Taller de Sistemas Operativos

Virtual File System

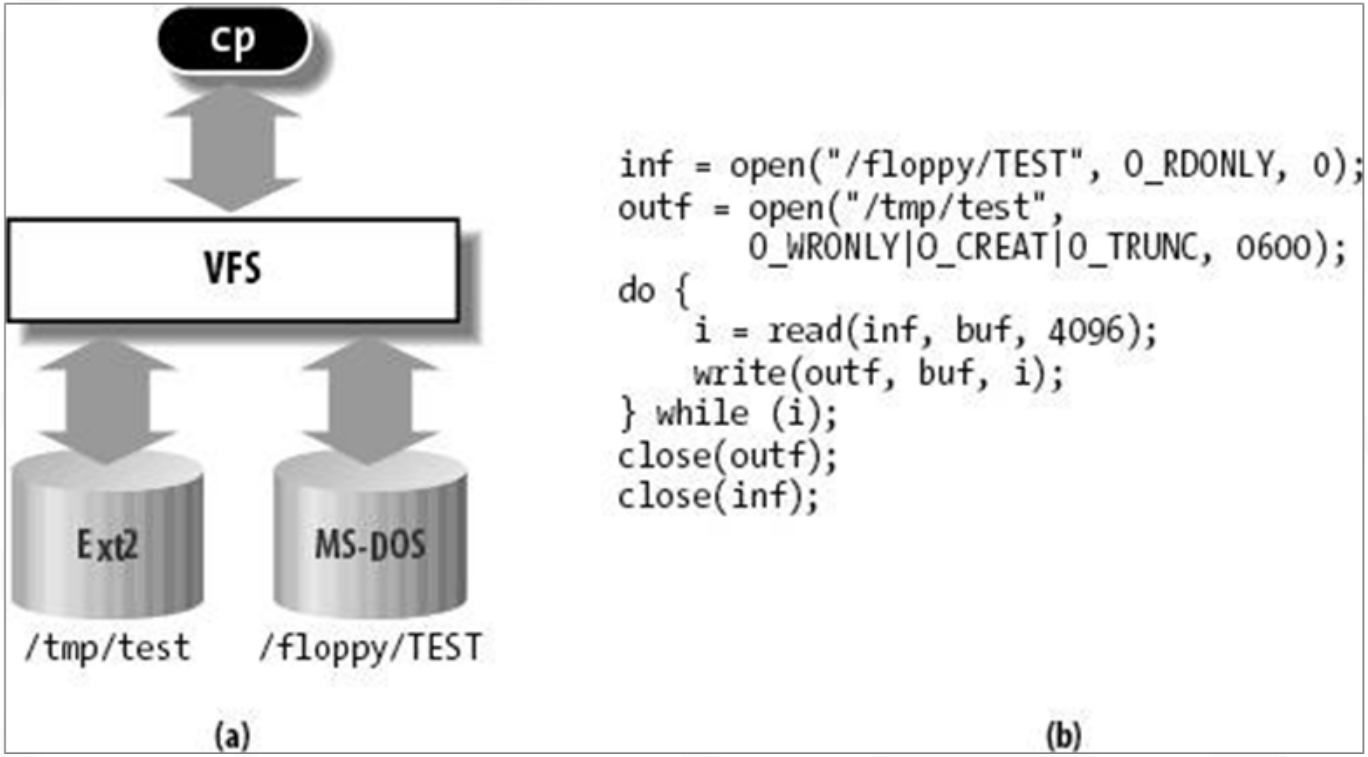
Conceptos generales

- Linux provee de una capa de software a nivel de núcleo para el acceso al sistema de archivos.
- La capa es denominada *Virtual File System* (VFS).
- Todas las llamadas a sistema que requieren acceso al sistema de archivos son resueltas a través de la capa VFS.
- La capa de abstracción define una interfaz conceptual básica y las estructuras de datos necesarias para el sistema de archivos.

