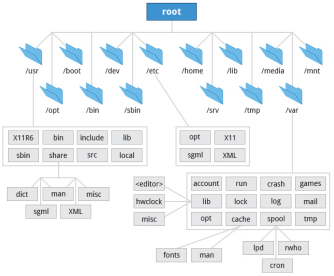


File Systems

Alejandro Furfaro

13 de septiembre de 2023

Temario

Ideas directrices detrás del diseño de UNIX

Ideas directrices detrás del diseño de UNIX

- El concepto de “*everything is a file*”, permite tratar cualquier clase de dispositivo de hardware de la misma forma que un archivo.

Ideas directrices detrás del diseño de UNIX

- El concepto de “*everything is a file*”, permite tratar cualquier clase de dispositivo de hardware de la misma forma que un archivo.

- Provee una capa de abstracción que permite tratar desde un punto de vista puramente lógico cualquier tipo de dispositivo mediante una interfaz común de Programación.

Ideas directrices detrás del diseño de UNIX

- El concepto de "*everything is a file*", permite tratar cualquier clase de dispositivo de hardware de la misma forma que un archivo.
- Provee una capa de abstracción que permite tratar desde un punto de vista puramente lógico cualquier tipo de dispositivo mediante una interfaz común de Programación.
- El problema del acceso a un recurso, entonces se resuelve mediante la distinción entre "*mecanismo*" y "*política*".

Ideas directrices detrás del diseño de UNIX

- El concepto de "*everything is a file*", permite tratar cualquier clase de dispositivo de hardware de la misma forma que un archivo.
- Provee una capa de abstracción que permite tratar desde un punto de vista puramente lógico cualquier tipo de dispositivo mediante una interfaz común de Programación.
- El problema del acceso a un recurso, entonces se resuelve mediante la distinción entre "*mecanismo*" y "*política*".

1. **Mecanismo**: Que capacidades deben proveerse.

Ideas directrices detrás del diseño de UNIX

- El concepto de "*everything is a file*", permite tratar cualquier clase de dispositivo de hardware de la misma forma que un archivo.
- Provee una capa de abstracción que permite tratar desde un punto de vista puramente lógico cualquier tipo de dispositivo mediante una interfaz común de Programación.
- El problema del acceso a un recurso, entonces se resuelve mediante la distinción entre "*mecanismo*" y "*política*".
  1. **Mecanismo**: Que capacidades deben proveerse.
  2. **Política**: Como deben utilizarse esas capacidades

¿Quien implementa el Mecanismo?

Device file

¿Quien implementa el Mecanismo?

Device file

- En UNIX un *device file* es la abstracción con la que se representa a cualquier dispositivo de hardware.

¿Quien implementa el Mecanismo?

Device file

- En UNIX un *device file* es la abstracción con la que se representa a cualquier dispositivo de hardware.
- Está asociado a un device driver, que es a su vez la pieza de código que se encarga de resolver el acceso al hardware (es decir "**el mecanismo**"), manejando todos los requerimientos de E/S, y proveyendo "hacia arriba" una interfaz independiente del hardware.

¿Quien implementa las políticas?

El File System

¿Quien implementa las políticas?

El File System

- Un ***file system*** es un conjunto de normativas, métodos y estructuras de datos que controlan la forma en que el sistema operativo almacena y accede a la información en un medio masivo de almacenamiento.

¿Quien implementa las políticas?

El File System

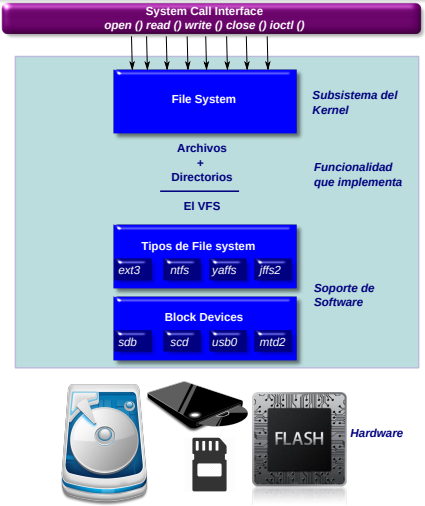
- Un ***file system*** es un conjunto de normativas, métodos y estructuras de datos que controlan la forma en que el sistema operativo almacena y accede a la información en un medio masivo de almacenamiento.
- Conforma “**las políticas**”de acceso

¿Quien implementa las políticas?

El File System

- Un ***file system*** es un conjunto de normativas, métodos y estructuras de datos que controlan la forma en que el sistema operativo almacena y accede a la información en un medio masivo de almacenamiento.
- Conforma "**las políticas**"de acceso
- Provee servicios que facilitan el acceso a los datos, y además que protegen la integridad y el acceso.

Filesystem + devices. Un subsistema



Contenido

Organizar la información

Organizar la información

- Un sistema de archivos es utilizado para controlar y organizar la forma en que los datos son almacenados y accedidos.

Organizar la información

- Un sistema de archivos es utilizado para controlar y organizar la forma en que los datos son almacenados y accedidos.

- Sin él, la información ubicada en un medio de almacenamiento sería un gran bloque de datos sin forma de saber donde termina un fragmento de información y donde comienza el siguiente.

Organizar la información

- Un sistema de archivos es utilizado para controlar y organizar la forma en que los datos son almacenados y accedidos.
- Sin él, la información ubicada en un medio de almacenamiento sería un gran bloque de datos sin forma de saber donde termina un fragmento de información y donde comienza el siguiente.
- Al separar el bloque monolítico de ’1’s y ’0’s en piezas individuales e identificándolas unívocamente con un nombre, podemos separar e identificar mucho más fácilmente cada trozo de información relevante como conjunto.

Organizar la información

- Un sistema de archivos es utilizado para controlar y organizar la forma en que los datos son almacenados y accedidos.
- Sin él, la información ubicada en un medio de almacenamiento sería un gran bloque de datos sin forma de saber donde termina un fragmento de información y donde comienza el siguiente.
- Al separar el bloque monolítico de ’1’s y ’0’s en piezas individuales e identificándolas unívocamente con un nombre, podemos separar e identificar mucho más fácilmente cada trozo de información relevante como conjunto.
- Cada pieza individual de información recibe genéricamente el nombre de archivo.

Política de organización

Política de organización

- Cada tipo de file system tiene su propio conjunto de reglas (***políticas***) para controlar la asignación de espacio de almacenamiento a los diferentes archivos.

Política de organización

- Cada tipo de file system tiene su propio conjunto de reglas (***políticas***) para controlar la asignación de espacio de almacenamiento a los diferentes archivos.

- Además estas reglas rigen también para asociar información útil para gestión de cada archivo (conocidos como *metadatos*): su nombre, su ruta relativa, sus permisos, y la fecha de creación y modificación, por ejemplo.

Política de organización

- Cada tipo de file system tiene su propio conjunto de reglas (***políticas***) para controlar la asignación de espacio de almacenamiento a los diferentes archivos.

- Además estas reglas rigen también para asociar información útil para gestión de cada archivo (conocidos como *metadatos*): su nombre, su ruta relativa, sus permisos, y la fecha de creación y modificación, por ejemplo.

- En función del paradigma “everything is a file” los sistemas POSIX (como Linux) tratan también a los recursos de entrada/salida, los dispositivos, conexiones de Red, y algunos mecanismos IPC, de forma muy similar a la forma de manejar un archivo en el disco.

Visión única

Visión única

- Los sistemas de archivos pueden ser usados en múltiples medios de almacenamiento: discos rígidos, memorias flash, cintas magnéticas, discos ópticos, entre otros.

Visión única

- Los sistemas de archivos pueden ser usados en múltiples medios de almacenamiento: discos rígidos, memorias flash, cintas magnéticas, discos ópticos, entre otros.
- En algunos casos, puede utilizarse la RAM como un file system temporal de alto rendimiento. Ejemplo: **`/proc`** en Linux.

Visión única

- Los sistemas de archivos pueden ser usados en múltiples medios de almacenamiento: discos rígidos, memorias flash, cintas magnéticas, discos ópticos, entre otros.
- En algunos casos, puede utilizarse la RAM como un file system temporal de alto rendimiento. Ejemplo: `/proc` en Linux.
- Algunos sistemas de archivos son usados en dispositivos de almacenamiento local, otros proveen acceso a archivos a través de un protocolo de red (como por ejemplo, NFS o SMB).

Visión única

- Los sistemas de archivos pueden ser usados en múltiples medios de almacenamiento: discos rígidos, memorias flash, cintas magnéticas, discos ópticos, entre otros.
- En algunos casos, puede utilizarse la RAM como un file system temporal de alto rendimiento. Ejemplo: **`/proc`** en Linux.
- Algunos sistemas de archivos son usados en dispositivos de almacenamiento local, otros proveen acceso a archivos a través de un protocolo de red (como por ejemplo, NFS o SMB).
- Incluso existen los sistemas de archivos “virtuales”, en los que cada “archivo” es creado o leído de manera dinámica (como por ejemplo, en *procfs* o *sysfs*).

Montaje

Montaje

- Montar un File System en cualquier sistema POSIX significa asociarlo a un dispositivo de almacenamiento.

Montaje

- Montar un File System en cualquier sistema POSIX significa asociarlo a un dispositivo de almacenamiento.
- El comando ***mount*** se utiliza para attachar un file system a la jerarquía de File Systems actual en el sistema (que comienza en root '/').

Montaje

- Montar un File System en cualquier sistema POSIX significa asociarlo a un dispositivo de almacenamiento.
- El comando ***mount*** se utiliza para attachar un file system a la jerarquía de File Systems actual en el sistema (que comienza en root '/').
- Al montar un file system además debemos indicar que tipo de File System es, y el punto de montaje, es decir, el punto de la jerarquía de directorios en el cual queremos attachar nuestro file system.

Montaje

- Montar un File System en cualquier sistema POSIX significa asociarlo a un dispositivo de almacenamiento.
- El comando ***mount*** se utiliza para attachar un file system a la jerarquía de File Systems actual en el sistema (que comienza en root '/').
- Al montar un file system además debemos indicar que tipo de File System es, y el punto de montaje, es decir, el punto de la jerarquía de directorios en el cual queremos attachar nuestro file system.
- Este criterio empleado para conectar y desconectar cualquier tipo de medio brinda una flexibilidad absoluta para proveer una visión única para un conjunto diverso de medios de almacenamiento.

Montaje

Cuatro comandos, dos archivos un par de trucos y ... dos FS

Montaje

Cuatro comandos, dos archivos un par de trucos y ... dos FS

1. ***dd***. Convierte y copia físicamente archivos

Montaje

Cuatro comandos, dos archivos un par de trucos y ... dos FS

1. ***dd***. Convierte y copia físicamente archivos
2. ***losetup***. Establece y controla loop devices (Loop devices???)

Montaje

Cuatro comandos, dos archivos un par de trucos y ... dos FS

1. ***dd***. Convierte y copia físicamente archivos
2. ***losetup***. Establece y controla loop devices (Loop devices???)
3. ***mke2fs***. Crea un filesystem ext2/ext3/ext4.

Montaje

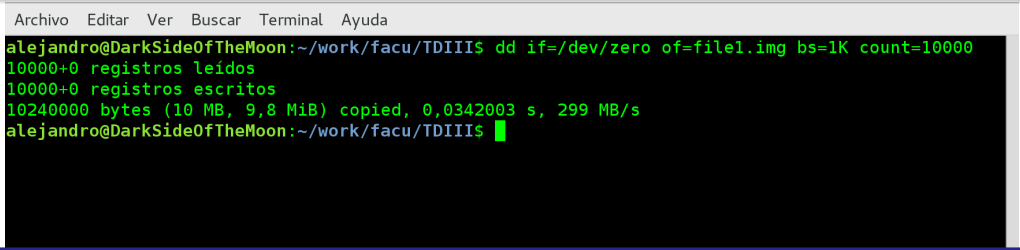
Cuatro comandos, dos archivos un par de trucos y ... dos FS

1. ***dd***. Convierte y copia físicamente archivos
2. ***losetup***. Establece y controla loop devices (Loop devices???)
3. ***mke2fs***. Crea un filesystem ext2/ext3/ext4.
4. ***mount***. Monta un file system de un tipo dado en un punto de una jerarquía existente

Montaje

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ dd if=/dev/zero of=file1.img bs=1K count=10000
10000+0 registros leídos
10000+0 registros escritos
10240000 bytes (10 MB, 9,8 MiB) copied, 0,0342003 s, 299 MB/s
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Montaje



Armamos un archivo lleno de ceros

El comando ***dd*** copia en forma física bloque a bloque. Es especial para construir imágenes de disco.

