

Capítulo 2

Estructura del Núcleo (Kernel)

2.1 Introducción

El Sistema Operativo UNIX se compone del **núcleo** (Kernel) y algunos programas de sistema. El núcleo es el corazón del sistema operativo¹. Controla los ficheros en los discos, ejecuta programas y multiplexa el procesador, y demás hardware, entre estos programas para proporcionar multitarea, asigna memoria y otros recursos a los procesos, recibe y envía paquetes a la red, etc. El núcleo proporciona las herramientas con las cuales se pueden realizar todos estos servicios e impide que nadie pueda acceder al hardware directamente, obligándole a usar estas herramientas. De esta forma el núcleo puede controlar quién hace cada cosa y proteger a unos usuarios de otros. Las herramientas proporcionadas por el kernel se usan a través de las **Llamadas al sistema**².

Los programas de sistema usan las herramientas proporcionadas por el núcleo para implementar los distintos servicios necesarios en el sistema operativo. Los programas de sistema y los programas de aplicación se ejecutan sobre el núcleo en **modo usuario**. La diferencia entre un programa de sistema y un programa de aplicación radica en su finalidad: un programa de sistema tiene como fin conseguir que el sistema funcione, mientras que un programa de aplicación se utiliza para obtener resultados finales para el usuario. Por ejemplo **vi** es un programa de aplicación, mientras que **telnet** es un programa de sistema.

Las partes más importantes del núcleo son: el gestor de sistemas de ficheros, los ficheros de dispositivos (Devices), el gestor de procesos, el gestor de memoria y el gestor de red, como se puede apreciar en la figura 2.1.

¹De hecho es considerado a veces, erróneamente, el sistema operativo en si mismo. Un sistema operativo proporciona muchos más servicios que los del núcleo.

²Son puntos de entrada al núcleo que pueden ser invocados desde programas de usuario.

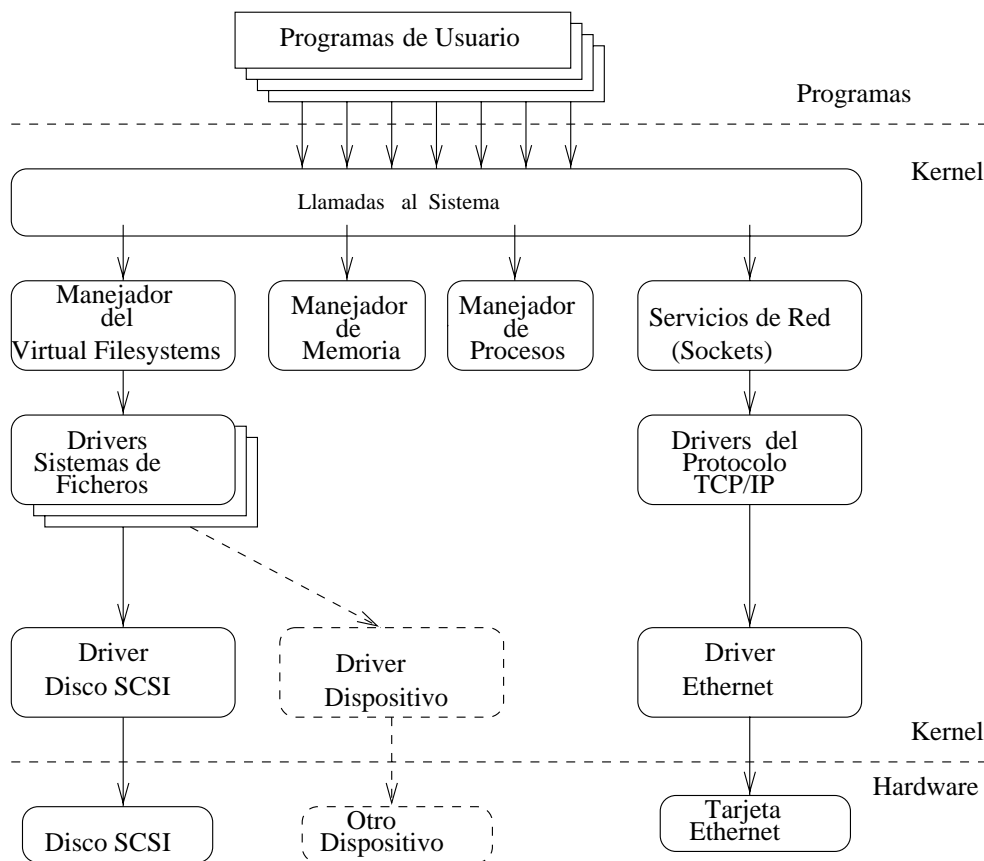


Figura 2.1: Estructura del núcleo de UNIX .

2.2 Sistemas de Ficheros

En un sistema UNIX la información se guarda en ficheros, los cuales a su vez están organizados dentro de lo que se conoce como **Sistemas de Ficheros**. Un sistema de ficheros es simplemente una estructura situada en un área del disco llamada **partición** o **sección**. Puede haber, por tanto, varios sistemas de ficheros en un mismo disco (en varias particiones o secciones de él). También, al contrario, puede ocurrir, al menos en sistemas UNIX más recientes, que un sistema de ficheros esté montado sobre varias unidades de disco, dando lugar a lo que se llama **sistemas de ficheros multivolumen**.

