Ancor Santiago Romero Gutiérrez

Daniel Suárez García

VFS - Introducción

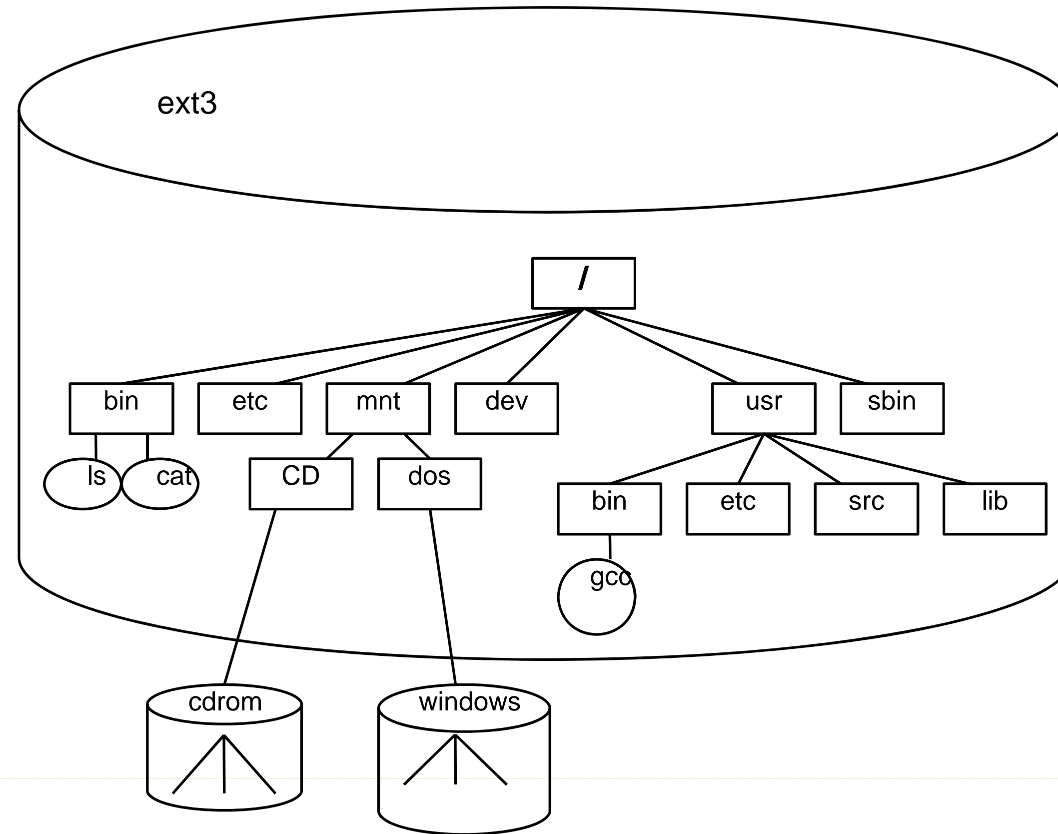
Sistema ficheros: Es una estructura de tipo árbol, en la que los nodos son los directorios y las hojas son los ficheros. Esto permite al sistema operativo mantener los ficheros con un cierto orden.

En el caso de linux, el sistema de ficheros estándar es el ext, que actualmente se encuentra en su versión 3.

Los sistemas soportados por linux se pueden dividir en 3 categorías:

- Basados en disco: discos duros, disquetes, CD-ROM, etc. (Estos sistemas son: ext2, ext3, ReiserFS, XFS, JFS, ISO9660, etc.)
- Sistemas remotos (de red): NFS, Coda, Samba, etc.
- Sistemas especiales: procfs, ramfs y devfs.

VFS - Introducción



VFS - Objetivo

- Objetivo: Transparencia del sistema de ficheros mediante el uso de una capa de software por encima de los sistemas de ficheros reales.

