

2020

# Sistemas Operativos

## Trabajo Práctico N°1: INODOS

Grupo: 13

Integrantes: Manuel Onetto,  
Lucas Maidana, Martin Rotelli,  
Nuria Delaude, Franco Molina

2-9-2020

# Índice

<b>1.DEFINICIÓN</b>	<b>2</b>
<b>a) Estructura</b>	<b>2</b>
<b>b) Tabla de Inodos</b>	<b>4</b>
<b>2. USO Y FUNCIONAMIENTO</b>	<b>5</b>
<b>a) Tablas de direccionamiento directo en sistemas de archivo ext4</b>	<b>7</b>
<b>b) Tablas de direccionamiento indirectas simples</b>	<b>8</b>
<b>c) Tablas de direccionamiento indirectas dobles y triples</b>	<b>8</b>
<b>3. BENEFICIOS</b>	<b>9</b>
<b>a) Localización</b>	<b>10</b>
<b>b) Gestión del espacio libre</b>	<b>10</b>
<b>c) Fragmentación</b>	<b>11</b>
<b>d) Directorios y enlaces duros</b>	<b>11</b>
<b>4. SISTEMA DE ARCHIVOS.</b>	<b>12</b>
<b>a) Definición de los sistemas de archivos que implementan inodos</b>	<b>12</b>
<b>b) Ejemplos de los sistemas de archivos que implementan inodos</b>	<b>13</b>
<b>5. COMPARACIÓN</b>	<b>15</b>
<b>a) Implementación de inodos en UNIX y LINUX</b>	<b>15</b>
<b>b) Implementación en Windows</b>	<b>19</b>
<b>BIBLIOGRAFÍA</b>	<b>20</b>

## 1. Definición

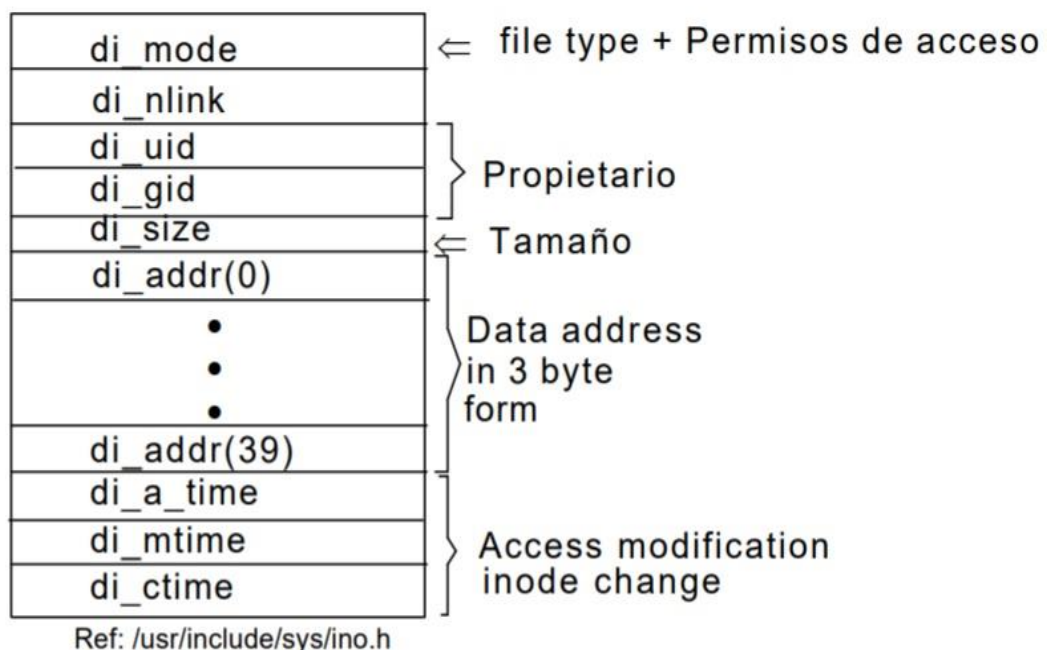
Todos los tipos de archivos de UNIX son administrados por el sistema operativo mediante los llamados INODOS, que son estructuras de control que usa el sistema operativo para describir un archivo en particular. Contiene los metadatos de un archivo regular, directorio, o cualquier otro objeto que pueda contener el sistema de archivos, con excepción de su nombre.

Muchos archivos pueden ser asociados con un i-nodo en particular, pero no sucede al revés, un i-nodo activo es asociado exactamente con un solo archivo, y cada archivo es controlado por un i-nodo.

Existen dos tipos de i-nodos, Disk o In-core, estos últimos son usados para representar al archivo en el kernel una vez abierto. Un i-nodo está residente en disco, o residente en la tabla de i-nodos del S.O. o en ambos.

### a) Estructura

La estructura de un inodo residente en disco está definida en `/usr/include/sys/ino.h`. Cada campo de la estructura de inodo residente en el disco comienza con los caracteres di- (disk inode).



1. El device ID en el que está almacenado el fichero o archivo (identificador del dispositivo).
2. El número de inodo que identifica al archivo dentro del sistema de archivos.
3. El tamaño del archivo en bytes.
4. El ID de usuario del creador o un propietario del archivo con derechos diferenciados
5. Banderas adicionales del sistema y del usuario para proteger el archivo (limitan su uso y modificación).
6. El ID de grupo de un grupo de usuarios con derechos diferenciados

7. El modo de acceso: donde se determina la capacidad de leer, escribir, y ejecutar el archivo por parte del propietario, del grupo y de otros usuarios.
8. Las marcas de tiempo con las fechas de última modificación (mtime), acceso (atime) y de la creación del propio inodo (ctime). Los tres campos contienen un valor entero que refleja el número de segundos desde las 00:00:00 hs del 01/01/1970. Las marcas de tiempo residen solamente en los inodos residentes en el disco.
9. El contador de hardlinks/enlaces, esto es, el número de nombres (entradas de directorio) asociados con este inodo. El número de enlaces se emplea por el sistema operativo para eliminar el archivo del sistema de ficheros, tanto el inodo como el contenido, cuando se han borrado todos los enlaces y el contador queda a cero.
10. La estructura de punteros (tabla de direccionamiento) detalla los bloques de disco en que está almacenado el archivo. Y a su vez está compuesta por:
  - a) Doce punteros que apuntan directamente a bloques de datos del archivo (punteros directos)
  - b) Un puntero de indirección simple (apunta a un bloque de punteros, los cuales apuntan a bloques de datos del archivo)
  - c) Un puntero de indirección doble (apunta a un bloque de punteros, los cuales apuntan a otros bloques de punteros, estos últimos apuntan a bloques de datos del archivo).
  - d) Un puntero de indirección triple (apunta a un bloque de punteros que apuntan a otros bloques de punteros que apuntan a otros bloques de punteros que luego apuntan a bloques de datos del archivo).

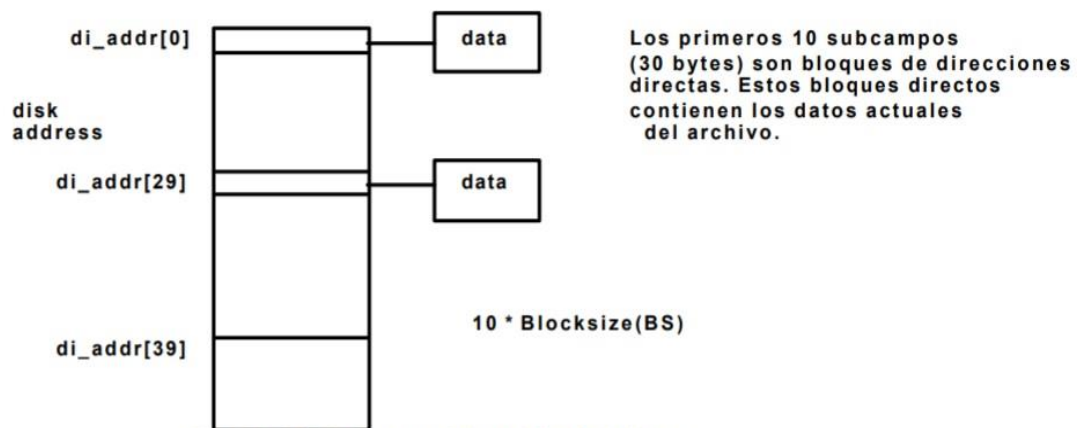
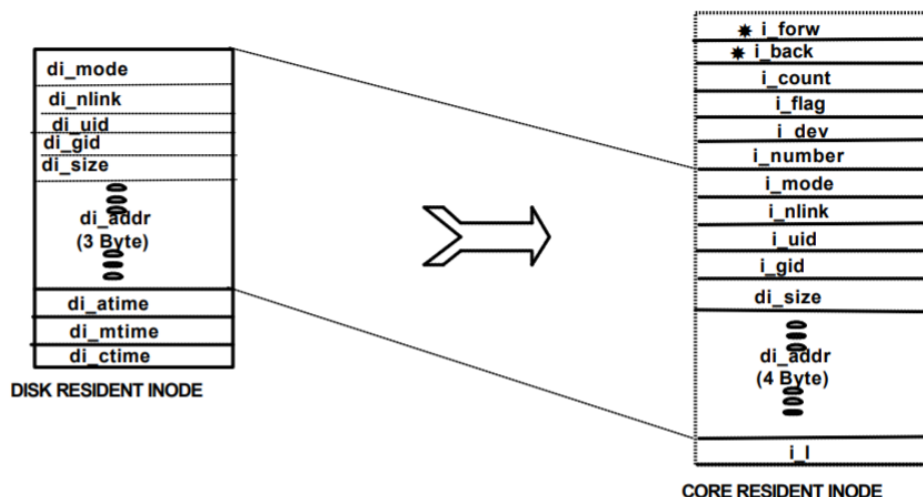