2.2.1 Estructura de los sistemas de ficheros

La mayoría de los sistemas de ficheros UNIX tienen una estructura parecida, lo que varía de unos a otros son los detalles de la implementación, lo que los hará más o menos eficientes. En cualquier caso los conceptos principales en torno a la estructura de un sistema de ficheros son: **superbloque**,

inodos, bloques de datos, bloques de directorio y bloques de datos indirectos.

2.2.1.1 Estructura del sistema de ficheros HFS (hp-ux), basado en BSD 4.2

Las características más destacables de los sistemas de ficheros HFS son:

- Un sistema de ficheros está compuesto de un superbloque principal y uno o más **grupos de cilindros**.
- Cada grupo de cilindros tiene una copia de la información estática del superbloque principal.
- Cada grupo de cilindros tiene una tabla de información del grupo de cilindros.
- Cada grupo de cilindros tiene su propia área de inodos, área de bloques de datos y tablas de información.
- Los bloques de datos se pueden dividir en fragmentos.
- Varios ficheros pueden compartir el mismo bloque de datos.

Una aproximación gráfica elemental a esta estructura sería la de la figura 2.2.

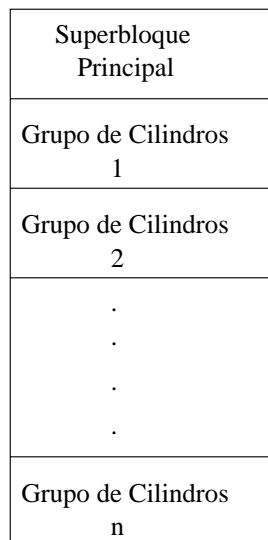


Figura 2.2: Estructura general del sistema de ficheros HFS.

2.2.1.1.1 El superbloque

La función del superbloque es controlar todo el sistema de ficheros HFS. Cualquier información que el sistema necesite conocer sobre el sistema de ficheros la obtendrá del superbloque. Contiene información estática que se conoce cuando se crea el sistema de ficheros como:

-
- Tamaño del sistema de ficheros.
 - Tamaño del bloque.
 - Tamaño del fragmento.
 - Características del disco.

El superbloque también mantiene información actualizada sobre el sistema de ficheros como:

- Número total de bloques libres.
- Número total de inodos libres.
- Flag de limpieza del sistema de ficheros.

Los datos del superbloque son cruciales. Consecuentemente el sistema mantiene un superbloque principal al principio del sistema de ficheros, y una copia del mismo en cada grupo de cilindros. Estas copias son sólo necesarias si el superbloque principal se pierde, daña o corrompe.

En `/usr/include/sys/fs.h` se encuentra la declaración en lenguaje C del superbloque.

2.2.1.1.2 El Grupo de Cilindros

Una aproximación gráfica elemental a la estructura de un grupo de cilindros sería la de la figura 2.3.

Copia del Superbloque
Informacion del Grupo de Cilindros
Tabla de Inodos
Bloques de Datos
·
·
·
·

Figura 2.3: Estructura de un grupo de cilindros.

Un grupo de cilindros es una colección de varios cilindros consecutivos. Cada grupo de cilindros tiene una copia del superbloque, una tabla de información del grupo de cilindros, una tabla de inodos

y uno o más bloques de datos. El grupo de cilindros controla todos los accesos a un fichero y sus datos asociados.

Se reservan 8 Kbytes de cada grupo de cilindros para una copia del superbloque. Dado que el superbloque contiene principalmente información estática, estas copias se generan cuando se crea el sistema de ficheros. Estas copias del superbloque no necesitan actualizarse regularmente ya que generalmente sólo se usan cuando el superbloque principal se pierde o corrompe y se necesita reconstruir el sistema de ficheros. Si hubiera necesidad de hacer esto, se pueden calcular los valores actuales que reflejan el estado del sistema de ficheros y se puede actualizar el superbloque principal de acuerdo a esos valores.

La tabla de información del grupo de cilindros contiene:

- El número de inodos y bloques de datos.
- Punteros al último bloque, fragmento e inodo utilizado.
- El número de fragmentos disponibles.
- El mapa de los inodos utilizados.
- El mapa de los fragmentos libres.

La tabla de inodos contiene entradas para un conjunto de inodos. Los inodos contienen información sobre ficheros individuales. El número de inodos asignados por grupo de cilindros se determina cuando se crea el sistema de ficheros y no se puede cambiar una vez creado.

La información de los directorios se mantiene dentro de los bloques de datos y no ocupa una región distinta dentro del grupo de cilindros. Cualquier bloque de datos no utilizado se puede utilizar para contener los datos de un directorio.

2.2.1.1.3 La tabla de inodos

La tabla de inodos contiene todos los inodos de un grupo de cilindros. Se accede a un fichero a través de su correspondiente inodo, el cual contiene toda la información del fichero. Consecuentemente todos los ficheros de un sistema de ficheros tienen asociado un inodo. Los inodos se mantienen dentro de una tabla de inodos en un grupo de cilindros. En el momento en que se crea el sistema de ficheros se determina el número máximo de ficheros que soportará ese sistema de ficheros.

Se accede muy a menudo a la tabla de inodos dado que el inodo es el único lugar del sistema de ficheros donde se almacena la información de los bloques que utiliza un fichero. La naturaleza secuencial de la tabla de inodos hace que el acceso sea sencillo. La tabla de inodos se sitúa físicamente en el mismo grupo de cilindros inmediatamente después de la estructura de información del grupo de cilindros. Los inodos se direccionan a partir de su posición relativa en la tabla de inodos. Por ejemplo, dado que el inodo tiene una longitud de 128 bytes, el inodo número 5 de la lista comenzará en el byte número 512 a partir del principio de la tabla de inodos, y dado que la estructura de información del grupo de cilindros conoce la longitud de la tabla de inodos, es fácil determinar cuándo se ha alcanzado el final de la misma.

