

El software libre suele estar disponible gratuitamente, o al precio de coste de la distribución a través de otros medios; sin embargo, no es obligatorio que sea así, por lo tanto no hay que asociar «software libre» a «software gratuito» (denominado usualmente freeware), ya que, conservando su carácter de libre, puede ser distribuido comercialmente. Análogamente, el software gratis o gratuito incluye en ocasiones el código fuente; no obstante, este tipo de software no es «libre» en el mismo sentido que el software libre, a menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa.



Autor: De René Mérou - http://es.gnu.org/~reneme/map/es/, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=8903504

Tampoco debe confundirse software libre con «software de dominio público». Este último es aquel que no requiere de licencia, pues sus derechos de explotación son para toda la humanidad, porque pertenece a todos por igual. Cualquiera puede hacer uso de él, consignando su autoría original. Este software sería aquel cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es del dominio público.

En palabras de Richard Stallman (https://es.wikipedia.org/wiki/Copyleft):

"La forma más simple de hacer que un programa sea libre es ponerlo en el dominio público, sin derechos reservados. Esto le permite compartir el programa y sus mejoras a la gente, si así lo desean. Pero le permite a gente no cooperativa convertir el programa en software privativo. Ellos pueden hacer cambios, muchos o pocos, y distribuir el resultado como un producto privativo. Las personas que reciben el programa con esas modificaciones no tienen la libertad que el autor original les dio; el intermediario se las ha quitado.

En el proyecto GNU, nuestro objetivo es el dar a todo usuario la libertad de redistribuir y cambiar software GNU. Si los intermediarios pudieran quitar esa libertad, nosotros tendríamos muchos usuarios, pero esos usuarios no tendrían libertad. Así en vez de poner software GNU en el dominio público, nosotros lo protegemos con Copyleft. Copyleft dice que cualquiera que redistribuye el software, con o sin cambios, debe dar la libertad de copiarlo y modificarlo más. Copyleft garantiza que cada usuario tiene libertad."

Libertades del Software libre

El software es "libre" si garantiza las siguientes libertades:

Libertad	Descripción
0	la libertad de **usar** el programa, con cualquier propósito (uso).
1	la libertad de **estudiar** cómo funciona el programa y modificarlo, adaptándolo a las propias necesidades (estudio).
2	la libertad de **distribuir** copias del programa, con lo cual se puede ayudar a otros usuarios (distribución).
3	la libertad de **mejorar** el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie (mejora).
Las libertades 1 y 3 requieren acceso al código fuente porque estudiar y modificar *software* sin su código fuente es muy poco viable.	

Una de las licencias de software libre más utilizadas es la **Licencia Pública General de GNU** (GNU GPL). El autor conserva los derechos de autor (*copyright*) y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que sea imposible crear un producto con partes no licenciadas GPL: el conjunto tiene que ser GPL.

Tipos de licencias: https://es.wikipedia.org/wiki/Software_libre#Tipos_de_licencias

8.2. Software de código abierto

https://es.wikipedia.org/wiki/Software_de_c%C3%B3digo_abierto

El software de código abierto (en inglés **open source** software u OSS) es el software cuyo código fuente y otros derechos que normalmente son exclusivos para quienes poseen los derechos de autor, son publicados bajo una licencia de código abierto o forman parte del dominio público.

En las licencias compatibles con la Open Source Definition el propietario de los derechos de autor permite a los usuarios utilizar, cambiar y redistribuir el software, a cualquiera, para cualquier propósito, ya sea en su forma modificada o en su forma original.

El movimiento del software libre surgió en 1983. En 1998, un grupo de personas defendieron la idea de cambiar la expresión *free software* (software libre) por *open source software* (software de código abierto), debido a la ambigüedad del primero (en inglés, free significa tanto gratis como libre) y al atractivo del segundo respecto a las empresas.

Algunas licencias aprobadas por la Open Source Initiative (opensource.org) son Apache, BSD, GPL o LGPL..

Puedes encontrar un listado de licencias en el siguiente enlace: https://opensource.org/licenses/category

8.3. Software libre y de código abierto

https://es.wikipedia.org/wiki/Software_libre_y_de_c%C3%B3digo_abierto

El software libre y de código abierto (también conocido como FOSS o FLOSS, siglas de free/libre and open source software, en inglés) es el software que está licenciado de tal manera que los usuarios pueden estudiar, modificar y mejorar su diseño mediante la disponibilidad de su código fuente.

El término "software libre y de código abierto" abarca los conceptos de software libre y software de código abierto, que, si bien comparten modelos de desarrollo similares, tienen diferencias en sus aspectos filosóficos. La principal diferencia entre los términos "open source" y "free software" es que este último tiene en cuenta los aspectos éticos y filosóficos de la libertad, mientras que el *open source* se basa únicamente en los aspectos técnico y en las ventajas de su modelo de desarrollo (como el beneficio que podía suponer para las empresas el que el código estuviera accesible a los ojos de muchas personas que podían mejorarlo). "FOSS" sería un término imparcial respecto a ambas filosofías.

El software gratis no necesariamente tiene que ser libre o de código abierto (o viceversa).

8.4. Software privativo

El software no libre es llamado también software propietario, software **privativo**, software privado y software con propietario. Se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), o que su código fuente no está disponible o el acceso a éste se encuentra restringido.

