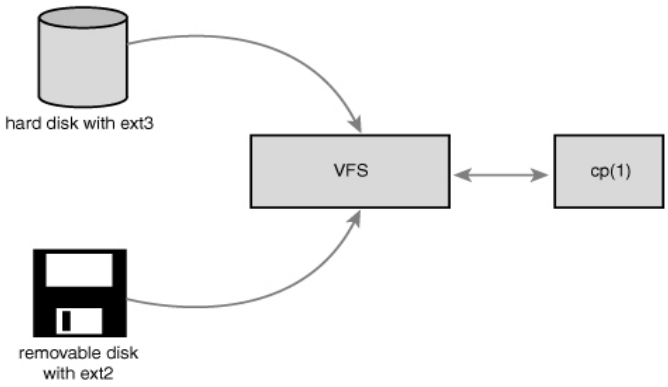
THE VIRTUAL FILESYSTEM

Subsistema del kernel que implementa las interfaces relacionadas con los ficheros y el sistema de ficheros proporcionadas a los programas del espacio de usuario. Todos los sistemas de archivos dependen del VFS para poder coexistir e interoperar. Esto permite a los programas utilizar las llamadas estándar para leer y escribir.

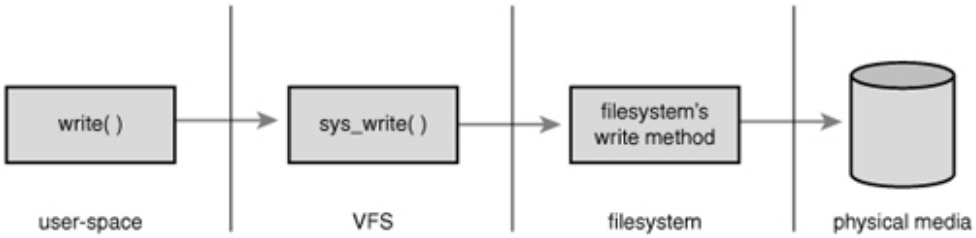


- **CAPA DE ABSTRACCIÓN DEL SISTEMA DE FICHEROS**

La capa de abstracción funciona definiendo las interfaces conceptuales básicas y las estructuras de datos que soportan todos los sistemas de archivos. El código real del sistema de archivos oculta los detalles de la implementación. Sin embargo, para la capa del VFS y el resto del núcleo, todos los sistemas de archivos tienen el mismo aspecto.

Todos admiten archivos y directorios, y operaciones como la creación y eliminación de ficheros.

Resultado de realizar write:



- **SISTEMAS DE FICHEROS DE UNIX**
- Un **sistema de ficheros** es un almacenamiento jerárquico de datos que se adhiere a una estructura específica. Los sistemas de archivos contienen archivos, directorios e información de control asociada. Las operaciones típicas que se realizan son la creación, el borrado y el montaje. En Unix, los sistemas de archivos se montan en un punto de montaje específico en una jerarquía global conocida como namespace.

- Un **fichero** es una cadena ordenada de bytes. El primer byte marca el comienzo del fichero, y el último byte marca el final. A cada fichero se le asigna un nombre legible para su identificación, tanto por el sistema como por el usuario. Las operaciones típicas de los ficheros son leer, escribir, crear y borrar. Están organizados en directorios
- Un **directorio** es análogo a una carpeta y suele contener archivos relacionados. Los directorios también pueden contener otros directorios, llamados subdirectorios. De este modo, los directorios pueden anidarse para formar rutas.

Los sistemas Unix separan el concepto de archivo de cualquier información asociada a él, como los permisos de acceso... Esta información se denomina a veces metadatos del archivo y se almacena en un inodo.

Toda esta información está unida a la información de control propia del sistema de archivos, que se almacena en el superbloque. El superbloque es una estructura de datos que contiene información sobre el sistema de archivos en su conjunto.

- **OBJETOS DEL VFS Y SUS ESTRUCTURAS**

Cuatro tipos de objetos:

- El objeto **superbloque**, que representa un sistema de ficheros específico montado.
- El objeto **inodo**, que representa un fichero específico.
- El objeto **dentry**, que representa una entrada de directorio, que es un componente único de una ruta.
- El objeto **file**, que representa un fichero abierto asociado a un proceso.

Cuatro tipos de operaciones, una para cada objeto:

- El objeto **super_operations**, que contiene los métodos que el núcleo puede invocar en un sistema de archivos específico, como write_inode() y sync_fs()
- El objeto **inode_operations**, que contiene los métodos que el núcleo puede invocar en un archivo específico, como create() y link()
- El objeto **dentry_operations**, que contiene los métodos que el núcleo puede invocar en una entrada de directorio específica, como d_compare() y d_delete()
- El objeto **file_operations**, que contiene los métodos que un proceso puede invocar en un archivo abierto, como read() y write()

Las operaciones se implementan como una estructura de punteros a funciones que operan sobre el objeto padre.