Fig.7.B.8 Subcampos: Direct Block Address.

#### COMPARACION DE INODOS RESIDENTES EN MEMORIA Y EN DISCO



## b) Tabla de Inodos

La tabla de inodos contiene una copia de los inodos activos residentes en el disco (campo `i_list`) de todos los File System montados, para una rápida referencia o modificación. Además, provee un punto de control simple para los diferentes procesos. La Tabla de inodos es accesible solamente vía System Call. Es un arreglo de la estructura de inodos que reside en la Memoria Central en el espacio de direcciones del kernel. El nombre del arreglo es `inode [ ]`. Con respecto al tamaño, el número de entradas en la Tabla de inodos (usualmente entre 100 y 300) es definido por el administrador del Sistema en el momento de la generación del sistema. Este número corresponde al total de inodos que requieren acceso simultáneo.

Las acciones que requiere una entrada en la Tabla son:

### Semi-permanente:

- 1) I/O (open, read, write, close ).
- 2) Creación de Archivos.
- 3) Cambio del Directorio Actual.
- 4) Montaje del File System.
- 5) Ejecución del Archivo.

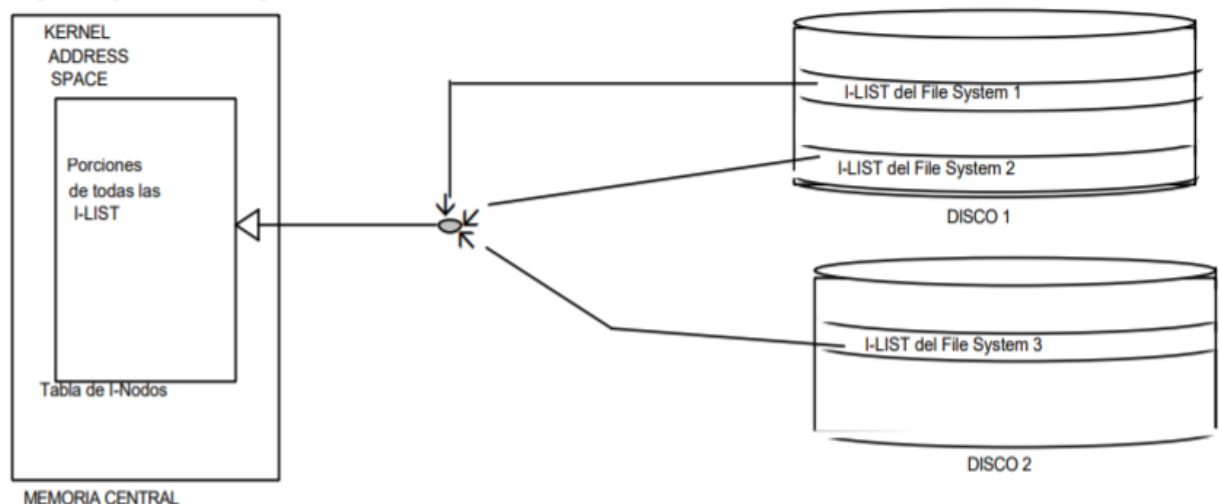
### Temporarios:

- 1) Obtener el estado del Archivo.
- 2) Cambios del estado del Archivo.
- 3) Vinculación/Desvinculación de Archivos.

### Permanente:

- 1) El root inode es la única entrada permanente en la Tabla.

## RELACIÓN CON LA I-LIST:



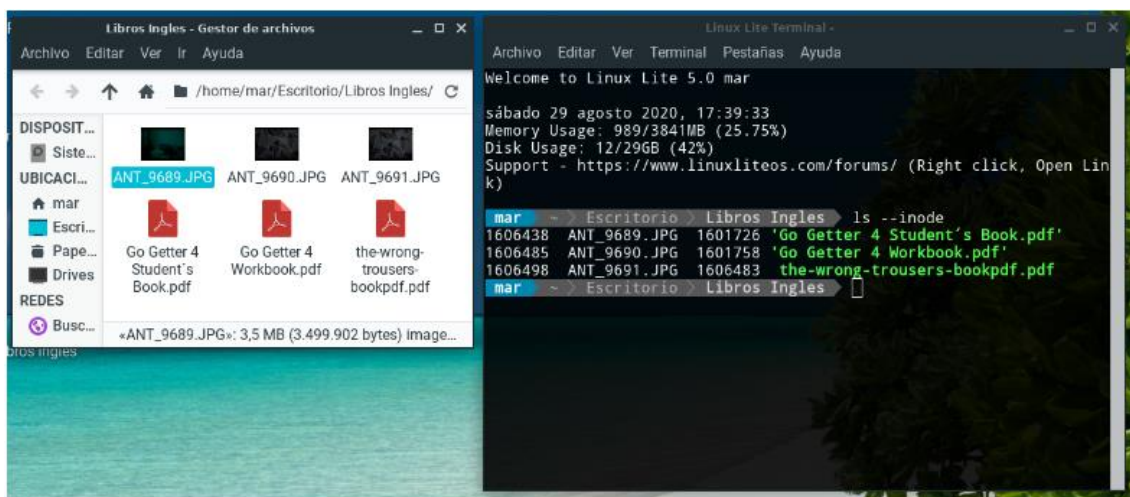
## 2. Uso y funcionamiento

La forma en que el sistema operativo sabe dónde está un archivo y qué archivo es, no es a través de su nombre, es a través de su Inode Number. Para eso, el sistema operativo almacena tablas a las que llama las Inode Tables, en las que se almacenan los Inode Numbers y el archivo al que corresponde ese número.

De esta manera, cuando tu como usuario haces referencia a un archivo por su nombre, el sistema lo buscará en las Inode Tables para encontrar su Inode Number, y ahora sí, con este último dato identificará al archivo que necesitas.

Cada vez que creas un nuevo archivo a este se le asigna un nombre, y, además, un Inode Number, para que como mencionamos, el sistema operativo pueda identificar tu archivo.

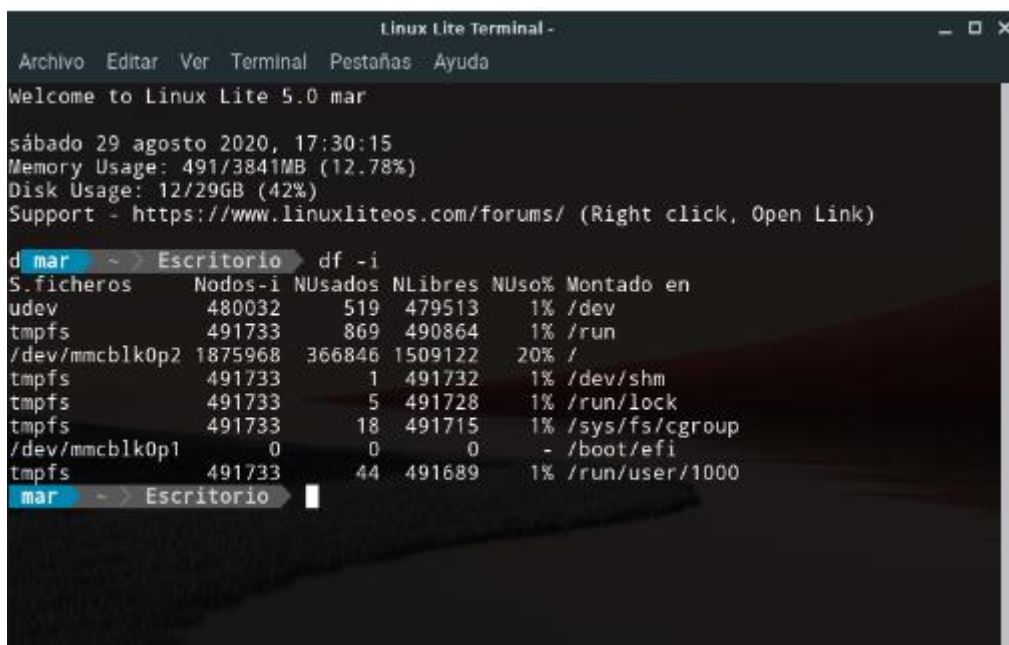
Si quisieras saber el Inode Number de tus archivos, puedes ejecutar el siguiente comando en tu terminal: **ls --inode**



La ejecución de este comando enlistará los archivos contenidos dentro del directorio, junto con el Inode Number de cada uno de ellos. Un aspecto muy importante de conocer acerca de este concepto es que un sistema operativo tiene un número limitado de identificadores (Inode numbers), mismos que fueron asignados al momento de la instalación del sistema operativo.