Conceptos generales

- El sistema de archivos debe amoldar sus conceptos para implementar los requerimientos que impone la capa VFS.
- VFS está pensada para sistemas de archivo “tipo unix” pero se puede adaptar a otros como FAT o NTFS
- Para el resto del sistema operativo la capa VFS brinda una vista uniforme del sistema de archivos, independientemente del sistema de archivo que este accediendo.
- Ej.:
  - Una llamada al system call write se traduce en la invocación a la función vfs_write, que posteriormente invoca a la función write del sistema de archivos específico.

Conceptos generales



Conceptos generales

```
asmlinkage ssize_t sys_write(unsigned int fd, const char
__user * buf, size_t count)
{
        struct file *file;
        ssize_t ret = -EBADF;
        int fput_needed;

        file = fget_light(fd, &fput_needed);
        if (file) {
                loff_t pos = file_pos_read(file);
                ret = vfs_write(file, buf, count, &pos);
                file_pos_write(file, pos);
                fput_light(file, fput_needed);
        }

        return ret;
}
```

Conceptos generales

```
ssize_t vfs_write(struct file *file, const char __user
   *buf, size_t count, loff_t *pos)
{
    ssize_t ret;
   //Controles de acceso
   …
   if (file->f_op->write)
      ret = file->f_op->write(file, buf, count,pos);
   else
      ret = do_sync_write(file, buf, count, pos);
   if (ret > 0) {
      fsnotify_modify(file->f_dentry);
      current->wchar += ret;
   }
   current->syscw++;
   return ret;
}
```

Objetos del VFS

- Se proveen cuatro tipos de objetos en VFS:
  - **superblock**: Representa un sistema de archivos montado.
  - **inode**: Representa a un archivo del sistema de archivos.
  - **dentry**: Representa una entrada de directorio. Un componente simple del camino. Puede ser un archivo o un directorio.
  - **file**: Representa un archivo abierto asociado con un proceso.
- VFS es orientado a objetos

El objeto superbloque

- El superbloque describe el sistema de archivos específico.
- En general, cada sistema de archivos contiene un bloque (sector) que lo describe.
- VFS propone una estructura superbloque (struct super_block) que contiene campos que deben ser llenados por el sistema de archivos específico (aunque no contenga una estructura del tipo superbloque).
- Los campos definen tamaños máximos, estado del sistema de archivos, contadores, etc.
- La estructura es definida en el archivo linux/fs.h
- El campo s_op es un puntero a la tabla de operaciones asociada al superbloque.
- El campo s_fs_info es un puntero a información específica del sistema de archivos.

El objeto superbloque

```
struct super_block {
 struct list_head        s_list;         /* list of all superblocks*/
 dev_t                   s_dev;           /* identifier */
 unsigned long           s_blocksize;     /* block size in bytes */
 unsigned char           s_blocksize_bits;/* block size in bits*/
 loff_t                  s_maxbytes;      /* max file size */
 struct file_system_type s_type;          /* filesystem type */
 struct super_operations *s_op;           /* superblock methods*/
 struct dquot_operations *dq_op;          /* quota methods */
 struct quotactl_ops     *s_qcop;         /* quota control methods */
 unsigned long           s_flags;         /* mount flags */
 struct dentry           *s_root;         /* directory mount point */
 struct rw_semaphore     s_umount;        /* unmount semaphore */
 int                     s_count;         /* superblock ref count */
}
```

No es el objeto completo

Operaciones del superbloque

- Las funciones del superbloque operan sobre el sistema de archivos y los inodos.
- Cuando es necesario aplicar una operación sobre el sistema de archivos se aplica sobre el campo s_op

```
sb->s_op->put_super(sb)
```

- Debido a que C no es un lenguaje orientado a objetos, es necesario pasar como parámetro el superbloque.

