

Fundamentos de Sistemas de Operação

*Unix Windows NT Netware Mac OS DOS/VMS Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus*

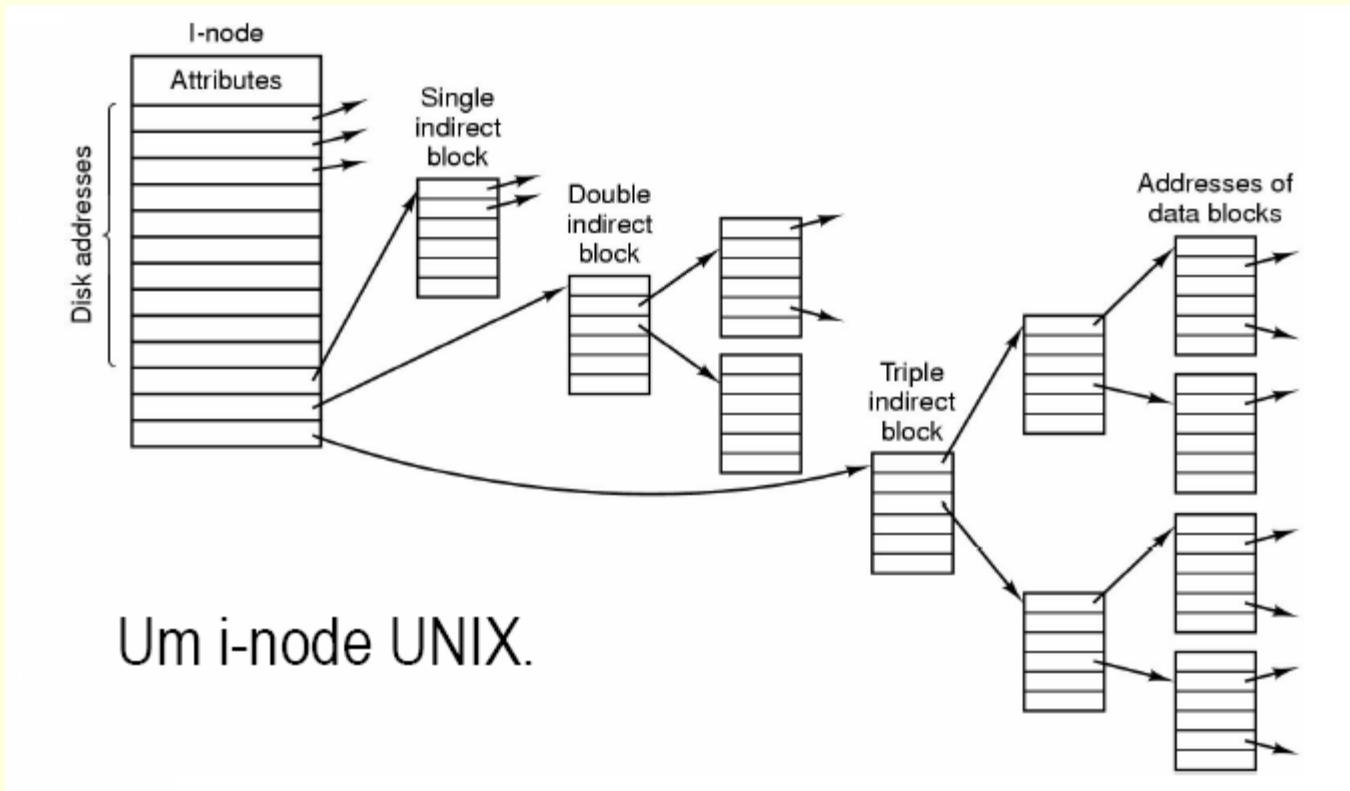
Gestão de Ficheiros

Estudo de casos –
ext2, um SF do Linux (estudo muito simplificado)

Um SF nativo: ext2

- O nome *ext2* tem a sua origem na palavra “extended”, versão 2
- Um SF *ext2* tem/descreve
 - Uma organização física dos dados num(a partição de um) disco formatado(a) em *ext2*
 - Uma forma como essa informação é colocada em memória (no EE do kernel) - ver /fs/ext2/ext2_fs.h
- As estruturas chave são:
 - Superbloco, Inode, File e Dentry (vem de directory entry)

i-node (1)



i-node (2)

- O nome *i-node* tem a sua origem em “index node”
 - É a estrutura de dados que guarda todas as informações sobre um ficheiro **excepto** o seu nome
 - Em particular, tem 15 apontadores que referenciam quais os blocos que pertencem a esse ficheiro:
 - Os 12 primeiros apontadores apontam directamente (cada um) um bloco. Ou seja, são suficientes para ficheiros de comprimento inferior a $12 \times 1024 = 12KB$ (se cada bloco de dados for um “cluster” de 2 blocos de disco – o ext2 suporta “clusters” de 1, 2 e 4KB)
 - Se o ficheiro for maior de 12KB, o 13º apontador aponta um bloco de índice que contém “n” apontadores – 128 se estivermos a usar blocos de índice de 512B e apontadores de 4B – e o ficheiro poderá crescer até $12K + 128 \times 1024 = 140KB$

(cont.)

i-node (3)