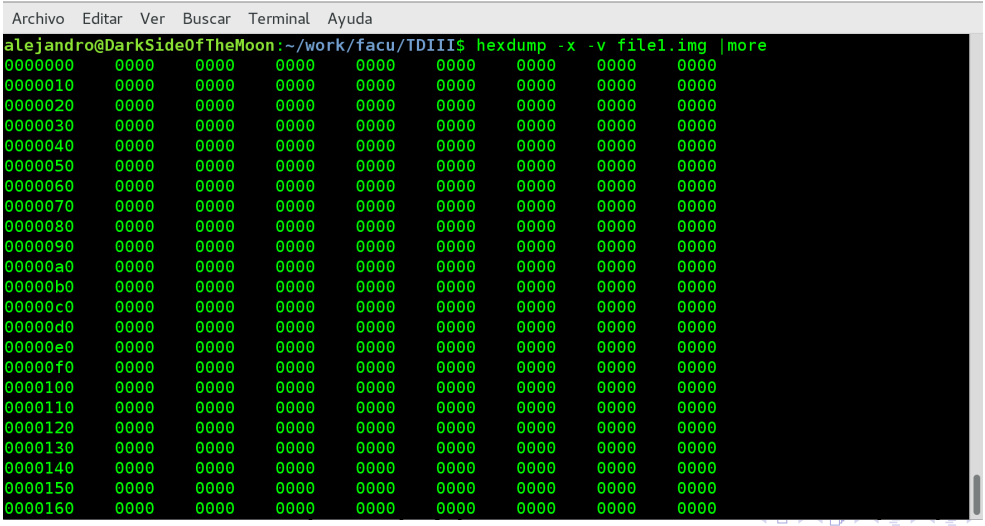
Como archivo de origen utilizamos un dispositivo virtual como `/dev/zero` que al ser leído provee una secuencia de ceros.

Utilizamos un tamaño de bloque de 1 Kbyte y establecemos que la copia se componga de 10000 bloques de este tamaño.

Como resultado tenemos un archvo de 10Mbytes lleno de ceros.

Montaje

Para visualizar el contenido del archivo podemos ejecutar ***hexdump***.



Montaje

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo losetup /dev/loop0 file1.img
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ losetup
NAME       SIZELIMIT OFFSET AUTOCLEAR RO BACK-FILE
/dev/loop0         0      0         0  0 /home/alejandro/work/facu/TDIII/file1.img
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Montaje

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo losetup /dev/loop0 file1.img
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ losetup
NAME       SIZELIMIT OFFSET AUTOCLEAR RO BACK-FILE
/dev/loop0         0      0         0  0 /home/alejandro/work/facu/TDIII/file1.img
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Asociamos el archivo a un ***loop device***

El comando ***losetup*** permite controlar y levantar ***loop devices***.
Los ***loop devices*** son abstracciones disponibles (al igual que ***/dev/zero***) en el directorio ***/dev*** con el objeto de permitir al sistema operativo el montaje de archivos que contienen una imagen de un DVD, CD, o de un disco cualquiera a partir de un punto cualquiera de la jerarquía de directorio del File System actual.

Montaje

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo mke2fs -c /dev/loop0 10000
mke2fs 1.42.13 (17-May-2015)
Descartando los bloques del dispositivo: hecho
Se está creando un sistema de ficheros con 10000 bloques de 1k y 2512 nodos-i
UUID del sistema de ficheros: 93127843-003a-4dc7-a664-501d0fe7f244
Respaldo del superbloque guardado en los bloques:
        8193

Se están revisando los bloques dañados (prueba de sólo lectura):   0.00% hecho, 0:00 transcurrid
hecho
Reservando las tablas de grupo: hecho
Escribiendo las tablas de nodos-i: hecho
Escribiendo superbloques y la información contable del sistema de ficheros: hecho

alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Montaje

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo mke2fs -c /dev/loop0 10000
mke2fs 1.42.13 (17-May-2015)
Descartando los bloques del dispositivo: hecho
Se está creando un sistema de ficheros con 10000 bloques de 1k y 2512 nodos-i
UUID del sistema de ficheros: 93127843-003a-4dc7-a664-501d0fe7f244
Respaldo del superbloque guardado en los bloques:
        8193

Se están revisando los bloques dañados (prueba de sólo lectura):   0.00% hecho, 0:00 transcurrid
```

Formateamos el medio como un File System

El comando ***mke2fs*** Construye una estructura de File System en un medio de almacenamiento.

El file system puede especificarse mediante la opción ***-t*** seguida del tipo de File system, ej: ***-t vfat*** en el caso de un FAT32. En ***/etc/mke2fs.conf***, está definido el tipo que se creará por default.

La opción ***-c*** hace que se chequeen todos los sectores en busca de sectores defectuosos antes de crear el File System.

Montaje

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo mount -t ext4 /dev/loop0 /mnt/point1
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ ls -las /mnt/point1
total 17
 1 drwxr-xr-x 3 root root  1024 ago 23 12:24 .
 4 drwxr-xr-x 4 root root  4096 ago 22 14:26 ..
12 drwx------ 2 root root 12288 ago 23 12:24 lost+found
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Montaje

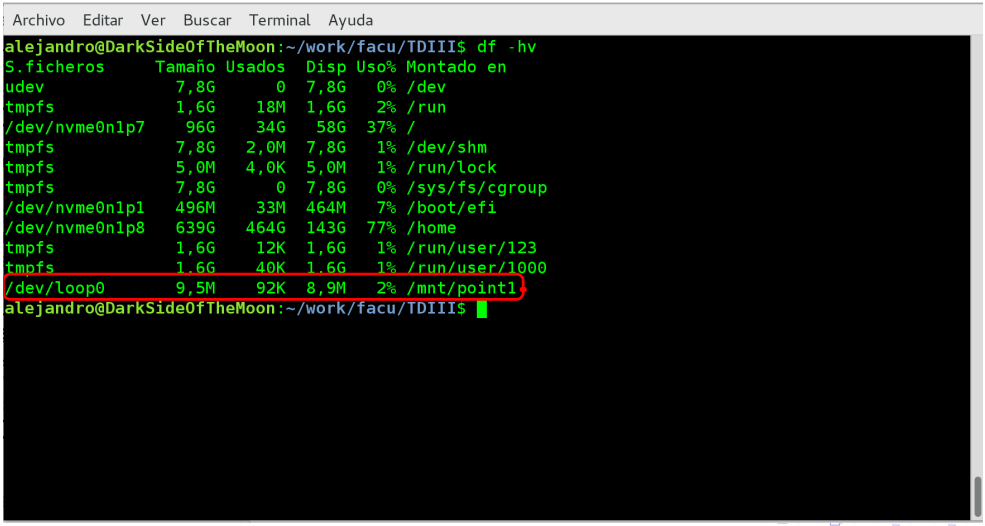
```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo mount -t ext4 /dev/loop0 /mnt/point1
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ ls -las /mnt/point1
total 17
 1 drwxr-xr-x 3 root root  1024 ago 23 12:24 .
 4 drwxr-xr-x 4 root root  4096 ago 22 14:26 ..
12 drwx------ 2 root root 12288 ago 23 12:24 lost+found
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Montamos File System

El comando ***mount*** attacha el dispositivo que contiene el medio de almacenamiento en el punto de la jerarquía de directorios que le indicamos.
Luego de montarlo el File System se encuentra accesible

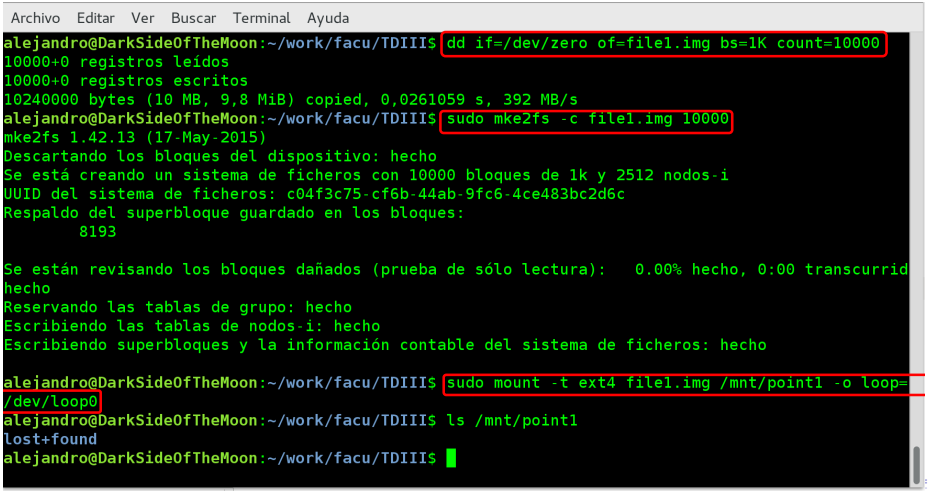
Montaje

Con el comando ***df*** es muy simple de comprobar su correcto montaje y disponibilidad.



Montaje

La opción ***-o*** del comando ***mount*** nos permite lograr el mismo resultado con tres comandos en lugar de cuatro.



Montaje

Armamos un segundo file system mapeado en un archivo que está en el primer file system...

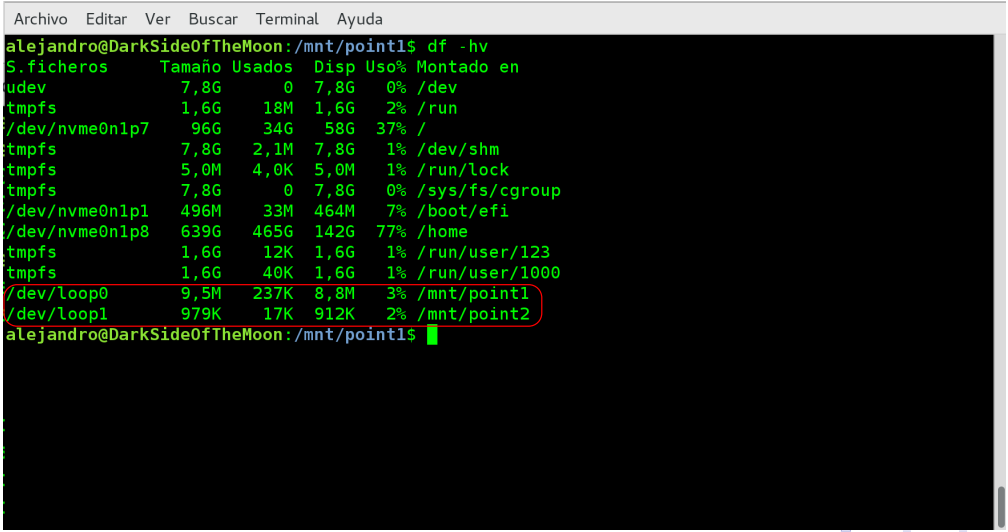
```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ cd /mnt/point1
alejandro@DarkSideOfTheMoon:/mnt/point1$ sudo dd if=/dev/zero of=file2.img bs=1K count=1000
1000+0 registros leídos
1000+0 registros escritos
1024000 bytes (1,0 MB, 1000 KiB) copied, 0,00244513 s, 419 MB/s
alejandro@DarkSideOfTheMoon:/mnt/point1$ sudo mke2fs -c file2.img
mke2fs 1.42.13 (17-May-2015)
Descartando los bloques del dispositivo: hecho
Se está creando un sistema de ficheros con 1000 bloques de 1k y 128 nodos-i

Se están revisando los bloques dañados (prueba de sólo lectura):   0.00% hecho, 0:00 transcurrid
hecho
Reservando las tablas de grupo: hecho
Escribiendo las tablas de nodos-i: hecho
Escribiendo superbloques y la información contable del sistema de ficheros: hecho

alejandro@DarkSideOfTheMoon:/mnt/point1$ sudo mount -t ext4 file2.img /mnt/point2 -o loop=/dev/l
oop1
alejandro@DarkSideOfTheMoon:/mnt/point1$ ls -las ../point2
total 17
 1 drwxr-xr-x 3 root root  1024 ago 23 14:51 .
 4 drwxr-xr-x 4 root root  4096 ago 23 14:29 ..
12 drwx------ 2 root root 12288 ago 23 14:51 lost+found
alejandro@DarkSideOfTheMoon:/mnt/point1$
```