Esto significa que nuestra computadora tiene un límite de archivos que puede crear, además, que este límite no depende sólo del espacio en disco duro, también depende de si aún hay identificadores disponibles.

Para conocer la cantidad de números disponibles, cuántos has usado y el porcentaje que se ha ocupado, puedes usar el siguiente comando: **df -i**



```
Linux Lite Terminal -
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
Welcome to Linux Lite 5.0 mar
sábado 29 agosto 2020, 17:30:15
Memory Usage: 491/3841MB (12.78%)
Disk Usage: 12/29GB (42%)
Support - https://www.linuxliteos.com/forums/ (Right click, Open Link)

d mar ~ > Escritorio df -i
S.ficheros  Nodos-i  NUsados  NLibres  NUsado%  Montado en
udev       480032   519      479513   1%       /dev
tmpfs      491733   869      490864   1%       /run
/dev/mmcblk0p2 1875968 366846   1509122  20%      /
tmpfs      491733   1        491732   1%       /dev/shm
tmpfs      491733   5        491728   1%       /run/lock
tmpfs      491733   18       491715   1%       /sys/fs/cgroup
/dev/mmcblk0p1 0        0        0        -        /boot/efi
tmpfs      491733   44       491689   1%       /run/user/1000

mar ~ > Escritorio
```

El comando **df** entrega información acerca del espacio en disco usado por el sistema de archivos, o en palabras más mortales, el espacio en el disco duro. La opción **-i** le indica que en lugar de mostrar la información del espacio usado, muestre información relacionada con los inodes.

Esto quiere decir que nuestra computadora se puede quedar sin espacio de dos formas:

- Ocupamos toda la capacidad del disco duro
- Nos acabamos los inodes disponibles para nuestro sistema operativo.

Si en algún punto llegarás a acabar la cantidad de Inodes disponibles para tu computadora, tu solución sería eliminar archivos. Recuerda que como cada archivo ocupa un Inode Number distinto, liberar tu capacidad en este caso significa borrar muchos archivos no borrar archivos grandes, de hecho, el peso de cada archivo no importa cuando quieres liberar los inodes disponibles para tu computadora.

En el caso de Linux, se crean por defecto, un inode por cada 2k bytes que contenga el sistema de archivos, esta regla puede ser configurada por el usuario, para reducir la cantidad de inodes disponibles o eventualmente incrementarla. La función de la tabla de direccionamiento de un inodo es indicar la posición del disco duro en que está almacenado un fichero.

Los inodos tienen un tamaño fijo. Por lo tanto, el tamaño de un inodo es finito y en muchas ocasiones no es suficiente para almacenar la totalidad de datos que necesitamos para acceder a un archivo. Frente a esta situación se usan tablas de direccionamiento indirecto simples, dobles y triples.

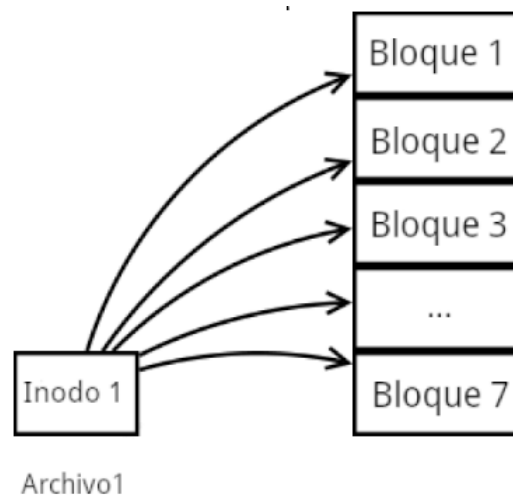
Los punteros son direcciones de bloques de disco: cada puntero contiene la información necesaria para identificar un bloque en el disco. Dado que cada bloque de disco tiene al menos 512 bytes (a veces 4096 u 8192 bytes), al usar direcciones de 32 bits, el disco puede direccionar hasta  $512 * 4 * 10243 = 2 \text{ TiB}$  (Tebibytes, más comúnmente llamado Terabytes) asumiendo

bloques de 1/2 KiB; tamaños correspondientemente más grandes a medida que aumenta el tamaño del bloque (por lo tanto, 32 TiB a un tamaño de bloque de 8 KiB). Para un esquema de direccionamiento para discos más grandes, tendría que pasar a tamaños de bloque más grandes o direcciones de disco más grandes. Por lo tanto, las direcciones de 48 o 64 bits podrían ser posibles.

#### a) Tablas de direccionamiento directo en sistemas de archivo ext4

Un inodo tiene una tabla de direccionamiento de 15 entradas. 12 de las 15 entradas permiten un direccionamiento directo a un bloque de datos del disco duro. Si un archivo se almacena en 7 bloques, un inodo nos puede direccionar de forma directa al contenido del archivo.

A modo de ejemplo, el archivo archivo1 está vinculado al inodo 1. Si consultamos el inodo 1 vemos que el contenido del archivo1 está almacenado en los bloques 1, 2, 3, 4, 5, 6 y 7. Por lo tanto, mediante un direccionamiento directo podemos acceder al contenido del archivo.

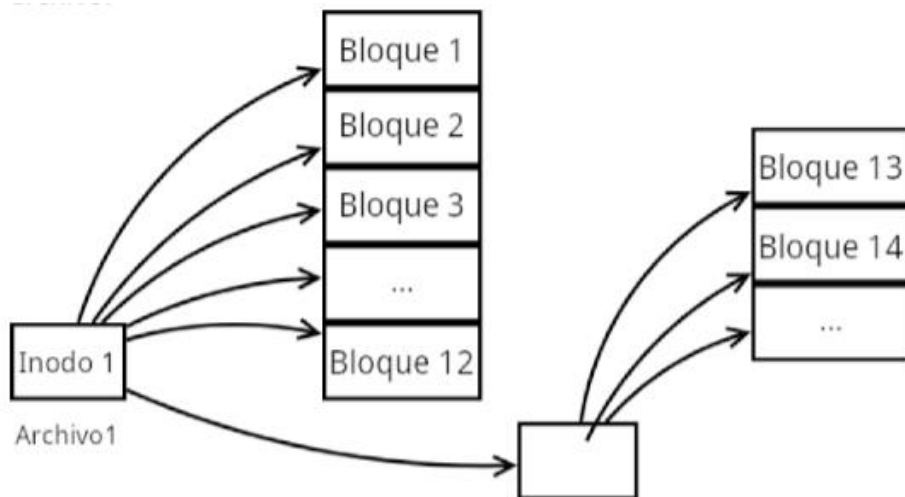


Cuando el archivo crece, el controlador de disco asigna un único bloque indirecto y lo registra en el inodo. Cuando el controlador necesita obtener un bloque, lee el bloque indirecto en la memoria y luego encuentra la dirección del bloque que necesita del bloque indirecto. Por lo tanto, requiere (nominalmente) dos lecturas para llegar a los datos, aunque, por supuesto, la indirecta tiende a almacenarse en la memoria caché.



## b) Tablas de direccionamiento indirectas simples

Si finalizamos el espacio que tenemos para las 12 entradas directas tendremos que usar un direccionamiento indirecto. La treceava posición de la tabla de direccionamiento del inodo nos dirigirá a un bloque de datos que contendrá una nueva tabla de direcciones hacia los bloques que almacenan el contenido de nuestro archivo.