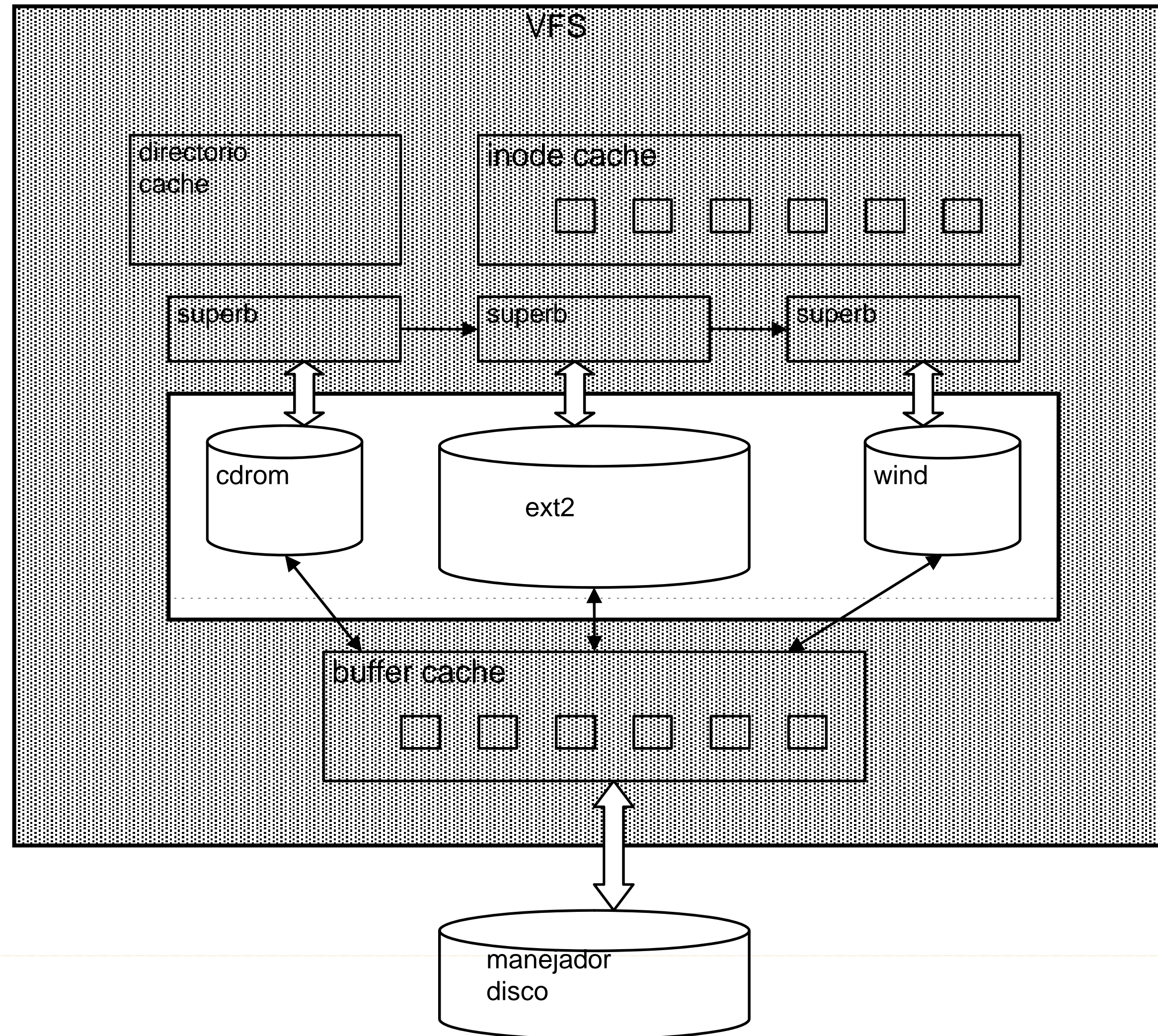
- Para cada lectura, escritura u otra operación que realice el usuario, el VFS apunta a la operación correspondiente para cada sistema de ficheros que sustituirá a la función inicialmente referenciada.

VFS - Funcionamiento

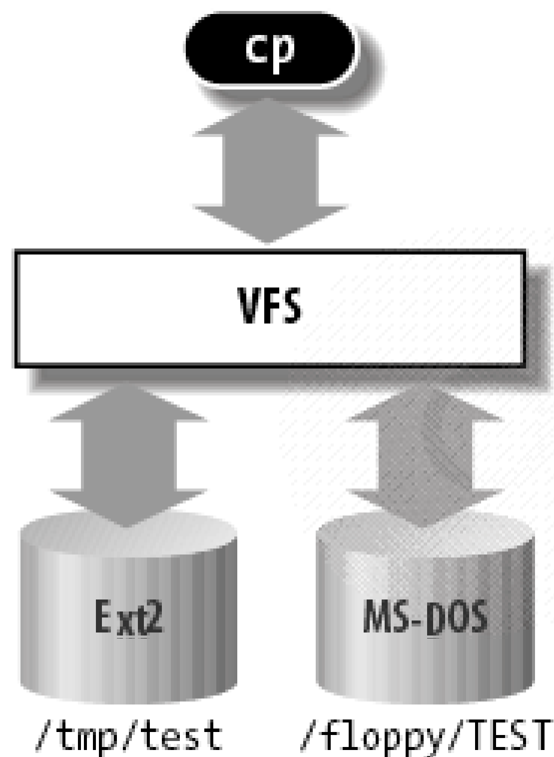
La filosofía de funcionamiento es:

- Se produce una llamada al sistema que implica un acceso a un sistema de ficheros.
- El VFS determina a qué sistema de ficheros vamos a acceder.
- Se comprueba si existe ya una entrada en el caché y si dicha entrada es válida.
- En el caso de que se requiera, se accede a la función del sistema de ficheros específico a través de las estructuras del VFS.
- La función puede, o bien completar la llamada, o bien requerir una operación de E/S sobre un dispositivo, que en su caso puede provocar el paso a modo "sleep" del proceso que invocó la operación sobre el sistema de ficheros , hasta que la operación se complete.

VFS - Funcionamiento



VFS – Ejemplo Funcionamiento




```
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);
```

(a) (b)