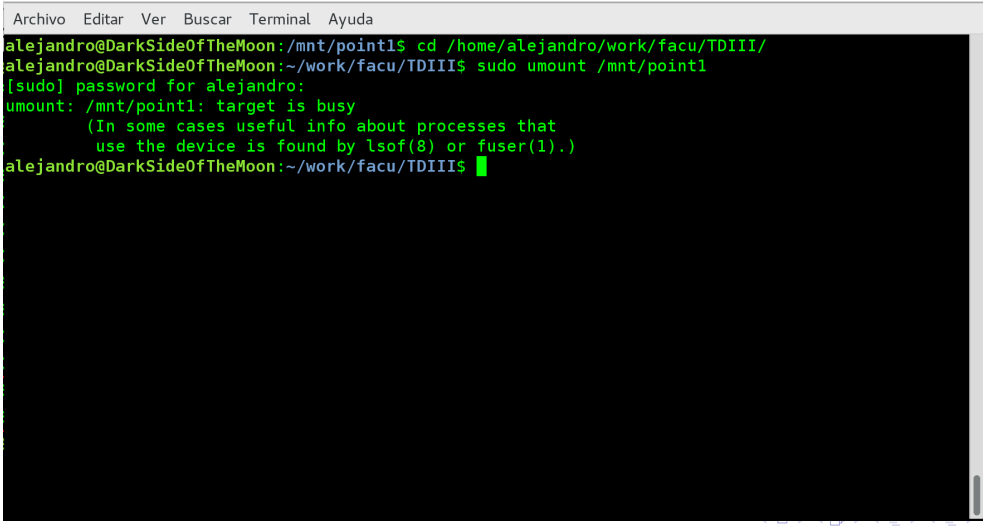
Montaje

... resultado:



Desmontaje

En este caso particular el orden no es trivial:



Desmontaje

Este es el orden correcto (***umount*** es mudo cuando sale bien):

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@DarkSideOfTheMoon:/mnt/point1$ cd /home/alejandro/work/facu/TDIII/
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo umount /mnt/point1
[sudo] password for alejandro:
umount: /mnt/point1: target is busy
        (In some cases useful info about processes that
         use the device is found by lsof(8) or fuser(1).)
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo umount /mnt/point2
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ sudo umount /mnt/point1
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$ df -hv
S.ficheros     Tamaño Usados  Disp Uso% Montado en
udev             7,8G      0  7,8G   0% /dev
tmpfs            1,6G     18M  1,6G   2% /run
/dev/nvme0n1p7    96G     34G   58G  37% /
tmpfs            7,8G    2,1M  7,8G   1% /dev/shm
tmpfs            5,0M    4,0K  5,0M   1% /run/lock
tmpfs            7,8G      0  7,8G   0% /sys/fs/cgroup
/dev/nvme0n1p1   496M     33M  464M   7% /boot/efi
/dev/nvme0n1p8   639G    465G  142G  77% /home
tmpfs            1,6G     12K  1,6G   1% /run/user/123
tmpfs            1,6G     40K  1,6G   1% /run/user/1000
alejandro@DarkSideOfTheMoon:~/work/facu/TDIII$
```

Contenido

Conceptos Básicos

Funciones

Conceptos Básicos

Funciones

Conceptos Básicos

Funciones

Principales

- El recolector de basura

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso
- Código de corrección de errores

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso
- Código de corrección de errores

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso
- Código de corrección de errores

Deseables

- Compresión

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso
- Código de corrección de errores

Deseables

- Compresión
- Ejecución in situ

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso
- Código de corrección de errores

Deseables

- Compresión
- Ejecución in situ
- Sincronización selectiva

Conceptos Básicos

Funciones

Principales

- El recolector de basura
- Gestión de bloques inválidos
- Tasa de uso
- Código de corrección de errores

Deseables

- Compresión
- Ejecución in situ
- Sincronización selectiva
- Tolerancia a apagados no programados

El recolector de basura

El recolector de basura

- El recolector de basura (garbage collector) es un proceso por medio del cual son recuperados los bloques inválidos (es decir, aquellos que contienen ciertos datos obsoletos).

El recolector de basura

- El recolector de basura (garbage collector) es un proceso por medio del cual son recuperados los bloques inválidos (es decir, aquellos que contienen ciertos datos obsoletos).

- Este proceso por lo general se ejecuta en segundo plano o cuando el sistema de archivos dispone de poco espacio libre.

El recolector de basura

- El recolector de basura (garbage collector) es un proceso por medio del cual son recuperados los bloques inválidos (es decir, aquellos que contienen ciertos datos obsoletos).
- Este proceso por lo general se ejecuta en segundo plano o cuando el sistema de archivos dispone de poco espacio libre.
- La recuperación de bloques inválidos, no solo debe efectuar el movimiento de los datos que aún son útiles en dicho bloque, sino también el borrado completo del mismo para etiquetarlo como disponible.

Gestión de bloques inválidos

Gestión de bloques inválidos

- La tasa de uso de los diferentes bloques de un dispositivo de almacenamiento no es homogénea.

Gestión de bloques inválidos

- La tasa de uso de los diferentes bloques de un dispositivo de almacenamiento no es homogénea.
- A través de la vida útil del dispositivo comienzan a aparecer bloques defectuosos.

Gestión de bloques inválidos

- La tasa de uso de los diferentes bloques de un dispositivo de almacenamiento no es homogénea.
- A través de la vida útil del dispositivo comienzan a aparecer bloques defectuosos.
- También es posible encontrar bloques en mal estado desde la fabricación misma del dispositivo.

Gestión de bloques inválidos

- La tasa de uso de los diferentes bloques de un dispositivo de almacenamiento no es homogénea.
- A través de la vida útil del dispositivo comienzan a aparecer bloques defectuosos.
- También es posible encontrar bloques en mal estado desde la fabricación misma del dispositivo.
- Si el controlador del dispositivo intenta acceder a los bloques defectuosos, habrá demoras asociadas con la verificación de la escritura.

Gestión de bloques inválidos

Gestión de bloques inválidos

- Un sistema de archivos debe disponer de un método que permita identificar los mencionados bloques y mantener un registro de los mismos a fin de evitar su utilización.

Gestión de bloques inválidos

- Un sistema de archivos debe disponer de un método que permita identificar los mencionados bloques y mantener un registro de los mismos a fin de evitar su utilización.
- Los dispositivos de buena calidad proveen este mecanismo a través de hardware relevando al sistema de archivos de esta operatoria, optimizando así los tiempos de acceso.

Tasa de uso

Tasa de uso

- El principal objetivo del mecanismo de tasa de uso (wear leveling) es permitir que el sistema de archivos distribuya las operaciones de escritura en forma homogénea a través de los diferentes bloques que componen un dispositivo de almacenamiento masivo.

Tasa de uso

- El principal objetivo del mecanismo de tasa de uso (wear leveling) es permitir que el sistema de archivos distribuya las operaciones de escritura en forma homogénea a través de los diferentes bloques que componen un dispositivo de almacenamiento masivo.

- La utilización localizada de bloques, debido a la falta del mecanismo descripto, provoca la reducción de la vida útil del dispositivo y un mayor esfuerzo por parte del gestor de bloques inválidos.

Tasa de uso

- El principal objetivo del mecanismo de tasa de uso (wear leveling) es permitir que el sistema de archivos distribuya las operaciones de escritura en forma homogénea a través de los diferentes bloques que componen un dispositivo de almacenamiento masivo.
- La utilización localizada de bloques, debido a la falta del mecanismo descripto, provoca la reducción de la vida útil del dispositivo y un mayor esfuerzo por parte del gestor de bloques inválidos.
- Adicionalmente el mecanismo debe asegurar que los datos con tiempos de vida muy prolongados, se encuentren siempre disponibles (sin errores) a través de las sucesivas lecturas.

Código de corrección de errores

Código de corrección de errores

- Tanto la tasa de uso como la gestión de bloques inválidos, permite al SO garantizar el acceso sin errores a la unidad de almacenamiento.

Código de corrección de errores

- Tanto la tasa de uso como la gestión de bloques inválidos, permite al SO garantizar el acceso sin errores a la unidad de almacenamiento.

- Sin embargo a fin de prolongar la vida útil del dispositivo es fundamental disponer de algún mecanismo que permite corregir cierta cantidad de errores en un bloque antes de declararlo como invalido.

Código de corrección de errores

- Tanto la tasa de uso como la gestión de bloques inválidos, permite al SO garantizar el acceso sin errores a la unidad de almacenamiento.
- Sin embargo a fin de prolongar la vida útil del dispositivo es fundamental disponer de algún mecanismo que permite corregir cierta cantidad de errores en un bloque antes de declararlo como invalido.
- Este mecanismo es el código corrector de errores (ECC),

Código de corrección de errores

Código de corrección de errores

- El ECC permite recuperar correctamente una cadena de información aunque esta contenga un número limitado de errores.

Código de corrección de errores

- El ECC permite recuperar correctamente una cadena de información aunque esta contenga un número limitado de errores.

- La principal restricción de la implementación de un ECC es el costo a nivel de tiempo de acceso, en contrapartida con la mayor disponibilidad del dispositivo de almacenamiento.

Código de corrección de errores

- El ECC permite recuperar correctamente una cadena de información aunque esta contenga un número limitado de errores.
- La principal restricción de la implementación de un ECC es el costo a nivel de tiempo de acceso, en contrapartida con la mayor disponibilidad del dispositivo de almacenamiento.
- Entre los ECC más conocidos se encuentran Reed-Solomon (Chen et al., 2008; Micron,2007) o Bose-Chaudhuri-Hocquenghem [BCH] (Choi et al., 2010; Junho & Wonyong, 2009; Micheloni et al., 2008).

Contenido

File systems soportados por Linux

(ver **`/proc/filesystems`**).

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple
6. Un rango de File System de Red:

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple
6. Un rango de File System de Red:
   - NFS de Sun (disponible en http://nfs.sourceforge.net/)

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple
6. Un rango de File System de Red:
   - NFS de Sun (disponible en http://nfs.sourceforge.net/)
   - SMB (IBM y Microsoft)

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple
6. Un rango de File System de Red:
   - NFS de Sun (disponible en http://nfs.sourceforge.net/)
   - SMB (IBM y Microsoft)
   - NCP de Novell

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple
6. Un rango de File System de Red:
   - NFS de Sun (disponible en http://nfs.sourceforge.net/)
   - SMB (IBM y Microsoft)
   - NCP de Novell
   - Coda (desarrollado en la Universidad Carnegie Mellon)

File systems soportados por Linux

(ver **`/proc/filesystems`**).

1. El tradicional *ext2* file system
2. Diferentes file system nativos de UNIX, como los file system de BSD, Minix, y System V.
3. Los file system de Microsoft como FAT, FAT32, y NTFS,
4. El file system ISO 9660 para CD-ROM
5. HFS de Apple
6. Un rango de File System de Red:
   - NFS de Sun (disponible en http://nfs.sourceforge.net/)
   - SMB (IBM y Microsoft)
   - NCP de Novell
   - Coda (desarrollado en la Universidad Carnegie Mellon)
7. Un rango amplio de journaling File System, including *ext3*, *ext4*, *Reiserfs*, *JFS*, *XFS*, *Btrfs*.

El Virtual File System

El Virtual File System

- Los Sistemas Operativos del slide anterior tienen diferencias (grandes por lo general) en sus detalles de implementación.

El Virtual File System

- Los Sistemas Operativos del slide anterior tienen diferencias (grandes por lo general) en sus detalles de implementación.
- Desde el punto de vista del usuario y del programador es necesario incluir una visión unificada de File System capaz de independizarse de estos detalles

El Virtual File System

- Los Sistemas Operativos del slide anterior tienen diferencias (grandes por lo general) en sus detalles de implementación.
- Desde el punto de vista del usuario y del programador es necesario incluir una visión unificada de File System capaz de independizarse de estos detalles
- El Virtual File System de Linux cumple con suma eficiencia esta tarea

El Virtual File System

- Los Sistemas Operativos del slide anterior tienen diferencias (grandes por lo general) en sus detalles de implementación.
- Desde el punto de vista del usuario y del programador es necesario incluir una visión unificada de File System capaz de independizarse de estos detalles
- El Virtual File System de Linux cumple con suma eficiencia esta tarea
- Engloba en una vista única (es decir dentro de la estructura jerárquica única que inicia en "/".), todos los file system, que tenga disponibles y montados.

El Virtual File System

- Los Sistemas Operativos del slide anterior tienen diferencias (grandes por lo general) en sus detalles de implementación.
- Desde el punto de vista del usuario y del programador es necesario incluir una visión unificada de File System capaz de independizarse de estos detalles
- El Virtual File System de Linux cumple con suma eficiencia esta tarea
- Engloba en una vista única (es decir dentro de la estructura jerárquica única que inicia en "/".), todos los file system, que tenga disponibles y montados.
- Dispone de una estructura alojada en RAM, denominada "dentry cache", que almacena los objetos (dentry) más recientemente utilizados, permitiendo acelerar el tiempo de acceso a un archivo especifico.

La visión del usuario

El usuario ve una estructura de Directorios

La visión del *file system* por parte del usuario, es una disposición jerárquica de directorios que contienen archivos y otros directorios, los cuales son identificados por sus nombres, conformando una estructura de datos clásica de árbol binario.

La visión del usuario

El usuario ve una estructura de Directorios

La visión del *file system* por parte del usuario, es una disposición jerárquica de directorios que contienen archivos y otros directorios, los cuales son identificados por sus nombres, conformando una estructura de datos clásica de árbol binario.

Punto de inicio

Esta jerarquía en sistemas *"UNIX like"* comienza en un único directorio conocido como *root*, el cual es representado como **"/"**.

La visión del usuario

El usuario ve una estructura de Directorios

La visión del *file system* por parte del usuario, es una disposición jerárquica de directorios que contienen archivos y otros directorios, los cuales son identificados por sus nombres, conformando una estructura de datos clásica de árbol binario.

Punto de inicio

Esta jerarquía en sistemas *"UNIX like"* comienza en un único directorio conocido como *root*, el cual es representado como **"/"**.

Elementos del arbol binario

Cada elemento que integra la estructura de datos se denomina **nodo**.

File systems en Linux

File systems en Linux

- Desde el punto de vista del usuario, el *File system Hierarchy Standard (FHS)* define los directorios y sus contenidos en los sistemas simil UNIX.

File systems en Linux

- Desde el punto de vista del usuario, el *File system Hierarchy Standard (FHS)* define los directorios y sus contenidos en los sistemas simil UNIX.
- Todos los archivos y directorios están organizados bajo el directorio *root*, incluso si están en diferentes particiones o dispositivos físicos.

File systems en Linux

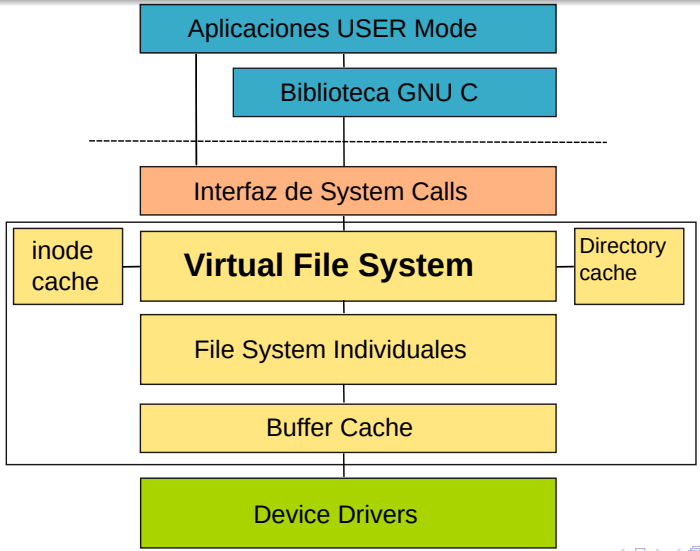
- Desde el punto de vista del usuario, el *File system Hierarchy Standard (FHS)* define los directorios y sus contenidos en los sistemas simil UNIX.
- Todos los archivos y directorios están organizados bajo el directorio *root*, incluso si están en diferentes particiones o dispositivos físicos.
- Por otro lado, desde el punto de vista del sistema operativo, el sistema de archivos es completamente plano. No tiene una estructura jerárquica, ni directorios, ni archivos, ni programas.

File systems en Linux

- Desde el punto de vista del usuario, el *File system Hierarchy Standard (FHS)* define los directorios y sus contenidos en los sistemas simil UNIX.
- Todos los archivos y directorios están organizados bajo el directorio *root*, incluso si están en diferentes particiones o dispositivos físicos.
- Por otro lado, desde el punto de vista del sistema operativo, el sistema de archivos es completamente plano. No tiene una estructura jerárquica, ni directorios, ni archivos, ni programas.

El kernel utiliza una entidad llamada *inodo* para representar un archivo.

Arquitectura a nivel componentes



Componentes

Componentes

- El Virtual File System exporta un grupo de funciones comunes a todos los File Systems que operan en la capa infrerior, generando hacia la capa de System Calls un nivel de abstracción respecto del file system particular al que se accede.

Componentes

- El Virtual File System exporta un grupo de funciones comunes a todos los File Systems que operan en la capa infrerior, generando hacia la capa de System Calls un nivel de abstracción respecto del file system particular al que se accede.
- Así las aplicaciones y las funciones de las bibliotecas de C y GNU usan un set común de funciones.

Componentes

- El Virtual File System exporta un grupo de funciones comunes a todos los File Systems que operan en la capa infrerior, generando hacia la capa de System Calls un nivel de abstracción respecto del file system particular al que se accede.
- Así las aplicaciones y las funciones de las bibliotecas de C y GNU usan un set común de funciones.
- Cada File System individual exporta un grupo de funciones para que el VFS las pueda acceder.

Componentes

- El Virtual File System exporta un grupo de funciones comunes a todos los File Systems que operan en la capa infrerior, generando hacia la capa de System Calls un nivel de abstracción respecto del file system particular al que se accede.
- Así las aplicaciones y las funciones de las bibliotecas de C y GNU usan un set común de funciones.
- Cada File System individual exporta un grupo de funciones para que el VFS las pueda acceder.
- En el buffer cache se encolan los pedidos de escritura y lectura de cada file system a los block devices.

Estructuras principales

Linux mira los file systems a través de cuatro objetos

Estructuras principales

Linux mira los file systems a través de cuatro objetos

- ***Superblock***.
  Reside en el root y en él están definidas todas las características del File System

Estructuras principales

Linux mira los file systems a través de cuatro objetos

- ***Superblock***.
  Reside en el root y en él están definidas todas las características del File System
- ***inode***
  Cada objeto mantenido dentro de un file system (archivo, directorio, device, etc) tiene asociado un ***inode***.

Estructuras principales

Linux mira los file systems a través de cuatro objetos

- ***Superblock***.
  Reside en el root y en él están definidas todas las características del File System
- ***inode***
  Cada objeto mantenido dentro de un file system (archivo, directorio, device, etc) tiene asociado un ***inode***.
- ***dentry***.
  Se emplean para trasladar nombres a ***inodes***

Estructuras principales

Linux mira los file systems a través de cuatro objetos

- ***Superblock***.
  Reside en el root y en él están definidas todas las características del File System
- ***inode***
  Cada objeto mantenido dentro de un file system (archivo, directorio, device, etc) tiene asociado un ***inode***.
- ***dentry***.
  Se emplean para trasladar nombres a ***inodes***
- ***file***.
  Cada vez que un proceso abre un archivo se genera para ese proceso un objeto ***file*** para mantener allí toda la información lógica relacionada con el acceso al archivo por parte del proceso.

Registro en el kernel de un File System

Registro en el kernel de un File System

- El kernel mantiene una lista de los diferentes tipos de File System que soporta (no necesariamente pueden estar activos o montados, sino que solo son los que puede aceptar)

Registro en el kernel de un File System

- El kernel mantiene una lista de los diferentes tipos de File System que soporta (no necesariamente pueden estar activos o montados, sino que solo son los que puede aceptar)
- Estos están accesibles incluso desde el espacio de usuario en ***/proc/filesystems***.

Registro en el kernel de un File System

- El kernel mantiene una lista de los diferentes tipos de File System que soporta (no necesariamente pueden estar activos o montados, sino que solo son los que puede aceptar)
- Estos están accesibles incluso desde el espacio de usuario en ***/proc/filesystems***.
- Para agregar un nuevo file system a la lista está disponible la función `register_filesy`

Registro en el kernel de un File System

- El kernel mantiene una lista de los diferentes tipos de File System que soporta (no necesariamente pueden estar activos o montados, sino que solo son los que puede aceptar)
- Estos están accesibles incluso desde el espacio de usuario en ***/proc/filesystems***.
- Para agregar un nuevo file system a la lista está disponible la función `register_filesy`
- Esta función recibe como único argumento un puntero a `struct file_system_type`, y la enlaza a la lista de file systems soportados.

Registro en el kernel de un File System