Operaciones del superbloque

```
struct super_operations {
  struct inode *(*alloc_inode) (struct super_block *sb);
  /* crea un inodo */
  void (*destroy_inode) (struct inode *);
  /* borra un inodo de la memoria */
  void (*dirty_inode) (struct inode *);
  /* se invoca cuando se cambia un inodo */
  void (*write_inode) (struct inode *, int);
  /* escribe el inodo en el disco */
  void (*drop_inode) (struct inode *);
  /* se invoca cuando no hay más referencias al inodo */
  void (*delete_inode) (struct inode *);
  /* borra el inodo del disco */
  void (*write_super) (struct super_block *);
  /* escribe el superbloque en el disco */
  int (*sync_fs) (struct super_block *, int);
  /* sincroniza las estructuras de memoria con el disco*/
  int (*statfs) (struct super_block *, struct statfs *);
  /* da estadísticas del filesystem */
}
```

No son todas las operaciones

Operaciones del superbloque

Ej.: (ext2)

```
static struct super_operations ext2_sops = {
        .alloc_inode    = ext2_alloc_inode,
        .destroy_inode  = ext2_destroy_inode,
        .write_inode    = ext2_write_inode,
        .delete_inode   = ext2_delete_inode,
        .put_super      = ext2_put_super,
        .write_super    = ext2_write_super,
        .statfs         = ext2_statfs,
        .remount_fs     = ext2_remount,
        .clear_inode    = ext2_clear_inode,
        .show_options   = ext2_show_options,
#ifdef CONFIG_QUOTA
        .quota_read     = ext2_quota_read,
        .quota_write    = ext2_quota_write,
#endif
};
```

El objeto inodo

- El inodo representa los archivos y los directorios de los sistemas de archivos.
- La estructura que lo representa es la struct inode, que contiene un conjunto de campos de información, tamaños, estado, tipo, etc.
- La estructura se construye en el momento de acceder a un archivo
- Algunos campos son dependientes del tipo de archivo al cual se hace referencia el inodo (pipe, block device, char device)
- La estructura es definida en el archivo linux/fs.h
- Está en alguna de las siguientes listas
  - Inutilizado
  - Utilizado
  - Sucio
- Se guardan también en un hash

El objeto inodo

```
struct inode {
 struct hlist_node        i_hash;              /* hash node */
 struct list_head         i_lru;               /* lru list of inodes */
 struct list_head         i_sb_list;       /* list of inodes for superblock */
 unsigned long            i_ino;               /* inode number */
 atomic_t                 i_count;             /* reference counter */
 umode_t                  i_mode;              /* access permissions */
 unsigned int             i_nlink;             /* number of hard links */
 kuid_t                   i_uid;               /* user id of owner */
 kgid_t                   i_gid;               /* group id of owner */
 loff_t                   i_size;              /* file size in bytes */
 struct timespec          i_atime;             /* last access time */
 struct timespec          i_mtime;             /* last modify time */
 struct timespec          i_ctime;             /* last change time */
 unsigned int             i_blkbits;           /* block size in bits */
 blkcnt_t                 i_blocks;            /* file size in blocks */
 struct super_block       *i_sb;               /* associated superblock */
 struct inode_operations *i_op;                /* inode ops table */
 union {
  struct pipe_inode_info *i_pipe;              /* pipe information */
  struct block_device    *i_bdev;              /* block device driver */
  struct cdev            *i_cdev;              /* character device driver */
};
```

No es la estructura completa

Operaciones sobre inodo

```
struct inode_operations {
 int (*create) (struct inode *, struct dentry *,int);
 /* crea un nuevo inodo */
 struct dentry * (*lookup) (struct inode *, struct dentry *);
 /* busca un inodo (directorio o archivo) en un directorio */
 int (*link) (struct dentry *, struct inode *, struct dentry *);
 /* agrega un hard link de un inodo en un directorio */
 int (*unlink) (struct inode *, struct dentry *);
 /* borra un inodo de un directorio */
 int (*symlink) (struct inode *, struct dentry *, const char *);
 /* agrega un link simbólico de un inodo en un directorio */
 int (*mkdir) (struct inode *, struct dentry *, int);
 /* crea un directorio */
 int (*rmdir) (struct inode *, struct dentry *);
 /* borra un directorio */
 int (*rename) (struct inode *, struct dentry *,
                struct inode *, struct dentry *);
 /* mueve un archivo */
 int (*follow_link) (struct dentry *, struct nameidata *);
 /* traduce un link simbólico */
 int (*permission) (struct inode *, int);
 /* indica si se tiene permiso para acceder a este inodo */
}
```

No son todas las operaciones

El objeto dentry

- El objeto dentry representa las entradas de un directorio.
- Esta estructura permite mantener en memoria la estructura del sistema de archivos.
- No corresponde a una estructura en disco, se crean para facilitar la navegación por el *filesystem*
- Es utilizado, por ejemplo, para la resolución de caminos.
- Ej.:
  - /mnt/cdrom/foo
  - Se compone de cuatro estructuras dentry: ‘/’ ‘mnt’ ‘cdrom’ y ‘foo’.
  - Cada una tendrá un puntero al inodo correspondiente.

