LFS (Log File System)

Las máquinas y los discos son cada vez mas rápidas.
Las cantidades de memoria disponibles son cada vez mayores, lo cual permite cachés de gran tamaño, aliviando las lecturas, perod de todas formas, por cuestiones de seguridad, hay que ecribir los cachés en disco.

En definitiva, el factor limitante de velocidad del software es la gran cantidad de accesos random a disco.
La idea de LFS es desvincular a las aplicaciones de la escritura a disco y reemplazar las pequeñas escrituras de tipo random por escrituras masivas secuenciales, pero manteniendo la posibilidad de recuperar los datos del disco de forma eficiente.

LFS divide el disco estáticamente en **segmentos** de tamaño fijo (normalmente ½ Mb).
El ordenamiento lógico de estos segmentos crea un único log continuo.

LFS mantiene las estructuras estructuras de Ext2. Tiene un superblock, los inodes tienen la misma información (incluyendo punteros indirectos, doble indirectos y triple indirectos).

LFS reúne varias páginas que necesiten ser escritas y se prepara para escribirlas en el disco en el próximo segmento libre. En este punto, ordena los bloques por número de bloque, les asigna direcciones de disco y actualiza los metadatos necesarios. Los bloques de metadatos modificados se toman junto con los bloques de datos y se escriben todos juntos en un segmento del disco.

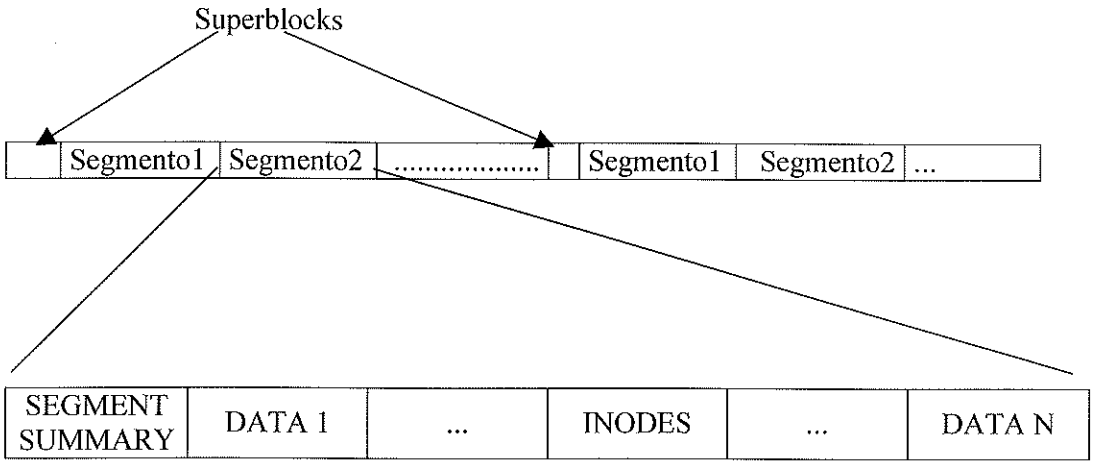
Checkpoint

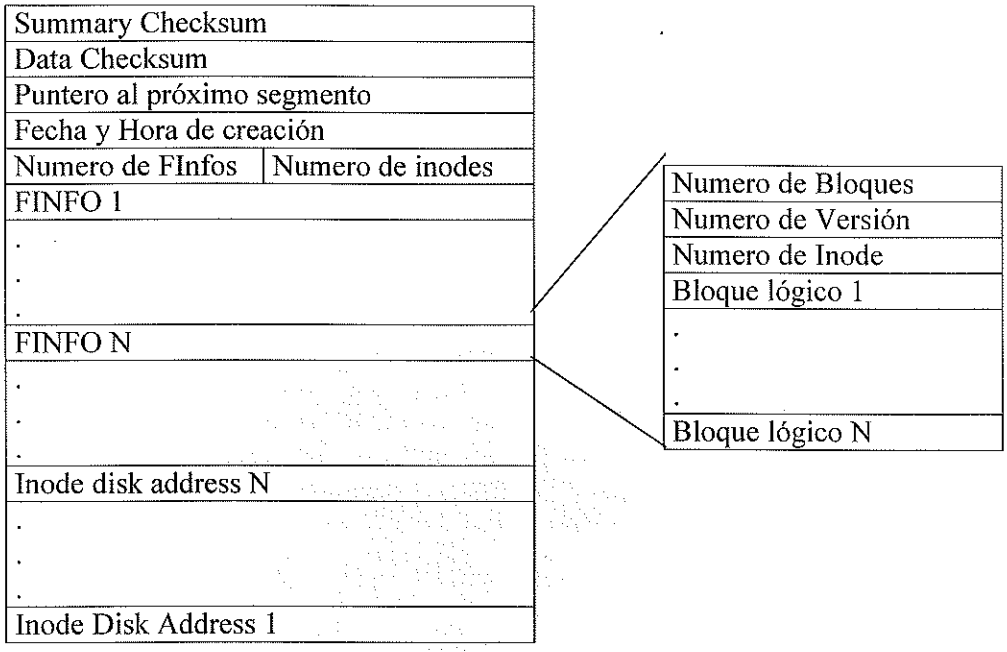
Es una operación que provee el file system durante la cual, en un punto fijo en el disco se escriben todas las estructuras de datos del file system residentes en memoria durante una operación de Checkpoint. Un checkpoint marca un estado consistente del file system que puede ser usado para la recuperación después de un problema.
Tambien se escriben a disco todos los bloques de datos y metadatos modificados.
La política de recuperación consiste en recuperar el estado del file system a partir del último checkpoint. Todo lo hecho en el medio se pierde. Es posible regular el riesgo que esto implica aumentando o disminuyendo el intervalo de tiempo con el que se efectúan los checkpoints.
Aunque no esté implementado, LFS permite una mejor recuperación por medio del análisis del log (al igual que la mayoría de los motores de bases de datos). Una ventaja adicional es que le alcanza con revisar la cola del log desde el último checkpoint mientras que en los otros file systems es necesario recorrer todo el disco en busca de errores para permitir la recuperación.