2.2.1.1.4 Estructura de un inodo

Una aproximación gráfica a la estructura de un inodo sería la de la figura 2.4.

Tipo del fichero y Permisos
Numero de enlaces del fichero
Informacion del propietario y grupo
Tamaño del fichero en bytes
Fechas (acceso, modificacion y mod. inodo)
Bloque directo 1
Bloque directo 2
⋮

⋮
Bloque directo 12
Bloques de indireccion simple 13
Bloques de indireccion doble 14
Bloques de indireccion triple 15

Figura 2.4: Estructura de un inodo.

El inodo de un fichero contiene toda la información correspondiente a ese fichero. En concreto un inodo contiene:

- El tipo de fichero (regular, directorio, especial).
- Los permisos del fichero.
- El número de enlaces del fichero.
- El propietario y el grupo del fichero.
- Fechas del último acceso, última modificación del fichero y última modificación del inodo.
- El tamaño del fichero en bytes.
- 12 direcciones a 12 bloques de datos en disco.
- Un puntero a un bloque de 2048 direcciones de bloques de datos (Indirección simple).
- Un puntero a un bloque de 2048 punteros a 2048 direcciones de bloques de datos cada uno (Indirección doble).
- Un puntero a un bloque de 2048 punteros cada uno de los cuales apunta a otro bloque de 2048 punteros que a su vez apuntan a un bloque de 2048 direcciones de bloques de datos (Indirección triple).

En la figura 2.5 se representa el acceso a los bloques de datos de un fichero, tanto a los accedidos de forma directa, como a los que se hace a través de indirección simple, doble o triple.

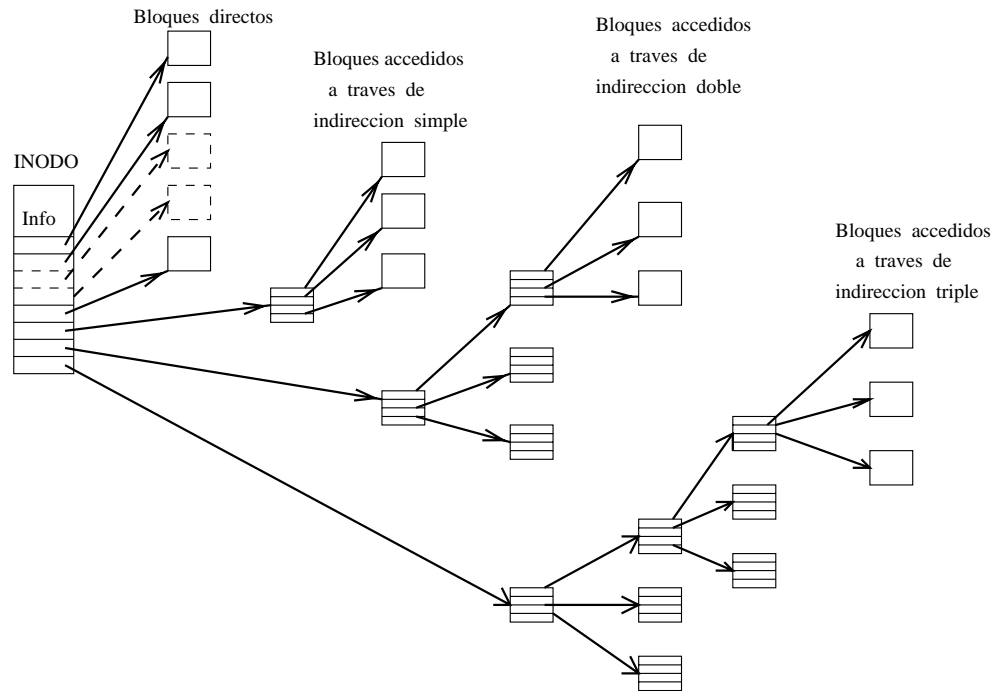


Figura 2.5: Acceso a los bloques de datos de un fichero.

El inodo contiene slots para 15 direcciones: 12 de acceso directo a los bloques de datos y 3 de acceso a través de indirección simple, doble y triple. Estas direcciones son de acceso a los bloques de datos o bien a los fragmentos de datos. Cuando se crea un fichero o bien cuando este crece, se realiza una entrada en uno de estos slots, y cuando el fichero decrece se desasignan los correspondientes fragmentos de datos, eliminándose las correspondientes direcciones del inodo tan pronto como se libera un bloque de datos.

La declaración en lenguaje C del inodo se encuentra en `/usr/include/sys/ino.h`. Esta declaración corresponde a la copia en disco de un inodo. Por cada fichero abierto en el sistema, el sistema mantiene en memoria una copia del inodo del mismo. Esta copia en memoria tiene una información ligeramente diferente de la existente en el disco y es la definida en `/usr/include/sys/inode.h`.