El objeto dentry

```
struct dentry {
 struct lockref    d_lock_ref   /* per-dentry lock and refcount*/
 struct inode             *d_inode;  /* associated inode */
 struct list_head         d_lru;     /* LRU list */
 struct list_head         d_child;   /* list of dentries within*/
 struct list_head         d_subdirs; /* subdirectories */
 struct list_head         d_alias;   /* list of alias inodes */
 struct dentry_operations *d_op;     /* dentry operations table*/
 struct super_block       *d_sb;     /* superblock of file */
 unsigned int             d_flags;   /* dentry flags */
 struct dentry            *d_parent; /* dentry object of parent*/
 struct qstr              d_name;    /* dentry name */
 struct hlist_bl_node     d_hash;    /* lookup hash list */
 unsigned char        d_iname[DNAME_INLINE_LEN]; /* short name */
}
```

No es la estructura completa

Operaciones sobre dentry

```
struct dentry_operations {
 int (*d_revalidate) (struct dentry *, int);
 /* indica si el dentry es todavía válido */
 int (*d_hash) (struct dentry *, struct qstr *);
 /* retorna un valor de hash para el dentry */
 int (*d_compare) (struct dentry *, struct qstr
  *, struct qstr *);
 /* compara dos dentry (es necesario para fs no
  case sensitive) */
 int (*d_delete) (struct dentry *);
 /* se invoca cuando el dentry no está en uso */
}
```

No son todas las operaciones

Estado de dentry

- Los objetos *dentry* pueden estar en 4 estados:
  - **Libre**: El *dentry* no contiene información válida y no está siendo utilizado por el VFS
  - **Usado**: El *dentry* tiene un inodo asignado que es válido y está siendo usado (d_lockref.count > 0).
  - **Sin uso**: El *dentry* tiene un inodo asignado que es válido, pero no está siendo usado en el momento (d_lockref.count = 0).
  - **Negativo o inválido**: El *dentry* no está asociado a ningún inodo (d_inode = NULL). Inodo inexistente (borrado) o ruta a archivo invalido
- Existe una cache de *dentry* que conserva los objetos que fueron utilizados.
  - Esto permite no estar continuamente creando y destruyendo estructuras dentry.
  - A su vez, permite que el VFS no tenga que “caminar” por todo el sistema de archivos cada vez que se accede a un archivo (se usa un hash).

Estado dentry

- El cache consiste de 3 partes
  - Lista de entradas en uso
  - Lista LRU de entradas sin uso e inválidas
  - Tabla de hash con todos los dentry
- El valor en la tabla de hash es dado por d_hash y se accede a esta tabla desde la función d_lookup
- Sirve guardar las entradas por si se vuelve a acceder a las mismas
- El cache es útil porque:
  - Los procesos acceden varias veces a los mismos archivos
  - Los procesos acceden a varios archivos en el mismo directorio
- Se usa el *slab allocator* para crear todas las estructuras del VFS

El objeto file

- El objeto file es utilizado para representar los archivos abiertos de los procesos.
- La estructura es creada una vez que un proceso invoca al *system call* open y es destruida cuando se invoca a close.
- Cada proceso del sistema tiene una tabla de archivos abiertos.
- Para cada entrada válida se tendrá un puntero a una estructura file.
- Se mantiene un puntero al dentry correspondiente.
- Se dispone de un conjunto de operaciones que se aplican al archivo abierto (read, write, lseek, mmap, etc.).