```
struct file_system_type {
        const char *name;
        int fs_flags;
#define FS_REQUIRES_DEV         1
#define FS_BINARY_MOUNTDATA     2
#define FS_HAS_SUBTYPE          4
#define FS_USERNS_MOUNT         8       /* Can be mounted by userns root */
#define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move() during rename() internally. */
        struct dentry *(*mount) (struct file_system_type *, int,
                        const char *, void *);
        void (*kill_sb) (struct super_block *);
        struct module *owner;
        struct file_system_type * next;
        struct hlist_head fs_supers;

        struct lock_class_key s_lock_key;
        struct lock_class_key s_umount_key;
        struct lock_class_key s_vfs_rename_key;
        struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];

        struct lock_class_key i_lock_key;
        struct lock_class_key i_mutex_key;
        struct lock_class_key i_mutex_dir_key;
};
```

Registro en el kernel de un File System

```
struct file_system_type {
        const char *name;
        int fs_flags;
#define FS_REQUIRES_DEV         1
#define FS_BINARY_MOUNTDATA     2
#define FS_HAS_SUBTYPE          4
#define FS_USERNS_MOUNT         8       /* Can be mounted by userns root */
#define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move() during rename() internally. */
        struct dentry *(*mount) (struct file_system_type *, int,
                        const char *, void *);
        void (*kill_sb) (struct super_block *);
        struct module *owner;
        struct file_system_type * next;
        struct hlist_head fs_supers;

        struct lock_class_key s_lock_key;
        struct lock_class_key s_umount_key;
        struct lock_class_key s_vfs_rename_key;
        struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];

        struct lock_class_key i_lock_key;
        struct lock_class_key i_mutex_key;
        struct lock_class_key i_mutex_dir_key;
};
```

```
struct file_system_type {
        const char *name;
        int fs_flags;
#define FS_REQUIRES_DEV         1
#define FS_BINARY_MOUNTDATA     2
#define FS_HAS_SUBTYPE          4
#define FS_USERNS_MOUNT         8       /* Can be mounted by usern
#define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move()
        struct dentry *(*mount) (struct file_system_type *, int,
                        const char *, void *);
        void (*kill_sb) (struct super_block *);
        struct module *owner;
        struct file_system_type * next;
        struct hlist_head fs_supers;

        struct lock_class_key s_lock_key;
        struct lock_class_key s_umount_key;
        struct lock_class_key s_vfs_rename_key;
        struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];

        struct lock_class_key i_lock_key;
        struct lock_class_key i_mutex_key;
        struct lock_class_key i_mutex_dir_key;
};
```

Super Block

Super Block

- Aquí se describen las características del File System.

Super Block

- Aquí se describen las características del File System.
- Información vital para su operación.

Super Block

- Aquí se describen las características del File System.
- Información vital para su operación.
- Normalmente está generado de manera permanente en el Primer Bloque del File system en el medio de almacenamiento.

Super Block

- Aquí se describen las características del File System.
- Información vital para su operación.
- Normalmente está generado de manera permanente en el Primer Bloque del File system en el medio de almacenamiento.
- Si no lo está puede se generado on the fly en el momento del montaje del File System (es lo menos común)

Super Block