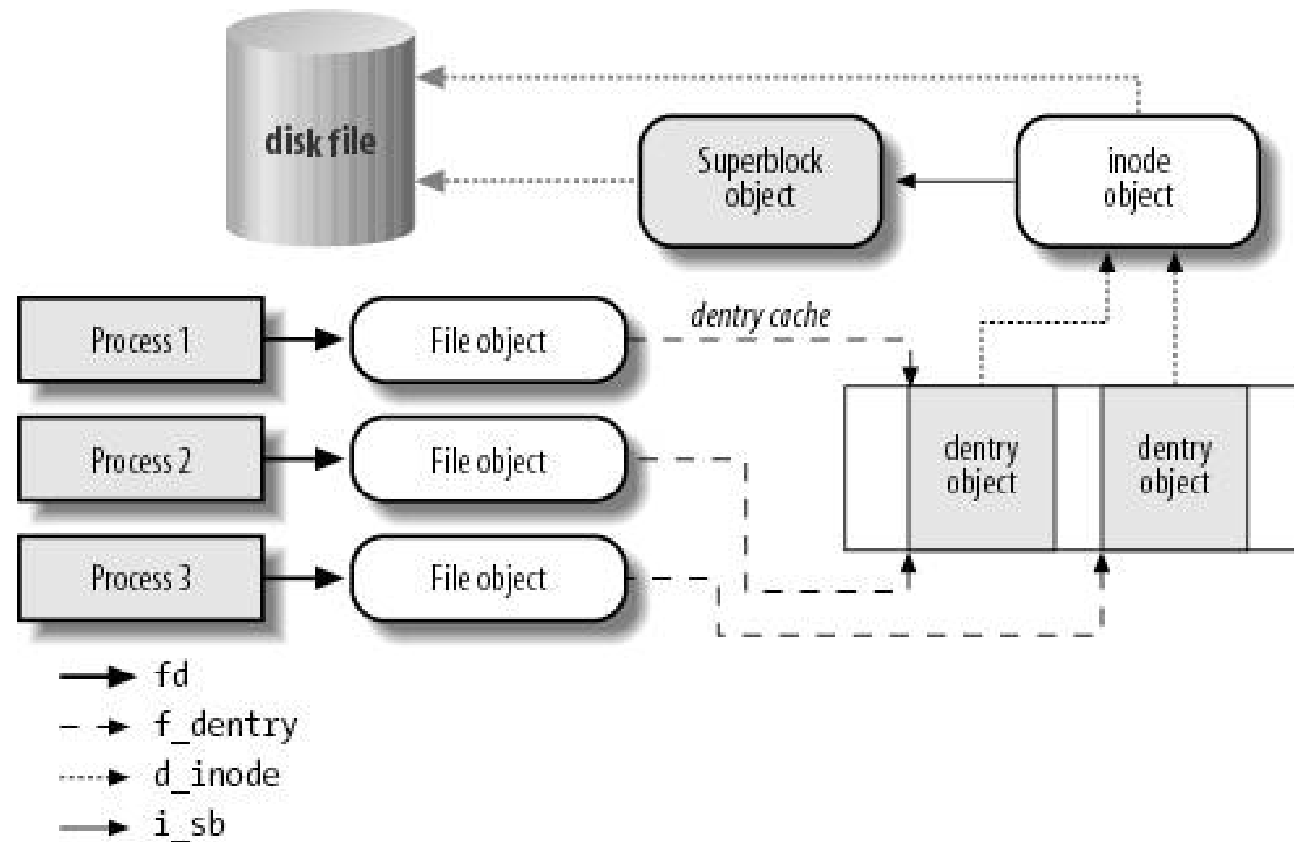
VFS - Estructuras

- VFS tiene un conjunto de estructuras en memoria principal, definidas en el archivo include/linux/fs.h:
    - lista de sistemas soportados
    - lista de sistemas montados
    - superbloque
    - inode
    - buffer cache
    - cache de directorios
    - File
- Una serie de funciones para poder manejarlas.

VFS - Estructuras

- Por su parte, cada sistema de ficheros tiene sus propias estructuras de datos en disco.
- Y unas funciones propias para cada sistema para el manejo de dichas estructuras.

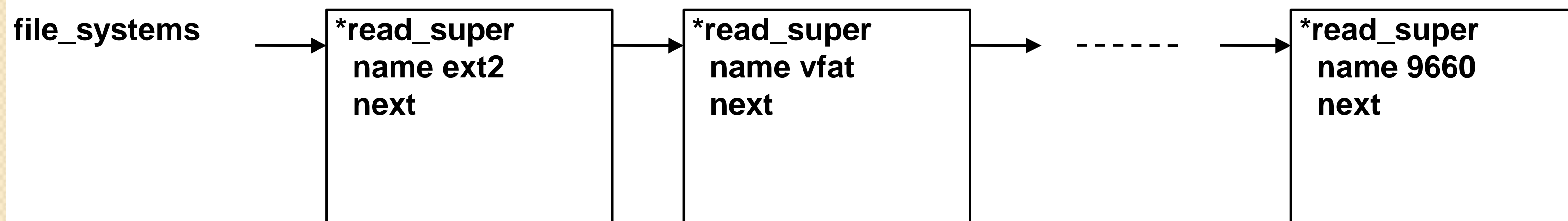
VFS - Interacción entre procesos y entidades VFS



VFS - Sistemas soportados

¿Qué pasa cuando arrancamos el sistema?

Cuando el sistema arranca cada sistema de ficheros se inicializa y se registra en el VFS llamando a la función register_file_system en una lista de estructuras file_system_type, apuntadas por la variable file_systems.



VFS – file_system_type

```
struct file_system_type
{
	const char *name;
	int fs_flags;
	int (*get_sb) (struct file_system_type *, int,
	                         const char *, void *);

	void (*kill_sb) (struct super_block *);
	struct module *owner;
	struct file_system_type * next;
	struct list_head fs_supers;
	struct lock_class_key s_lock_key;
	struct lock_class_key s_umount_key;
	struct lock_class_key i_lock_key;
	struct lock_class_key i_mutex_key;
	struct lock_class_key i_mutex_dir_key;
	struct lock_class_key i_alloc_sem_key
};
```

VFS – file_system_type (campos)

name: nombre, aparece en el fichero /proc/filesystems y es usado como clave para encontrar un sistema de ficheros por su nombre

fs_flags: Banderas,
FS_REQUIRES_DEV para sistemas de ficheros que sólo pueden ser montados como dispositivos de bloque.
FS_SINGLE para sistemas de ficheros que pueden tener sólo un superbloque.
FS_NOMOUNT para los sistemas de ficheros que no pueden ser montados desde el espacio de usuario.

***get_sb**: Copia el superbloque del sistema de ficheros real en el VFS. Se invoca cuando montamos un sistema de ficheros.

***kill_sb**: deshace las inicializaciones y las asignaciones realizadas por get_sb(). Se invoca al desmontar el sistema de ficheros.