Si un archivo crece aún más, el controlador asigna un bloque indirecto doble. Cada puntero del bloque indirecto doble apunta a un solo bloque indirecto. Por lo tanto, puede tener 2048 bloques indirectos más, cada uno de los cuales puede apuntar efectivamente a 16 MiB, lo que lleva a que se puedan almacenar archivos de hasta 32 GiB (aproximadamente).

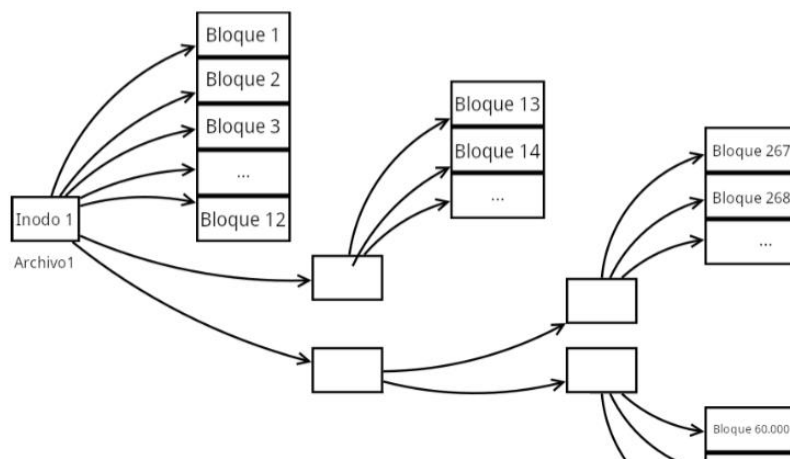
Si un archivo crece aún más, el controlador asigna un bloque indirecto triple. Cada uno de los 2048 punteros en un bloque indirecto triple apunta a un bloque doble. Entonces, bajo el esquema de direccionamiento de 32 bits con direcciones de 32 bits, se podrían direccionar archivos de hasta aproximadamente 64 TiB. Excepto que se ha quedado sin direcciones de disco antes de eso (32 TiB máximo debido a las direcciones de 32 bits a bloques de 8 KiB).

Por lo tanto, la estructura de inodo puede manejar archivos más grandes que los que pueden manejar las direcciones de disco de 32 bits.

## c) Tablas de direccionamiento indirectas dobles y triples

Del mismo modo que se hacen direccionamientos indirectos simples, también podemos hacer direccionamientos indirectos dobles y triples con las entradas 14 y 15.

A continuación, se muestra la estructura de un direccionamiento indirecto doble.



De este modo podemos llegar a conseguir archivos con un tamaño de hasta 16TB en sistemas de ficheros ext4.

Frente a este funcionamiento podemos llegar a las siguientes conclusiones:

- a) El acceso a los archivos será más lento cuando se hace un direccionamiento indirecto.
- b) Podremos acceder de forma mucho más rápida y eficiente en archivos de poco tamaño. Por este motivo los sistemas de archivos basados en inodos son ideales para manejar servidores web o servidores de email.

### 3. Beneficios

Los sistemas basados en inodos son conceptualmente algo más complejos que los basados en FAT. Comparten además limitaciones comunes, usan más espacio de disco, aunque suelen ser más rápidos.

Actualmente existen sistemas basados en inodos mucho más avanzados y complejos que UFS y ext2/3/4 y que mejoran este sistema de forma sustancial. Por ejemplo, en XFS los inodos tienen una estructura totalmente distinta donde se indica un bloque de inicio y una longitud en número de bloques contiguos que pertenecen al archivo. Muchos sistemas además usan estructuras de árbol B+ como pueden ser ZFS o Btrfs.

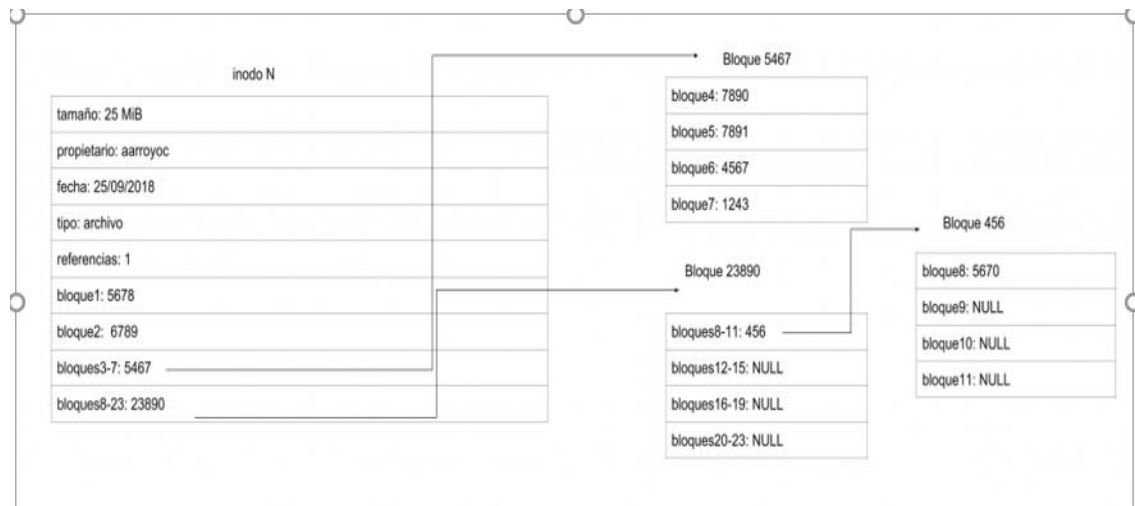
Tomando la base de FAT, una partición de un sistema basado en inodos también contiene un sector de arranque y un superbloque con metadatos. También es necesario un bloque dedicado al directorio raíz presente en el disco. Además, es necesario espacio para almacenar todos los inodos y un mapa de bits de espacio libre que en FAT no hacía falta, ya que la propia tabla nos indicaba que bloques del disco estaban libres.

Este sistema tiene una ventaja de rendimiento respecto a FAT en cuanto al acceso aleatorio a los archivos, ya que es mucho más rápido de esta forma que con FAT. En FAT para hacer lo mismo tenemos que ir recorriendo la tabla arriba y abajo siguiendo los números de bloque hasta encontrar el bloque deseado.

Normalmente un inodo tiene un tamaño fijo, lo que implica que el índice no se puede alargar hasta el infinito. Esto hace que haya un número limitado de bloques que puede usar un archivo y, por ende, que haya un tamaño máximo de archivo que no es muy elevado. Para

solventar este problema hay varias soluciones. El enfoque de UFS y de ext2/3/4 consiste en múltiples niveles de indexado progresivo.

Esto significa que los primeros bloques son números de bloque directos pero los siguientes son números de bloque que llevan a tablas de inodo secundarias que ya sí, hacen referencia al archivo real. Más adelante los números de bloque hacen referencias a tablas de inodo secundarias que a su vez llaman a tablas de inodos terciarias.



Esto provoca algo que en principio puede parecer paradójico y es que es más lento leer una zona final de un archivo que una zona del principio, aunque en una lectura secuencial no se nota demasiado.

Otra solución a esto es enlazar los inodos a modo de lista enlazada añadiendo al final de cada inodo un número de inodo al que continuar la lectura de índices.

#### a) Localización

Los inodos se crean cuando se formatea el disco y permanecen en un estado libre. No se pueden añadir más inodos ni quitar inodos y no puede haber más archivos y directorios que inodos por esta misma razón. Esto es una desventaja respecto a FAT, ya que en FAT puede haber tantos archivos como bloques haya/entradas tenga la tabla. En sistemas de inodos como ext2/3/4 puede ocurrir que no queden inodos libres pero haya todavía bloques libres, dejando todo ese espacio inutilizado (aunque en realidad lo podrían usar los archivos existentes si crecieran).

Los inodos se pueden ubicar de dos formas distintas. Un enfoque consiste en ponerlos al principio del disco todos juntos. Otro enfoque, el que sigue ext3, consiste en dividir el disco en 4 zonas y ubicar inodos en cada inicio de zona. La idea es que los inodos de esa zona usen bloques de esa zona y de esta forma reducir los desplazamientos de las cabezas lectoras del disco (en unidades SSD esto da completamente igual como podréis suponer).

#### b) Gestión del espacio libre

Una de las grandes ventajas de FAT era que la tabla podía mantener a la vez un listado de bloques libres, listos para ser usados. Con inodos no tenemos esa facilidad y tenemos que recurrir a otros tipos de estructura. Aquí hay muchos planteamientos siendo el más común el mapa de bits. El mapa de bits es una estructura que se compone de un listado de bits. Cada bit

corresponde a un bloque y dependiendo de si el bit es 1 o 0 podemos saber si el bloque está libre u ocupado.