```
struct super_block {
        struct list_head        s_list;         /* Keep this first */
        dev_t                   s_dev;          /* search index; _not_ kdev_t */
        unsigned char           s_blocksize_bits;
        unsigned long           s_blocksize;
        loff_t                  s_maxbytes;     /* Max file size */
        struct file_system_type *s_type;
        const struct super_operations   *s_op;
        const struct dquot_operations   *dq_op;
        const struct quotactl_ops       *s_qcop;
        const struct export_operations *s_export_op;
        unsigned long           s_flags;
        unsigned long           s_iflags;       /* internal SB_I_* flags */
        unsigned long           s_magic;
        struct dentry           *s_root;
        struct rw_semaphore     s_umount;
        int                     s_count;
        atomic_t                s_active;
#ifdef CONFIG_SECURITY
        void                    *s_security;
#endif
        const struct xattr_handler **s_xattr;
```

Información en Super Block

```
struct super_operations {
        struct inode *(*alloc_inode)(struct super_block *sb);
        void (*destroy_inode)(struct inode *);

        void (*dirty_inode) (struct inode *, int flags);
        int (*write_inode) (struct inode *, struct writeback_control *wbc);
        int (*drop_inode) (struct inode *);
        void (*evict_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        int (*sync_fs)(struct super_block *sb, int wait);
        int (*freeze_super) (struct super_block *);
        int (*freeze_fs) (struct super_block *);
        int (*thaw_super) (struct super_block *);
        int (*unfreeze_fs) (struct super_block *);
        int (*statfs) (struct dentry *, struct kstatfs *);
        int (*remount_fs) (struct super_block *, int *, char *);
        void (*umount_begin) (struct super_block *);

        int (*show_options)(struct seq_file *, struct dentry *);
        int (*show_devname)(struct seq_file *, struct dentry *);
        int (*show_path)(struct seq_file *, struct dentry *);
        int (*show_stats)(struct seq_file *, struct dentry *);
#ifdef CONFIG_QUOTA
        ssize_t (*quota_read)(struct super_block *, int, char *, size_t, loff_t);
        ssize_t (*quota_write)(struct super_block *, int, const char *, size_t, loff_t);
        struct dquot **(*get_dquots)(struct inode *);
#endif
        int (*bdev_try_to_free_page)(struct super_block*, struct page*, gfp_t);
        long (*nr_cached_objects)(struct super_block *,
                                  struct shrink_control *);
        long (*free_cached_objects)(struct super_block *,
                                    struct shrink_control *);
};
```

Estructura inode

```
struct inode {
        umode_t                 i_mode;
        unsigned short          i_opflags;
        kuid_t                  i_uid;
        kgid_t                  i_gid;
        unsigned int            i_flags;

#ifdef CONFIG_FS_POSIX_ACL
        struct posix_acl        *i_acl;
        struct posix_acl        *i_default_acl;
#endif

        const struct inode_operations   *i_op;
        struct super_block      *i_sb;
        struct address_space    *i_mapping;

#ifdef CONFIG_SECURITY
        void                    *i_security;
#endif

        /* Stat data, not accessed from path walking */
        unsigned long           i_ino;
        /*
         * Filesystems may only read i_nlink directly.
         * following functions for modification:
         *
         *    (set|clear|inc|drop)_nlink
         *    inode_(inc|dec)_link_count
         */
        union {
                const unsigned int i_nlink;
                unsigned int __i_nlink;
        };
        dev_t                   i_rdev;
        loff_t                  i_size;
        struct timespec         i_atime;
        struct timespec         i_mtime;
        struct timespec         i_ctime;
        spinlock_t              i_lock; /* i_blocks, i_b
        unsigned short          i_bytes;
        unsigned int            i_blkbits;
        blkcnt_t                i_blocks;

#ifdef __NEED_I_SIZE_ORDERED
        seqcount_t              i_size_seqcount;
#endif

        /* Misc */
        unsigned long           i_state;
        struct rw_semaphore     i_rwsem;

        unsigned long           dirtied_when;   /* jiffi
        unsigned long           dirtied_time_when;

        struct hlist_node       i_hash;
        struct list_head        i_io_list;      /* backi
#ifdef CONFIG_CGROUP_WRITEBACK
        struct bdi_writeback    *i_wb;          /* the a

        /* foreign inode detection, see wbc_detach_inode
        int                     i_wb_frn_winner;
        u16                     i_wb_frn_avg_time;
        u16                     i_wb_frn_history;
#endif
```

Estuctura inode

```
        struct list_head        i_lru;          /* inode LRU list */
        struct list_head        i_sb_list;
        struct list_head        i_wb_list;      /* backing dev writeback list */
        union {
                struct hlist_head       i_dentry;
                struct rcu_head         i_rcu;
        };
        u64                     i_version;
        atomic_t                i_count;
        atomic_t                i_dio_count;
        atomic_t                i_writecount;
#ifdef CONFIG_IMA
        atomic_t                i_readcount; /* struct files open R
#endif
        const struct file_operations    *i_fop;/* former ->i_op->d
        struct file_lock_context        *i_flctx;
        struct address_space    i_data;
        struct list_head        i_devices;
        union {
                struct pipe_inode_info  *i_pipe;
                struct block_device     *i_bdev;
                struct cdev             *i_cdev;
                char                    *i_link;
                unsigned                i_dir_seq;
        };

        __u32                   i_generation;

#ifdef CONFIG_FSNOTIFY
        __u32                   i_fsnotify_mask; /* all events this
        struct fsnotify_mark_connector __rcu    *i_fsnotify_marks;
#endif
```

```
#if IS_ENABLED(CONFIG_FS_ENCRYPTION)
        struct fscrypt_info     *i_crypt_info;
#endif

        void                    *i_private; /* fs or devi
};
```

Estructura inode_operations

```
struct inode_operations {
        struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int);
        const char * (*get_link) (struct dentry *, struct inode *, struct delayed_call *);
        int (*permission) (struct inode *, int);
        struct posix_acl * (*get_acl)(struct inode *, int);

        int (*readlink) (struct dentry *, char __user *,int);

        int (*create) (struct inode *,struct dentry *, umode_t, bool);
        int (*link) (struct dentry *,struct inode *,struct dentry *);
        int (*unlink) (struct inode *,struct dentry *);
        int (*symlink) (struct inode *,struct dentry *,const char *);
        int (*mkdir) (struct inode *,struct dentry *,umode_t);
        int (*rmdir) (struct inode *,struct dentry *);
        int (*mknod) (struct inode *,struct dentry *,umode_t,dev_t);
        int (*rename) (struct inode *, struct dentry *,
                       struct inode *, struct dentry *, unsigned int);
        int (*setattr) (struct dentry *, struct iattr *);
        int (*getattr) (const struct path *, struct kstat *, u32, unsigned int);
        ssize_t (*listxattr) (struct dentry *, char *, size_t);
        int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                      u64 len);
        int (*update_time)(struct inode *, struct timespec *, int);
        int (*atomic_open)(struct inode *, struct dentry *,
                           struct file *, unsigned open_flag,
                           umode_t create_mode, int *opened);
        int (*tmpfile) (struct inode *, struct dentry *, umode_t);
        int (*set_acl)(struct inode *, struct posix_acl *, int);
} ____cacheline_aligned;
```

Estuctura file_operations

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
        unsigned (*mmap_capabilities)(struct file *);
#endif
```

Estuctura file_operations

```
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
        int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
                        u64);
        ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,
                        u64);
};
```

Estuctura file_operations

```
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
        int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
                        u64);
        ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,
                        u64);
};
```

Importante

- La estructura file_operations es un componente fundamental

Estuctura file_operations

```
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
        int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
                        u64);
        ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,
                        u64);
};
```

Importante

- La estructura file_operations es un componente fundamental
- Es uno de los pilares sobre los que se implementa el paradigma “everything is a file”

Estuctura file_operations

```
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
        int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
                        u64);
        ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,
                        u64);
};
```

Importante

- La estructura file_operations es un componente fundamental
- Es uno de los pilares sobre los que se implementa el paradigma “everything is a file”
- Al abordar el desarrollo de un device driver sobre Linux terminaremos de valorar su importancia.

Estructura dentry

```
struct dentry {
        /* RCU lookup touched fields */
        unsigned int d_flags;           /* protected by d_lock */
        seqcount_t d_seq;               /* per dentry seqlock */
        struct hlist_bl_node d_hash;    /* lookup hash list */
        struct dentry *d_parent;        /* parent directory */
        struct qstr d_name;
        struct inode *d_inode;          /* Where the name belongs to - NULL is
                                         * negative */
        unsigned char d_iname[DNAME_INLINE_LEN];        /* small names */

        /* Ref lookup also touches following */
        struct lockref d_lockref;       /* per-dentry lock and refcount */
        const struct dentry_operations *d_op;
        struct super_block *d_sb;       /* The root of the dentry tree */
        unsigned long d_time;           /* used by d_revalidate */
        void *d_fsdata;                 /* fs-specific data */

        union {
                struct list_head d_lru;         /* LRU list */
                wait_queue_head_t *d_wait;      /* in-lookup ones only */
        };
        struct list_head d_child;       /* child of parent list */
        struct list_head d_subdirs;     /* our children */
        /*
         * d_alias and d_rcu can share memory
         */
        union {
                struct hlist_node d_alias;      /* inode alias list */
                struct hlist_bl_node d_in_lookup_hash;  /* only for in-lookup ones */
                struct rcu_head d_rcu;
        } d_u;
};
```

Estucturas dentry_operation y wait_queue_head

```
struct dentry_operations {
        int (*d_revalidate)(struct dentry *, unsigned int);
        int (*d_weak_revalidate)(struct dentry *, unsigned int);
        int (*d_hash)(const struct dentry *, struct qstr *);
        int (*d_compare)(const struct dentry *,
                         unsigned int, const char *, const struct qstr *);
        int (*d_delete)(const struct dentry *);
        int (*d_init)(struct dentry *);
        void (*d_release)(struct dentry *);
        void (*d_prune)(struct dentry *);
        void (*d_iput)(struct dentry *, struct inode *);
        char *(*d_dname)(struct dentry *, char *, int);
        struct vfsmount *(*d_automount)(struct path *);
        int (*d_manage)(const struct path *, bool);
        struct dentry *(*d_real)(struct dentry *, const struct inode *,
                                 unsigned int);
} ____cacheline_aligned;
```

```
struct __wait_queue_head {
        spinlock_t              lock;
        struct list_head        task_list;
};
typedef struct __wait_queue_head wait_queue_head_t;
```

Estuctura file

```
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
        struct inode            *f_inode;       /* cached
        const struct file_operations    *f_op;

        /*
         * Protects f_ep_links, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t              f_lock;
        atomic_long_t           f_count;
        unsigned int            f_flags;
        fmode_t                 f_mode;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;

        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        /* needed for tty driver, and maybe others */
        void                    *private_data;

#ifdef CONFIG_EPOLL
        /* Used by fs/eventpoll.c to link all the hooks to tl
        struct list_head        f_ep_links;
        struct list_head        f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
        struct address_space    *f_mapping;
} __attribute__((aligned(4)));  /* lest something weird decid

struct file_handle {
        __u32 handle_bytes;
        int handle_type;
        /* file identifier */
        unsigned char f_handle[0];
};
```

Organización de la información

Organización de la información

- Los medios de almacenamiento se organizan físicamente en sectores.

