# MC504/MC514 - Sistemas Operacionais Sistemas de Arquivos (Parte III)

Islene Calciolari Garcia

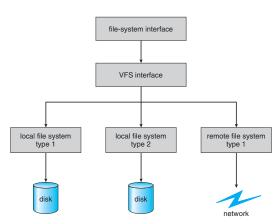
Instituto de Computação - Unicamp

Segundo Semestre de 2016

### Sumário

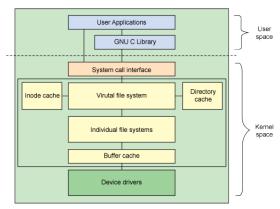
- Virtual File System
- 2 ext2
- 3 ext3
- 4 ext4
- S NFS
- **6** FUSE

## Virtual File System



Silberschatz: Figura 12.04

## Arquitetura em camadas



M. Tim Jones

## Sistemas de arquivos implementados

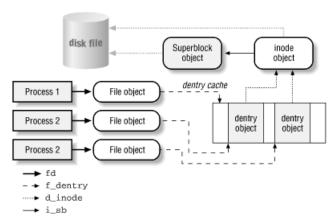
```
file systems ----
                      struct file_system_type {
                        const char *name:
                        int fs flags;
                        struct super block *get sb;
                        void (*kill sb);
                        struct module *owner;
                        struct file_system_type *next;
                        struct list head fs supers;
                      struct file system type {
                        const char *name;
                        int fs flags;
                        struct super block *get sb;
                        void (*kill sb);
                        struct module *owner;
                        struct file system type *next;
                        struct list head fs supers;
```

M. Tim Jones

cat /proc/filesystems

- sistemas de arquivos podem ser adicionados e removidos dinamicamente
- $supported \neq mounted$

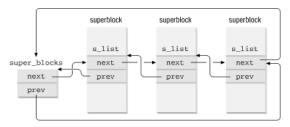
## Objetos do VFS e interações com os processos



Understanding the Linux Kernel: Figura 12-2

## Superblock

- Metadados sobre os sistemas de arquivos
- Estrutura existente em disco e em memória

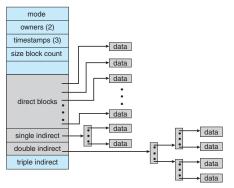


Understanding the Linux Kernel: Figura 12-2

## Estruturas do 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;
                                                        struct super_operations {
                                                            struct inode*(*alloc_inode)(struct super_block *sb);
   unsigned long
                                     s_flags;
                                                            void (*destroy inode)(struct inode *);
  struct dentry
                                     *s root:
                                                            int (*write inode) (struct inode *, int);
   struct rw semaphore
                                     s umount:
                                                            void (*delete inode) (struct inode *):
   struct list head
                                     s inodes;
                                                            void (*write super) (struct super block *);
  struct list head
                                     s dirty:
                                                            int (*sync fs)(struct super block *sb, int wait);
  struct list head
                                     s io;
                                                            int (*statfs) (struct dentry *, struct kstatfs *);
   struct list head
                                     s more io:
                                                            int (*remount fs)(struct super block *, int *, char *);
  struct list head
                                     s files;
                                                            void (*umount_begin) (struct super_block *);
  char
                                     s id[32]:
                                                            int(*show_options)(struct seq_file *,struct vfsmount *);
  void
                                     *s fs info;
  fmode t
                                     s mode:
  struct mutex
                         s vfs rename mutex;
```

#### I-node



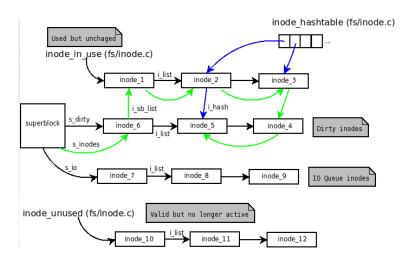
Silberschatz: Figura 12.09

- metadados de um arquivo ou diretório
- uma estrutura em disco e outra em memória

#### Estruturas do inode

