3.4 Administración de archivos en Linux

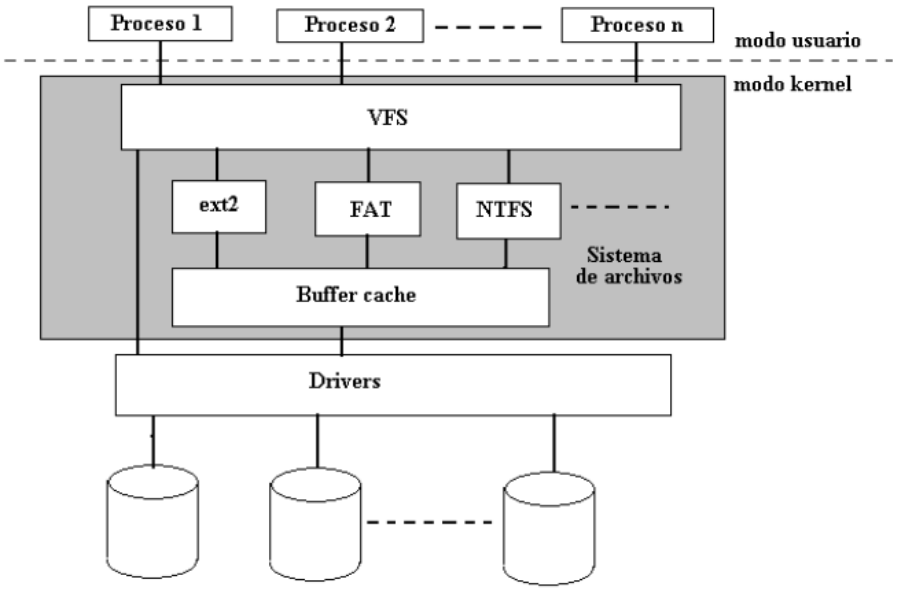
Comenzaremos la explicación del sistema de archivos de Linux, exponiendo la idea de lo que se denomina **Sistema de Archivos Virtual** o simplemente **VFS** (Virtual File system).

El sistema de archivos VFS

Un sistema de archivos virtual (VFS) o conmutador de sistema de archivos virtual, es una capa de abstracción encima de un sistema de archivos más específico o real. El propósito de un VFS es permitir que las aplicaciones cliente tengan acceso a diversos tipos de sistemas de archivos específicos (FAT, ext3, NTFS, etc.) de una manera uniforme. Puede ser utilizada para transparentar las diferencias en los sistemas de archivos de Windows, Mac OS, Unix, etc. de modo que las aplicaciones pudieran tener acceso a archivos en los sistemas de archivos locales de esos tipos sin tener que saber a qué tipo de sistema de archivos están teniendo acceso.

Un VFS especifica un interfaz entre el kernel y un sistema de archivos en específico. Por lo tanto, es fácil agregar nuevos sistemas de archivos al kernel simplemente satisfaciendo la interfaz.

Por lo tanto VFS provee una capa de abstracción que permite a los procesos trabajar con un modelo de archivos común, el cual deberá representar cualquier comportamiento de un sistema de archivos específico. En la figura se muestra un esquema que expresa esta idea en una forma simple.



El sistema VFS está implementado usando el concepto de objetos, pero con estructuras del lenguaje C, ya que Linux usa lenguaje C y no C++. Los principales objetos de la estructura VFS son:

- El objeto **Super Block**, que representa a un sistema de archivos específico montado
- El objeto **inodo** que representa a un archivo específico
- El objeto **dentry**, que representa a una entrada de directorio, como elemento de un path
- El objeto **file**, que representa a un archivo abierto asociado con un proceso

Cada uno de estos objetos tiene definido funciones u operaciones (con la estructura de datos asociada para su administración) que usa el kernel para manejar dichos objetos. Estas operaciones son básicamente:
- El objeto ***super_operations***, que es una estructura que usa el kernel para leer y grabar inodos, como así también al Super Block y registro de estadísticas requeridos por llamadas al sistema operativo, como *fstatfs* y *status*.
- El objeto ***inode_operations***, que contiene funciones que usa el kernel para usar sobre un archivo específico como *create()* y *link()*.
- El objeto ***dentry_operations***, que contiene las funciones que usa el kernel para actuar sobre las entradas de directorio como *d_compare( )* y *d_delete( )*
- El objeto ***file***, que contiene las operaciones que un proceso puede realizar como *read()* y *write()*.