Next: puntero al siguiente nodo de la lista.

fs_supers: puntero al primer nodo de una lista que contiene todos los superbloques del mismo tipo.

VFS - Sistemas montados

¿Qué pasa cuando se monta un sistema de ficheros real?

Al invocar una orden del tipo:

mount –t sistemaficheros –o ro dispositivofisico puntomontaje

Ej: mount –t iso9660 –o ro /dev/cdrom /mnt/CD

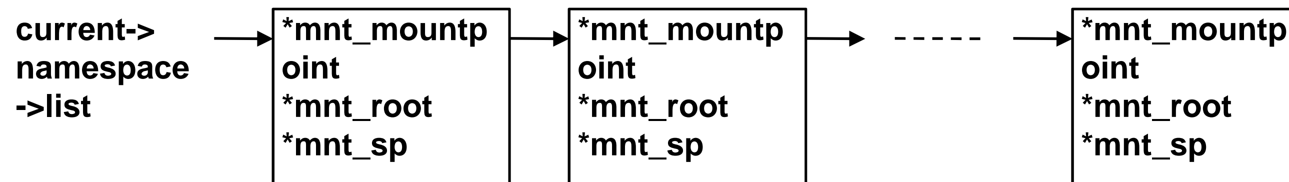
VFS buscará el nombre del sistema de ficheros en la lista apuntada por file_sytems y llamará a la función get_sb que cargará el superbloque correspondiente.

VFS – Sistemas montados

¿Qué pasa cuando se monta un sistema de ficheros real?

file_system_type ofrece poca información acerca del sistema de ficheros montado.

Por ello, cada sistema de ficheros montado se describe por una estructura vfsmount. Todas forman una lista apuntada por current->namespace->list, en /include/linux/mount.h.



VFS - vfsmount

```
struct vfsmount {
    struct list_head mnt_hash;
    struct vfsmount *mnt_parent;            /* fs we are mounted on */
    struct dentry *mnt_mountpoint;          /* dentry of mountpoint */
    struct dentry *mnt_root;        /* root of the mounted tree */
    struct super_block *mnt_sb;     /* pointer to superblock */
    struct list_head mnt_mounts;    /* list of children, anchored here */
    struct list_head mnt_child;     /* and going through their mnt_child */
    int mnt_flags;                  /* 4 bytes hole on 64bits arches */
    const char *mnt_devname;        /* Name of device e.g. /dev/dsk/hda1 */
    struct list_head mnt_list;
    struct list_head mnt_expire;    /* link in fs-specific expiry list */
    struct list_head mnt_share;     /* circular list of shared mounts */
    struct list_head mnt_slave_list;        /* list of slave mounts */
    struct list_head mnt_slave;             /* slave list entry */
```

VFS - vfsmount

```
struct vfsmount *mnt_master;        /* slave is on master->mnt_slave_list */
struct mnt_namespace *mnt_ns;       /* containing namespace */
int mnt_id;                         /* mount identifier */
int mnt_group_id;                   /* peer group identifier */
/*
 * We put mnt_count & mnt_expiry_mark at the end of struct vfsmount
 * to let these frequently modified fields in a separate cache line
 * (so that reads of mnt_flags wont ping-pong on SMP machines)
 */
atomic_t mnt_count;
int mnt_expiry_mark;                /* true if marked for expiry */
int mnt_pinned;
int mnt_ghosts;
/*
 * This value is not stable unless all of the mnt_writers[] spinlocks
 * are held, and all mnt_writer[]s on this mount have 0 as their ->count
 */
atomic_t __mnt_writers;
};
```

VFS – Desmontando un FS

¿Y que pasa cuando desmontamos?

umount /mnt/CD

- El sistema comprueba que el directorio pasado es un punto de montaje, que el usuario tenga privilegios para desmontarlo y que ninguno de sus archivos este en uso.
- Comprueba que en el superbloque el campo s_dirt del superbloque y si este esta activado guarda los cambios en el disco.
- Libera la estructura vfsmount y la la estructura super_block si nadie mas la esta usando.

VFS – Algunas funciones

- alloc_vfsmnt(name)

  Ubica e inicializa un sistema de ficheros montado

- free_vfsmnt(mnt)

  Libera una descriptor de fs apuntado por mnt

- lookup_mnt(mnt, dentry)

  Realiza la función de hash correspondiente y devuelve la dirección de el struct vfsmount correspondiente.

VFS - Superbloque

- Mantiene metadatos sobre los sistemas de ficheros montados. Está representado por un bloque de control de sistema almacenado en disco.

- Cuando los sistemas de ficheros reales se montan en el sistema son representados en el sistema en la tabla super_blocks (linux/fs/super.c) de NR_SUPER elementos de estructura super_block. Esta tabla se encuentra en memoria principal.

VFS – super_block

```
struct super_block {
struct list_head            s_list;       /* Keep this first */
dev_t                       s_dev;        /* search index; _not_ kdev_t */
unsigned long               s_blocksize;
unsigned char               s_blocksize_bits;
unsigned char               s_dirt;
unsigned long long          s_maxbytes;   /* Max file size */
struct file_system_type     *s_type;
const struct super_operations     *s_op;
struct dquot_operations     *dq_op;
struct quotactl_ops         *s_qcop;
const struct export_operations    *s_export_op;
unsigned long               s_flags;
unsigned long               s_magic;
```

VFS – super_block

```
struct dentry                    *s_root;
struct rw_semaphore      s_umount;
struct mutex                     s_lock;
int                              s_count;
int                              s_need_sync_fs;
atomic_t                         s_active;
#ifdef CONFIG_SECURITY
   void                  *s_security;
#endif
struct xattr_handler     **s_xattr;
struct list_head         s_inodes;      /* all inodes */
struct list_head         s_dirty;       /* dirty inodes */
struct list_head         s_io;          /* parked for writeback */
struct list_head         s_more_io;     /* parked for more writeback
   */
```