```
struct inode {
  struct hlist node
                                       i hash;
                                                              struct file_operations {
                                       i list;
  struct list head
                                                                ssize t (*read) (struct file *, char user *, size t, loff t *);
  struct list head
                                       i sb list;
                                                                ssize t (*write) (struct file *, const char user *, size t, loff t *);
  struct list head
                                        i dentry;
                                                                int (*open) (struct inode *, struct file *);
  unsigned long
                                       i ino;
                                                                int (*flush) (struct file *, fl owner t id);
  unsigned int
                                       i nlink;
  uid t
                                       i uid;
                                                              };
                                        i gid;
  gid t
  loff t
                                       i size:
  struct timespec
                                       i atime:
  struct timespec
                                       i mtime:
                                                            struct inode operations {
  umode t
                                       i mode:
                                                              int (*create) (struct inode *,struct dentry *,int, struct nameidata *);
  struct mutex
                                       i mutex:
                                                              struct dentry *(*lookup) (struct inode *,struct dentry *, struct nameidata*);
  struct rw semaphore
                                       i alloc sem:
                                                              int (*link) (struct dentry *,struct inode *,struct dentry *);
  const struct file operations
                                        *i fop:
                                                              int (*unlink) (struct inode *,struct dentry *);
  const struct inode operations
                                        *i_op;
                                                              int (*symlink) (struct inode *,struct dentry *,const char *);
  struct super_block
                                        *i sb:
                                                              int (*mkdir) (struct inode *,struct dentry *,int);
  unsigned int
                                 i flags:
                                                              int (*rmdir) (struct inode *,struct dentry *);
                                                              int (*rename) (struct inode *, struct dentry *,
};
                                                                   struct inode *, struct dentry *);
```

#### Inode cache



### Dentry

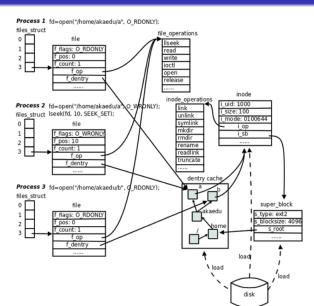
- Estrutura em memória
- Mapeamento de nomes e i-nodes
- Estados de um dentry no cache
  - Free: sem uso e sem informações válidas.
  - Unused: contém informações válidas.
  - In use: em uso e com informações válidas.
  - Negative: corresponde a um caminho inválido.

### Estrutura do file object

- Estrutura em memória
- Arquivo aberto

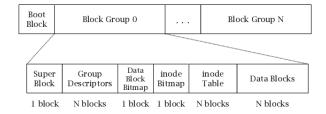
```
struct file {
  const struct file operations
                                  *f op:
  spinlock t
                                  f lock:
  atomic_long_t
                                  f count:
  unsigned int
                                  f flags:
  fmode t
                                  f mode:
  loff t
                                  f pos:
  struct fown struct
                                  f owner:
  const struct cred
                                  *f cred:
                                                        struct file operations {
  struct file ra state
                                  f ra:
                                                           struct module *owner:
  1164
                                  f version:
                                                           loff t (*llseek) (struct file *, loff t, int);
  struct address space
                           *f mapping:
                                                           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 *);
```

### File descriptor table



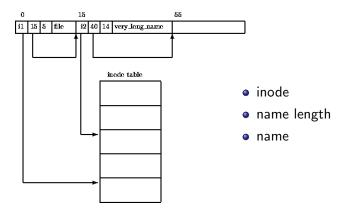


#### Estrutura do Ext2



- Super block: número de blocos no disco, tamanho dos blocos,
   ...
- Group descriptors: localização dos bitmaps inodes/blocos e da tabela de i-nodes, ..

#### Diretórios no Ext2



#### Ext2 e crashes

- Operações como criação e remoção envolvem várias escritas em disco
- Um crash pode ocorrer durante o processo e
  - um bloco foi incluído no inode, mas não está marcado como em uso no bitmap
  - um bloco foi removido do inode, mas n\u00e3o est\u00e1 marcado como livre no bitmap
  - um bloco foi reutilizado, mas o inode original ainda aponta para ele.
  - um inode foi marcado como em uso, mas não há entrada em diretório apontando para ele.
  - ...

#### fsck e recuperação Testando com debugfs

- Crie um sistema de arquivos ext2 com mkfs.ext2
- Utilize o debugfs para corromper este sistema
- Verifique se o fsck encontra os erros

## Criando e fazendo verificação inicial