El objeto file

```
struct file {
 struct path         *f_path; /* associated dentry */
 struct file_operations
                     *f_op;   /* file operations table */
 atomic_long_t       f_count; /* file object's usage count*/
 unsigned int        f_flags; /* flags specified on open */
 fmode_t             f_mode;  /* file access mode */
 loff_t              f_pos;   /* file offset(file pointer)*/
 struct fown_struct f_owner; /* owner data for signals */
 const struct cred  *f_cred; /* file credentials */
 spinlock_t          f_lock   /* file struct lock */
}
```

No es la estructura completa

Operaciones sobre file

```
struct file_operations {
 loff_t (*llseek) (struct file *, loff_t, int);
 /* cambia de posición dentro del archivo */
 ssize_t (*read) (struct file *, char *, size_t, loff_t
  *);
 /* lee algunos bytes del archivo a un buffer */
 ssize_t (*write) (struct file *, const char *, size_t,
  loff_t *);
 /* escribe en el archivo */
 unsigned int (*poll) (struct file *, struct
  poll_table_struct *);
 /* avisa cuando el archivo cambió */
 int (*mmap) (struct file *, struct vm_area_struct *);
 /* mapea en memoria una parte del archivo */
 int (*open) (struct inode *, struct file *);
 /* crea un nuevo objeto file mapeado al inodo */
 int (*fsync) (struct file *, struct dentry *, int);
 /* sincroniza los datos en cache con el disco */
 int (*flock) (struct file *filp, int cmd, struct
  file_lock *fl);
 /* bloquea el archivo */
}
```

No son todas las operaciones

Directorio actual

- Linux mantiene una estructura que se mantiene actualizada con el directorio corriente de trabajo de cada proceso.
- Esta estructura es apuntada desde cada PCB (fs).
- Es compartida por los *threads*.

```
struct fs_struct {
  int             users;        /* user count */
  spinlock_t      lock;         /* lock protecting structure */
  seqcount_t      seq;          /* para sincronizar sin lock */
  int             umask;        /* default file permissions*/
  int             in_exec       /* currently executing a file */
  struct path     root;         /* root directory */
  struct path     pwd;          /* current working directory */
}
```