Los sistemas de archivos soportados por VFS pueden agruparse en tres clases principales:

Sistema de archivos basados en disco: consiste en el manejo del espacio de disco u otro dispositivo que emula a un disco como un dispositivo de memoria flash. Alguno de estos sistemas de archivos son:
- Sistemas de archivos nativos de Linux como: ext2, ext3 y ReiserFS
- Sistemas de archivos derivados de Unix como: System V FS, Xenix, BSD FS, Solaris FS, Minix FS
- Sistema de archivos de Microsoft como: MS-DOS, VFAT, NTFS
- Sistemas de archivos de CD-ROM como: ISO9660, sistema de archivos UDF (Universal Disk Format para DVD)
- Sistemas de archivos propietarios de IBM, Apple, Amiga, etc.

Sistemas de archivos de red: permiten el acceso a archivos almacenados en servidores de red y también sistemas de archivos distribuidos como: NFS, Coda, AFS (Andrew filesystem), CIFS (Common Internet File System, usados en Microsoft Windows), y NCP (Novell NetWare Core Protocol).

Sistemas de archivos especiales: no manejan espacio de almacenamiento en disco en forma local ni en forma remota. Tal es el caso de sistemas emulados en memoria como el bien conocido */proc* de Unix|Linux.

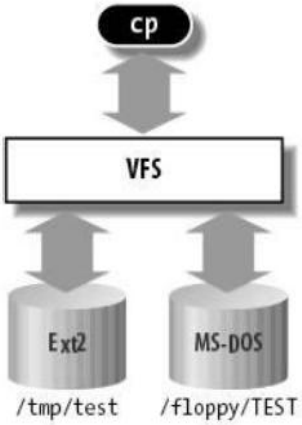
Como trabaja VFS

Como vimos, VFS maneja las llamadas al sistema operativo que realiza un proceso, usando la semántica Unix estándar. El objetivo mayor es proveer una interfaz común con el resto de los sistemas de archivos específicos.
Por ejemplo, un usuario ejecuta el siguiente comando del shell:

```
$ cp /floppy/TEST /tmp/test
```

Donde /floppy es el punto de montaje de un disquete con sistema de archivos **DOS**, y /tmp es un directorio normal del sistema de archivos **ext2**. Con la ayuda de VFS, el programa *cp* no necesita saber las particularidades de ambos sistemas de archivos. El

programa *cp* interactúa con VFS mediante una llamada al sistema operativo usando la semántica genérica de Unix como se muestra a continuación.



(a)