2.2.1.1.5 Estructura del directorio

Un directorio se comporta como un fichero ordinario en un sistema de ficheros, con la excepción de que un usuario no puede escribir en él directamente. La función de un directorio es situar un nivel por debajo de él a todos los ficheros y/o directorios que contenga, y para realizar esto, el directorio tiene una entrada por cada fichero y/o directorio que contiene. Esta entrada consta de 4 campos:

-
- Un número de inodo binario (4 bytes).
 - Longitud de la entrada en el directorio (2 bytes).
 - La longitud del nombre del fichero (2 bytes).
 - El nombre del fichero (14 bytes para nombres cortos o 255 para nombres largos).

Para el caso de nombres cortos el sistema de ficheros HFS utiliza 32 bytes para cada entrada de directorio. Así por ejemplo si creamos un nuevo fichero llamado *fich1* dentro de un cierto directorio, el contenido de la nueva entrada de este directorio será:

27 32 5 fich1

suponiendo que 27 sea el primer inodo libre en la tabla de inodos del grupo de cilindros correspondiente.

Cuando se crea un directorio por primera vez, contiene dos entradas: "." y "..", y se utilizan para direccionamiento relativo de paths. La entrada "." contiene el número de inodo de este directorio, mientras que la entrada ".." contiene el número de inodo del directorio padre. La única excepción ocurre cuando se trata del directorio raíz de cualquier sistema de ficheros. En este caso el número de inodo será "2" y dado que no tiene ningún ancestro permanente el número "2" se coloca tanto en la entrada "." como en la "..". Esto también será válido para el directorio raíz de todos los sistemas de ficheros.

Cuando se crea un fichero, el nombre del fichero y el número del inodo del fichero se colocan en el primer slot del directorio que tenga un cero en el campo del número del inodo. Cuando se borra un fichero, se pone un cero en el campo del número del inodo. Esto significa que ahora el slot está vacío y disponible para su uso cuando sea necesario.

2.2.1.1.6 Asignación del espacio libre

Técnicamente el inodo contiene la dirección de un fragmento. La dirección se puede interpretar como que referencia a un bloque completo o a uno o más fragmentos, dependiendo del número de bytes almacenados en la dirección. Sólo se puede asignar un bloque parcial al final de un fichero, así si existen 3 punteros a datos, los 2 primeros referenciarán a bloques completos, y el tercero podrá referenciar a un bloque parcial.

En la figura 2.6 se puede observar cómo se realiza la asignación de bloques y fragmentos cuando crece un fichero. Se trata de un fichero de 14 Kbytes almacenado en bloques de 8 Kbytes con fragmentos de 1 Kbyte. El número de bloques que se necesitan es un bloque completo más 6 fragmentos de otro bloque. Por lo tanto el primer puntero apunta a un bloque completo, mientras que el segundo apunta al primero de los 6 fragmentos restantes.

Para gestionar el espacio libre de la forma más eficiente posible, el sistema permite a distintos ficheros reclamar fragmentos dentro de un mismo bloque. Esto no provoca ninguna dificultad en la gestión del espacio, dado que sólo la última dirección del inodo puede apuntar a un bloque parcial

En este ejemplo se indican en el bitmap los fragmentos libres con un **uno** y los ocupados con un **cero**. Por tanto los fragmentos libres son los números 15-22 y 25-32, y los ocupados los números 1-14 y 23-24. Dado que los números de fragmento 25-32 constituyen 8 fragmentos contiguos libres que comienzan en el límite de un bloque, se pueden utilizar para formar un bloque completo. Por otro lado, los fragmentos 15-22 no se pueden utilizar para formar un bloque completo. Si se asigna un bloque parcial, los fragmentos deben ser consecutivos y no saltar los límites de un bloque. Por ejemplo, si se necesitasen 3 fragmentos para un fichero, se podrían utilizar los fragmentos 17-19, pero no los fragmentos 15-17. Dado que varios ficheros pueden reclamar fragmentos dentro del mismo bloque, es posible que los fragmentos del medio del bloque, o los últimos, estén utilizados, mientras que los del principio no.

Siempre que el sistema necesite asignar menos de un bloque, se asignarán del bloque que contenga suficientes fragmentos libres contiguos. De esta forma, es posible que diferentes ficheros utilicen diferentes fragmentos dentro del mismo bloque. Por ejemplo, en el bloque 3 del bitmap, los fragmentos números 17-22 están libres mientras que los 23 y 24 fueron reclamados por un fichero diferente. Puede ocurrir la siguiente situación, si en un momento anterior se reclamaron los fragmentos 17-22 por un fichero, mientras los fragmentos 23 y 24 lo fueron por otro fichero: si se eliminan los datos del primer fichero, esos fragmentos ya no son necesarios, así puede ocurrir que los primeros fragmentos del bloque estén libres, mientras que los últimos estén todavía ocupados; aunque por un fichero diferente del que ocupaba los primeros fragmentos del bloque.

Cada vez que necesite escribir datos en un fichero que ya existe, el sistema chequea para ver si puede incrementar su tamaño. En este caso pueden ocurrir las tres siguientes condiciones:

1. Hay suficiente espacio en el bloque o fragmento. En este caso los nuevos datos se escriben dentro del espacio no asignado.
2. El fichero contiene sólo bloques completos, y no hay suficiente espacio en el último bloque para mantener los datos adicionales. Si se necesita escribir más de un bloque completo de datos, se asigna un nuevo bloque y el primer bloque de datos adicionales se almacena ahí. Este proceso se repite hasta que se necesita escribir menos de un bloque de datos. Cuando esto ocurre, se asigna un bloque que contenga los suficientes fragmentos contiguos para escribir ahí los datos.
3. El fichero contiene fragmentos, pero no son suficientes para almacenar los nuevos datos. Si el tamaño de los datos que ya existen en los fragmentos, más los nuevos datos, exceden el tamaño de un bloque completo, se asigna un nuevo bloque, y tanto los datos antiguos como los nuevos se escriben en el nuevo bloque utilizando el procedimiento descrito en el punto anterior. Si el tamaño de los datos antiguos y los nuevos es menos de un bloque, se asignan a un bloque que contenga los suficientes fragmentos contiguos libres (o un bloque completo).