VFS – super_block

```
struct hlist_head          s_anon;/*   anonymous   dentries   for   (nfs)
    exporting */
struct list_head           s_files;  /* s_dentry_lru and
    s_nr_dentry_unused are protected by dcache_lock */
struct list_head           s_dentry_lru;   /* unused dentry lru */
int                   s_nr_dentry_unused;     /* # of dentry on lru */
struct block_device        *s_bdev;
struct mtd_info            *s_mtd;
struct list_head           s_instances;
struct quota_info          s_dquot;        /* Diskquota specific options */
int                        s_frozen;
wait_queue_head_t          s_wait_unfrozen;
char                  s_id[32];                   /* Informational name */
void                       *s_fs_info;     /* Filesystem private info */
fmode_t                    s_mode;
```

VFS - Superbloque

s_list: una lista doblemente enlazada de todos los superbloques activos.

s_dev: para sistemas de archivos que requieren un bloque para ser montado en él.

s_blocksize, s_blocksize_bits: tamaño del bloque y log2(tamaño del bloque).

s_lock: indica cuando un superbloque está actualmente bloqueado por lock_super()/unlock_super().

s_dirt: establece cuando superbloque está modificado, y limpiado cuando es vuelto a ser escrito a disco.

s_type: puntero a struct file_system_type del sistema de archivos correspondiente.

s_op: puntero a la estructura super_operations, la cual contiene métodos específicos del sistema de archivos para leer/escribir inodos, etc.

VFS - Superbloque

s_flags: banderas de superbloque.

s_magic: número mágico del sistema de archivos. Usado por el sistema de archivos de minix para diferenciar entre múltiples tipos de él mismo.

s_root: dentry de la raiz del sistema de archivos.

s_wait: cola de espera de los procesos esperando para que el superbloque sea desbloqueado.

s_dirty: una lista de todos los inodos sucios.

s_files: una lista de todos los archivos abiertos en este superbloque.

s_bdev: para FS_REQUIRES_DEV, esto apunta a la estructura block_device describiendo el dispositivo en el que el sistema de archivos está montado.

s_mounts: una lista de todas las estructuras vfsmount, una por cada instancia montada de este superbloque.

VFS – Operaciones de superbloque

Las operaciones de superbloque están descritas en la estructura super_operations declarada en include/linux/fs.h:

```
struct super_operations {
    struct inode *(*alloc_inode)(struct super_block *sb);
    void (*destroy_inode)(struct inode *);
        void (*dirty_inode) (struct inode *);
    int (*write_inode) (struct inode *, int);
    void (*drop_inode) (struct inode *);
    void (*delete_inode) (struct inode *);
    void (*put_super) (struct super_block *);
    void (*write_super) (struct super_block *);
    int (*sync_fs)(struct super_block *sb, int wait);
    int (*freeze_fs) (struct super_block *);
    int (*unfreeze_fs) (struct super_block *);
```

VFS – Operaciones de superbloque

```
int (*statfs) (struct dentry *, struct kstatfs *);
int (*remount_fs) (struct super_block *, int *, char *);
void (*clear_inode) (struct inode *);
void (*umount_begin) (struct super_block *);
int (*show_options)(struct seq_file *, struct vfsmount *);
int (*show_stats)(struct seq_file *, struct vfsmount *);
#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);
#endif
int (*bdev_try_to_free_page)(struct super_block*, struct page*,
    gfp_t);
};
```

VFS – Operaciones de superbloqe

read_inode: lee un inodo desde el sistema de archivos.

write_inode: escribe un inodo de vuelta al disco.

put_inode: llamado cuando la cuenta de referencia es decrementada.

delete_inode: llamado cuando inode->i_count y inode->i_nlink llegana a 0.

put_super: llamado en las últimas etapas de la llamada al sistema umount para notificar al sistema de archivos que cualquier información mantenida por el sistema de archivos sobre esa instancia tiene que ser liberada.

write_super: llamado cuando el superbloque necesita ser vuelto a escribir en el disco.

statfs: implementa las llamadas al sistema fstatfs / statfs.

28

VFS – Operaciones de Superbloque

clear_inode: llamado desde el nivel VFS clear_inode(). Los sistemas que

incluyen datos privados a la estructura del inodo (a través del campo generic_ip) deben liberarse aquí.

umount_begin: llamado durante el desmontaje forzado para notificarlo al sistema de archivos de antemano, por lo tanto puede ser lo mejor para asegurarse que nada mantiene al sistema de archivos ocupado.

VFS - Inodos

- Todo fichero y directorio del sistema de ficheros está representado por un inodo del VFS.
- La información que contiene un inodo se construye obteniendo información del sistema de ficheros real mediante funciones específicas de cada uno de ellos.
- Todo fichero o directorio que está siendo accedido, se representa en el VFS por medio de un struct “inode” en la memoria principal del núcleo. Permanecen ahí mientras sean útiles para el sistema.