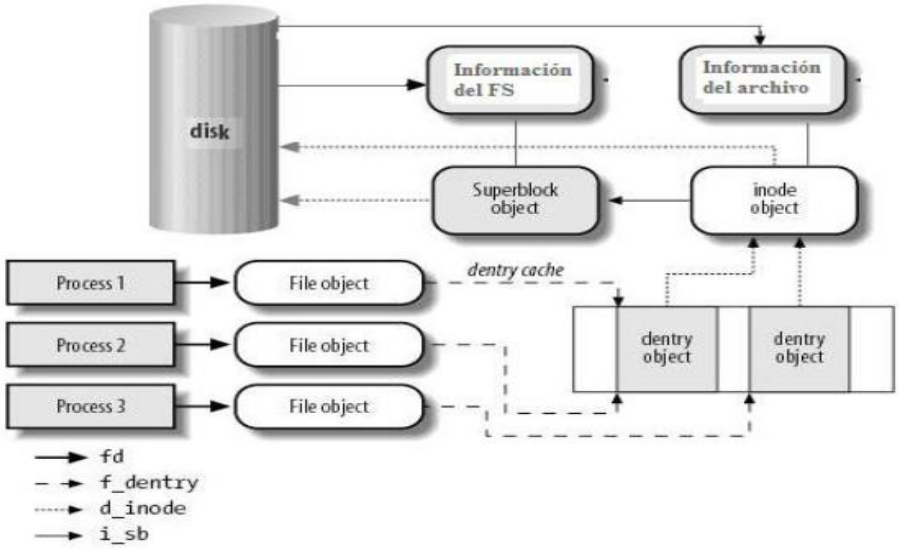
```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

(b)

Vemos que hace uso de las llamadas al sistema ***open***, ***read***, ***write*** y ***close*** que usa el sistema de archivos de Unix.

El modelo de archivo común

Para logra el objetivo que se plantea con el ejemplo anterior, VFS se basa en un **modelo de archivo común** que permite simular un sistemas de archivos estándar de Unix, de tal manera que genere la mínima sobrecarga a las operaciones con archivos nativos de Linux como son los que pertenecen a sistemas de archivos ext2 y ext3.
Este modelo se basa en el uso de los objetos básicos ya indicados. En la siguiente figura se muestra el modelo de archivo común que hace uso de ellos. Se muestra un ejemplo de tres procesos que tienen abierto un mismo archivo, en donde dos de ellos acceden al mismo enlace duro (misma entrada de directorio). Allí vemos la relación entre las estructuras de los objetos file, dentry, inodo y superblock.



Como ya vimos en cada bloque del diagrama (situado en la memoria), se realiza la traducción del sistema que corresponde a cada sistema de archivos (en el disco cada sistema mantiene su estructura específica). Por ejemplo, en el objeto archivo se realizan las operaciones relacionadas con los parámetros propios de un archivo abierto, que se acceden mediante un descriptor de archivo que llamamos **fd**. Desde allí se accede a la estructura que contiene la información de la entrada del directorio (dentry object). Luego se accede al objeto inodo, donde se maneja información relativa a las propiedades o atributos del archivo. Finalmente se accede al objeto superblock que contiene información administrativa específica del sistema de archivos en general.
Como vemos en estas cuatro etapas de administración se tienen en cuenta las características propias de cada sistema de archivos que involucra un archivo abierto, una entrada de directorio, las propiedades del archivo y las características globales del sistema de archivo. Esta estructura es la más adecuada para minimizar la sobrecarga cuando se accede a archivos de un sistema de archivos nativo como lo son ext2 y ext3.

Llamadas al sistema operativo manejadas en FVS

A continuación se muestran las llamadas al sistema más importantes en el sistema de archivo VFS.

Nombre de la llamada	Descripción
`mount( ) umount( ) umount2( )`	Monta/desmonta un sistema de archivo
`sysfs( )`	Obtiene información del sistema
`statfs( ) fstatfs( ) statfs64( ) fstatfs64( )` `ustat( )`	Obtiene estadísticas del sistema
`chroot( ) pivot_root( )`	Cambia el directorio raíz
`chdir( ) fchdir( ) getcwd( )`	Manejo del directorio actual

Nombre de la llamada	Descripción
`mkdir( ) rmdir( )`	Crea y elimina directorios
`getdents( ) getdents64( ) readdir( ) link( )` `unlink( ) rename( ) lookup_dcookie( )`	Maneja entradas de directorio
`readlink( ) symlink( )`	Maneja enlaces simbólicos