Cuando se asigna un bloque o un fragmento, la dirección se almacena en el inodo y se actualiza el bitmap de fragmentos libres.

2.2.1.1.7 Políticas de asignación de bloques y fragmentos

Además de las consideraciones de asignación de bloques y fragmentos libres, el sistema utiliza determinadas políticas de asignación dependiendo del tipo de datos que se necesiten escribir. Por

ejemplo, si un usuario crea un directorio o fichero nuevo, el sistema debe de tener ciertos criterios para determinar qué grupo de cilindros se utilizará para el nuevo objeto. Los criterios que utiliza el sistema se conocen como **políticas globales** y **políticas locales**.

Las políticas de asignación a nivel global, se utilizan para determinar la situación de los directorios y los nuevos ficheros y las políticas de asignación a nivel local se utilizan para determinar la situación actual de los bloques de datos.

En el nivel global, se toma la decisión de en qué grupo de cilindros poner un fichero o directorio. El sistema intentará poner todos los ficheros de un único directorio en el mismo grupo de cilindros. Cuando se crea un nuevo directorio, se sitúa en el grupo de cilindros que tenga el mayor número de inodos libres y el menor número de directorios. Existe un problema potencial con esta estrategia: si el sistema intenta mantener todos los ficheros de un directorio dentro del mismo grupo de cilindros, un número excepcionalmente pequeño de ficheros grandes puede agotar todo el espacio disponible. Para controlar esto el sistema mantiene un número límite de bloques por cilindro que puede utilizar un fichero (conocido como MAXBPG en `/usr/include/sys/fs.h`). Una vez que un fichero alcanza este número límite, el sistema asignará bloques de otro grupo de cilindros a ese fichero.

Las políticas globales llaman a las políticas locales con peticiones de bloques de datos adicionales. Son estas políticas de asignación local las que determinan qué bloque asignar. Las políticas locales asignarán bloques en el siguiente orden:

1. Asignación para un bloque pedido.
2. Asigna un bloque en el mismo cilindro que esté rotacionalmente más cerca del bloque pedido.
3. Asigna cualquier bloque dentro del mismo grupo de cilindros.
4. Busca un nuevo grupo de cilindros y asigna un bloque en cualquier lugar de él.
5. Emplea una búsqueda de fuerza bruta para encontrar un bloque disponible.

2.2.1.2 El sistema de ficheros Ext2

2.2.1.2.1 Introducción

Ext2 es el tipo de sistema de ficheros más potente y avanzado de los existentes sobre **Linux**. Linux inicialmente fue implementado como una extensión del sistema operativo **Minix** y su primera versión incluía soporte únicamente para el sistema de ficheros Minix. El sistema de ficheros Minix tiene dos limitaciones serias:

1. El direccionamiento de bloques está implementado sobre enteros de 16 bits, lo que hace que el tamaño máximo del sistema de ficheros esté limitado a 64 MBytes.
2. Las entradas de directorios son de tamaño fijo y el tamaño máximo para los nombres de ficheros está limitado a 14 caracteres.

Para facilitar la adición de otros sistemas de ficheros al kernel de Linux, se desarrolló una capa de software intermedio entre los sistemas de ficheros y las llamadas al sistema, llamada **Virtual File**

System (VFS), tras lo cual se desarrolló un nuevo sistema de ficheros llamado **The Extended File System**, el cual eliminaba las dos grandes limitaciones de Minix:

1. Su tamaño máximo llegaba hasta 2 GBytes.
2. El tamaño máximo para el nombre de los ficheros llegaba hasta 255 caracteres.

Este sistema de ficheros supuso una mejora sobre el de Minix, pero adolecía de algunos problemas, por ejemplo, usaba listas enlazadas para controlar los inodos y bloques libres, y esto producía malos rendimientos: a medida que se usaba el sistema de ficheros, las listas se desordenaban provocando la fragmentación del sistema de ficheros.

Como respuesta a estos problemas se desarrollaron en breve dos nuevos sistemas de ficheros: el **Xia** y el **Second Extended File System (Ext2)**. El sistema de ficheros Xia fue desarrollado a partir del sistema de ficheros de Minix, al cual se le añadieron algunas mejoras. Básicamente, se le añadió soporte para nombres de fichero largos, soporte para particiones más grandes y para fechas (de acceso, de modificación del fichero y de modificación del inodo). Por otro lado, **Ext2** se basó en el código de **Ext**, al cual se le añadieron muchas mejoras. Fue diseñado pensando en futuras evoluciones del sistema de ficheros.

Estos dos sistemas de ficheros cuando fueron liberados proporcionaban esencialmente las mismas características, sólo que Xia, debido a su mínimo diseño, era más estable que Ext2. Pero a medida que Ext2 se fue usando, se le fueron corrigiendo los errores iniciales y añadiéndole mejoras y nuevas características, siendo en la actualidad muy estable. Ext2 es en la actualidad el sistema de ficheros estándar de Linux.

