MO806I – Tópicos em Sistemas Operacionais
Universidade Estadual de Campinas – Instituto de Computação
Profa. Islene Calciolari Garcia – 2o. Sem/2009

Virtual Filesystem (VFS)

Bruno Azevedo
bpires@ic.unicamp.br

Wilson Akio Higashino
wilson.higashino@terra.com.br

Agenda

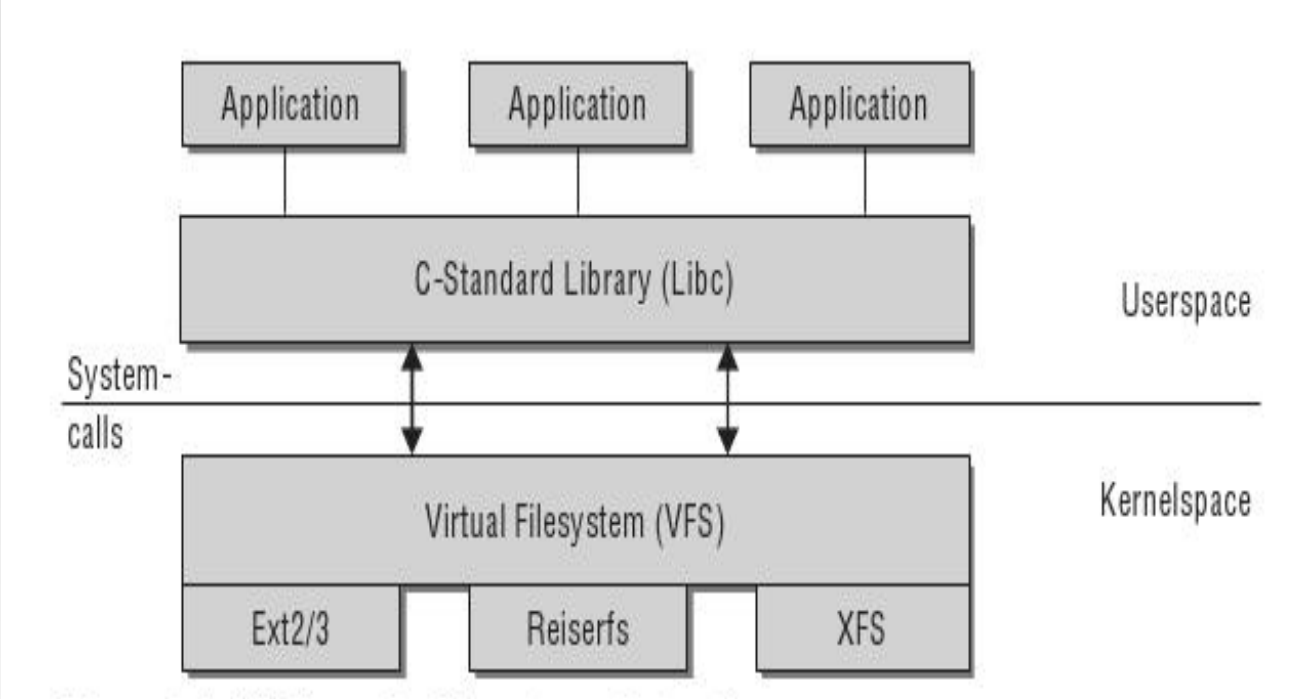
- Introdução
- Common File Model
  - Superblock
  - Inode
  - Informações do Processo
  - Dentry
- Operações com Filesystem
- Operações com arquivos
- Demo – Alteração no Kernel

Introdução

- Linux
  - Mais de 50 sistemas de arquivos
  - Transparente
- Como?