En las licencias asociadas, se protege al software contra uso, copia o redistribución. La empresa no vende en realidad el sistema operativo o la aplicación, sino una licencia de uso de los mismos. En el software no libre una persona física o jurídica (compañía, corporación, fundación, etc.) posee los derechos de autor sobre un software, negando la posibilidad de usar el programa con cualquier propósito; de estudiar cómo funciona el programa y adaptarlo a las propias necesidades (donde el acceso al código fuente es una condición previa); de distribuir copias o de mejorar el programa y hacer públicas las mejoras (para esto el acceso al código fuente es un requisito previo).

De esta manera, un software sigue siendo no libre aún si el código fuente es hecho público, cuando se mantiene la reserva de derechos sobre el uso, modificación o distribución (por ejemplo, la versión comercial de SSH o el programa de licencias Shared Source de Microsoft, https://www.microsoft.com/en-us/sharedsource/).

La familia Windows es un ejemplo de sistema operativo privativo.

Dentro de los sistemas operativos comerciales[6] y privativos, nos podemos encontrar con diversos tipos de licencia de uso:

8.4.1. O.E.M.

OEM (abreviatura del inglés Original Equipment Manufacturer, en español sería fabricante de equipamiento original). Este tipo de licencias se las otorga el desarrollador del sistema operativo al fabricante de hardware, de modo que cuando nosotros compramos uno de sus productos, este viene con una licencia de uso del sistema operativo de tipo OEM.

La particularidad de este tipo de licencias es el que el sistema operativo viene preparado para ese hardware específicamente, de modo que no tenemos realmente una licencia de uso del sistema operativo, sino una licencia de uso del sistema operativo únicamente para ese hardware en concreto.

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

8.4.2. Retail

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

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

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

8.4.3. VLM (licencias por volumen)

Para una empresa con cientos de ordenadores, es complicado controlar las licencias individuales de cada una de sus máquinas. Existe la posibilidad de contratar con el

[6] El software comercial es el desarrollado por una empresa que pretende ganar dinero por su uso.

desarrollador un tipo de licencia especial, de modo que, con una única clave de licencia, podemos utilizar varias máquinas a la vez. Es habitual que existan licencias de 25 usos concurrentes, 50, etc.

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

8.4.4. Licencias académicas

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

8.5. Más información

- Licencia de software: https://es.wikipedia.org/wiki/Licencia_de_software
- Comparación de licencias de software libre: https://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n_de_licencias_de_software_libre
- Aclaraciones sobre licencias GNU: https://www.gnu.org/licenses/gpl-faq.es.html
- Cómo elegir una licencia GNU: https://www.gnu.org/licenses/license-recommendations.html
- Análisis de las licencias de software libre (Miriam Ruiz): https://www.miriamruiz.es/slides/2008_CTIC_Doc_Licencias_de_Software_Libre.odt
- Free and open-source software: https://en.wikipedia.org/wiki/Free_and_open-source_software
- Free Software Foundation: http://www.fsf.org
- Open Source Initiative: https://opensource.org/
- Charla de Richard Stallman acerca del software libre: Free software, free society: Richard Stallman at TEDxGeneva 2014 https://www.youtube.com/watch?v=Ag1AKIl_2GM
- Otras licencias:
    - Creative Commons: https://es.wikipedia.org/wiki/Licencias_Creative_Commons

      https://creativecommons.org/share-your-work/licensing-types-examples/

9 BIBLIOGRAFÍA Y MATERIAL UTILIZADO

- MUÑOZ LÓPEZ, F. J., et alt. *Sistemas operativos en entornos monousuario y multiusuario*. Madrid: McGraw-Hill, 2005.
- STALLINGS, W. *Sistemas operativos. Aspectos internos y principios de diseño*. Madrid: Pearson Prentice Hall, 2005.
- RAYA GONZÁLEZ, L. *Sistemas Informáticos Monousuario y Multiusuario*. Ra-Ma: Paracuellos de Jarama, 2007.
- Apuntes de Raúl Escribano Alcaide sobre Sistemas Operativos.
- Apuntes de Víctor Marco Boix sobre Sistemas Operativos.
- Apuntes de José Antonio Carrasco Díaz sobre Sistemas Operativos.
- Wikipedia: es.wikipedia.org
- Comparación de Sistemas Operativos: http://es.wikipedia.org/wiki/Anexo:Comparación_de_sistemas_operativos
- Otro material de consulta: http://somebooks.es/conceptos-basicos-sobre-sistemas-operativos/#conte

10 AMPLIACIÓN: GESTIÓN DE ARCHIVOS

10.1. Métodos de asignación

Los distintos métodos existentes para asignar espacio a cada fichero dentro del disco es un aspecto totalmente transparente al usuario: al usuario no le interesa, o no tiene porqué interesarle, la forma en que se almacenan físicamente los ficheros. Esto sólo interesa al desarrollador del sistema operativo o al programador de sistemas que necesita estar en contacto con las peculiaridades físicas del dispositivo.