La tabla 2.2 nos muestra un resumen de las características de los sistemas de ficheros soportados por Linux.

Tabla 2.2: Resumen de las características de los sistemas de ficheros de Linux.

	Minix	Ext	Ext2	Xia
Tamaño Max. SF	64 MB	2 GB	4 TB	2 GB
Tamaño Max. Fichero	64 MB	2 GB	2 GB	64 MB
Máx. Nombre Fichero	16/30 c	255 c	255 c	248 c
Soporte 3 Fechas	No	No	Si	Si
Extensible	No	No	Si	No
Tamaño Bloque Var.	No	No	Si	No
Soportado	Si	No	Si	?

2.2.1.2.2 El Sistema de Ficheros Virtual (VFS)

El kernel de Linux contiene una capa intermedia de software entre los sistemas de ficheros y las llamadas al sistema, que es el **Sistema de Ficheros Virtual (VFS)**. El VFS es una capa de

indirección que maneja las llamadas al sistema orientadas a fichero y llama a las funciones necesarias en el código del sistema de ficheros físico para hacer las operaciones de Entrada/Salida.

Este mecanismo de indirección es frecuentemente usado en los sistemas operativos UNIX para facilitar la integración de varios tipos de sistemas de ficheros.

Cuando un proceso realiza una llamada al sistema orientada a fichero, el kernel llama a una función contenida en el VFS. Esta función redirige la llamada a una función contenida en el código del sistema de ficheros físico, la cual es la responsable del manejo de la estructura dependiente de las operaciones de E/S. El código del sistema de ficheros usa las funciones del **buffer caché** para hacer peticiones de Entrada/Salida a los dispositivos. Podemos ver este esquema en la figura 2.7

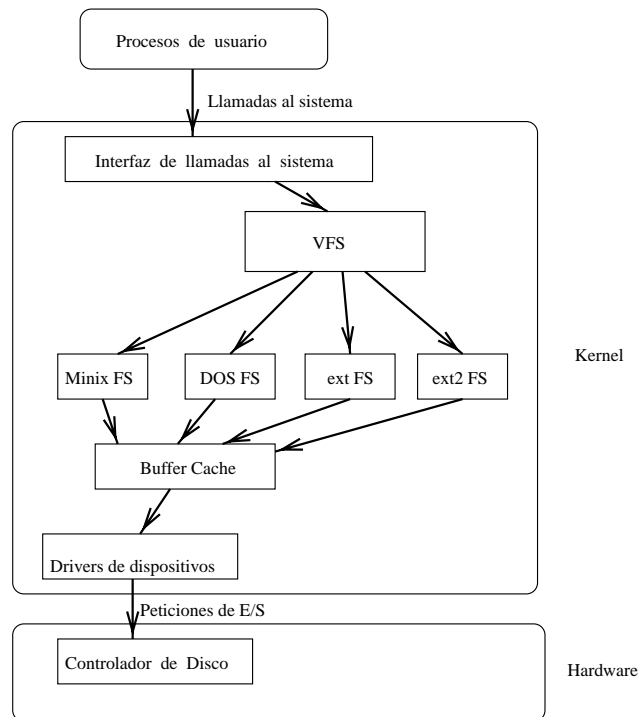


Figura 2.7: La capa intermedia VFS.

La capa VFS define un conjunto de funciones que cada sistema de ficheros tiene que implementar. Este interfaz está construido en base a un conjunto de operaciones asociadas a tres clases de objetos: sistemas de ficheros, inodos y ficheros abiertos.

El VFS conoce los tipos de sistemas de ficheros soportados por el kernel. Para ello usa una tabla definida durante la configuración del kernel. Cada entrada de esta tabla describe un tipo de sistema de ficheros: contiene el nombre del tipo de sistema de ficheros y un puntero a una función, que es llamada durante la operación de montaje. Cuando se monta un sistema de ficheros, se invoca la función de montaje apropiada (según el tipo de sistema de ficheros). Esta función es responsable de leer el superbloque, inicializar sus variables internas y devolver al VFS un descriptor de sistema de

ficheros montado. Una vez que el sistema de ficheros ha sido montado, las funciones del VFS pueden usar este descriptor para acceder a las rutinas del sistema de ficheros físico.

Un descriptor de sistema de ficheros montado contiene varias clases de datos: información común a cualquier tipo de sistema de ficheros, punteros a funciones proporcionadas por el código del kernel del sistema de ficheros físico, y datos privados mantenidos por el código del sistema de ficheros físico. Los punteros a funciones contenidos en los descriptores de sistemas de ficheros permite al VFS acceder a las rutinas internas del sistema de ficheros.

El VFS usa además otras dos clases de descriptores: descriptores de inodos y descriptores de ficheros abiertos. Cada descriptor contiene información relacionada con los ficheros en uso, y un conjunto de operaciones proporcionadas por el código del sistema de ficheros físico. Mientras que el descriptor de inodo contiene punteros a funciones que pueden ser usadas para operar sobre cualquier fichero (por ejemplo `create`, `unlink`), el descriptor de fichero contiene punteros a funciones que pueden sólo actuar sobre ficheros abiertos (por ejemplo, `read`, `write`).

2.2.1.2.3 Características estándar del sistema de ficheros Ext2

El sistema de ficheros Ext2 soporta los tipos de ficheros estándares de UNIX : ficheros regulares, ficheros de dispositivos, directorios y enlaces simbólicos.