Organización de la información

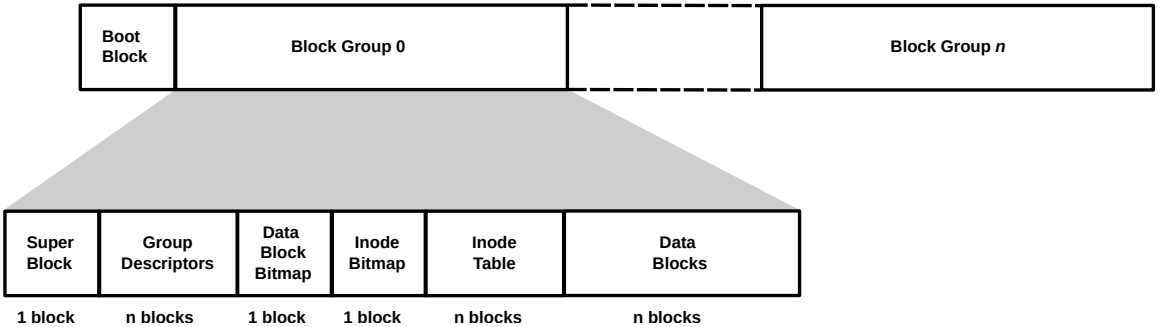
- Los medios de almacenamiento se organizan físicamente en sectores.
- El tamaño del sector depende del tipo de medio. Típicamente es de 512 bytes en discos magnéticos, 2048 bytes en memorias NAND Flash y para discos ópticos.

Organización de la información

- Los medios de almacenamiento se organizan físicamente en sectores.
- El tamaño del sector depende del tipo de medio. Típicamente es de 512 bytes en discos magnéticos, 2048 bytes en memorias NAND Flash y para discos ópticos.
- El Virtual File System los agrupa en bloques cada uno de los cuales tiene 8192 bytes

Organización de la información

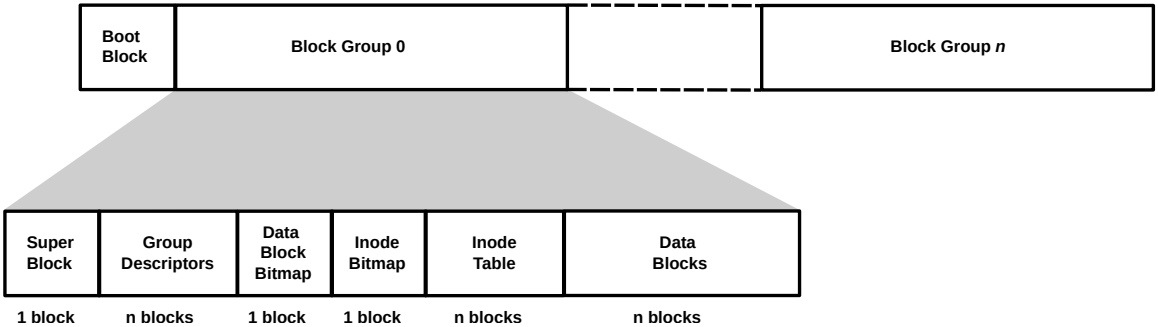
- Los medios de almacenamiento se organizan físicamente en sectores.
- El tamaño del sector depende del tipo de medio. Típicamente es de 512 bytes en discos magnéticos, 2048 bytes en memorias NAND Flash y para discos ópticos.
- El Virtual File System los agrupa en bloques cada uno de los cuales tiene 8192 bytes



Caso de ejemplo Ext2

Organización de la información: Ext2

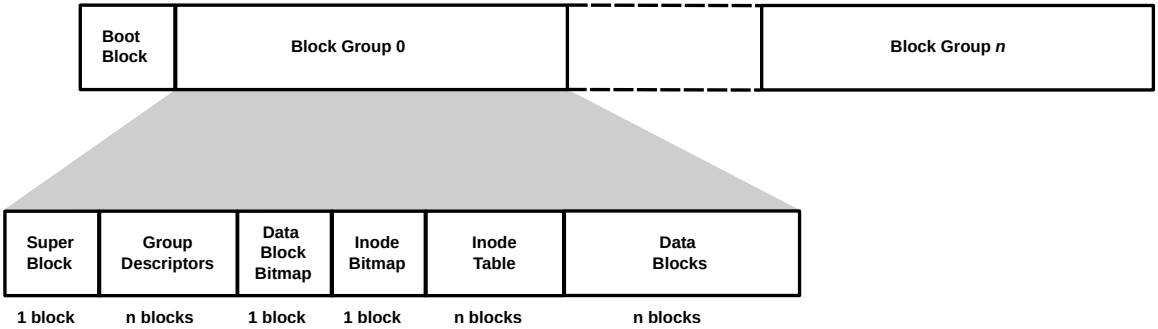
- El criterio de dividir el disco en Grupos de Bloques responde a una estrategia para reducir a fragmentación.



Caso de ejemplo Ext2

Organización de la información: Ext2

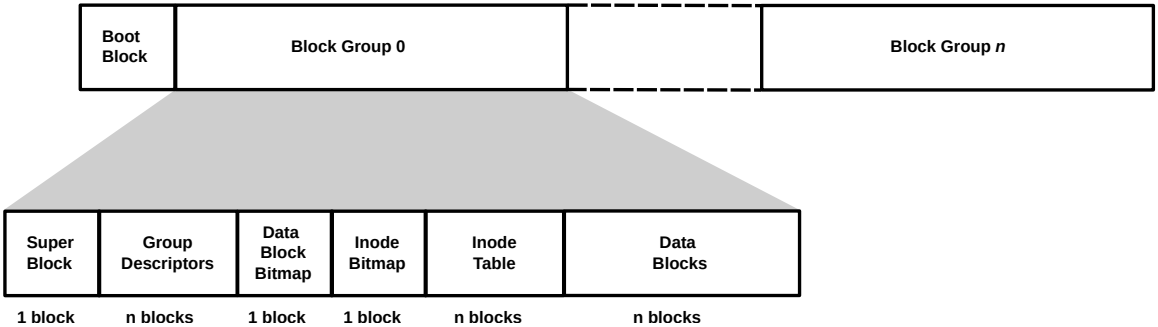
- El criterio de dividir el disco en Grupos de Bloques responde a una estrategia para reducir a fragmentación.
- El Virtual File System Manager tratará por todos los medios de ubicar todos los bloques de datos de un mismo archivo dentro del mismo Grupo de Bloques.



Caso de ejemplo Ext2

Organización de la información: Ext2

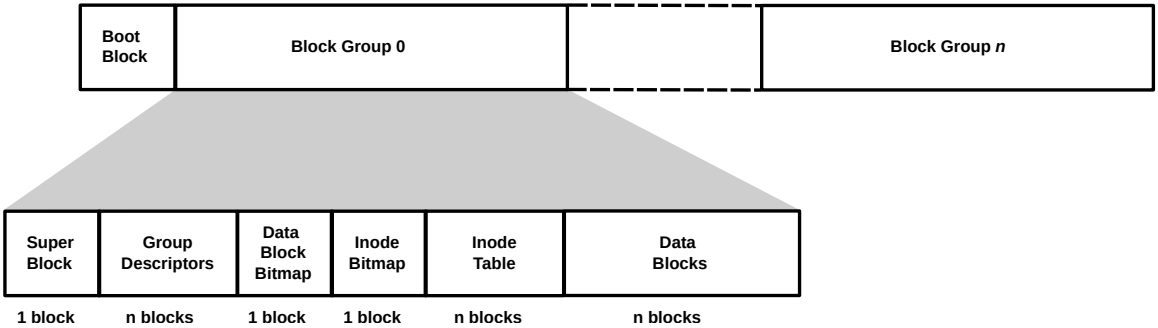
- El kernel utiliza solo el SuperBlock y el Group Descriptor del primer bloque. Los demás son copias de seguridad y son idénticos a éste.



Caso de ejemplo Ext2

Organización de la información: Ext2

- El kernel utiliza solo el SuperBlock y el Group Descriptor del primer bloque. Los demás son copias de seguridad y son idénticos a éste.
- El tamaño de un Block Group depende del tamaño de la partición del disco en la que se implementa el File System y del tamaño de bloque del disco.



Caso de ejemplo Ext2

Organización de la información: Ext2

Organización de la información: Ext2

Superbloque Para Ext2 contiene los siguientes objetos:

- Número total de inodos

Organización de la información: Ext2

Superbloque Para Ext2 contiene los siguientes objetos:

- Número total de inodos
- Tamaño del File System expresado en Bloques

Organización de la información: Ext2

Superbloque Para Ext2 contiene los siguientes objetos:

- Número total de inodos
- Tamaño del File System expresado en Bloques
- Contadores de inodos libres y de Bloques libres

Organización de la información: Ext2

Superbloque Para Ext2 contiene los siguientes objetos:

- Número total de inodos
- Tamaño del File System expresado en Bloques
- Contadores de inodos libres y de Bloques libres
- Cantidad de inodos por Grupo de Bloque y de Bloques por Grupo de Bloque

Organización de la información: Ext2

Superbloque Para Ext2 contiene los siguientes objetos:

- Número total de inodos
- Tamaño del File System expresado en Bloques
- Contadores de inodos libres y de Bloques libres
- Cantidad de inodos por Grupo de Bloque y de Bloques por Grupo de Bloque
- Identificador del File System (128 bits)

Organización de la información: Ext2

Superbloque Para Ext2 contiene los siguientes objetos:

- Número total de inodos
- Tamaño del File System expresado en Bloques
- Contadores de inodos libres y de Bloques libres
- Cantidad de inodos por Grupo de Bloque y de Bloques por Grupo de Bloque
- Identificador del File System (128 bits)
- Contador de montajes (llevaba la cuenta para disparar fsck antes del siguiente montaje de una cantidad de montajes consecutivos predefinida

Organización de la información: Ext2

Organización de la información: Ext2

Group Descriptor Para Ext2 contiene los siguientes objetos:

Organización de la información: Ext2

Group Descriptor Para Ext2 contiene los siguientes objetos:

- Números de bloque en que se encuentran respectivamente el Bitmap de inodos y el Bitmap de Bloques

Organización de la información: Ext2

Group Descriptor Para Ext2 contiene los siguientes objetos:

- Números de bloque en que se encuentran respectivamente el Bitmap de inodos y el Bitmap de Bloques
- Números de bloque en que comienza la Tabla de inodos

Organización de la información: Ext2

Group Descriptor Para Ext2 contiene los siguientes objetos:

- Números de bloque en que se encuentran respectivamente el Bitmap de inodos y el Bitmap de Bloques
- Números de bloque en que comienza la Tabla de inodos
- Cantidad de inodos libres y de Bloques libres

Organización de la información: Ext2

Group Descriptor Para Ext2 contiene los siguientes objetos:

- Números de bloque en que se encuentran respectivamente el Bitmap de inodos y el Bitmap de Bloques
- Números de bloque en que comienza la Tabla de inodos
- Cantidad de inodos libres y de Bloques libres
- Cantidad de directorios en el Grupo montajes consecutivos predefinida

Organización de la información: Ext2

Organización de la información: Ext2

Tabla de inodos Se compone de una sucesión de estructuras inode, similares a la vista para VFS. Todods los nodos miden la misma cantidad de bytes. Ocupa bloques consecutivos. Para Ext2 contiene:

Organización de la información: Ext2

Tabla de inodos Se compone de una sucesión de estructuras inode, similares a la vista para VFS. Todods los nodos miden la misma cantidad de bytes. Ocupa bloques consecutivos. Para Ext2 contiene:

- Tipo de nodo y derechos de acceso.

Organización de la información: Ext2

Tabla de inodos Se compone de una sucesión de estructuras inode, similares a la vista para VFS. Todods los nodos miden la misma cantidad de bytes. Ocupa bloques consecutivos. Para Ext2 contiene:

- Tipo de nodo y derechos de acceso.
- Identificadores de dueño y grupo.

Organización de la información: Ext2

Tabla de inodos Se compone de una sucesión de estructuras inode, similares a la vista para VFS. Todods los nodos miden la misma cantidad de bytes. Ocupa bloques consecutivos. Para Ext2 contiene:

- Tipo de nodo y derechos de acceso.
- Identificadores de dueño y grupo.
- Tamaño en bytes del nodo.