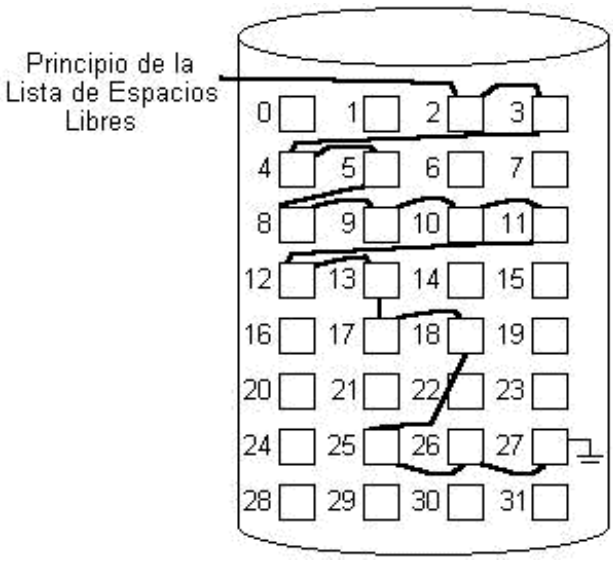
Para simplificar se entenderá que la unidad mínima de asignación de un bloque de datos equivale a un *cluster*.

Parece totalmente lógico que la asignación del espacio a los ficheros deba hacerse de modo que ésta sea tan eficiente como sea posible y que las operaciones a realizar sobre los ficheros sean rápidas. Contemplaremos tres métodos de asignación diferentes: la asignación contigua, la asignación enlazada y la asignación indexada. Algunos sistemas soportan los tres, pero lo más normal es que un sistema determinado sólo soporte uno. Para empezar, es conveniente conocer cómo se administra el espacio libre del disco.

10.2. Gestión del espacio libre

En un sistema informático, los ficheros se crean y se destruyen con frecuencia. Debido a que el espacio de disco no es ilimitado, se hace necesario reutilizar el espacio ocupado por ficheros que han sido borrados para almacenar nuevos ficheros. Dos métodos de gestión de *clusters* libres es mediante un mapa de bits o con punteros a *clusters*.

Con el método **de punteros a *cluster*s** el sistema sólo mantiene un puntero al primer *cluster* libre, este primer *cluster* libre apunta al siguiente, y así sucesivamente hasta llegar al último *cluster* libre. Es en realidad una lista encadenada.

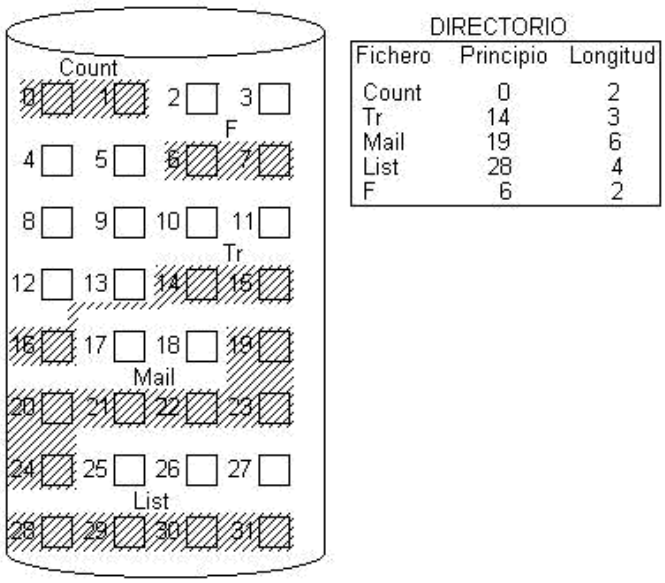


En un **mapa de bits** (también llamado vector de bits o, en inglés, bitmap), cada *cluster* del disco se representa por medio de un bit, que estará puesto a un valor lógico concreto si el bloque está asignado a algún fichero y a su complementario cuando el bloque esté libre. En el disco del ejemplo mostrado en la figura anterior, estaban libres los bloques 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 y 27; el mapa de bits de espacios libres correspondiente sería (suponiendo que el valor lógico 1 indica los bloques ocupados):

11000011000000111001111100011111

10.3. Gestión del espacio ocupado

El método de **asignación contigua** funciona de forma que cada fichero ocupe un conjunto de bloques consecutivos en el disco. Con la asignación contigua, la situación de un fichero queda perfectamente determinada por medio de la dirección de su primer *cluster* y su longitud. En la figura podemos ver un ejemplo de este tipo de asignación.



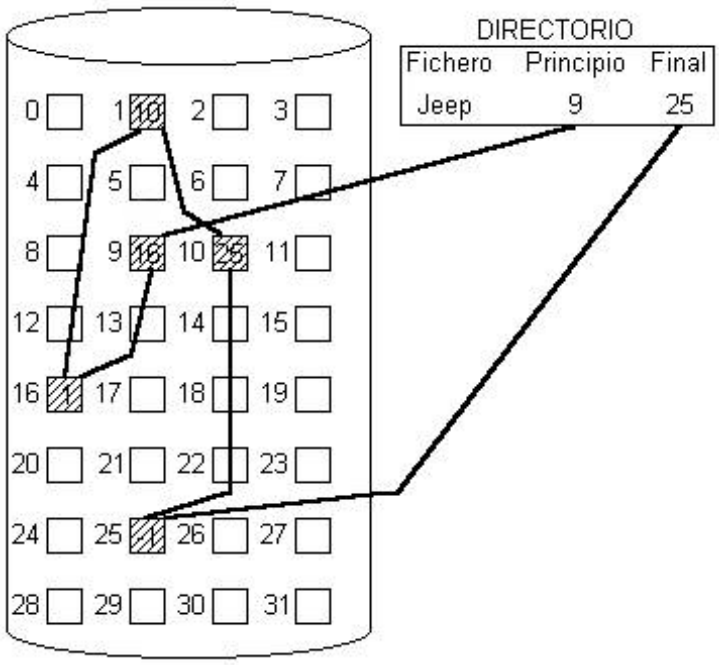
Como puede verse, el acceso a un fichero cuyo espacio ha sido asignado de forma contigua es bastante sencillo. El verdadero problema aparece a la hora de encontrar espacio para un nuevo fichero. Si el fichero a crear debe tener n *clusters* de longitud, habrá que localizar n bloques consecutivos en la lista de bloques libres. Así, en la figura anterior sería imposible almacenar un fichero que tuviera un tamaño de 7 *clusters*, aunque en realidad tenemos libres 15 *clusters*. Se ha producido fragmentación externa.