Como se ve, los sistemas basados en inodos tienen que mantener varias estructuras de datos de forma paralela. Esto aumenta las probabilidades de inconsistencias en el sistema. Por este motivo, muchos sistemas más avanzados como ext4 mantienen un journal o diario.

### c) Fragmentación

Veamos cómo se comportan los sistemas de inodos ante los tres tipos de fragmentación. Respecto a la fragmentación interna, que es aquella que sucede cuando asignamos espacio de más a archivos que no necesitan tanto, tiene los mismos valores que FAT, ya que los bloques siguen pudiendo pertenecer únicamente a un archivo. Realmente, para mejorar la fragmentación interna tenemos que saltar a sistemas como XFS o JFS.

La fragmentación externa de los sistemas basados en inodos es la misma que la de FAT, 0, ya que todo se asigna mediante bloques del mismo tamaño.

Respecto a la fragmentación de datos, históricamente las implementaciones como UFS o ext2/3 se han comportado relativamente bien, aunque a nivel teórico nada impide que se comporten igual de mal que FAT, no existen mecanismos preventivos. Ext4 por contra, sí que tiene mecanismos de prevención (delayed allocation).

### d) Directorios y enlaces duros

En los sistemas basados en inodos los directorios son más sencillos que en FAT, ya que no tienen que almacenar los metadatos del archivo en cuestión. En un fichero de directorio simplemente se almacena el nombre y el número de inodo correspondiente.

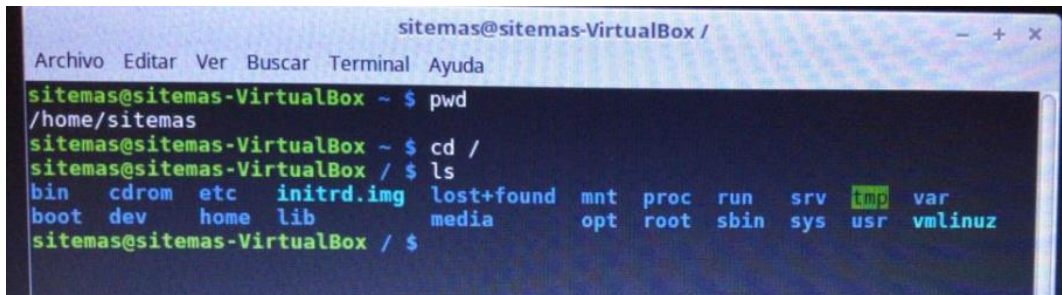
Un enlace duro no es más que otra entrada de directorio que apunta a un mismo inodo. Realmente desde el punto de vista del sistema de archivos no existe un fichero original y su enlace. Simplemente dos entradas diferentes apuntando al mismo inodo. Los inodos no se liberan hasta que no quede ninguna entrada de directorio apuntando a ellos. Para eso sirve el campo referencias dentro del inodo, para llevar la cuenta de cuántas veces se hace referencia al inodo en el sistema de archivos. Cuando el valor de referencias llega a cero, se puede liberar sin problema.

## 4. Sistema de archivos.

### a) Definición de los sistemas de archivos que implementan inodos

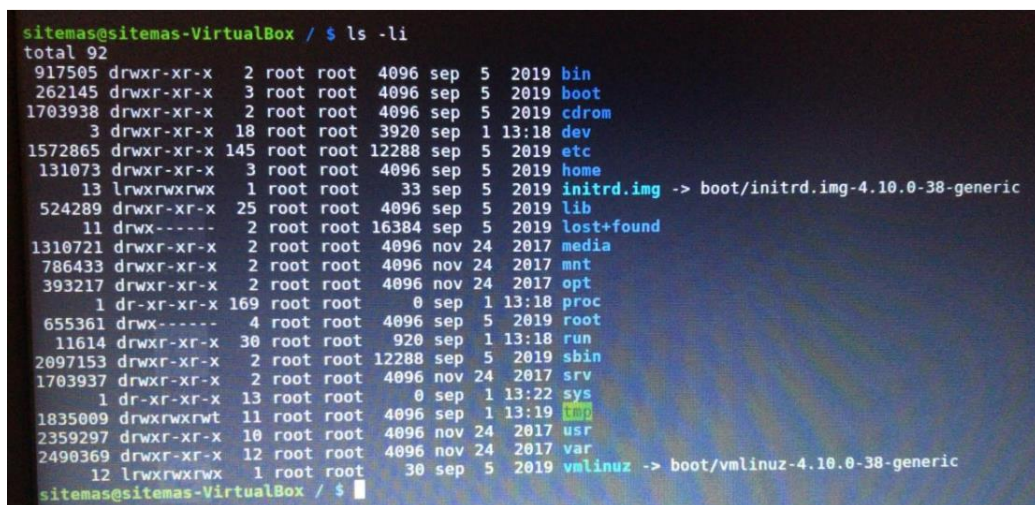
Los sistemas de archivos basados en el disco (diskbased) están formados por una colección de archivos y directorios que tienen las siguientes características:

1. Contienen un directorio root (/) que contiene a los archivos y directorios.



```
sistemas@sistemas-VirtualBox /
Archivo Editar Ver Buscar Terminal Ayuda
sistemas@sistemas-VirtualBox ~ $ pwd
/home/sistemas
sistemas@sistemas-VirtualBox ~ $ cd /
sistemas@sistemas-VirtualBox / $ ls
bin  cdrom  etc  initrd.img  lost+found  mnt  proc  run  srv  tmp  var
boot dev  home lib      media      opt  root  sbin  sys  usr  vmlinuz
sistemas@sistemas-VirtualBox / $
```

2. Cada archivo o directorio tiene un nombre, directorio donde reside, y un INODO como identificador único. Esto podemos verlo con el comando: `ls -li`

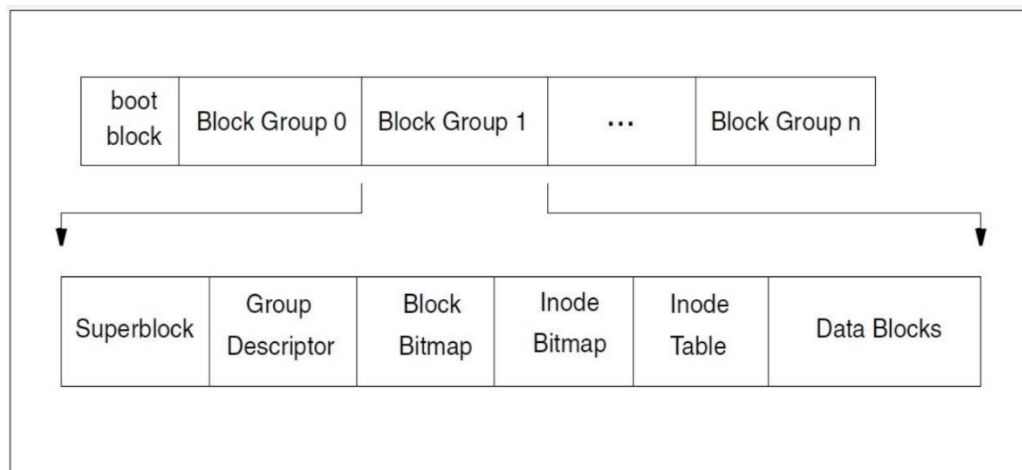


```
sistemas@sistemas-VirtualBox / $ ls -li
total 92
917505 drwxr-xr-x  2 root root  4096 sep  5  2019 bin
262145 drwxr-xr-x  3 root root  4096 sep  5  2019 boot
1703938 drwxr-xr-x  2 root root  4096 sep  5  2019 cdrom
      3 drwxr-xr-x 18 root root 3920 sep  1 13:18 dev
1572865 drwxr-xr-x 145 root root 12288 sep  5  2019 etc
131073 drwxr-xr-x  3 root root  4096 sep  5  2019 home
      13 lrwxrwxrwx  1 root root    33 sep  5  2019 initrd.img -> boot/initrd.img-4.10.0-38-generic
524289 drwxr-xr-x 25 root root  4096 sep  5  2019 lib
      11 drwx-----  2 root root 16384 sep  5  2019 lost+found
1310721 drwxr-xr-x  2 root root  4096 nov 24  2017 media
786433 drwxr-xr-x  2 root root  4096 nov 24  2017 mnt
393217 drwxr-xr-x  2 root root  4096 nov 24  2017 opt
      1 dr-xr-xr-x 169 root root    0 sep  1 13:18 proc
655361 drwx-----  4 root root  4096 sep  5  2019 root
      11614 drwxr-xr-x 30 root root  920 sep  1 13:18 run
2097153 drwxr-xr-x  2 root root 12288 sep  5  2019 sbin
1703937 drwxr-xr-x  2 root root  4096 nov 24  2017 srv
      1 dr-xr-xr-x 13 root root    0 sep  1 13:22 sys
1835009 drwxrwxrwt 11 root root  4096 sep  1 13:19 tmp
2359297 drwxr-xr-x 10 root root  4096 nov 24  2017 usr
2490369 drwxr-xr-x 12 root root  4096 nov 24  2017 var
      12 lrwxrwxrwx  1 root root    30 sep  5  2019 vmlinuz -> boot/vmlinuz-4.10.0-38-generic
sistemas@sistemas-VirtualBox / $
```