Organización de la información: Ext2

Tabla de inodos Se compone de una sucesión de estructuras inode, similares a la vista para VFS. Todods los nodos miden la misma cantidad de bytes. Ocupa bloques consecutivos. Para Ext2 contiene:

- Tipo de nodo y derechos de acceso.
- Identificadores de dueño y grupo.
- Tamaño en bytes del nodo.
- Timestamps diversos.

Organización de la información: Ext2

Tabla de inodos Se compone de una sucesión de estructuras inode, similares a la vista para VFS. Todods los nodos miden la misma cantidad de bytes. Ocupa bloques consecutivos. Para Ext2 contiene:

- Tipo de nodo y derechos de acceso.
- Identificadores de dueño y grupo.
- Tamaño en bytes del nodo.
- Timestamps diversos.
- Array de 15 punteros a bloque

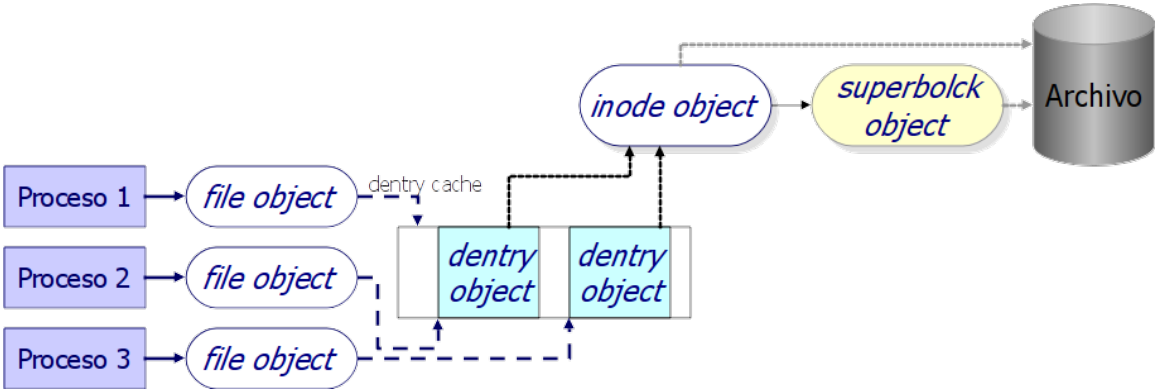
Organización de la información: Ext2

Organización de la información: Ext2

Bitmaps de Bloques y de inodos Se trata de un bloque cuyo mapeo es a nivel de bit y cada bit corresponde según corresponda al Bloque o inodo cuyo orden dentro del Block Group corresponde al indice del bit: El i-ésimo bit de Bitmap indica el estado del i-ésimo Bloque o inodo según corresponda dentro del Block Group.

Contenido

Common File Model



Contenido

File systems en Linux

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  número de *inodo*

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  número de *inodo*
  usuario owner

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo
  - tipo

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo
  - tipo
  - permisos

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo
  - tipo
  - permisos
  - fechas de acceso, creación y modificación

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo
  - tipo
  - permisos
  - fechas de acceso, creación y modificación
  - tamaño

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo
  - tipo
  - permisos
  - fechas de acceso, creación y modificación
  - tamaño
  - ubicación física en disco

File systems en Linux

- Un *inodo* es un ítem en una lista administrada por el kernel.
- contiene información única sobre un archivo
  - número de *inodo*
  - usuario owner
  - grupo
  - tipo
  - permisos
  - fechas de acceso, creación y modificación
  - tamaño
  - ubicación física en disco
- Podemos acceder a estos datos con los comandos `ls -i` y `stat`.

File systems en Linux

Por ejemplo:

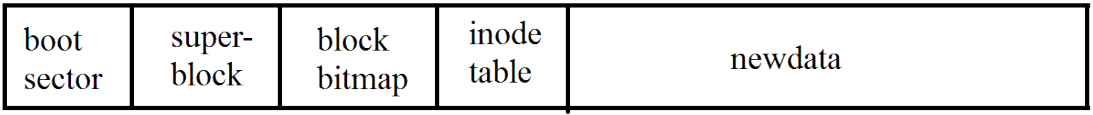
```
alejandro@DarkSideOfTheMoon:~$ ls -i Networking.pdf
28714759 Networking.pdf
alejandro@DarkSideOfTheMoon:~$ stat Networking.pdf
  Fichero: 'Networking.pdf'
  Tamaño: 3560397       Bloques: 6960       Bloque E/S: 4096   fichero regular
Dispositivo: 803h/2051d Nodo-i: 28714759    Enlaces: 1
Acceso: (0644/-rw-r--r--)  Uid: ( 1001/alejandro)   Gid: ( 1001/alejandro)
      Acceso: 2016-05-20 06:02:05.763992384 -0300
Modificación: 2015-04-22 07:18:32.159317238 -0300
      Cambio: 2015-09-07 19:31:26.122342774 -0300
    Creación: -
alejandro@DarkSideOfTheMoon:~$
```

Ejemplo de uso de `ls -i` y `stat`.

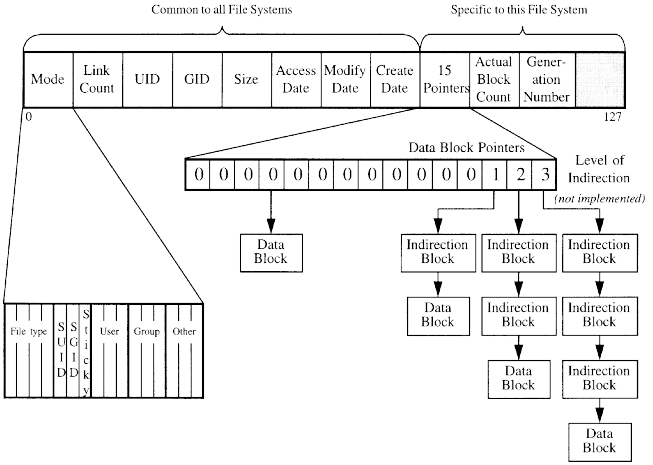
Organización de los archivos en el medio físico

layout

Los sistemas de archivos generalmente reservan espacio en sus particiones o volúmenes lógicos para varios datos extra que deben manejar, los cuales refieren al funcionamiento de éste.



Detalle de un inodo



Contenido

Servicios básicos

Un *filesystem* debe proveer varios servicios o funciones que mejoran la administración de los archivos, como son:

Servicios básicos

Un *filesystem* debe proveer varios servicios o funciones que mejoran la administración de los archivos, como son:

- Manejo de espacio

Servicios básicos

Un *filesystem* debe proveer varios servicios o funciones que mejoran la administración de los archivos, como son:

- Manejo de espacio
- Nombres de archivo y directorios

Servicios básicos

Un *filesystem* debe proveer varios servicios o funciones que mejoran la administración de los archivos, como son:

- Manejo de espacio
- Nombres de archivo y directorios
- Metadatos

Servicios básicos

Un *filesystem* debe proveer varios servicios o funciones que mejoran la administración de los archivos, como son:

- Manejo de espacio
- Nombres de archivo y directorios
- Metadatos
- Administración de acceso

Servicios básicos

Un *filesystem* debe proveer varios servicios o funciones que mejoran la administración de los archivos, como son:

- Manejo de espacio
- Nombres de archivo y directorios
- Metadatos
- Administración de acceso
- Mantenimiento de la integridad

Servicios básicos

Manejo de espacio

Servicios básicos

Manejo de espacio

- Los File System administran el espacio libre de forma atómica, usualmente en múltiples bloques físicos en un dispositivo. Es responsable de organizar los archivos y directorios, y hacer un seguimiento de las áreas utilizadas por cada archivo y las que están libres.

Servicios básicos

Manejo de espacio

- Los File System administran el espacio libre de forma atómica, usualmente en múltiples bloques físicos en un dispositivo. Es responsable de organizar los archivos y directorios, y hacer un seguimiento de las áreas utilizadas por cada archivo y las que están libres.
- Esto resulta en espacio desperdiciado en aquellas ocasiones que el tamaño real de un archivo no sea un múltiplo exacto de la unidad mínima de asignación, determinada al momento de crear el *file system*.

Servicios básicos

Manejo de espacio

- Los File System administran el espacio libre de forma atómica, usualmente en múltiples bloques físicos en un dispositivo. Es responsable de organizar los archivos y directorios, y hacer un seguimiento de las áreas utilizadas por cada archivo y las que están libres.
- Esto resulta en espacio desperdiciado en aquellas ocasiones que el tamaño real de un archivo no sea un múltiplo exacto de la unidad mínima de asignación, determinada al momento de crear el *file system*.
- La fragmentación del sistema de archivos ocurre cuando el espacio libre o los archivos individuales no son contiguos. A medida que el *file system* es usado, se generan de manera aleatoria espacios libres entre los usados.

Servicios básicos

Nombres de archivo y directorios

Servicios básicos

Nombres de archivo y directorios

- Un nombre de archivo es usado para identificar una ubicación del archivo en el sistema de archivos. La mayoría de los *file systems* tienen un límite en la longitud de su nombre. En el caso de Linux, el nombre es *case-sensitive*, es decir, discrimina las mayúsculas de las minúsculas.

Servicios básicos

Nombres de archivo y directorios

- Un nombre de archivo es usado para identificar una ubicación del archivo en el sistema de archivos. La mayoría de los *file systems* tienen un límite en la longitud de su nombre. En el caso de Linux, el nombre es *case-sensitive*, es decir, discrimina las mayúsculas de las minúsculas.
- Además de esto, la mayoría permiten ordenar los archivos en directorios (también llamados carpetas) que permiten al usuario agrupar esos archivos de manera cómoda. Esto puede ser implementado asociando al nombre con un índice en una tabla de contenidos o en el caso de Linux, un inodo.

Servicios básicos

Metadatos

Típicamente hay más información asociada a cada uno de los archivos. Por ejemplo el tamaño del archivo, que puede ser representado por la cantidad de bloques asignados al archivo, o un conteo de *bytes*; la fecha de creación y última modificación del archivo; el tipo de archivo (unidad de bloques, caracter, socket, subdirectorio, etc); el usuario y grupo dueños del archivo; sus permisos...

Servicios básicos

Administración de acceso

Hay varios mecanismos para controlar el acceso y modificación de los archivos en un *file system*. Los usos más comunes son la prevención del acceso y la modificación de los archivos por algún usuario o grupo, o el control de la modificación de los archivos, como por ejemplo, las cuotas de disco.

Servicios básicos

Mantenimiento de la integridad

Servicios básicos

Mantenimiento de la integridad

- Una responsabilidad importante del File System es asegurar que la estructura de los archivos y directorios sea consistente. Esto incluye tomar acciones en caso de que un programa que modificaba datos termine anormalmente, o no informe de sus actividades al sistema operativo. En ese caso, debe ocuparse de actualizar los metadatos, las entradas, y cerrar buffers.

Servicios básicos

Mantenimiento de la integridad

- Una responsabilidad importante del File System es asegurar que la estructura de los archivos y directorios sea consistente. Esto incluye tomar acciones en caso de que un programa que modificaba datos termine anormalmente, o no informe de sus actividades al sistema operativo. En ese caso, debe ocuparse de actualizar los metadatos, las entradas, y cerrar buffers.
- Otras fallas que debe considerar son en caso de falla física del medio de almacenamiento, pérdida de energía del sistema, o desconexión remota, en caso de un sistema de archivos en red.

Contenido

¿Que esperamos de los FS modernos?

Aparte de todas estas características, los *file system* modernos incorporan diseños y funciones más avanzadas, que permiten mejorar la integridad y la velocidad de acceso a archivos, entre ellas:

¿Que esperamos de los FS modernos?

Aparte de todas estas características, los *file system* modernos incorporan diseños y funciones más avanzadas, que permiten mejorar la integridad y la velocidad de acceso a archivos, entre ellas:

Journaling

¿Que esperamos de los FS modernos?

Aparte de todas estas características, los *file system* modernos incorporan diseños y funciones más avanzadas, que permiten mejorar la integridad y la velocidad de acceso a archivos, entre ellas:

Journaling

Log-structured filesystems

¿Que esperamos de los FS modernos?