La fragmentación externa se puede resolver, una solución se llama compactación. El único coste es el tiempo empleado en llevar a cabo esta compactación. La asignación contigua presenta algunos otros problemas como que esta estrategia parte del

hecho de conocer el espacio máximo que ocupará un fichero en el instante mismo de su creación, algo que es imposible de predecir.

Concluyendo, la asignación contigua es muy eficiente a la hora de acceder a los ficheros, pero es excesivamente engorrosa a la hora de asignar el espacio a nuevos ficheros.

Podríamos pensar en no asignar el espacio de forma contigua. La **asignación enlazada** podría ser la estrategia elegida. Siguiendo este esquema, cada fichero no es más que una lista enlazada de bloques, que pueden encontrarse en cualquier lugar del disco. La entrada del directorio posee únicamente un puntero al primer *cluster* y un puntero al último. Cada *cluster*, a su vez, contendrá un puntero al siguiente *cluster*. En la figura siguiente podemos ver un fichero almacenado siguiendo esta estrategia. El fichero ocupa 5 *clusters* por este orden: 9, 16, 1, 10 y 25:

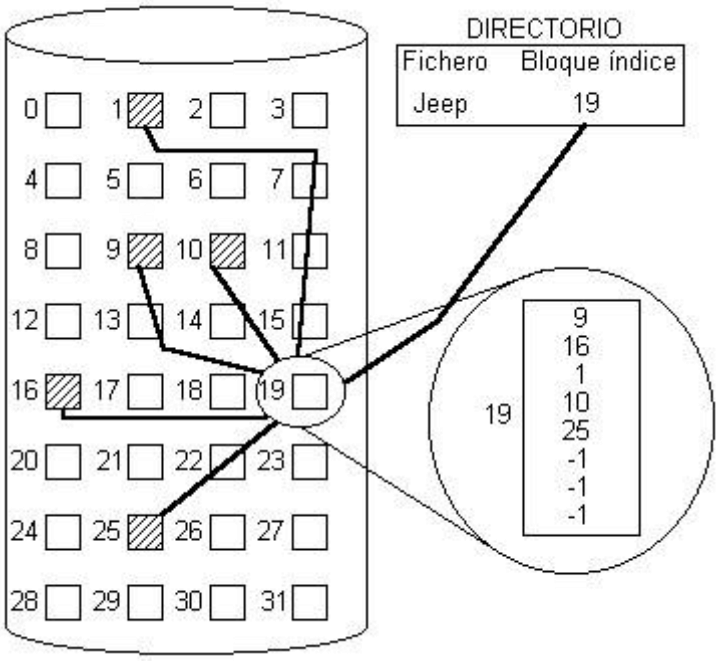


La asignación enlazada no provoca fragmentación externa, pues cualquier bloque de la lista de *clusters* libres es candidato a ser ocupado: basta con coger el primer *cluster* que se encuentre. Tampoco es preciso conocer el tamaño de un fichero en el momento de su creación: el fichero puede crecer mientras existan *clusters* libres en el disco

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

Otra desventaja reside en el espacio desaprovechado para almacenar los punteros y en el aspecto de la fiabilidad del sistema: si, por cualquier causa, se dañara un único puntero en un bloque asignado a un fichero, el resto del fichero sería ilocalizable.

La **asignación indexada** viene a resolver estos problemas reuniendo a todos los punteros en un mismo lugar: el bloque índice. A cada fichero le corresponde su propio bloque índice, que no es más que una tabla de direcciones de *clusters*, donde la entrada i apunta al *cluster* i del fichero. La entrada de un fichero en el directorio sólo necesita mantener la dirección del bloque índice para localizar todos y cada uno de sus *clusters*, como puede verse en la figura siguiente:



Ahora sí se permite el acceso directo, pues el acceso al bloque i sólo exige emplear la i-esima entrada del bloque índice. Como se ve, la asignación indexada soporta el acceso directo sin provocar fragmentación externa. Cualquier bloque libre en cualquier lugar del disco puede satisfacer una solicitud de espacio.

Sin embargo, también hay desventajas: los bloques índices son también *clusters* de disco, y dejan de estar disponibles para almacenar datos. Como la mayoría de los ficheros existentes en un sistema son pequeños, el despilfarro puede ser bastante considerable, pues se exige reservar todo un *cluster* como índice para cada fichero, aunque éste sólo ocupe dos *clusters*.

Como alternativa, se puede emplear una estructura arborescente en dos o tres niveles. Un fichero pequeño sólo emplearía direcciones del primer nivel. A medida que los ficheros crecen puede acudirse a los niveles segundo y tercero. El acceso a un bloque de datos exige acceder al puntero del primer nivel contenido en el bloque índice primario, el cual apunta a su vez a otro bloque índice secundario, etc.