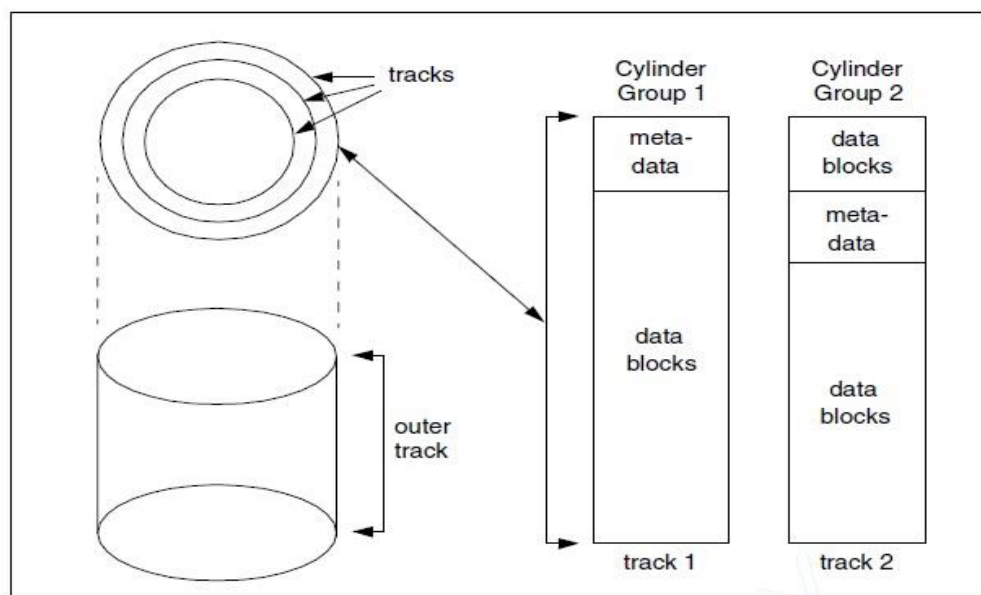
3. root: INODE number = 2, LOST+FOUND : iNODE NUMBER =3
4. Inode 0 es utilizado como un valor NULL, para indicar que no hay inodo. Inode 1 es utilizado para guardar los “bad blocks” del disco.
5. self-contained: No hay dependencias entre los sistemas de archivos (file systems).

## b) Ejemplos de los sistemas de archivos que implementan inodos

1. ext2: el sistema de archivos está dividido en un número determinado de grupos de bloques de tamaño fijo. Cada grupo de bloques maneja un set fijo de inodos, bloques de datos y contiene una copia del superbloque (registro de las características de un sistema de archivos)

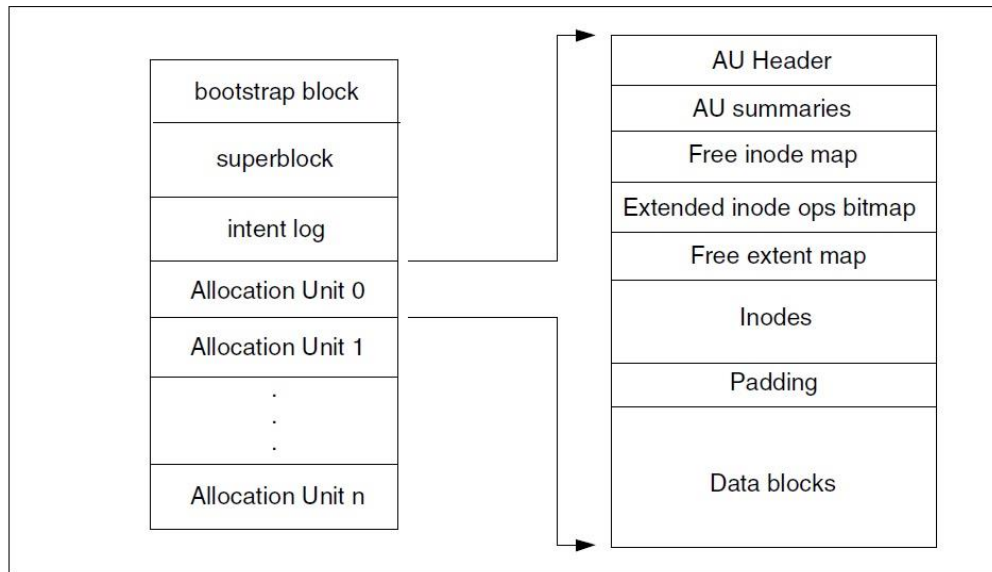


2. UFS. En la actualidad, los sistemas de archivos están divididos en bloques de cilindros. Cada bloque de cilindros contenía una copia del superbloque, un número fijo de inodos (El número de inodos por grupo de cilindros se calculó de manera que hubiera un inodo creado para cada 2048 bytes de datos), mapas de bits que describían inodos libres y bloques de datos, una tabla de resumen que describe el uso del bloque de datos y bloques de datos en sí mismos.



**Figure 9.5** Mapping the UFS filesystem to underlying disk geometries.

3. VxFS: El disco se divide en tres componentes principales: El **superbloque** contiene información fundamental sobre el tamaño, un resumen de los recursos disponibles y referencias a otros lugares del disco donde se puede encontrar información estructural adicional. Unidades de asignación. La **unidad de asignación (AU)** es aproximadamente equivalente a un grupo de cilindros UFS. En la versión de distribución de disco 1, cada AU contiene un conjunto de inodos y bloques de datos junto con mapas de bits de inodos y extensiones y resúmenes de extensiones. El **registro de intenciones** sigue inmediatamente al superbloque y la primera unidad de asignación sigue inmediatamente al registro.



**Figure 9.1** The VxFS version 1 disk layout.

4. XFS: El inodo XFS consta de tres partes: el núcleo del inodo, la bifurcación de datos y la bifurcación de atributo opcional. El núcleo de inodo contiene metadatos de inodo tradicionales de UNIX como propietario y grupo, número de bloques, marcas de tiempo y algunas adiciones específicas de XFS, como el ID del proyecto.
5. Ext2FS: (lógica) grupos de bloques en vez de cilindros. En cada bloque encontramos un array inodo. Almacena symbolic links en inodo. Asigna bloques contiguos.

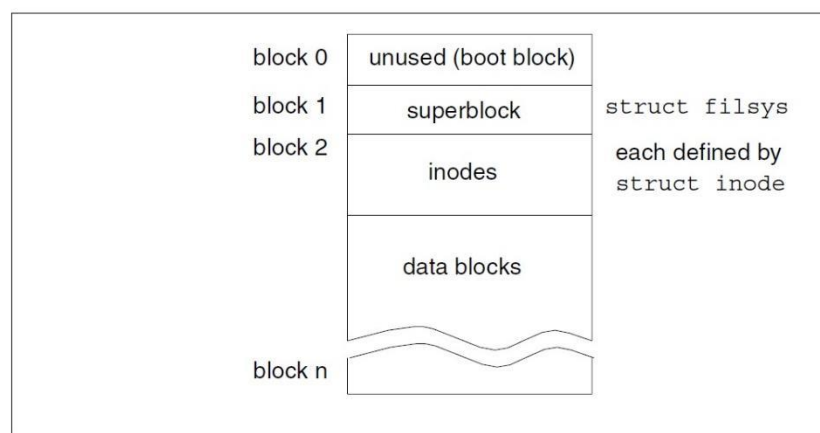


## 5. Comparación

### a) Implementación de inodos en UNIX y LINUX

A la hora de comparar las implementaciones de i-nodos entre los sistemas operativos de Unix y Linux cabe destacar que las diferencias son sutiles porque comparten una herencia y hasta estándares similares (POSIX).