VFS – Inode cache

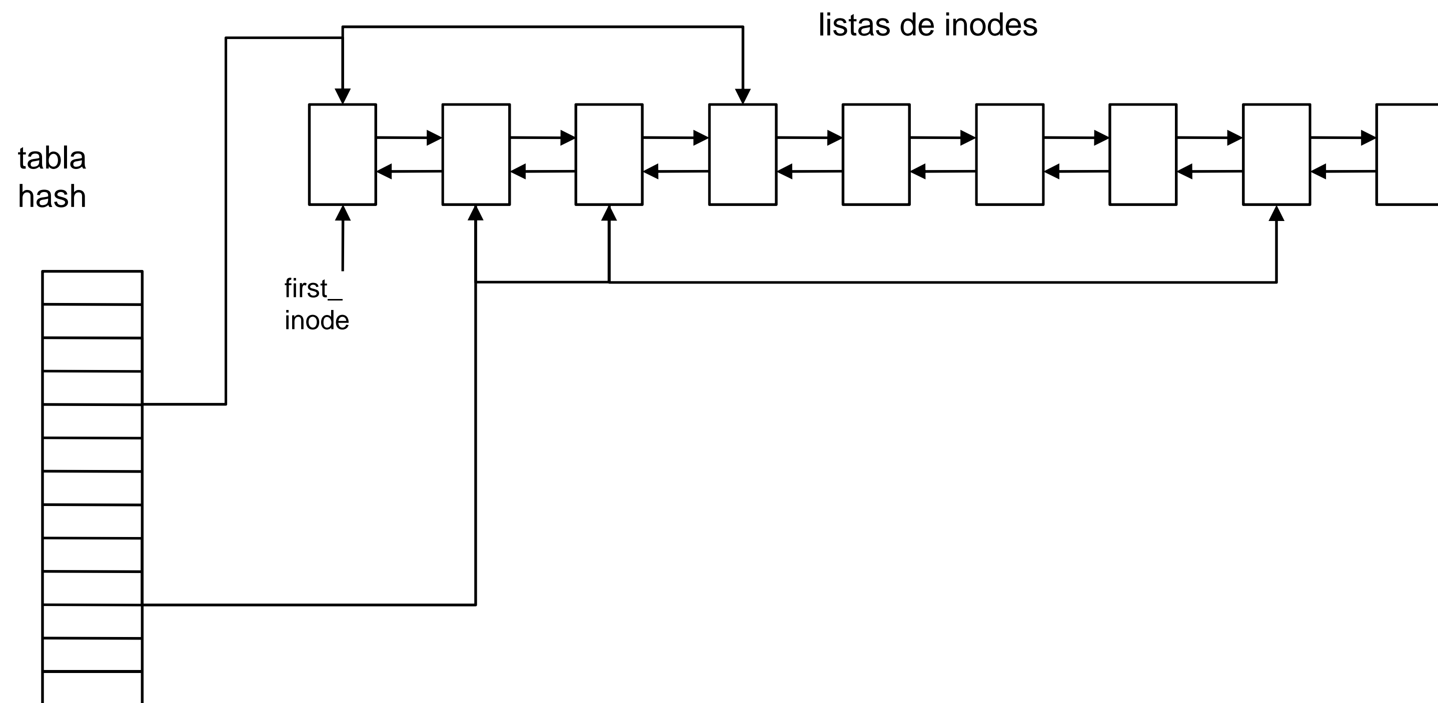
- Inode cache:
  - A medida que navegamos por el sistema de ficheros, sus inodes VFS son leídos o escritos continuamente. VFS mantiene una cache de inodes para acelerar este proceso.

  Esta cache se construye con unas tablas de hash cuyas entradas son punteros a listas doblemente encadenadas de inodes VFS con el mismo valor hash.

- Si un inode se necesita y no está en la cache, se busca en el disco el inodo y se almacena en ésta.

- Las funciones de lectura y escritura de inodos en disco son específicas de cada fs -> "inode_operations".

VFS – Inode cache



VFS – struct inode

```
struct inode {
    struct hlist_node      i_hash;
    struct list_head       i_list;
    struct list_head       i_sb_list;
    struct list_head       i_dentry;
    unsigned long          i_ino;
    atomic_t               i_count;
    unsigned int           i_nlink;
    uid_t                  i_uid;
    gid_t                  i_gid;
    dev_t                  i_rdev;
    u64                    i_version;
    loff_t                 i_size;
    #ifdef __NEED_I_SIZE_ORDERED
            seqcount_t             i_size_seqcount;
    #endif
    struct timespec        i_atime;
    struct timespec        i_mtime;
    struct timespec        i_ctime;
```

VFS – struct inode (II)

```
unsigned int                i_blkbits;
blkcnt_t                    i_blocks;
unsigned short              i_bytes;
umode_t                     i_mode;
spinlock_t                  i_lock;
struct mutex                i_mutex;
struct rw_semaphore         i_alloc_sem;
const struct inode_operations    *i_op;
const struct file_operations     *i_fop;
struct super_block          *i_sb;
struct file_lock            *i_flock;
struct address_space        *i_mapping;
struct address_space        i_data;
#ifdef CONFIG_QUOTA
    struct dquot            *i_dquot[MAXQUOTAS];
#endif
struct list_head            i_devices;
```

VFS – struct inode (III)

```
union {
    struct pipe_inode_info   *i_pipe;
    struct block_device      *i_bdev;
    struct cdev              *i_cdev;
};
int                          i_cindex;
__u32                        i_generation;
#ifdef CONFIG_DNOTIFY
    unsigned long            i_dnotify_mask;
    struct dnotify_struct    *i_dnotify;
#endif
#ifdef CONFIG_INOTIFY
    struct list_head         inotify_watches;
    struct mutex             inotify_mutex;
#endif
```

VFS – struct inode (IV)

```
unsigned long          i_state;
unsigned long          dirtied_when;
unsigned int           i_flags;
atomic_t               i_writecount;
#ifdef CONFIG_SECURITY
    void               *i_security;
#endif
void                   *i_private;
};
```

VFS – Inode - Campos

i_hash: lista de inodos con el mismo valor de hash.
i_list: lista global de inodos.
i_ino: número de inode (único).
i_count: número de usos del inode.
i_mode: flags.
i_nlink: número de enlaces a este inode.
i_uid: identificador de usuario propietario.
i_gid: identificador de grupo propietario.
i_size: tamaño del fichero asociado.
i_time: fecha del último acceso.
i_mtime: fecha de última modificación.
i_ctime: fecha de creación.
i_blkbits: tamaño del bloque en bits
i_blksize: tamaño del bloque
i_blocks: número de bloques que ocupa el inode.