Otra alternativa consiste en mantener los primeros punteros del bloque índice, unos 15, en el directorio de dispositivo. Si un fichero requiere de más de 15 *clusters*, un décimo sexto puntero apunta a una lista de bloques índice. De esta manera, los ficheros pequeños no precisan de un bloque índice. El antiguo sistema de ficheros MINIX de las primeras versiones de UNIX/LINUX empleaba un esquema análogo a éste.

10.4. Algunos sistemas de ficheros

10.4.1. Sistema de ficheros FAT

El sistema de ficheros FAT trabaja sobre la Tabla de Localización de Ficheros (File Allocation Table) de la cual toma su nombre. Cada volumen lógico tiene su propia FAT, que sirve para dos importantes funciones: contener la información de localización para cada fichero en el volumen, en forma de **listas enlazadas** de unidades de *clusters* e indicar qué *clusters* están libres para asignar a un fichero que está creándose o expandiéndose.

Cuando el sistema de ficheros FAT16 fue concebido, era una solución excelente al manejo del disco ya que los disquetes en los que se usaba eran raramente más grandes de 1Mb. En esos discos, la FAT, era suficientemente pequeña para mantenerse en memoria todo el tiempo, permitiendo un acceso aleatorio muy rápido a cualquier parte de un fichero.

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

Si, por ejemplo, se hace que el tamaño del clúster sea de 8 KB (16 sectores), se pueden manejar discos de tamaños hasta los 512 MB (65.536 * 8 / 1024). El problema es que todos los tamaños de los ficheros deben redondearse a un número entero de *clusters*, y 8 KB es un tamaño bastante grande. Esto implica que, por ejemplo, un

pequeño fichero de procesamiento por lotes con unas pocas docenas de caracteres ocupe 8 KB de espacio en el disco y en el disco puede haber un gran número de ficheros, por lo que el espacio desperdiciado es muy grande.

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

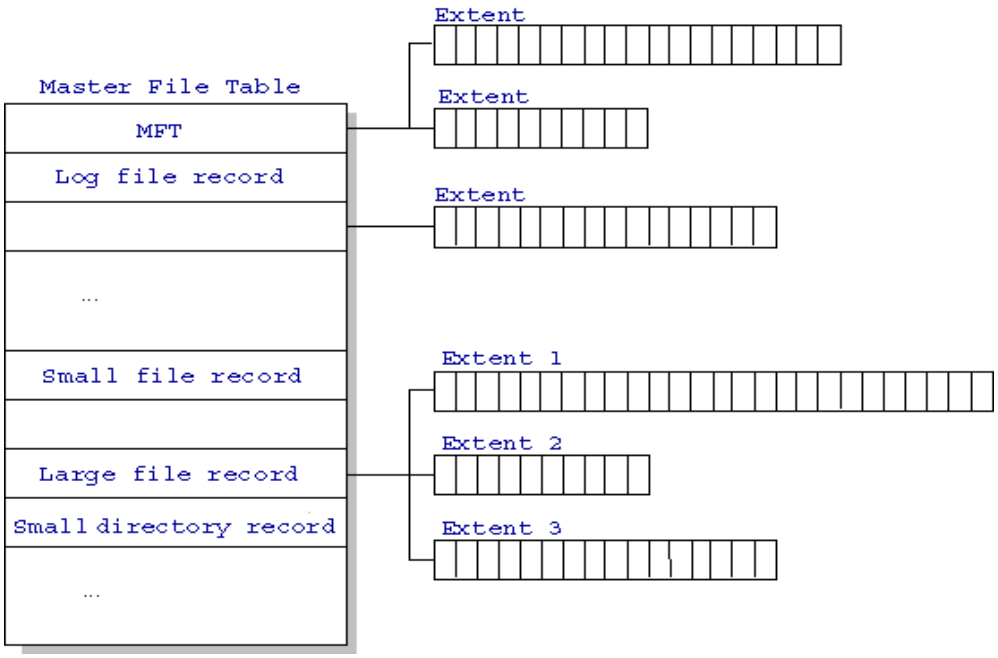
El principal cambio de una FAT a otra radica en la ampliación del tamaño de las entradas, que pasa de 16 a 32 bits. Esto permite mucho más espacio de disco, otra de las ventajas es la flexibilidad y robustez del sistema, ya que FAT32 usa *clusters* más pequeños, en concreto de 4Kb, por lo que la eficiencia aumenta y el espacio desperdiciado disminuye.

10.4.2. Sistema de ficheros NTFS

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

Cada fichero en un volumen NTFS está representado por un registro en un fichero especial llamado tabla de fichero maestro (MFT: Master File Table). NTFS reserva los 16 primeros registros de la tabla para información especial. El primer registro describe la propia MFT, seguido por un registro espejo (otras copia de la MFT). Si el primer registro MFT es erróneo, NTFS lee el segundo registro para encontrar el fichero espejo, cuyo contenido es idéntico al del primer registro.