Es necesario describir como el sistema de archivos original de Unix era almacenado en disco. Podría pensarse como una estructura en bloques. El primer bloque (512 bytes, boot block). El segundo, bloque nº1, conocido como super bloque, era una estructura que contenía la información sobre el sistema de archivos, entre ellas, el número de bloques en el sistema, el número de i-nodos (archivos) y el número de i-nodos y bloques de datos libres.



**Figure 6.1** The on-disk layout of the first UNIX filesystem.

Cada archivo del sistema estaba representado por un i-nodo único que contenía campos como:

- 1) `i_mode`: que especificaba si el archivo era un directorio (IFDIR), un bloque especial (IFBLK), o un archivo de caracteres especial (IFCHR). Si alguno de estos modos no era seteado, entonces se consideraba al archivo como regular.
- 2) `i_nlink`: guardaba el número de enlaces duros (hard links) al archivo. Si alcanzaba el 0, entonces el i-nodo era liberado.
- 3) `i_uid`: Id usuario del archivo.
- 4) `i_guid`: Id grupo del archivo.
- 5) `i_size`: el tamaño en bytes del archivo.
- 6) `i_addr`: este campo contiene las direcciones en el disco donde los bloques de datos son guardados.
- 7) `i_mtime`: el tiempo de la última modificación del archivo.
- 8) `i_atime`: el tiempo del último acceso al archivo.



El campo `i_addr` era un vector de 8 punteros. Cada puntero podía referenciar a un único bloque del disco, otorgando 512 bytes de almacenamiento o podía referenciar a un bloque indirecto. Cada bloque de indirección contenía 32 punteros, y cada uno de ellos podía apuntar a un bloque de 512 bytes de almacenamiento o a un bloque de indirección doble. Los bloques de indirección doble apuntan a bloques de datos indirectos.

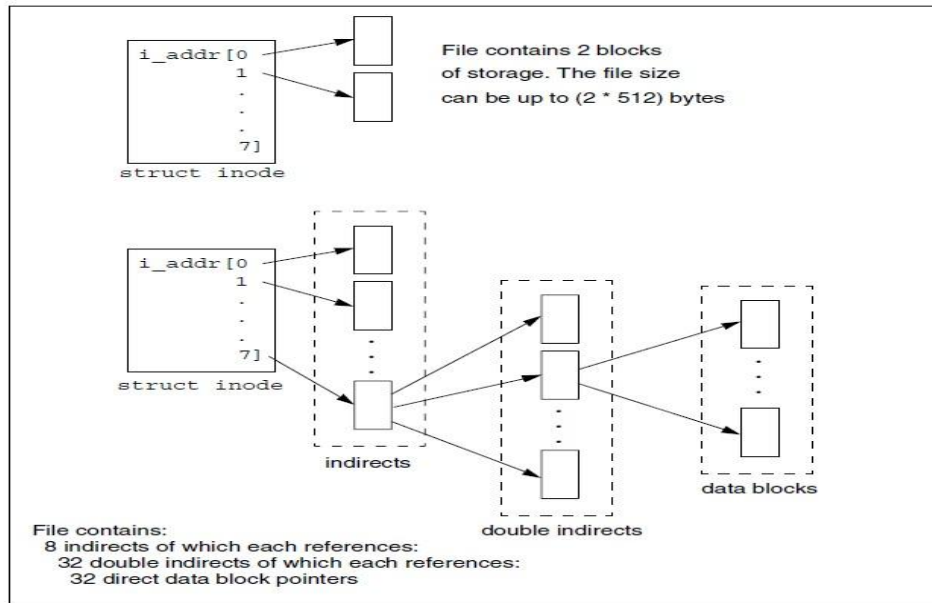


Figure 6.2 File storage through the use of indirect data blocks.

En el primer ejemplo se observa como el i-nodo referencia a 2 bloques de datos. En este caso el tamaño del archivo estaría dentro de 513 y 1024 bytes. Si el tamaño del archivo fuese menor a 512 bytes, un solo bloque de datos sería necesario. El valor del vector `i_addr[]` sería NULL de las posición 2 a la 7.

En el segundo caso muestra un archivo con el máximo tamaño posible, donde cada elemento de `i_addr[]` referencia a un bloque indirecto. Cada uno de estos bloques apunta a 32 bloques de indirección dobles, y cada bloque de indirección doble apunta a 32 bloques de datos. Siendo el tamaño máximo del archivo  $32 * 32 * 32 = 32768$  bloques de datos.

El sistema operativo Unix contaba además con una memoria cache para los i-nodos. Como cada archivo era representado en disco por un i-nodo, por ejemplo cuando se abría, el i-nodo debía ser recuperado del disco; esto también ocurría cuando se realizaban otras llamadas al sistema como `stat()` que tomaban mucha de su información de la estructura del i-nodo.

El i-nodo debía mantenerse en memoria mientras durase la apertura del archivo y hasta podía ser actualizado siempre y cuando una determinada operación provocase cambios en el contenido de su estructura. Por ejemplo, si se escribiesen 512 bytes de datos donde terminaba un archivo que tenía un bloque asignado(`i_addr[0]`) se tendrían que modificar los campos `i_size` por 1024 bytes, asignar un nuevo bloque al archivo y setear `i_addr[1]` para que apunte a este nuevo bloque. Luego de que el archivo fuese cerrado y que ningún otro proceso lo retenga abierto, el i-nodo podía ser liberado.

Como muchos i-nodos se acceden frecuentemente (ej: /usr, /user/bin), existe esta memoria cache i-nodos que los retiene aun cuando no están siendo usados.

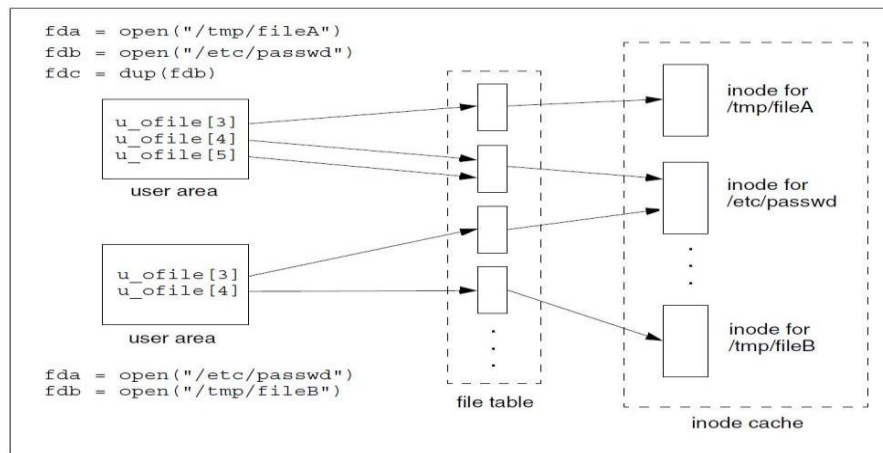


Figure 6.3 Mapping between file descriptors and the inode cache.

Para describir como estos subsistemas trabajan juntos ilustramos una llamada al sistema `open()`, `read()` y `close()`.