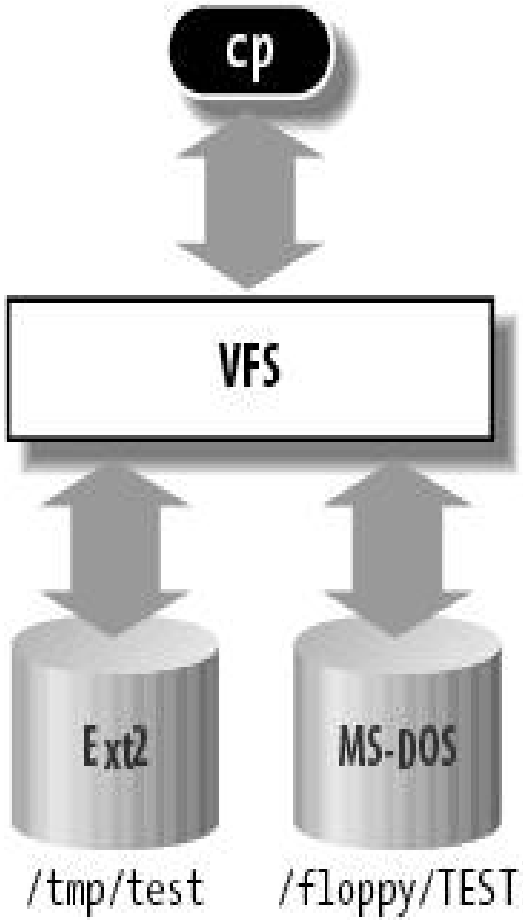
Virtual Filesystem

Camada de abstração entre as aplicações e os sistemas de arquivos usados



- Aplicações não interagem com os FS
- Conjunto comum de funções

Virtual Filesystem



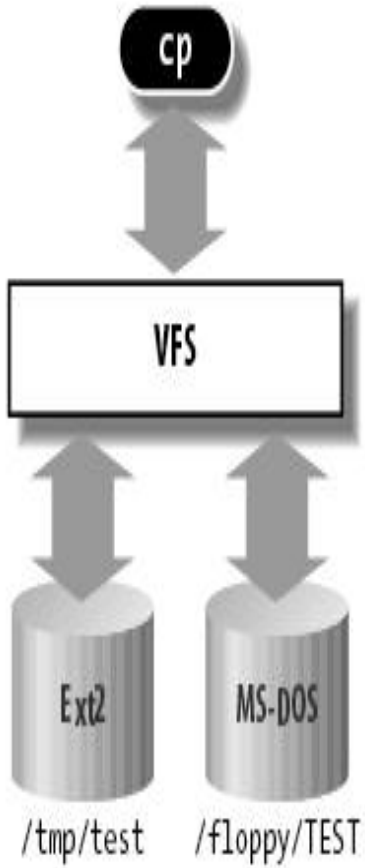

```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

Modelo Comum de Arquivos

- Introduz um modelo comum de arquivos
- Tudo é um arquivo
- Cada FS deve traduzir sua organização para o MCA
  - Exemplo: MCA (tudo é um arquivo) vs FAT

Modelo Comum de Arquivos

1. read()
2. read() é traduzida em sys_read()
3. O arquivo é representado por uma estrutura com o campo f_op
4. sysread() encontra o ponteiro para a funcao
5. file->f_op->read(...);



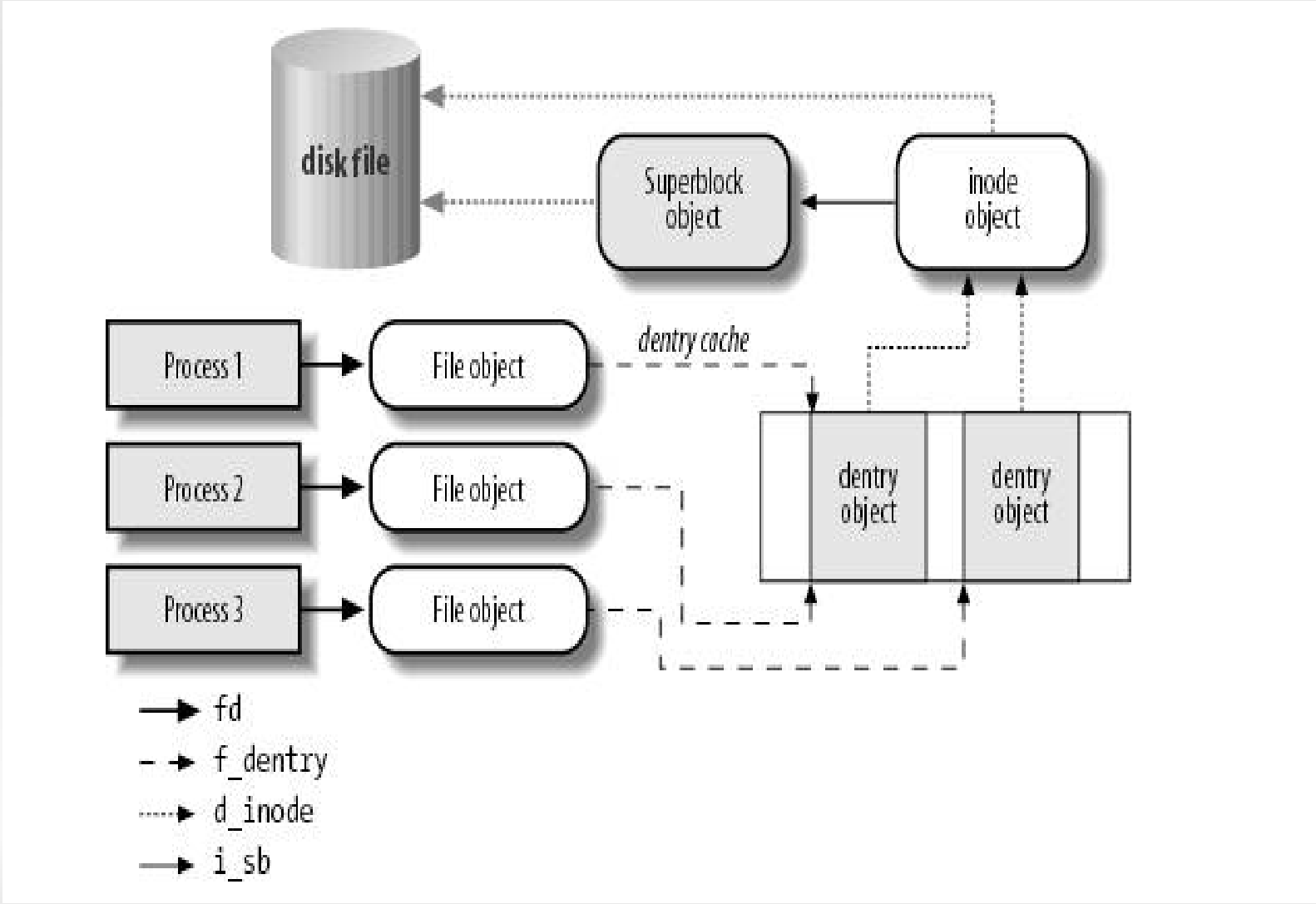