Aparte de todas estas características, los *file system* modernos incorporan diseños y funciones más avanzadas, que permiten mejorar la integridad y la velocidad de acceso a archivos, entre ellas:

- Journaling
- Log-structured filesystems
- Copy-on-write filesystems

¿Que esperamos de los FS modernos?

Journaling

¿Que esperamos de los FS modernos?

Journaling

- Un *journaling file system* es un sistema de archivos que mantiene un registro de los cambios que aún no fueron guardados registrando los cambios intencionados en una estructura de datos conocida como *journal*, que comúnmente es una lista circular.

¿Que esperamos de los FS modernos?

Journaling

- Un *journaling file system* es un sistema de archivos que mantiene un registro de los cambios que aún no fueron guardados registrando los cambios intencionados en una estructura de datos conocida como *journal*, que comúnmente es una lista circular.
- En caso de una falla del sistema o del suministro de energía, estos sistemas de archivos pueden recuperarse de manera mucho más rápida, con menor probabilidad de corrupción de sus datos.

¿Que esperamos de los FS modernos?

Journaling

- Un *journaling file system* es un sistema de archivos que mantiene un registro de los cambios que aún no fueron guardados registrando los cambios intencionados en una estructura de datos conocida como *journal*, que comúnmente es una lista circular.
- En caso de una falla del sistema o del suministro de energía, estos sistemas de archivos pueden recuperarse de manera mucho más rápida, con menor probabilidad de corrupción de sus datos.
- Ya que es necesario reservar espacio para esta estructura estos FS sufren una reducción del espacio disponible al usuario además de una pérdida de rendimiento, por tener que escribir dos veces los cambios a realizar en el disco.

¿Que esperamos de los FS modernos?

Log-structured file systems

¿Que esperamos de los FS modernos?

Log-structured file systems

- En estos sistemas de archivos, la estructura del sistema de archivos es una lista circular de los cambios realizados.

¿Que esperamos de los FS modernos?

Log-structured file systems

- En estos sistemas de archivos, la estructura del sistema de archivos es una lista circular de los cambios realizados.
- En caso de una falla, el sistema de archivos debe únicamente reconstruir su estado desde el último punto consistente.

¿Que esperamos de los FS modernos?

Log-structured file systems

- En estos sistemas de archivos, la estructura del sistema de archivos es una lista circular de los cambios realizados.
- En caso de una falla, el sistema de archivos debe únicamente reconstruir su estado desde el último punto consistente.
- Por ejemplo: F2FS.

¿Que esperamos de los FS modernos?

Copy-on-write file systems

¿Que esperamos de los FS modernos?

Copy-on-write file systems

- Los sistemas de archivos *copy-on-write* escriben los cambios a los archivos en nuevos bloques libres, y actualizan los metadatos y sus estructuras al finalizar la escritura.

¿Que esperamos de los FS modernos?

Copy-on-write file systems

- Los sistemas de archivos *copy-on-write* escriben los cambios a los archivos en nuevos bloques libres, y actualizan los metadatos y sus estructuras al finalizar la escritura.
- De esta manera, se evitan las corrupciones de los archivos en caso de fallas, y no se introduce mayor tiempo de escritura.

¿Que esperamos de los FS modernos?

Copy-on-write file systems

- Los sistemas de archivos *copy-on-write* escriben los cambios a los archivos en nuevos bloques libres, y actualizan los metadatos y sus estructuras al finalizar la escritura.
- De esta manera, se evitan las corrupciones de los archivos en caso de fallas, y no se introduce mayor tiempo de escritura.
- Por ejemplo: Btrfs.

Contenido

Casos de ejemplo

A continuación veremos algunos casos de *filesystems* usados mayormente en Linux embebido.

Casos de ejemplo

A continuación veremos algunos casos de *filesystems* usados mayormente en Linux embebido.

ext2: altamente popular, muy simple de implementar.

Casos de ejemplo

A continuación veremos algunos casos de *filesystems* usados mayormente en Linux embebido.

ext2: altamente popular, muy simple de implementar.

YAFFS/YAFFS2: muy usado en memorias Flash.

Casos de ejemplo

A continuación veremos algunos casos de *filesystems* usados mayormente en Linux embebido.

ext2: altamente popular, muy simple de implementar.

YAFFS/YAFFS2: muy usado en memorias Flash.

F2FS: Desarrollado por Samsung para su uso en dispositivos móviles.

Casos de ejemplo

ext2

Casos de ejemplo

ext2

- Es un sistema de archivos diseñado para el kernel Linux. No soporta journaling, por lo que es muy usado en sistemas con memorias Flash debido a que minimiza la cantidad de ciclos de escritura, los cuales pueden ser escasos en dispositivos de este tipo.

Casos de ejemplo

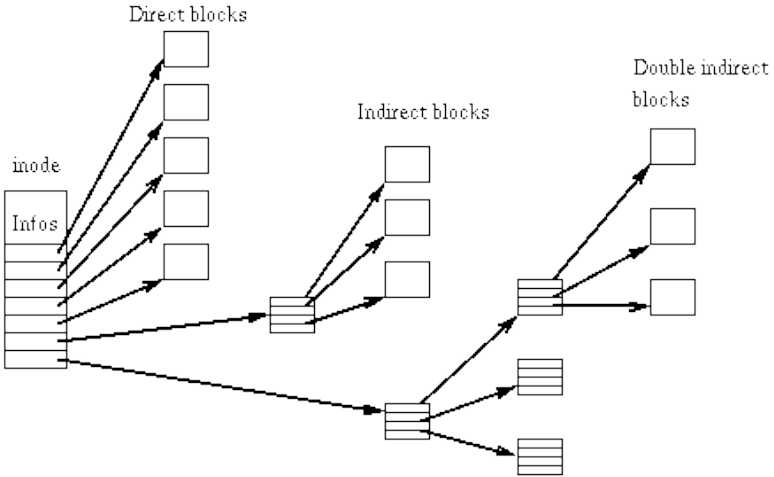
ext2

- Es un sistema de archivos diseñado para el kernel Linux. No soporta journaling, por lo que es muy usado en sistemas con memorias Flash debido a que minimiza la cantidad de ciclos de escritura, los cuales pueden ser escasos en dispositivos de este tipo.
- El espacio en el medio es dividido en bloques, el primero se conoce como *superblock*, que contiene información crucial para el funcionamiento del sistema operativo.

Casos de ejemplo

ext2

- Es un sistema de archivos diseñado para el kernel Linux. No soporta journaling, por lo que es muy usado en sistemas con memorias Flash debido a que minimiza la cantidad de ciclos de escritura, los cuales pueden ser escasos en dispositivos de este tipo.
- El espacio en el medio es dividido en bloques, el primero se conoce como *superblock*, que contiene información crucial para el funcionamiento del sistema operativo.
- Organiza sus contenidos en tablas de asignación, e intenta ubicar archivos en el mismo grupo que los de su directorio, para minimizar tiempos de acceso.



Ejemplo de estructura de inodos en ext2

Casos de ejemplo

YAFFS

Casos de ejemplo

YAFFS

- YAFFS (Yet Another Flash File System) fue diseñado en 2002 para dispositivos NAND. Está pensado para un bajo tiempo de lectura, tasa de uso estática, gestión de apagado no programado, y corrección de errores.

Casos de ejemplo

YAFFS

- YAFFS (Yet Another Flash File System) fue diseñado en 2002 para dispositivos NAND. Está pensado para un bajo tiempo de lectura, tasa de uso estática, gestión de apagado no programado, y corrección de errores.
- Utiliza una estructura *log* que prioriza la integridad de los datos del sistema de archivos, dejando la *performance* como una segunda prioridad.

Casos de ejemplo

YAFFS

- YAFFS (Yet Another Flash File System) fue diseñado en 2002 para dispositivos NAND. Está pensado para un bajo tiempo de lectura, tasa de uso estática, gestión de apagado no programado, y corrección de errores.
- Utiliza una estructura *log* que prioriza la integridad de los datos del sistema de archivos, dejando la *performance* como una segunda prioridad.
- Está diseñado para ser altamente portable, y es soportado por Linux, Windows CE, pSOS, eCos, ThreadX y varios más.

Casos de ejemplo

F2FS

Casos de ejemplo

F2FS

- F2FS (Flash Friendly File System) fue desarrollado por Samsung para su uso en Linux. Orientado desde el principio para su uso en dispositivos NAND, ordena sus contenidos en estructuras simil *log*, adaptadas a las nuevas formas de almacenamiento.

Casos de ejemplo

F2FS

- F2FS (Flash Friendly File System) fue desarrollado por Samsung para su uso en Linux. Orientado desde el principio para su uso en dispositivos NAND, ordena sus contenidos en estructuras simil *log*, adaptadas a las nuevas formas de almacenamiento.
- Además de esto, tiene en cuenta las particularidades internas de las memorias flash, como su manejo de memoria (a través de la FTL, Flash Translation Layer) y soporta de operaciones TRIM. También permite defragmentación online y encriptación.

Casos de ejemplo

Funcionamiento de F2FS

Casos de ejemplo

Funcionamiento de F2FS

- F2FS divide el volumen lógico en un conjunto de segmentos, cada uno de 2MB. Una sección se compone de varios segmentos, y una zona de varias secciones.

Casos de ejemplo

Funcionamiento de F2FS

- F2FS divide el volumen lógico en un conjunto de segmentos, cada uno de 2MB. Una sección se compone de varios segmentos, y una zona de varias secciones.
- F2FS separa todo el volumen en seis áreas, y todas ellas exceptuando el área de superbloque consiste de múltiples segmentos.

Casos de ejemplo

Segmentos de F2FS

Casos de ejemplo

Segmentos de F2FS

Superblock : ubicado al comienzo de la partición, contiene información básica sobre el FS. Tiene dos copias.

Casos de ejemplo

Segmentos de F2FS

Superblock : ubicado al comienzo de la partición, contiene información básica sobre el FS. Tiene dos copias.

Checkpoint: contiene información sobre el FS, mapas de conjuntos SIT/NAT, listas de inodos huérfanos, y un resumen de los segmentos activos.

Casos de ejemplo

Segmentos de F2FS

Superblock : ubicado al comienzo de la partición, contiene información básica sobre el FS. Tiene dos copias.

Checkpoint: contiene información sobre el FS, mapas de conjuntos SIT/NAT, listas de inodos huérfanos, y un resumen de los segmentos activos.

Segment Information Table: contiene la cantidad de bloques activos y el mapa de validez de los bloques de la Main Área.

Casos de ejemplo

Segmentos de F2FS

Casos de ejemplo

Segmentos de F2FS

Node Address Table: tabla de direcciones de los bloques de la Main Área.

Casos de ejemplo

Segmentos de F2FS

Node Address Table: tabla de direcciones de los bloques de la Main Área.

Segment Summary Área: contiene información sobre el dueño de los bloques de datos y nodos.

Casos de ejemplo

Segmentos de F2FS

Node Address Table: tabla de direcciones de los bloques de la Main Área.

Segment Summary Área: contiene información sobre el dueño de los bloques de datos y nodos.

Main Área: contiene los datos de los archivos y directorios con sus respectivos índices.

Casos de ejemplo

Integridad de datos en F2FS

Casos de ejemplo

Integridad de datos en F2FS

- F2FS utiliza un esquema de *checkpoints* para mantener la integridad del sistema de archivos.

Casos de ejemplo

Integridad de datos en F2FS

- F2FS utiliza un esquema de *checkpoints* para mantener la integridad del sistema de archivos.
- Al momento del montaje del FS, se comprueba el área Checkpoint en búsqueda del último válido. Para reducir el tiempo de búsqueda F2FS usa únicamente dos copias del CP, de las cuales una siempre indica el último dato válido.

Casos de ejemplo

Integridad de datos en F2FS

- F2FS utiliza un esquema de *checkpoints* para mantener la integridad del sistema de archivos.
- Al momento del montaje del FS, se comprueba el área Checkpoint en búsqueda del último válido. Para reducir el tiempo de búsqueda F2FS usa únicamente dos copias del CP, de las cuales una siempre indica el último dato válido.
- Para la integridad de los datos en el FS, cada CP apunta a las copias del NAT y SIT válidas.

¿Preguntas?