Ext2 es capaz de manejar sistemas de ficheros sobre particiones muy grandes. Si bien el código original del kernel restringía el tamaño máximo del sistema de ficheros a 2 GB, trabajos posteriores en la capa VFS han elevado este límite a 4 TB.

Ext2 permite nombres de ficheros largos. Para ello usa entradas de directorio de longitud variable. La longitud máxima permitida para un nombre de fichero es de 255 caracteres. Este límite podría extenderse a 1012 caracteres si fuese necesario.

Ext2 reserva algunos bloques del sistema de ficheros para el superusuario, normalmente el 5 % de los bloques. Esto permite al administrador del sistema, conectándose como **root** (superusuario), solucionar las situaciones en donde un proceso de usuario ha llenado el sistema de ficheros.

2.2.1.2.4 Características avanzadas del sistema de ficheros Ext2

Además de las características estándar de los sistemas de ficheros UNIX , Ext2 soporta algunas extensiones que no están normalmente presentes en sistemas de ficheros UNIX .

Los atributos (permisos) de un fichero permiten a los usuarios modificar el funcionamiento del kernel con respecto a este fichero. En tiempo de montaje del sistema de ficheros se puede seleccionar la semántica BSD, o bien la SVR4. Una opción en el montaje le permite al administrador elegir la semántica para la creación de ficheros. En un sistema de ficheros montado con semántica BSD, los ficheros son creados con el mismo GID que el directorio donde se crean. La semántica SVR4 es algo más compleja: si el directorio tiene el SGID activado, los nuevos ficheros heredan el GID del directorio y los subdirectorios heredan el GID y el bit SGID; en otro caso tanto los ficheros como los directorios son creados con el GID primario del proceso que los crea.

El sistema de ficheros tipo Ext2 permite actualizaciones síncronas al estilo BSD. Esto se hace

mediante una opción en tiempo de montaje del sistema de ficheros, y garantiza que los datos serán escritos en el disco cada vez que sean modificados; esto puede ser útil para mantener un estricto control de la consistencia, sin embargo el rendimiento será muy pobre.

◇ Ext2 permite al administrador elegir el tamaño del bloque lógico en tiempo de creación del sistema de ficheros. Los tamaños típicos de bloque lógico son 1K, 2K y 4K. El uso de tamaños grandes para los bloques acelerará las operaciones de E/S, ya que habrá menos peticiones de E/S y menos búsquedas de las cabezas del disco. En contrapartida, el consumo de espacio en disco será mayor: por término medio, el último bloque de un fichero tiene la mitad del espacio libre, por tanto cuanto más grande es el tamaño de bloque más espacio se derrocha en el último bloque de cada fichero.

Ext2 implementa enlaces simbólicos rápidos. Un enlace de este tipo no usa ningún bloque de datos del sistema de ficheros: el nombre del fichero destino no se almacena en un bloque de datos sino en el propio inodo. Esta política puede salvar algún espacio en disco, ya que no se necesita ningún bloque de datos, y acelerar la operación de enlace, ya que no se necesita leer ningún bloque de datos cuando se accede al enlace. El espacio disponible en un inodo es limitado y por tanto no todos los enlaces simbólicos pueden ser implementados como enlaces rápidos. El tamaño máximo del fichero destino para que un enlace simbólico pueda ser implementado de esta forma es de 60 caracteres. En un futuro próximo se piensa extender este sistema a ficheros pequeños.

Ext2 controla el estado del sistema de ficheros. El kernel usa un campo del superbloque que indica el estado del sistema de ficheros. Cuando se monta el sistema de ficheros en modo lectura/escritura, su estado se pone como "Not Clean". Cuando se desmonta o se vuelve a montar en modo solo-lectura, su estado se pone a "Clean". En tiempo de arranque, el comprobador del sistema de ficheros usa esta información para decidir si el sistema de ficheros debe ser comprobado.

Saltarse la comprobación del sistema de ficheros puede a veces ser peligroso, ya que Ext2 proporciona dos formas de forzar la comprobación a intervalos regulares. El superbloque mantiene un contador de montaje. Cada vez que el sistema de ficheros se monta en modo lectura/escritura, se incrementa el contador. Cuando alcanza un valor máximo, también grabado en el superbloque, el comprobador del sistema de ficheros fuerza la comprobación incluso si el sistema de ficheros está en estado "Clean". En el superbloque también se mantiene la fecha de la última comprobación y un intervalo máximo de comprobación. Estos dos campos permiten al administrador hacer comprobaciones periódicas. Cuando se alcanza el intervalo máximo de comprobación, el comprobador ignora el estado del sistema de ficheros y fuerza la comprobación.

Ext2 ofrece herramientas para poner a punto el funcionamiento del sistema de ficheros. El comando `tune2fs` se puede usar para modificar:

- El funcionamiento del error. Cuando el código del kernel detecta una inconsistencia, el sistema de ficheros se marca como "Erroneous" y se puede tomar una de las tres siguientes acciones: continuar con la ejecución normal, volver a montar el sistema de ficheros en modo solo-lectura para evitar su corrupción, o forzar un **kernel panic** y rebotar para ejecutar la comprobación del sistema de ficheros.
- El número máximo de montajes.
- El máximo intervalo de comprobación.

- El número de bloques lógicos reservados para el superusuario.