```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

Modelo Comum de Arquivos

- Orientado a objeto?
- Perspectiva do kernel: conjunto comum de objetos:
  - Superblock – informação de um FS montado
  - Inode – todo objeto manipulado pelo FS (arquivo ou diretório)
  - Dentry – associa nome com Inode
  - File – informações da interação entre um arquivo e um processo (arquivo aberto)

Modelo Comum de Arquivos

- Como se relacionam?



Superblock

- Informações chave sobre o FS (“contêiner de metadados”)
- Existe em disco e em memória
  - Em disco: informações sobre a estrutura do FS
  - Em memória: informações e estado do FS
- Operações como alocar, deletar inodes...
- Ponteiro para o dentry do root, para lista de inodes...
- /linux/include/fs.h

Superblock

```
struct super_block {
    struct list_head                    s_list;
    unsigned long                       s_blocksize;
    unsigned char                       s_blocksize_bits;
    struct file_system_type             *s_type;
    const struct super_operations       *s_op;
    unsigned long                       s_flags;
    struct dentry                       *s_root;
    struct rw_semaphore                 s_umount;
    struct list_head                    s_inodes;
    struct list_head                    s_dirty;
    struct list_head                    s_io;
    struct list_head                    s_more_io;
    struct list_head                    s_files;
    char                                s_id[32];
    void                                *s_fs_info;
    fmode_t                             s_mode;
    struct mutex            s_vfs_rename_mutex;
    ...
};
```

```
struct super_operations {
    struct inode*(*alloc_inode)(struct super_block *sb);
    void (*destroy_inode)(struct inode *);
    int (*write_inode) (struct inode *, int);
    void (*delete_inode) (struct inode *);
    void (*write_super) (struct super_block *);
    int (*sync_fs)(struct super_block *sb, int wait);
    int (*statfs) (struct dentry *, struct kstatfs *);
    int (*remount_fs)(struct super_block *, int *, char *);
    void (*umount_begin) (struct super_block *);
    int(*show_options)(struct seq_file *,struct vfsmount *);
    ...
};
```

Inode

- Diretórios como estruturas de dados? “Tipo especial de arquivo”.
- Representa um objeto com um identificador único
- Elementos do Inode
  - Metadados para descrever status (e.g. Permissão de acesso, data da última edição
  - Ponteiro para os dados

Inode

- Cache e tabela hash
- Operações inode como unlink, create...
- Operações file como open, read, write...
- /linux/include/fs.h