VFS – Inode – Campos (II)

i_lock: cerrojo para asegurar exclusión mutua.
i_op: puntero a estructura de operaciones de inode.
i_fop: puntero a estructura de operaciones de ficher.
i_sb: puntero al superbloque.
i_mapping: puntero a las áreas y páginas de memoria de este inode.
i_data
i_dquot[MAXQUOTAS]: cuotas de disco.
i_pipe: puntero a estructura pipe si el inode representa un pipe.
i_cdev: puntero a block_device si el inode representa un dispositivo de modo bloque.
dirtied_when: fecha de la primera modificación.
i_writecount: número de accesos de escritura.

VFS – Inode - address_space

Contiene las VMA y las páginas del inode

```
struct address_space {
struct inode                *host
struct radix_tree_root      page_tree;
spinlock_t                  tree_lock;
unsigned int                i_mmap_writable;
struct prio_tree_root       i_mmap;
struct list_head            i_mmap_nonlinear;
spinlock_t                  i_mmap_lock;
unsigned int                truncate_count;
unsigned long               nrpages;
pgoff_t                     writeback_index;
const struct address_space_operations      *a_ops;
```

VFS – Inode - address_space

```
unsigned long              flags;
struct backing_dev_info   *backing_dev_info;
spinlock_t                 private_lock;
struct address_space      *assoc_mapping;

} __attribute__((aligned(sizeof(long))));
```

VFS – Operaciones de Inode(I)

- Cada sistema de ficheros tiene sus propios métodos para manejar inodes.
- Las operaciones que utiliza VFS para manejarlos, no son mas que punteros a la función del manejador del filesystem correspondiente.

```
struct inode_operations {
int (*create) (struct inode *,struct dentry *,int, struct nameidata *);
struct dentry * (*lookup) (struct inode *,struct dentry *, struct
                        nameidata *);
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 *,int);
int (*rmdir) (struct inode *,struct dentry *);
int (*mknod) (struct inode *,struct dentry *,int,dev_t);
```

VFS – Operaciones de inode(II)

```
int (*rename) (struct inode *, struct dentry *, struct inode *, struct dentry
    *);
int (*readlink) (struct dentry *, char __user *,int);
void * (*follow_link) (struct dentry *, struct nameidata *);
void (*put_link) (struct dentry *, struct nameidata *, void *);
void (*truncate) (struct inode *);
int (*permission) (struct inode *, int);
int (*setattr) (struct dentry *, struct iattr *);
int (*getattr) (struct vfsmount *mnt, struct dentry *, struct kstat *);
int (*setxattr) (struct dentry *, const char *,const void *,size_t,int);
ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
```

VFS – Operaciones de inode(III)

```
ssize_t (*listxattr) (struct dentry *, char *, size_t);
int (*removexattr) (struct dentry *, const char *);
void (*truncate_range)(struct inode *, loff_t, loff_t);
long (*fallocate)(struct inode *inode, int mode, loff_t offset, loff_t len);
int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                              u64 len);
};
```

VFS - Caché de directorios

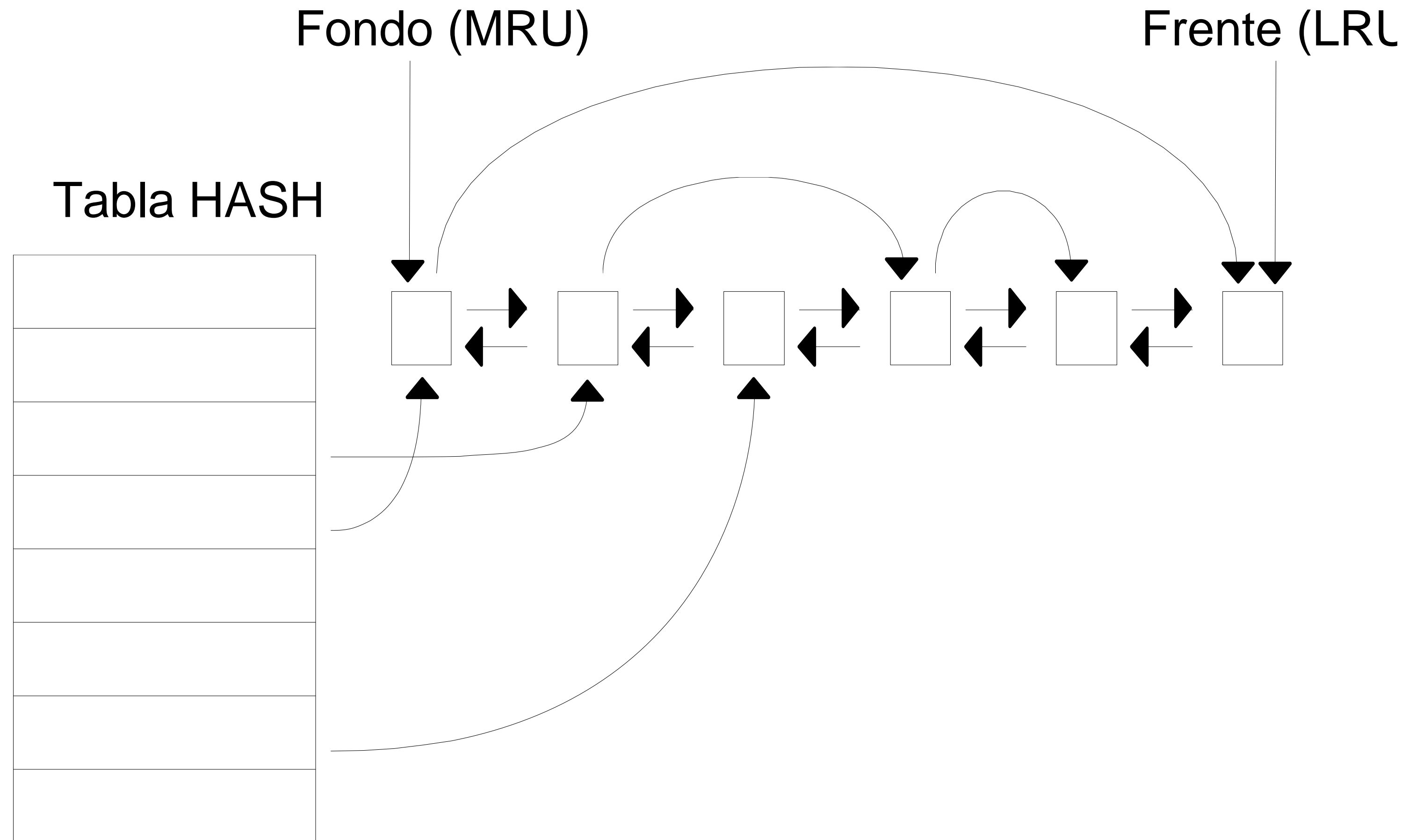
- Para agilizar las búsquedas VFS mantiene dos tablas caché de directorio, con las entradas de directorio más utilizadas y sus inodos.