Las localizaciones de los segmentos de datos para ambos (MFT y MFT espejo) están grabadas en el sector de arranque (Boot). En el centro lógico del disco está almacenado un duplicado del Boot. El tercer registro del MFT es el fichero de Log, usado para recuperación de ficheros. A partir del registro 16 de la MFT, los registros son para cada fichero y directorio del volumen. La siguiente figura proporciona una visión simplificada de la estructura del MFT:



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

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

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

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

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

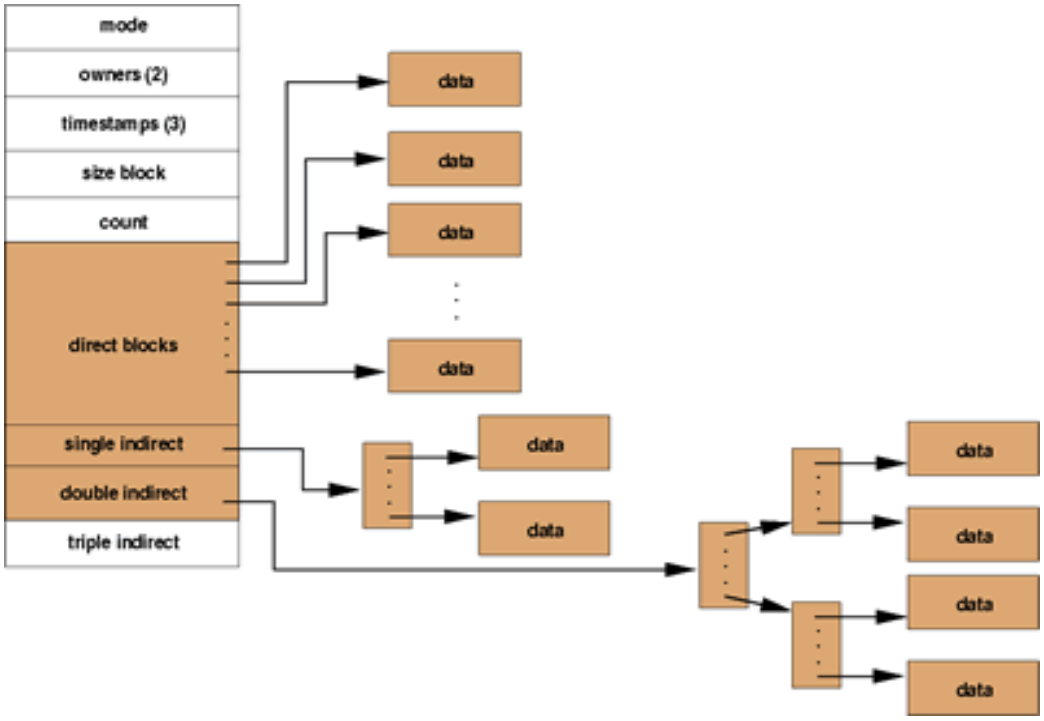
Por ejemplo, un flujo de datos asociado a un fichero en NTFS es el flujo de la ACL (Lista de Control de Acceso). En este flujo se apuntan todos los identificadores de usuarios y grupos de usuarios que tienen permiso para usar dicho fichero, así como que tipo de permiso tienen. Esta ACL puede tener un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

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

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

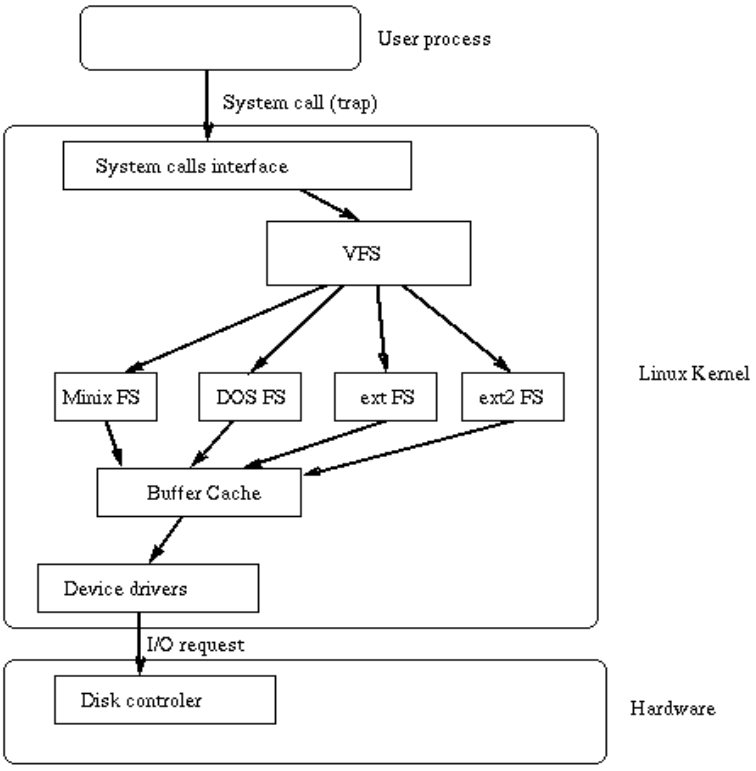
10.4.3. Sistemas de ficheros para Linux

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



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

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



Este inodo de Ext2 tiene un tamaño fijo entre 1 y 4 K, y no usa una FAT, sino una tabla de inodos distribuidos en un número determinable de grupos a través de la superficie, tiene un límite máximo (usando bloques de 4 kB) de 4 TB por archivo, 16 TB por partición y detección de desmontaje incorrecto. Sin embargo, su gran handicap es que no permite usar "journaling" (registro por diario). El journaling se basa en llevar

un registro de todas las transacciones de lectura/escritura que sufre un sistema de archivos.

Por la imposibilidad de usar journaling en Ext2, se ha creado el sistema de ficheros Ext3, que es una implementación de Ext2 con journaling. A parte de ext2 y ext3, gracias al VFS de Linux podemos usar diversos sistemas de ficheros, como ReiserFS, Reiser4, UFS, XFS, JFS, FAT, UFS, UMSDOS, ZFS...