Ext2 también permite el borrado seguro de ficheros. Cuando se borra un fichero se escriben datos aleatorios en los bloques de datos del mismo. Esto previene que se pueda acceder mediante un editor al contenido de un fichero borrado.

Por último, se han añadido nuevos tipos de ficheros al sistema de ficheros Ext2, inspirado en el sistema de ficheros BSD 4.4. **Ficheros inmutables**, que sólo pueden ser leídos: nadie puede escribir o borrarlos. Esto se puede usar para proteger ficheros de configuración críticos. **Ficheros a los que sólo se puede añadir**: estos ficheros pueden ser abiertos en modo escritura, pero los datos siempre son añadidos al final del fichero. Como los ficheros inmutables, no pueden ser borrados ni renombrados. Esto es especialmente útil para ficheros de **log**, los cuales sólo pueden crecer.

2.2.1.2.5 Estructura física

La estructura física del sistema de ficheros Ext2 ha estado fuertemente influenciada por la del sistema de ficheros BSD. El sistema de ficheros se compone de **grupos de bloques**. Los grupos de bloques son análogos a los grupos de cilindros de BSD. Sin embargo, los grupos de bloques no están ligados a la estructura física de los bloques en el disco, ya que los discos modernos tienden a estar optimizados para el acceso secuencial y esconden su geometría física al sistema operativo.

La estructura física del sistema de ficheros es la representada en la figura 2.8.

Sector de Boot	Grupo de Bloques 1	Grupo de Bloques 2	...	Grupo de Bloques N
-------------------	-----------------------	-----------------------	-----	-----------------------

Figura 2.8: Estructura física del sistema de ficheros Ext2.

Cada grupo de bloques contiene una copia redundante de información crucial del sistema de ficheros (superbloque y descriptores de los sistemas de ficheros) y también contiene una parte del sistema de ficheros (un bitmap de bloques, un bitmap de inodos, una tabla de inodos, y bloques de datos). La estructura de un grupo de bloques es la representada en la figura 2.9

Super Bloque	Descriptores de SF	Bitmap de Bloques	Bitmap de Inodos	Tabla de Inodos	Bloques de datos
-----------------	-----------------------	----------------------	---------------------	--------------------	------------------

Figura 2.9: Estructura de un grupo de bloques.

El uso de grupos de bloques es un gran logro en términos de fiabilidad. Como las estructuras de control están replicadas en cada grupo de bloques, es fácil recuperar un sistema de ficheros si el superbloque se corrompe. Estas estructuras también ayudan a conseguir buenos rendimientos: reduciendo la distancia entre la tabla de inodos y los bloques de datos es posible reducir las búsquedas de las cabezas del disco para el acceso a los ficheros.

En Ext2, los directorios se manejan como listas enlazadas de entradas de longitud variable. Cada entrada contiene el número de inodo, la longitud de la entrada, la longitud del fichero y su nombre.

Usando entradas de longitud variable, es posible implementar nombres de fichero largos sin derrochar espacio en los directorios. La estructura de una entrada de directorio es la mostrada en la figura 2.10.

numero de inodo	long. entrada	long. nombre	nombre_fichero
-----------------	---------------	--------------	----------------

Figura 2.10: Estructura de una entrada de directorio Ext2.

En la figura 2.11 podemos ver la estructura de un directorio que contiene tres ficheros: *fich1*, *fichero_grande*, y *f3*.

i1	16	05	fich1	i2	40	14	fichero_grande	i3	12	02	f3
----	----	----	-------	----	----	----	----------------	----	----	----	----

Figura 2.11: Ejemplo de un directorio con tres ficheros.

2.2.1.2.6 Optimización del rendimiento

El código del kernel del Ext2 contiene muchas optimizaciones que tienden a mejorar las operaciones de E/S en el acceso a ficheros.

Ext2 utiliza el bufer caché realizando lecturas por adelantado: cuando hay que leer un bloque, el código del kernel hace peticiones de E/S de varios bloques contiguos. De esta forma, intenta asegurar que el siguiente bloque a leer esté ya cargado en el bufer caché. Las lecturas por adelantado, son normalmente realizadas durante lecturas secuenciales sobre ficheros, y Ext2 también lo usa para lecturas de directorios.

Ext2 también contiene muchas optimizaciones de asignación. Los grupos de bloques se usan para agrupar inodos y bloques de datos relacionados: el código del kernel siempre intenta asignar bloques de datos a un fichero en el mismo grupo que su inodo. Con esto se consigue reducir el número de posicionamientos de las cabezas del disco cuando el kernel intenta leer un inodo y sus bloques de datos.

Al escribir datos en un fichero, Ext2 preasigna hasta 8 bloques de datos adyacentes cuando asigna un nuevo bloque. La preasignación consigue aciertos de hasta un 75% incluso en sistemas de ficheros muy llenos, y también alcanza buenos rendimientos en escritura bajo cargas pesadas.

2.2.2 Tipo de Sistemas de Ficheros. Área de Intercambio (Swap)

Linux, en particular, además de los tipos de sistemas de ficheros ya comentados (Minix, Ext, Xia y Ext2) soporta varios sistemas de ficheros ajenos, para facilitar el intercambio de ficheros con otros sistemas operativos, en concreto:

msdos	Para la compatibilidad con el sistema de ficheros FAT del MS-DOS.
umsdos	Extiende el sistema de ficheros msdos bajo Linux, de forma que desde Linux se pueden usar nombres de fichero largos, propiedad, permisos, enlaces y ficheros de