Archivos abiertos

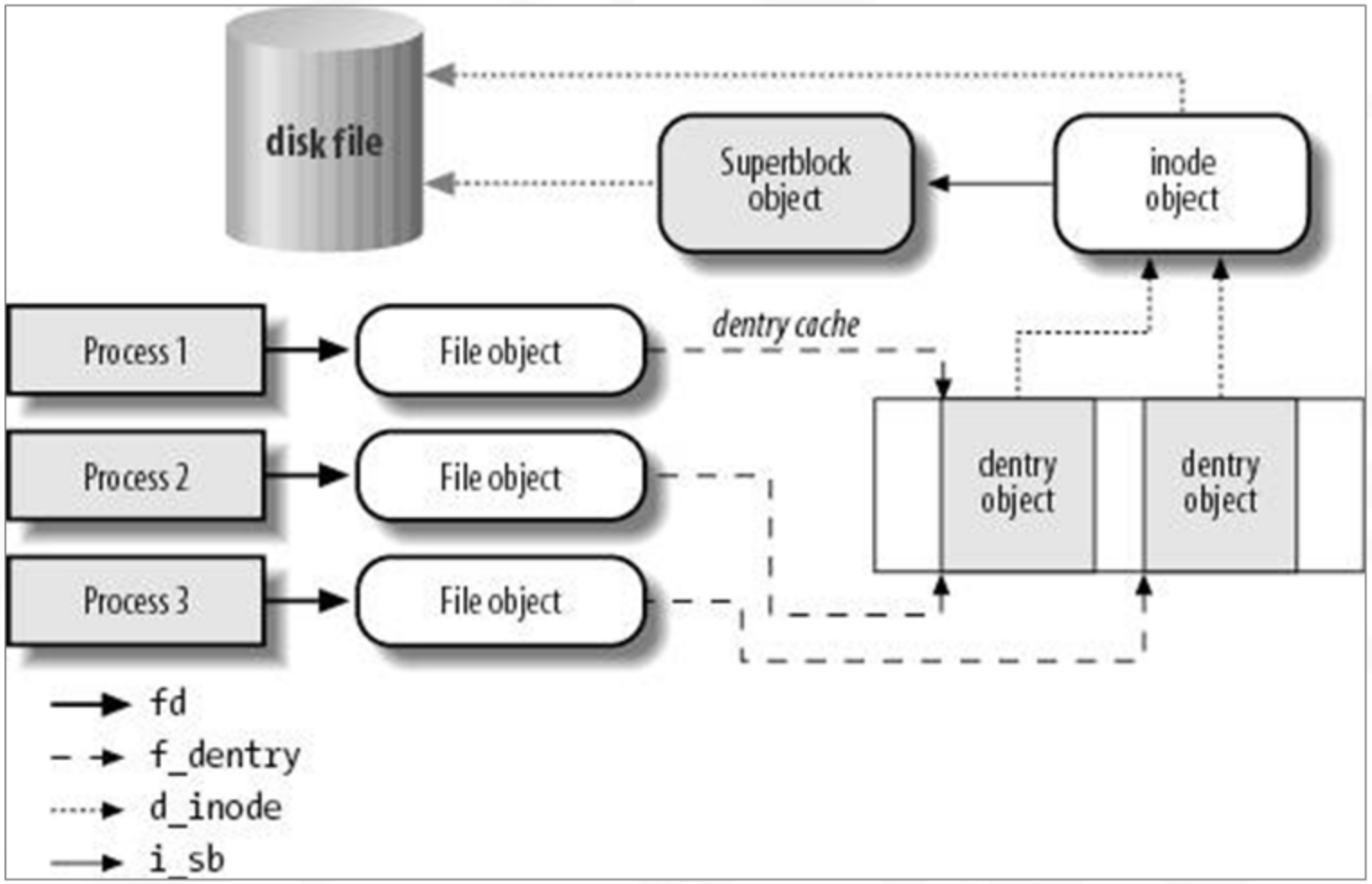
- Cada proceso del sistema tiene una tabla con los archivos abiertos.
- Campo files del PCB
- También es compartida por los threads

```
struct files_struct {
 atomic_t        count;      /* structure's usage count */
 struct fdtable *fdt;        /* pointer to other fd table */
 spinlock_t      file_lock;  /* per-file lock */
 int             next_fd;    /* cache of next fd number */
 struct file    *fd_array[NR_OPEN_DEFAULT];
                            /* default array of file objects */
}
```

No es la estructura completa

Objetos del VFS

- La interacción entre procesos y los objetos VFS:



Estructura de datos asociadas

- Para cada filesystem se tiene una estructura file_system_type que:
  - Describe el sistema de archivos.
  - Contiene el nombre, puntero a las operaciones de montado y desmontado, etc.
- Para cada filesystem montado se tiene una estructura vfs_mount que:
  - Representa una instancia del sistema de archivos a nivel del VFS.
  - Contiene punteros a la entrada dentry raíz del sistema de archivos, al vfs_mount padre, lista de hijos, dispositivo, etc.

file_system_type

```
struct file_system_type {
  const char        *name;       /* filesystem's name */
  int               fs_flags;    /* filesystem type flags */

  /* the following is used to read the superblock off the disk */
  struct dentry     *(*mount) (struct file_system_type *,
                        int, char *, void *);

  /* the following is used to terminate access to the superblock */
  void              (*kill_sb) (struct super_block *);

  struct module     *owner;      /* module owning the filesystem */
  struct file_system_type *next; /* next file_system_type */
  struct hlist_head fs_supers;   /* list of superblock objects */
}
```

No es la estructura completa

vfsmount

```
struct vfsmount {
 struct list_head   mnt_hash;        /* hash table list */
 struct vfsmount    *mnt_parent;     /* parent filesystem */
 struct dentry      *mnt_mountpoint; /* dentry of this mount point */
 struct dentry      *mnt_root;       /* dentry of root of this fs */
 struct super_block *mnt_sb;         /* superblock of this filesystem*/
 atomic_t           mnt_count;       /* usage count */
 int                mnt_flags;       /* mount flags */
 char               *mnt_devname;    /* device file name */
 struct namespace   *mnt_namespace   /* associated namespace */
}
```