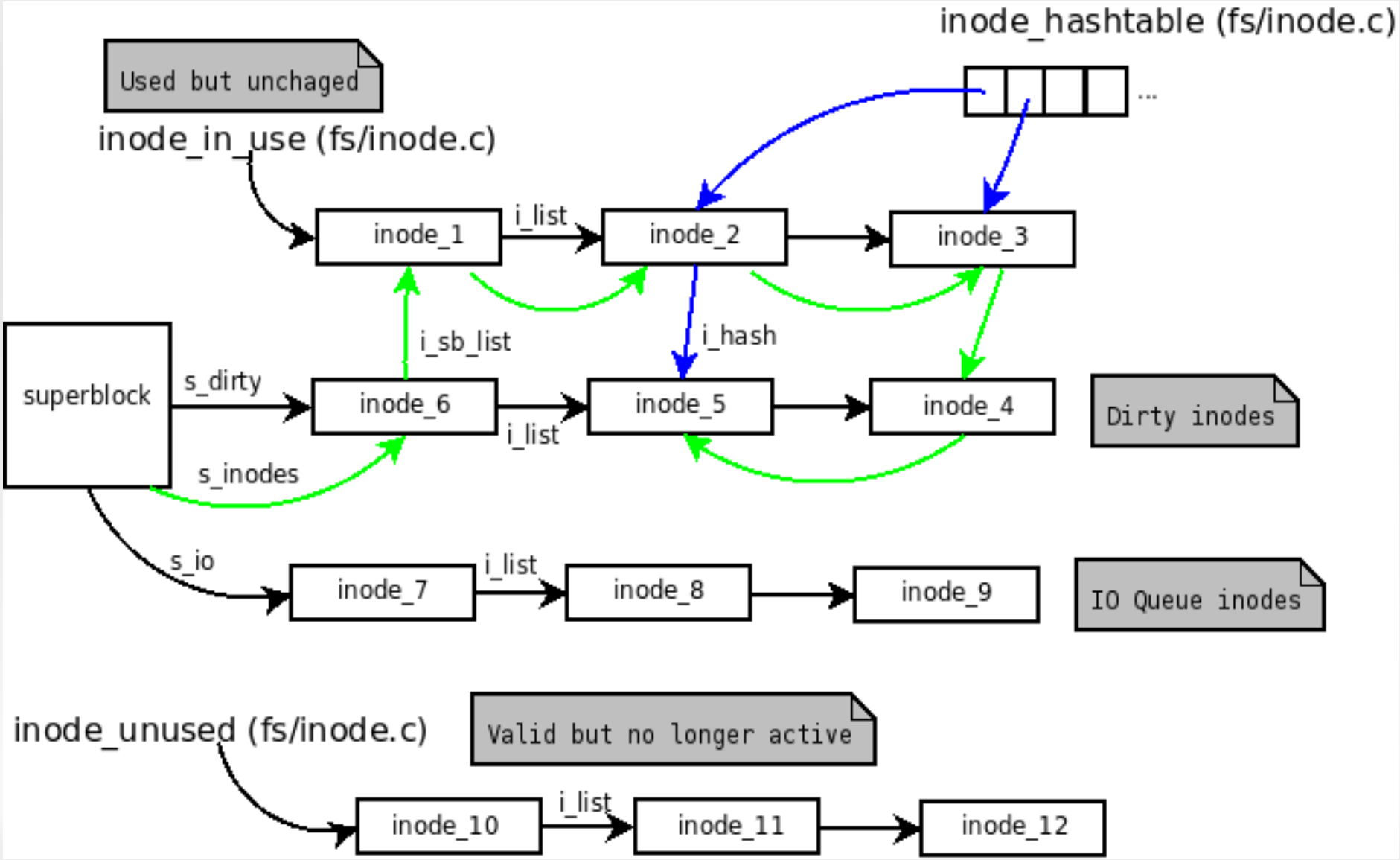
Inode

```
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;
    unsigned int                    i_nlink;
    uid_t                           i_uid;
    gid_t                           i_gid;
    loff_t                          i_size;
    struct timespec                 i_atime;
    struct timespec                 i_mtime;
    umode_t                         i_mode;
    struct mutex                    i_mutex;
    struct rw_semaphore             i_alloc_sem;
    const struct file_operations    *i_fop;
    const struct inode_operations   *i_op;
    struct super_block              *i_sb;
    unsigned int                i_flags;
    ...
};
```

```
struct file_operations {
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    ...
};
```

```
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 (*rename) (struct inode *, struct dentry *,
            struct inode *, struct dentry *);
    ...
};
```

Inode


inode_hashtable (fs/inode.c)
Used but unchaged
inode_in_use (fs/inode.c)
inode_1
i_list
inode_2
inode_3
i_sb_list
i_hash
superblock
s_dirty
inode_6
inode_5
inode_4
Dirty inodes
i_list
s_inodes
s_io
inode_7
i_list
inode_8
inode_9
IO Queue inodes
inode_unused (fs/inode.c)
Valid but no longer active
i_list
inode_10
inode_11
inode_12

Dentry

- Mapeia nome de arquivo para um inode
- root dentry (superblock) aponta para si mesmo, todos os outros tem pais e podem ter filhos.
  - /home/user/nome -> quatro dentry são criados
- Mapeia a natureza hierárquica de sistemas de arquivos
- dentry hash e cache – preenchido ao ler dados de diretórios e arquivos
- /linux/include/dcache.h

Dentry

- Cada dentry pode estar em um de quatro estados:
  - Free: não contém informações válidas.
  - Unused: no momento sem uso pelo kernel. Possui informações válidas. Pode ser descartado para liberar memória. d_count == 0 e d_inode aponta para o inode associado.
  - In use: usado pelo kernel no momento. d_count é positivo e d_inode aponta para o inode associado.
  - Negative: inode associado não existe. Mantido no cache para que operações de lookup possam ser resolvidas rapidamente.

Dentry

```
struct dentry {
    atomic_t                        d_count;
    spinlock_t                      d_lock;
    struct inode                    *d_inode;
    struct dentry                   *d_parent;
    struct dentry                   *d_child;
    struct qstr                     d_name;
    struct list_head                d_subdirs;
    struct list_head                d_alias;
    const struct dentry_operations *d_op;
    struct super_block              *d_sb;
    void                            *d_fsdata;
    ...
};
```

```
struct dentry_operations {
    int (*d_revalidate)(struct dentry *, struct nameidata *);
    int (*d_hash) (struct dentry *, struct qstr *);
    int (*d_compare) (struct dentry *,struct qstr *, struct qstr *);
    int (*d_delete)(struct dentry *);
    void (*d_release)(struct dentry *);
    void (*d_iput)(struct dentry *, struct inode *);
    char *(*d_dname)(struct dentry *, char *, int);
};
```

File

- Para cada arquivo aberto, existe um objeto File
- Contém informação específica da instância aberta
- Referencia o dentry e operações do objeto file