`chown( ) fchown( ) lchown( ) chown16( )` `fchown16( ) lchown16( )`	Cambio de propietario
`chmod( ) fchmod( ) utime( )`	Cambio de atributos

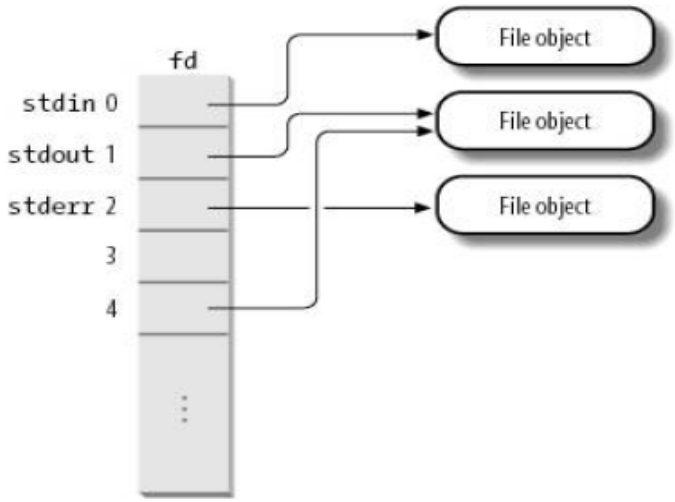
`stat( ) fstat( ) lstat( ) access( ) oldstat( ) oldfstat( ) oldlstat( )`
`stat64( ) lstat64( )`
`fstat64( )`
	Lee el estado de un archivo

`open( ) close( ) creat( ) umask( )`	Abrir, cerrar y crear archivos
`dup( ) dup2( ) fcntl( ) fcntl64( )`	Manejo de descriptores de archivos
`select( ) poll( )`	Espera eventos en un conjunto de decriptores
`truncate( ) ftruncate( ) truncate64( )` `ftruncate64( )`	Cambia el tamaño dela rchivo
`lseek( ) _llseek( )`	Maneja el puntero del archivo
`read( ) write( ) readv( ) writev( ) sendfile( ) sendfile64( ) readahead( )`	Operaciones con el archivo
io_setup( ) io_submit( ) io_getevents( ) io_cancel( ) io_destroy( )	E/S asincronas
`pread64( ) pwrite64( )`	Acceso en forma aleatoria
`mmap( ) mmap2( ) munmap( ) madvise( ) mincore( )` `remap_file_pages( )`	Manejo de proyección de archivos
`fdatasync( )` fsync( ) sync( ) msync( )	Sincronización con datos del archivo
`flock( )`	Manejo de bloqueos del archivo
setxattr( ) lsetxattr( ) fsetxattr( ) getxattr( ) lgetxattr( ) fgetxattr( ) listxattr( ) llistxattr( ) flistxattr( ) removexattr( ) lremovexattr( ) fremovexattr( )	Manejo de atributos extendidos

Como también vemos en la figura anterior, existen ciertas operaciones con archivos en donde se accede directamente al disco sin pasar por la traducción propia de un sistema de archivo en particular, como es el caso cuando un proceso cierra un archivo abierto.

Procesos y archivos

Cada proceso tiene en su *descriptor de proceso* (estructura que actúa como Bloque de Control de Proceso) punteros a sendas estructuras *fs_struct* y *files_struct*, que se usan para registrar datos administrativos relacionados al sistema de archivo y a los archivos abiertos de cada proceso. Allí se registran datos administrativos del sistema VFS que relacionan al proceso con su actividad sobre el sistema de archivos y los archivos abiertos por él. Justamente, desde la estructura *files_struct* se apunta a los *objetos archivo* que se muestra en la figura anterior. Como ya vimos en Unix, también aquí en VFS, cada proceso tiene por defecto tres archivos abiertos que permiten el acceso al dispositivo de
entrada estándar, dispositivo de salida estándar y al dispositivo de salida de error estándar mediante los descriptores de archivos 0, 1 y 2 respectivamente, como se indica en la siguiente figura.



En Linux como en Unix, los discos lógicos (dispositivos) o sistemas de archivos, deben montarse para poder ser accedidos. La forma clásica es mediante el comando *mount* como se indica a continuación.
Mount –t tipo_de_sistema_de_archivo dispositivo punto de montaje

Por ejemplo.
Mount –t ext3 /dev/hda1 /mnt/disco_uno
Montamos el disco lógico /dev/hda1 (equivalente al disco IDE0) en el directorio /mnt/disco_uno.

Para desmontar se usa el comando umount:

Umount /dev/hda1