```
$ dd if=/dev/zero of=ext2.dmp bs=1k count=256
$ mkfs.ext2 ext2.dmp
$ fsck ext2.dmp
e2fsck 1.42.13 (17-May-2015)
ext2.dmp: clean, 11/32 files, 23/256 blocks
$ fsck -f ext2.dmp
e2fsck 1.42.13 (17-May-2015)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
ext2.dmp: 11/32 files (0.0% non-contiguous), 23/256 blocks
```

### Alterando bitmap dos inodes

```
$ debugfs -w ext2.dmp
debugfs: write a.txt a.txt
debugfs: ls
2 (12) . 2 (12) .. 11 (20) lost+found 12 (980) a.
debugfs: freei <12>
```

Como o fsck consegue corrigir o erro?

#### lost+found

```
debugfs: write a.txt a.txt
debugfs: modify_inode <12>
        Link count [1] 2
debugfs: ls
2 (12) . 2 (12) .. 11 (20) lost+found 12 (980) a.txt
debugfs: rm a.txt
debugfs: quit
$ fsck.ext2 -f ext2.dmp
```

 Por que o inode <12> seria colocado no diretório de "achados e perdidos"?

### **Journaling**

- Anúncio das ações seguido de commit record
- redo log vs undo log
- ext3 adota redo log (delete?)
- teste mkfs.ext3 e veja o espaço reservado para o journal
- Veja o tamanho do log no debugfs com logdump

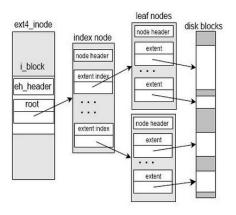
## Modos de journaling

- Journal: Dados + metadados no log
- Ordered
  - Apenas metadados
  - Metadados só podem ser gravados no log após os dados estarem atualizados
  - Possibilidade de falha em caso de sobrescrita de blocos de dados
- Writeback: Sem restrição para ordem de gravação dos dados e metadados

## Checkpointing

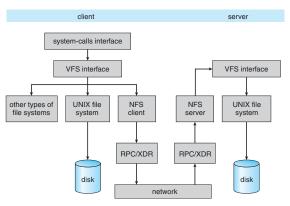
- Coleta de lixo das entradas no log
- Operações que já estão no disco podem ser descartadas
- Espaço pode ser reutilizado
- fsck poderia procurar inconsistências somente onde é necessário

#### ext4



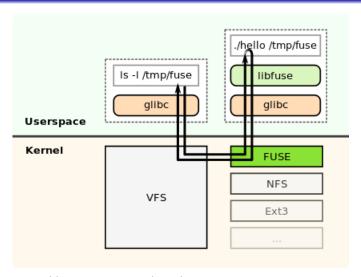
- journal
- arquivos maiores
- extents
- estrutura para diretórios: (HTree)

### **NFS**



Silberschatz: Figura 12.15

### **FUSE File System**



 $https://en.wikipedia.org/wiki/Filesystem\_in\_Userspace$ 



#### Referências adicionais

- Anatomy of the Linux virtual file system switch, M. Tim Jones
- Understanding the Linux Kernel, 3rd Edition.
- Bruno Azevedo e Wilson Higashino. Virtual File System.
   MO806 2s2009