File

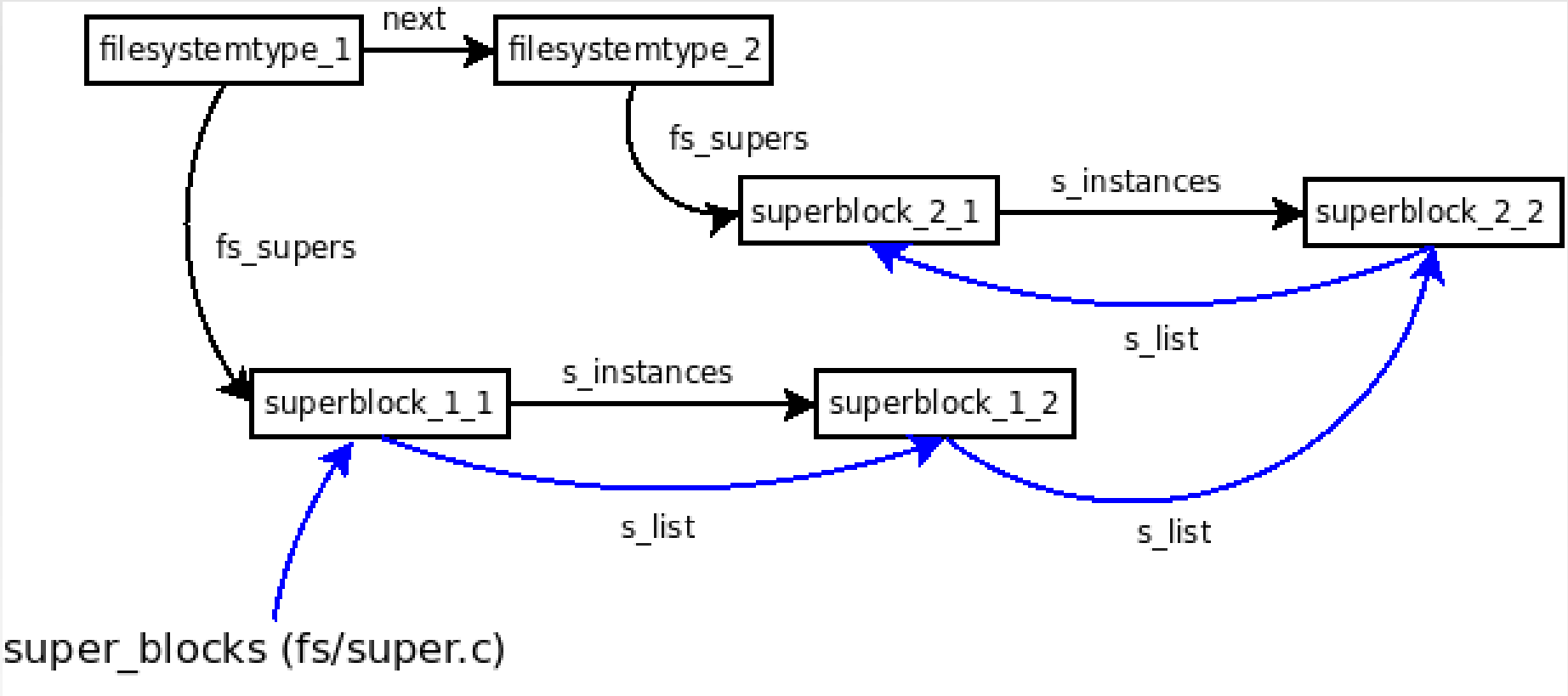
```
struct file {
    const struct file_operations    *f_op;
    spinlock_t                      f_lock;
    atomic_long_t                   f_count;
    unsigned int                    f_flags;
    fmode_t                         f_mode;
    loff_t                          f_pos;
    struct fown_struct              f_owner;
    const struct cred               *f_cred;
    struct file_ra_state            f_ra;
    u64                             f_version;
    struct address_space    *f_mapping;
    ...
};
```

```
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);
    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 (*lock) (struct file *, int, struct file_lock *);
    ...
};
```