- También existe el objeto namespace que permite definir un a vista distinta del filesystem para un proceso
- Por defecto todos los procesos comparten el mismo namespace

Objetos del VFS

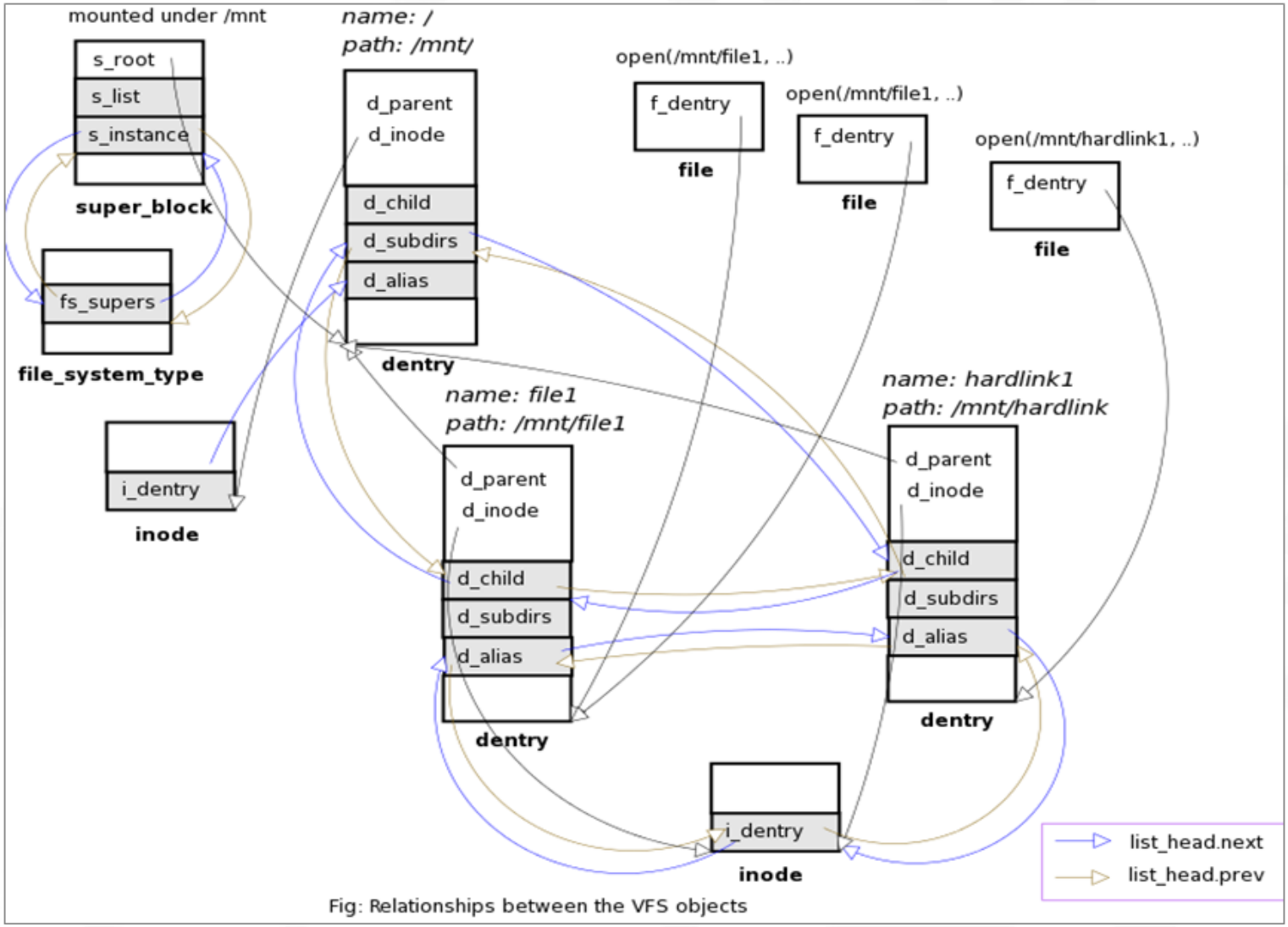


Fig: Relationships between the VFS objects