Mapa de inodes
Como resultado de esta técnica, los inodes ya no están siempre en el mismo lugar del disco sino que su ubicación va cambiando conforme son actualizados.
Debido a esto, se necesita un ***Mapa de inodes*** que mapea un número de inode a su dirección en el disco.

Segmentos Parciales
Si los datos a escribir no alcanzan para completar un segmento pueden escribirse segmentos parciales

Estructura en Disco



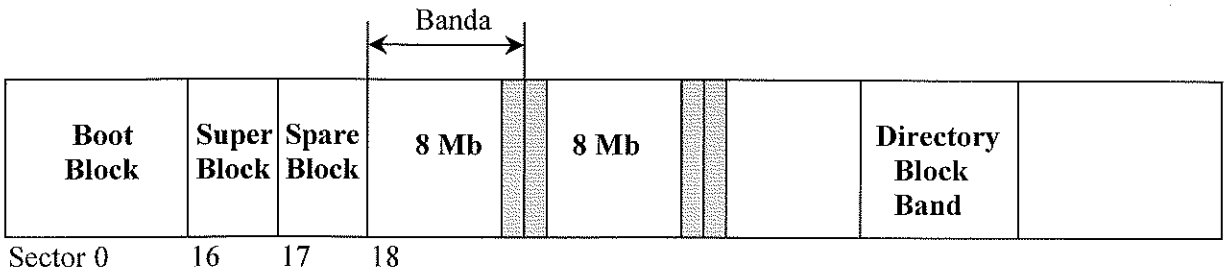


Cleaner
Los bloques modificados se escriben a disco en una posición diferente de la anterior. A esta técnica de reubicación se la llama política de no sobreescribir y necesita un mecanismo para recuperar el espacio resultante de bloques borrados o reescritos. Cleaner recorre los segmentos descartando aquellos bloques que ya no son válidos y agrega los nuevos. Para hacer esto necesita poder identificar los bloques dentro del segmento. esta información la obtiene del **Segment Summary.**

HPFS (OS/2)

HPFS se diseño para suplir las falencias de FAT. Es el file system de OS/2
Logra una excelente velocidad gracias al uso de técnicas de read ahead , write behind y caché inteligente.
Está diseñado para ser tolerante a fallas.

Estructura de un volumen HPFS



Boot Block

- ID del volumen (32 bit)
- Programa de bootstrap: Es bastante complejo, permite acceder en modo protegido al file system y leer los archivos del sistema operativo de cualquier lugar en el que estén.

Super Block

- Versión de HPFS
- Puntero al fnode del directorio raíz
- Cantidad de sectores
- Cantidad de sectores dañados
- Puntero a los bitmaps de espacio libre
- Puntero a la lista de bloques dañados
- Puntero al Directory Block Band
- Fecha de la última vez que se ejecutó un chequeo (CHKDSK)

Spare Block

- Flags y punteros (por ejemplo, puntero al pool de bloques libres)

Bandas

El resto del disco se divide en bandas de 8Mb. Cada una tiene un bitmap de espacio libre que se ubica alternativamente al inicio o al final de la banda. Si una banda tiene su bitmap al principio, la siguiente lo tendrá al final. De esta forma se puede disponer de un espacio máximo de 16Mb contiguos para asignar a archivos. El tamaño de la banda no está dado por una limitación del file system sino que es una decsión de implementación que puede cambiar.