Operações com Filesystem

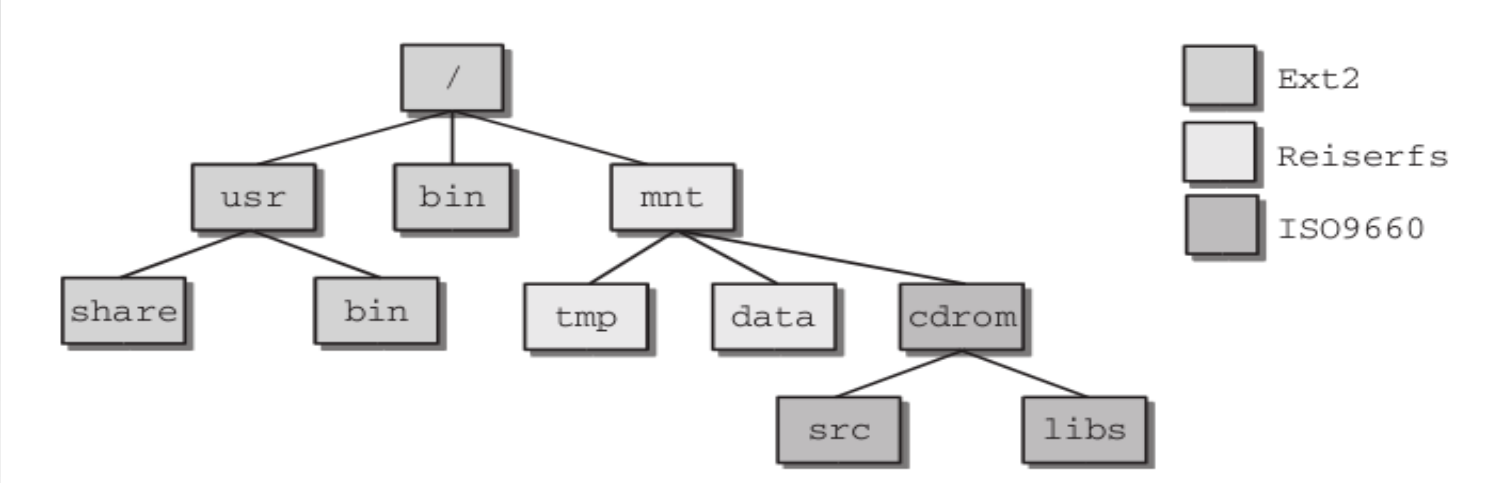
- Registro de Filesystem
  - Arquitetura modular
  - FS se registra no Kernel
  - Função **register_filesystem** em *fs/super.c*
    - FS são mantidos em lista ligada de estruturas **file_system_type**
      - Insere nesta lista
    - Estrutura **file_system_type**
      - Lista de todos superblocks deste tipo (*fs_supers*)
        - Um para cada ponto de montagem
      - Apontador para funções que obtem um superblock (*get_sb*)