- **OBJETO SUPERBLOQUE Y OPERACIONES**

El objeto superbloque es implementado por cada sistema de archivos y se utiliza para almacenar información que describe ese sistema de ficheros específico. Este objeto suele corresponder al superbloque del sistema de archivos o al bloque de control del sistema de archivos, que se almacena en un sector especial del disco.

En su estructura nos importa **s_op**, que es un puntero a la estructura que realiza las operaciones:

Operaciones importantes:

`void write_inode(struct inode *inode, int wait):Escribe el inodo al disco.`

`void write_super(struct super_block *sb): Actualiza el superbloque del disco con el superbloque especificado.`

`int sync_fs(struct super_block *sb, int wait): Sincroniza los metadatos del sistema de archivos con el sistema de archivos en disco.`

- **OJETO INODO Y OPERACIONES**

El objeto inodo representa toda la información que necesita el kernel para manipular un fichero o directorio. En los sistemas de ficheros tipo Unix, esta información se lee simplemente del inodo del disco. Sin embargo, si un sistema de ficheros no tiene inodos, el sistema de ficheros debe obtener la información de donde sea que esté almacenada en el disco. Los sistemas de archivos sin inodos generalmente almacenan la información específica del fichero como parte del fichero.

Nos interesa i_op, puntero a una tabla de funciones.

Operaciones importantes:

`int create(struct inode *dir, struct dentry *dentry, int mode): Crea inodo.`

`int mkdir(struct inode *dir,struct dentry *dentry,int mode): Crea directorio.`

`int rmdir(struct inode *dir,struct dentry *dentry):Borra directorio.`

`int mknod(struct inode *dir,struct dentry *dentry,int mode, dev_t rdev):Crea un fichero especial.`

`void truncate(struct inode *inode):Modifica el tamaño del fichero dado.`

`int getattr(struct vfsmount *mnt,struct dentry *dentry,struct kstat *stat): Invocado por el VFS al notar que un inodo necesita ser refrescado desde el disco.`

- **OBJETO DENTRY Y OPERACIONES**

Un dentry es un componente específico en un path. Los objetos Dentry son todos los componentes de una ruta, incluidos los archivos. También pueden incluir puntos de montaje.

Nos interesa d_op.

Operaciones importantes

`int d_compare(struct dentry *dentry,struct qstr *name1,struct qstr *name2): Compara dos nombres de fichero.`

`int d_delete (struct dentry *dentry): Borra si no se usa el dentry.`

`void d_release(struct dentry *dentry): Se hace cuando el dentry va a ser liberado.`

Estados del dentry: usado, no usado o negativo,

Dentry usado corresponde a un inodo válido (d_inode apunta a un inodo asociado) e indica que hay uno o más usuarios del objeto. Esta en uso por el VFS y apunta a datos válidos.

Dentry no usado corresponde a un inodo válido, pero el VFS no está utilizando actualmente el objeto dentry. Debido a que el objeto dentry todavía apunta a un objeto válido, el dentry se mantiene en la caché en caso de que se necesite de nuevo.

Dentry negativo no está asociado con un inodo válido (d_inode es NULL) porque o bien el inodo fue borrado o el nombre de la ruta no es correcto. Sin embargo, se mantiene para que las futuras búsquedas se resuelvan rápidamente.

Por otro lado tenemos la dentry cache que consiste en:

Listas dentry "usadas" vinculadas a su inodo asociado. Dado que un inodo dado puede tener múltiples enlaces, puede haber múltiples objetos dentry.

Una lista doblemente enlazada de dentry objects no utilizados y negativos.

Una tabla de hash y una función de hashing utilizadas para resolver rápidamente una ruta dada en el objeto de dentry asociado.

Para acceder a rutas el VFS mira en la cache de dentry, por lo que se aceleran las búsquedas si se encuentra en ella

- **OBJETO FICHERO Y OPERACIONES**

El objeto fichero representa a un fichero abierto por un proceso.

Nos interesa f_op ya que es un puntero que apunta a la tabla de funciones.

Funciones importantes:

`ssize_t read(struct file *file,char *buf, size_t count,loff_t *offset):` lee la cantidad de bytes del archivo dado en la posición offset en buf.

`ssize_t write(struct file *file,const char *buf, size_t count,loff_t *offset):` Escribe la cantidad de bytes de buf en el archivo dado en la posición offset.

`int mmap(struct file *file,struct vm_area_struct *vma):` Mapea en memoria el fichero dado en el espacio de direcciones dado

`int fsync(struct file *file, struct dentry *dentry, int datasync):` Escribe en disco todos los datos en cache del archivo.