10.5. Planificación de las solicitudes de acceso

Otro factor importante, además de la asignación del espacio en disco es la eficiencia con que se escribe o recupera la información del disco. El sistema operativo puede mejorar el tiempo promedio de servicio del disco planificando las solicitudes del acceso.

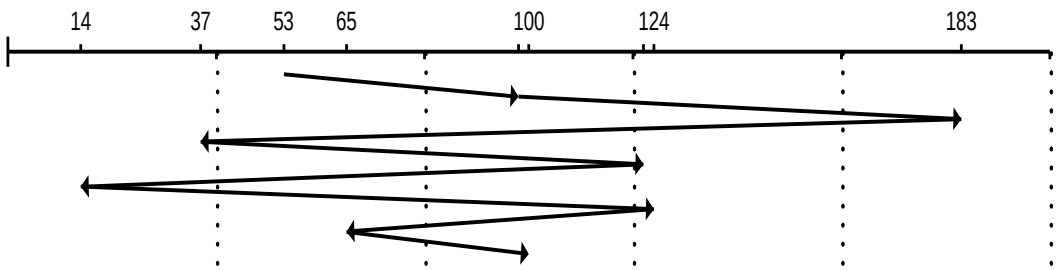
La velocidad del disco se compone de tres partes. Para acceder a un bloque en el disco, el sistema primero debe mover la cabeza a la pista o cilindro correspondiente. A este movimiento se le denomina posicionamiento, una vez que la cabeza se encuentra en la pista correcta debe esperar a que el bloque deseado pase por debajo de la cabeza de lectura y escritura; a esta demora se le conoce como tiempo de latencia. Por último, puede efectuarse la transferencia real de los datos entre el disco y la memoria principal. Esta última espera es el tiempo de transferencia. El tiempo total para servir una solicitud del disco es la suma de los tiempos de posicionamiento, latencia y transferencia.

Si la unidad y el controlador del disco deseado están disponibles, es posible servir de inmediato la solicitud. Sin embargo, si la unidad o el controlador están ocupados sirviendo una solicitud, será necesario colocar en una cola todas las solicitudes adicionales, que generalmente provienen de otros procesos. Algunos de los algoritmos de planificación del disco son los siguientes:

FCFS (First Come, Firs Served): la forma más sencilla para planificar un disco es la planificación de servicio por orden de llegada. Este algoritmo es fácil de programar, aunque normalmente no ofrece los mejores tiempos de respuesta.

Ejemplo

Supongamos una cola de peticiones a disco que afectan a las pistas: 98, 183, 37, 122, 14, 124, 65 y 100. Si en un principio, la cabeza de lectura y escritura está en la pista 53, se tendrá la siguiente situación:

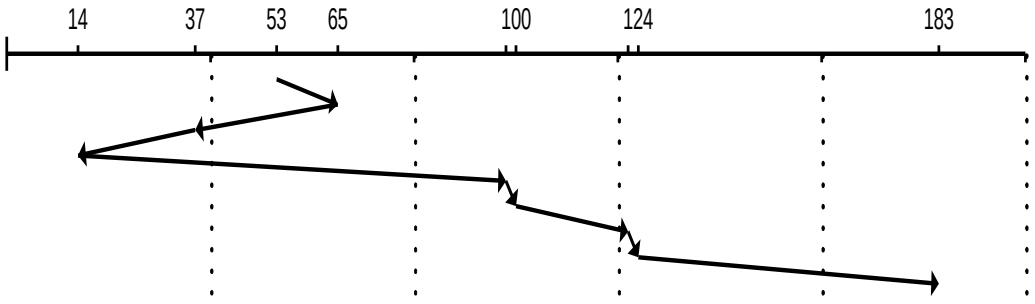


Atender las solicitudes de la correspondiente cola implica un movimiento total de la cabeza del disco de 673 pistas. Nótese que se ha asumido que el disco posee 200 pistas. El problema principal con FCFS es que al no tomarse en cuenta las pistas que serán accedidas, puede forzarse a la cabeza lectora a moverse de un extremo a otro, reduciéndose la productividad del disco.

SSTF (Shortest Seek Time First): el algoritmo **primero la de menor tiempo de posicionamiento** esta basado en la idea de servir primero las solicitudes más próximas a la posición actual del disco.

Ejemplo

Si la cabeza está inicialmente en la pista 53 y se tienen la siguiente lista de solicitudes: 98, 183, 37, 122, 14, 124, 65 y 100. Estas serán atendidas en el siguiente orden: 65, 37, 14, 98, 100, 122, 124 y 183.



Lo cual da como resultado un total de 232 pistas recorridas

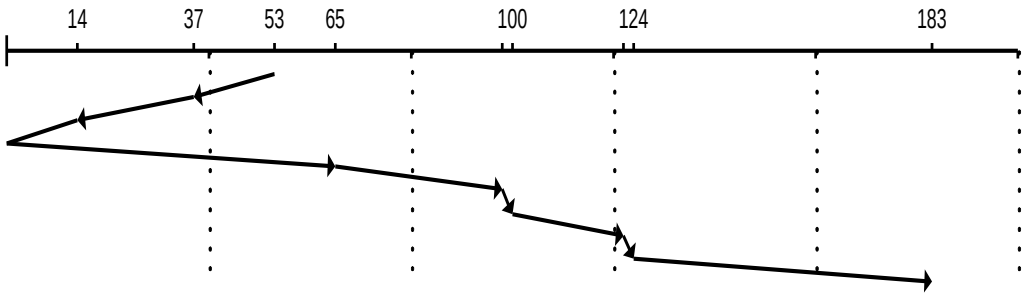
SCAN: la cabeza de lectura y escritura comienza en un extremo del disco y se desplaza hacia el otro lado sirviendo las solicitudes al llegar a cada pista, hasta

alcanzar el extremo opuesto. Luego se invierte la dirección del movimiento de la cabeza y continúa el servicio, rastreando continuamente el disco de un extremo al otro.