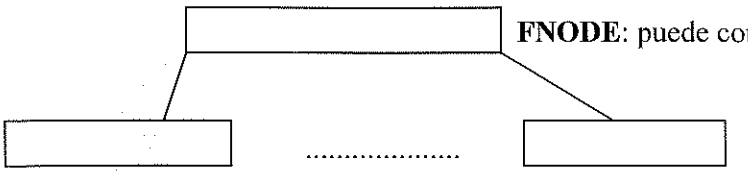
Representación de los archivos

HPFS ve cada archivo como una colección de uno o mas secuencias (o runs) de conjuntos (extents) de uno o mas sectores contiguos. Cada secuencia se representa con 32 bits para indicar el número de sector inicial y otros 32 bits para la cantidad de sectores contiguos de ese extent.
Un archivo o directorio se identifica por una estructura llamada Fnode.

Fnode (ocupa un solo sector y se guarda próximo al archivo o directorio que describe)

- Longitud del nombre del archivo
- Primeros 15 caracteres del nombre del archivo
- Información de control
- Historia de accesos
- Atributos extendidos
- ACL's
- Puntero al archivo.

Un Fnode puede almacenar punteros para hasta 8 runs de hasta 16Mb cada uno (este límite de 16Mb se debe al tamaño de las bandas y la ubicación de los bitmaps de espacio libre) . Los archivos cuyo tamaño y fragmentación lo permitan, son descriptos completamente dentro del Fnode. Aquellos que sean mas grandes o estén muy fragmentados, se representan utilizando una estructura de árbol B+ de la siguiente forma:



El nivel del árbol puede crecer tanto como sea necesario.

Ventajas de esta estructura:

El agregado de espacio contiguo a un archivo sólo requiere modificar la longitud del run correspondiente y actualizar el bitmap de libres.
Traducir un offset en un archivo a un número de sector sólo requiere acceder al fnode y recorrer la lista de runs (o el árbol B+ si el archivo no estuviera integramente descripto en su fnode) hasta que llega al rango correcto. Con un simple cálculo puede identificarse el número de sector dentro del run.

Directorios

Los directorios también se describen por medio de un fnode. En el superblock hay un puntero al fnode del directorio raiz.
Los punteros a los fnodes de cualquier subdirectorio se encuentran en el directorio padre.

Los directorios están formados por uno a varios directory blocks de 2kb (se obtienen asignando 4 sectores contiguos). Siempre que sea posible, los directory block se asignan en la Directory Band (es

una banda especial que se encuentra cerca del centro del disco.), pero cuando se encuentre llena, se asignan en donde se encuentre lugar.
Cada directory block puede contener ona o varias entradas de directorio válidas, mas una entrada de directorio dummy al final que indica el fin del bloque. Por lo tanto, siempre contendrá por lo menso dos entradas.

Entrada de directorio

Longitud de la entrada (32 bit)
Fecha y Hora de creación y modificación
Longitud del nombre del archivo o subdirectorio
Nombre del archivo o subdirectorio
Puntero al fnode
Puntero al árbol B

Las entradas de directorio están ordenadas alfabéticamente.
Si un directorio es demasiado largo para ser descripto dentro de un directory block, se asignan mas directory blocks y se organizan como un árbol B.

Busqueda en directorios

Búsqueda de /dir1/dir2/arch

```
Busqueda(){
        Se lee del superblock la dirección del fnode del raíz
        Fnode= fnode del raiz obtenido del superblock
        Mientras queden arch o dir y Fnode no está vacío{
                Dir=parsear subdirectorio o nombre de arch de la cadena completa
                Fnode=BuscarEnDirectoryBlock(Fnode, Dir)
        }
        Si fnode está vacío
                Return (No se encontró el archivo)
        Sino
                Return Fnode
}

BuscarEnDirectoryBlock(Fnode , Dir ){
        Leer fnode y encontrar ubicación del directory block
        Mientras queden entradas de directorio con nombre menor a Dir
                Leer siguiente entrada de directorio;
        Si no encontró la entrada de Dir {
                Leer puntero al árbol B
                Si es puntero válido entonces
                        Return BuscarEntradaEnArbolB()
                Else
                        Return vacío
        }
        Else{
                Leer puntero al fnode_encontrado
                Return fnode_encontrado
        }
}
```

Desventaja

El uso de árboles B obliga a operaciones de mantenimiento bastante complejas. Podría pasar que sin cambiar el tamaño del archivo, falte espacio en disco. Por eso, para casos de emergencia con directorios, existe un pool de bloques libres.