  - Cada caché se estructura como una tabla hash con varias listas.

  - Cada entrada de la tabla apunta a una lista que tiene el mismo valor hash.

  - La función hash utiliza el número de dispositivo del sistema de ficheros, y el nombre del directorio.

VFS - Caché de directorios



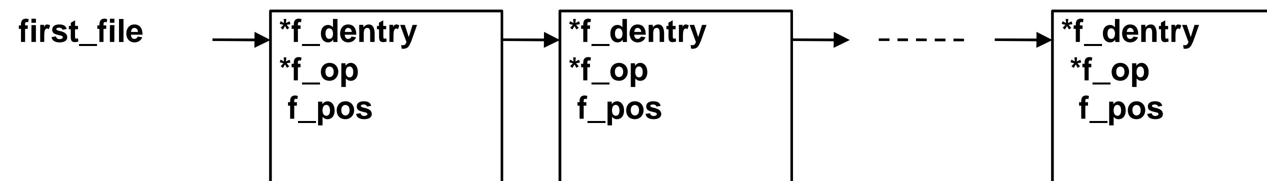
VFS – Buffer Cache

- Para minimizar los accesos a disco, VFS implementa un conjunto de buffers, con estructura “buffer_head” en memoria principal.
- Mantiene copias de bloques de disco individuales. Las entradas del caché están identificadas por el dispositivo y número de bloque
- El buffer es compartido por todos los sistemas de ficheros reales montados en el sistema.
- Es independiente de las particularidades físicas de los discos.
- Los buffers se mantienen en una de varias listas enlazadas: sin usar (unused), libres (free), no modificadas (clean), modificadas (dirty), bloqueadas (locked), etc.
- Periódicamente, el proceso “update” llama a la primitiva “sync()” para forzar la reescritura de todos los buffers modificados con esto el sistema experimenta una notable reducción de las E/S a disco.

VFS - Estructura File

- Cuando un proceso de usuario abre un fichero, VFS crea un objeto file compuesto de una estructura file. Esta estructura no tiene imagen en disco, por tanto, no se incluye dirty bit.

- Todas ellas se encadenan en una lista apuntada por first_file



VFS - Estructura File

```
struct file {
union {
    struct list_head        fu_list;
    struct rcu_head         fu_rcuhead;
} f_u;
struct path                         f_path;
#define f_dentry                    f_path.dentry
#define f_vfsmnt                    f_path.mnt
const struct file_operations   *f_op;
atomic_long_t                       f_count;
unsigned int                        f_flags;
fmode_t                             f_mode;
loff_t                              f_pos;
struct fown_struct                  f_owner;
const struct cred                   *f_cred;
```

VFS - Estructura File

```
struct file_ra_state        f_ra;
u64                         f_version;
#ifdef CONFIG_SECURITY
    void                    *f_security;
#endif
void                        *private_data;
#ifdef CONFIG_EPOLL
    struct list_head        f_ep_links;
    spinlock_t              f_ep_lock;
#endif
struct address_space        *f_mapping;
#ifdef CONFIG_DEBUG_WRITECOUNT
    unsigned long           f_mnt_write_state;
#endif
};
```

VFS – 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 (*aio_read) (struct kiocb *, const struct iovec *, unsigned long,
    loff_t);
ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long,
    loff_t);
int (*readdir) (struct file *, void *, filldir_t);
unsigned int (*poll) (struct file *, struct poll_table_struct *);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
```

VFS – file_operations

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 *, struct dentry *, int datasync);

int (*aio_fsync) (struct kiocb *, 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);

VFS – file_operations

```
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 **);
};
```

VFS-Estructura Dentry

- Consideramos cada directorio como un fichero que contiene una lista de ficheros y otros directorios.
- Cuando leemos una entrada de directorio, el kernel crea un objeto dentry para cada componente de su inode.
- Dentry no tiene imagen en disco

- Ejemplo: Cuando buscamos en la ruta /tmp/test, el kernel crea un objeto dentry para el directorio / , un segundo objeto para la entrada tmp del directorio raíz y un tercer objeto dentry para la entrada test del directorio /tmp/
- Definido en dcache.h