Ejemplo

La cabeza se encuentra en la pista 53, se mueve hacia el extremo inferior y se reciben las solicitudes: 98, 183, 37, 122, 14, 124, 65 y 100

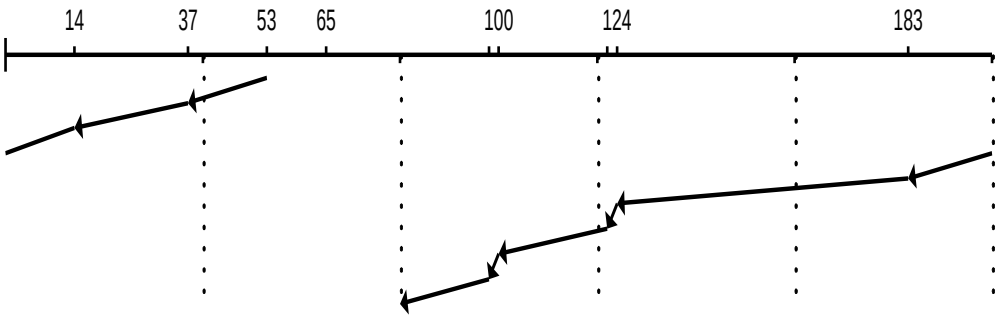
Entonces estas serán atendidas en el orden 37, 14, 65, 98, 100, 122, 124 y 183, teniéndose un movimiento de la cabeza lectora de 236 pistas.



C-SCAN: es una variante de la planificación SCAN que al igual que éste mueve la cabeza de un extremo al otro del disco; sin embargo, cuando llega al extremo opuesto, regresa de inmediato al inicio del disco, sin servir ninguna solicitud en el camino, considerando al disco como circular.

Ejemplo.

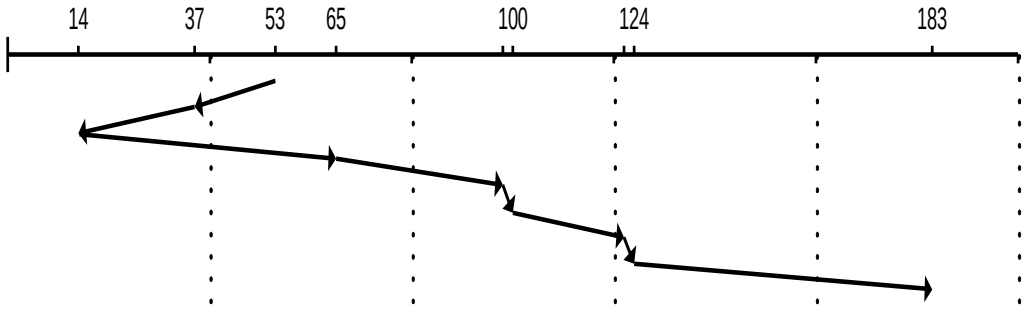
La cabeza se encuentra en la pista 53, se mueve hacia el extremo inferior y se reciben las solicitudes: 98, 183, 37, 122, 14, 124, 65 y 100



LOOK: los planificadores SCAN y C-SCAN siempre mueven la cabeza hasta el extremo del disco. Si la cabeza se mueve hasta la última solicitud en cada dirección, se obtiene el algoritmo LOOK, esto es, mira si hay una solicitud antes de moverse en esa dirección.

Ejemplo

La cabeza se encuentra en la pista 53, se mueve hacia el extremo inferior y se reciben las solicitudes: 98, 183, 37, 122, 14, 124, 65 y 100

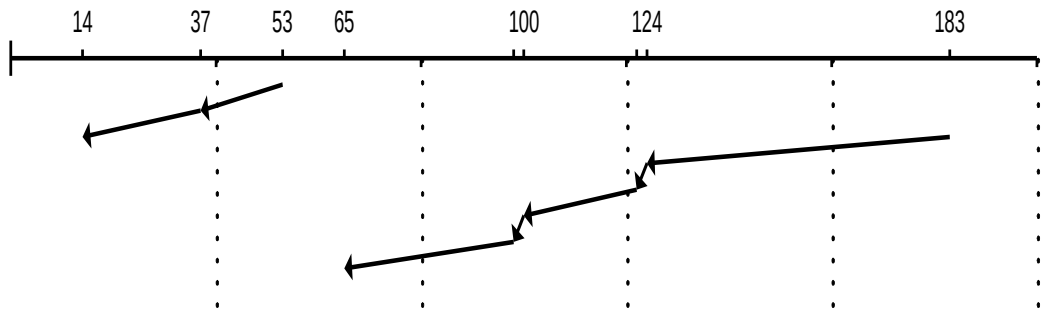


Para un total de 208 pistas recorridas.

C-LOOK: es similar a LOOK, pero manejando el recorrido de forma circular, es decir en un único sentido.

Ejemplo

La cabeza se encuentra en la pista 53, se mueve hacia el extremo inferior y se reciben las solicitudes: 98, 183, 37, 122, 14, 124, 65 y 100



-