Operações com Filesystem



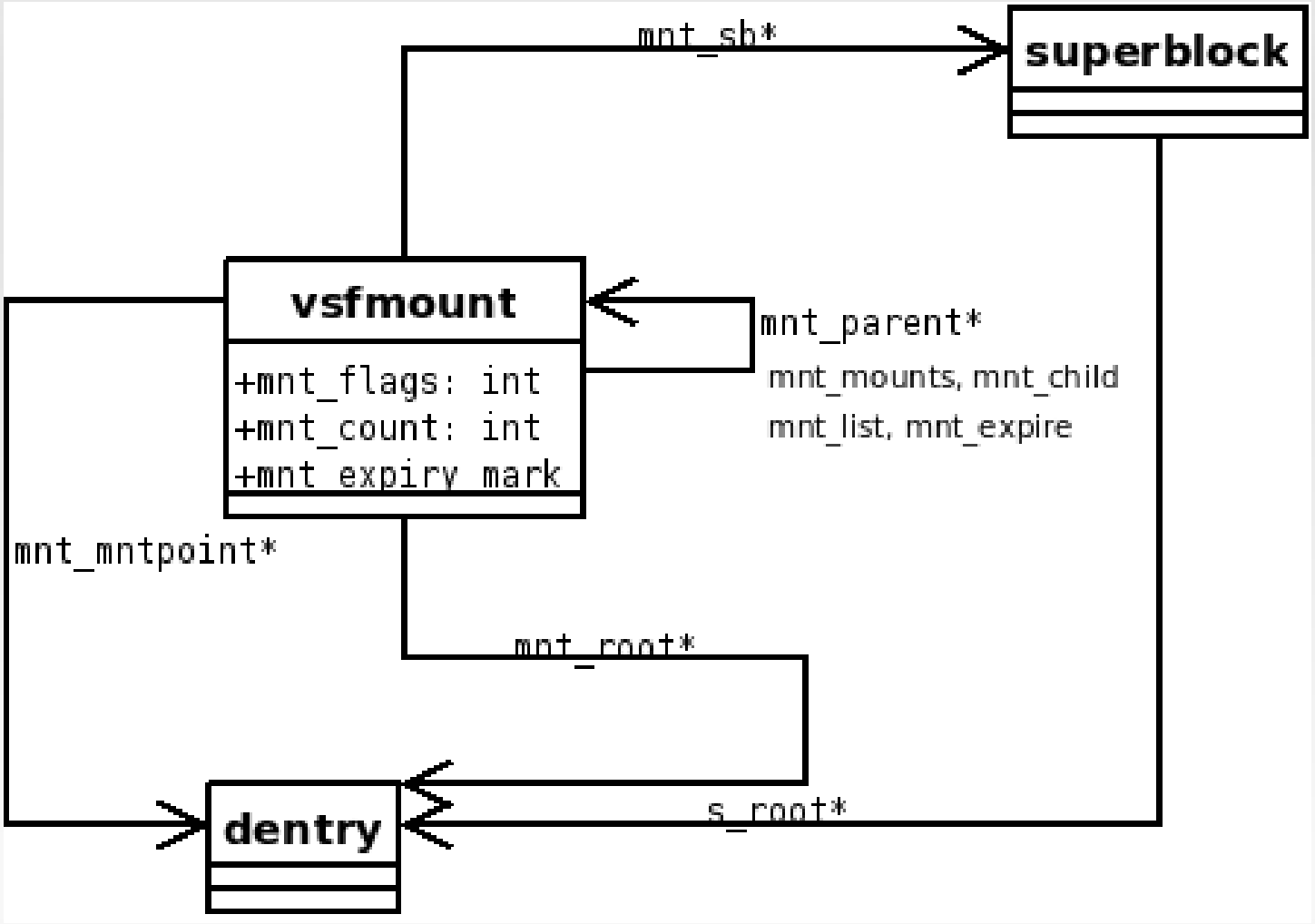
Operações com Filesystem

- Montagem
  - Operação complexa

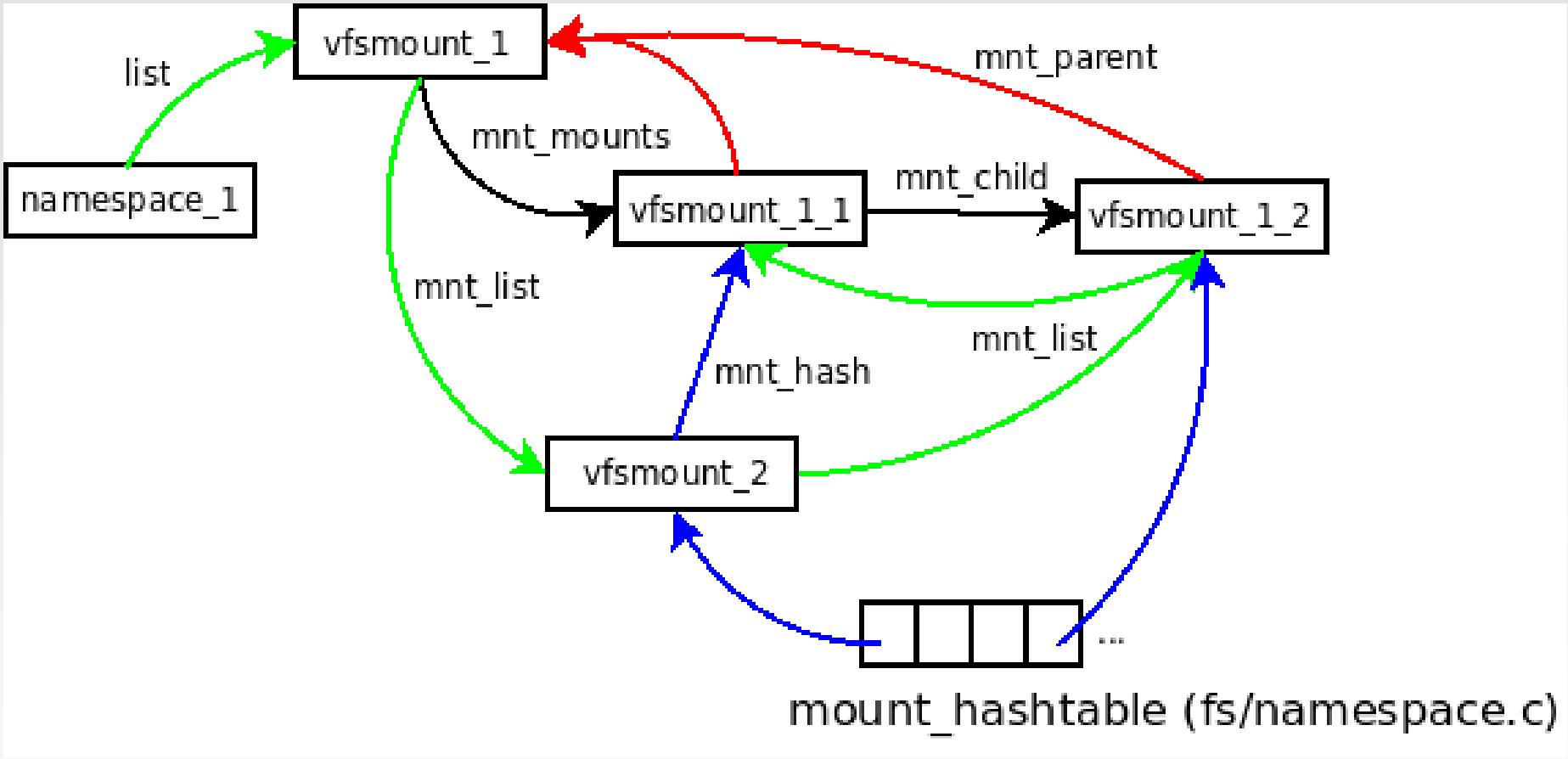


  - /mnt e /mnt/cdrom são mount points
    - Conteúdo substituído com o diretório raiz relativo do novo FS
  - Mountpoints podem ser aninhados (hierárquico)

Operações com Filesystem



Operações com Filesystem



Operações com Filesystem

- Montagem
  - System call **mount**
  - Ponto de entrada: **sys_mount** (*fs/namespace.c*)
    - Copia dados para espaço do Kernel
    - Invoca **do_mount**
      - Encontra *dentry* do ponto de montagem (**path_lookup**)
      - Multiplexador
        - **do_remount**
        - **do_move_mount**
        - **do_new_mount**