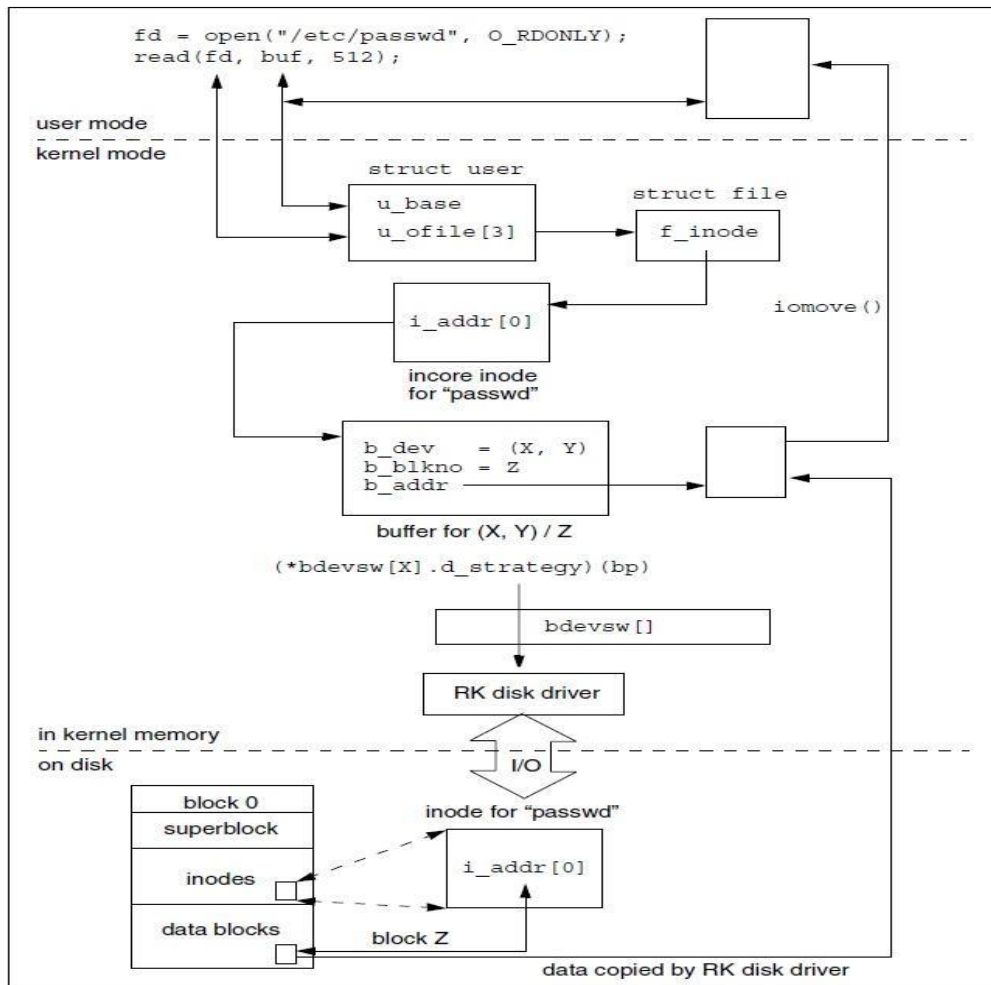
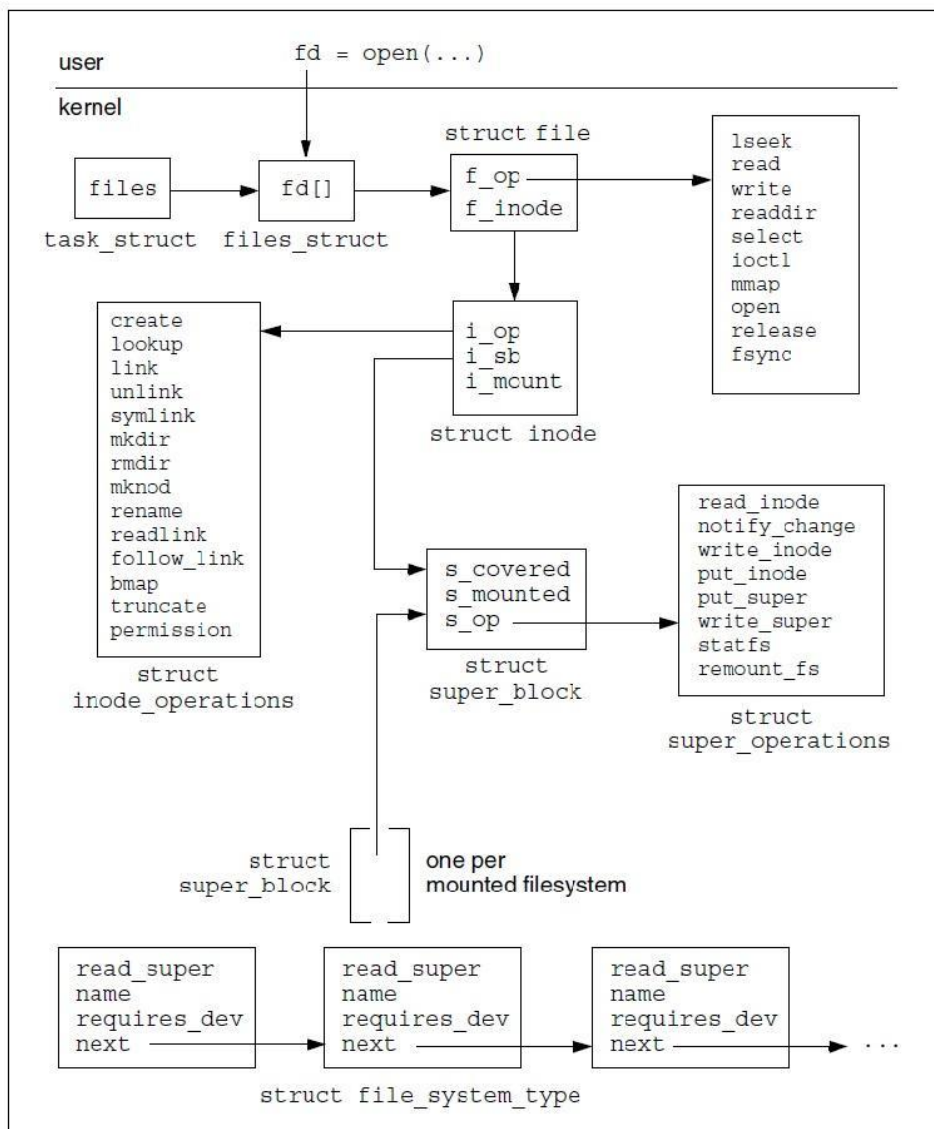


Figure 6.4 Kernel structures used when reading from a file.

Las principales estructuras del sistema de archivos de Linux VFS se muestran a continuación.



**Figure 8.4** Main structures of the Linux 2.2 VFS architecture.

Como en otras implementaciones de Unix, los descriptores de archivos se utilizan para indexar dentro de vectores del proceso que contienen punteros a las tablas de archivos del sistema.

Linux tiene una memoria cache i-nodo centralizada como en versiones anteriores de Unix, cuenta con una estructura i-nodo y todos los i-nodos se encuentran en una lista enlazada encabezada por `first_inode` una variable kernel.

```

Struct inode {
    unsigned long    i_ino;        /*Inode number*/
    atomic_t         i_count;      /*Reference count*/
    kdev_t           i_dev;        /*Filesystem device/
    umode_t          i_mode;       /*Type/
    nlink_t          i_nlink;      /*# hard links*/
    uid_t            i_uid;        /*user ID*/
    gid_t            i_gid;        /*group ID*/
    kdev_t           i_rdev;       /*for device files*/
    loff_t           i_size;       /*File size*/
    time_t           i_atime;      /*access*/
    time_t           i_mtime;      /*modification*/
    time_t           i_ctime;      /*creation*/
    unsigned long    i_blksize;    /*Fs block size*/
    unsigned long    i_blocks;     /*# blocks in file*/
    struct inode_operation *i_op; /*inode operations*/
    struct super_block *i_sb;     /*superblock/mount*/
    struct vm_area_struct *i_mmap; /*mapped file areas*/
    unsigned char    i_update;     /*is inode current?*/
    union {
        struct minix_inode_info minix_i;
        struct ext2_inode_info ext2_i;
        ...
        void *generic_ip;
    } u;
};

```

En vez de tener un único puntero por archivo el elemento u al final de la estructura contiene una unión de todas las posibles estructuras de datos privadas del sistema de archivos. Asociado a cada i-nodo existe un conjunto de operaciones que pueden ser realizadas en el archivo: create, lookup, link, unlink, symlink, mkdir, rmdir, mknod, rename, readlink, follow\_link, bmap, truncate, permission.

## b) Implementación en Windows

Por último, Windows no utiliza i-nodos porque sus sistemas de archivos principales son NTFS y FAT. El sistema de archivos NTFS contiene un archivo llamado Master File Table. Existe al menos una entrada en esta tabla por cada archivo que pertenece al sistema. Toda la información sobre el archivo, incluyendo su tamaño, tiempo, permisos, etc es guardado en esta tabla.

Mientras que FAT consiste en un sistema de archivos del sector descriptor (sector de booteo), una tabla de sistema de archivos de asignación de bloques y el espacio de almacenamiento sin formato para almacenar archivos y carpetas. Los archivos de FAT se almacenan en directorios de conjuntos de registros de 32 bytes. Registro de atributos de archivo del primer bloque de un archivo. Cualquier bloque siguiente se puede encontrar a través de una tabla de asignación de bloque, utilizándola como una lista enlazada.

## BIBLIOGRAFÍA

- Pate, Steve. (2003). UNIX Filesystems Evolution, Design, and Implementation. Wiley.
- La Catedra. (2006). Notas sobre Sistemas Operativos. Ghia.
- Wolf, Ruiz, Bergero. (2015). Fundamentos Sistemas Operativos. UNAM Facultad Ingenieria.
- Carles. (2018). Inodos y dentries en un sistema de archivos. <https://geekland.eu/inodos-dentires-sistema-archivos/>
- Heiser, Gernot. (2020). Advanced Operating Systems - The Berkeley Fast File System (FFS). <https://www.engineering.unsw.edu.au/>