Operações com Filesystem

- **do_new_mount**
  - **do_kern_mount**
    - Obtém *file_system_type* correto
    - Lê *superblock* (função *get_sb()*)
  - **do_add_mount**
    - Adiciona na lista de mounts do *namespace*
    - Ajusta estruturas *vfsmount*
      - mnt_parent/mnt_mounts, mnt_mountpoint
    - Adiciona na hash table global

Operações com Filesystem

- Desmontagem
  - System call **umount**
  - Ponto de entrada: **sys_umount** (*fs/namespace.c*)
    - Encontra *vfsmount* e *dentry* do ponto de montagem (**path_lookup**)
    - **do_umount**
      - Decrementa **d_mount** do *dentry* **mnt_mountpoint**
      - Remove *vfsmount* das listas
      - Restaura estado original
        - Utiliza *mnt_parent* e *mnt_mountpoint*

Operações com Filesystem

- Expiração automática
  - Desmonta quando mount não está em uso
    - AFS / NFS
    - Montagens são mantidos em lista
      - **vfsmount->mnt_expire**
  - Peridiocamente executa **mark_mounts_for_expiry**
    - Percorre lista
    - Marca **mnt_expiry_mark** se não está em uso
    - Remove na próxima passagem

Operações com Arquivos

- Encontrando um inode pelo nome (**lookup**)
  - Usa estrutura **nameidata**

```
struct nameidata {
    struct dentry           *dentry;
    struct vfsmount         *mnt;
    struct qstr             last;
    ...
};
```

  - **dentry** e **mnt** armazenam resultados
  - **last** é o nome procurado

```
int fastcall path_lookup(const char *name, unsigned int flags,
        struct nameidata *nd);
```

Operações com Arquivos

- *nameidata* é usado como área de trabalho
- Inicializado com diretório root ou corrente
- Função **link_path_walk**
  - Processa pathname componente por componente em um loop
  - Complexa
    - Verifica ciclos (links simbólicos)
    - Detecta pontos de montagens
    - Verifica permissões de acesso
    - Trata “.” e “..”

Operações com Arquivos

- Verifica permissão
  - Método **permission** em *inode_operations* ou **exec_lite**
- Path é varrido até encontrar "/"
  - Cada componente é processado em uma iteração
  - Se for ".", vai para próxima iteração
  - Se for ".."
    - Se diretório atual não é mount point, usa **d_parent** do *dentry* atual
    - CC, utiliza campos *mnt_mountpoint* e *mnt_parent* do *vfsmount* atual para definir novos *dentry* e *vfsmount*
  - Invoca **do_lookup**

Operações com Arquivos

- **do_lookup**
  - Recebe *dentry* atual e nome procurado
  - Procura *inode* no cache de dentries
  - Se não encontrar, executa **real_lookup**
    - Aloca estruturas de dados
    - Invoca função **lookup** de *inode_operations*
    - Verifica se diretório é um mount point
      - Extrai *vfsmount* da *mount_hashtable*
      - *mnt_root* é usado como novo *dentry*

Operações com Arquivos

- Abertura de arquivo
  - Função **open**
    - Retorna file descriptor
    - Índice em array de arquivos abertos de um processo
      - **task_struct->files->fd_array**
  - **sys_open** (*fs/open.c*)
    - Encontra posição livre no array
    - Encontra *Inode* (*path_lookup*)

Operações com Arquivos

- **sys_open** (*fs/open.c*)
  - Cria nova estrutura *file*
  - Adiciona na lista *s_files* do *superblock*
  - Invoca **file_operations->open**
    - Específico do filesystem
  - Adiciona nas estruturas do processo

Operações com Arquivos

- Leitura de arquivo
  - Função **read** (*fs/read_write.c*)
    - Recebe file descriptor, buffer e número de bytes
    - Obtém *file* associado ao descriptor
    - Lê **file->f_pos**
    - Invoca **file->f_op->read**
      - Específico do filesystem
      - Se não tiver, invoca **do_sync_read**
    - Atualiza **file->f_pos**

Operações com Arquivos

- Escrita de arquivo
  - Função **write** (*fs/read_write.c*)
    - Análoga ao read
    - Mesmos argumentos
    - Invoca **file->f_op->write** ou **do_sync_write**

Referências

- **Access the Linux Kernel using the /proc filesystem**. M. Tim Jones. http://www.ibm.com/developerworks/library/l-proc.html
- **Linux Kernel Procfs Guide.** Erik Mouew. http://www.kernel.org/doc/htmldocs/procfs-guide.html
- **Procfs from the inside.** Fernando Apesteguia. http://www.linuxforums.org/articles/procfs-from-the-inside_86.html
- **Professional Linux Kernel Architecture**. Wolfgang Mauerer, Wiley Publishing, 2008.
- **QEMU.** http://www.qemu.org.
- **Understanding the Linux Kernel, 3rd. Edition**. Daniel P. Bovet, Marco Cesati. O'Reilly, 2005.