- Se o ficheiro for maior que 140 KB o 14º apontador aponta um bloco de índice de nível 2, que poderá apontar 128 blocos de índice de nível 1, cada um com 128 apontadores... $12K+(128+128^2)*1K \sim 16MB$
- Se ... o último apontador ... $12K+(128+128^2+128^3)*1K \sim 4GB$
- Outras informações guardadas no inode:
 - Utilizador dono do ficheiro (UID)
 - Permissões
 - Timestamps da criação, última modificação, último acesso
- Um inode ext2 tem 128 bytes (8 inodes em 1K)

i-node (4)

```
struct ext2_inode {  
    __le16  i_mode;                      /* File mode */  
    __le16  i_uid;                       /* Low 16 bits of Owner Uid */  
    __le32  i_size;                      /* Size in bytes */  
    __le32  i_atime;                     /* Access time */  
    __le32  i_ctime;                     /* Creation time */  
    __le32  i_mtime;                     /* Modification time */  
    __le32  i_dtime;                     /* Deletion Time */  
    __le16  i_gid;                       /* Low 16 bits of Group Id */  
    __le16  i_links_count;               /* Links count */  
    __le32  i_blocks;                    /* Blocks count */  
    __le32  i_flags;                     /* File flags */  
    ...  
    __le32  i_block[EXT2_N_BLOCKS];     /* Pointers to blocks */  
    ...  
}
```

i-node (5)

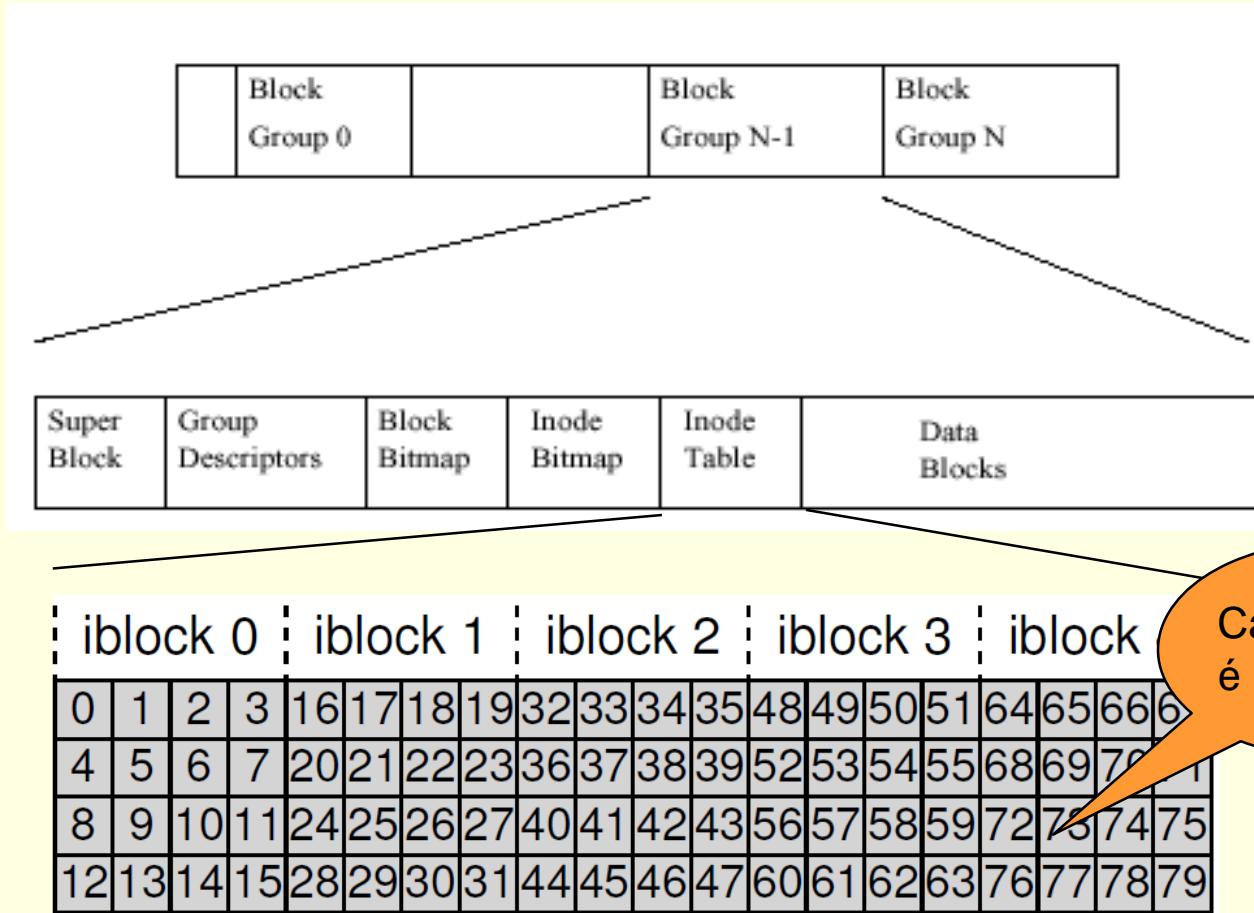
```
...
/*
 * Constants relative to the data blocks
 */
#define EXT2_NDIR_BLOCKS           12
#define EXT2_IND_BLOCK
#define EXT2_DIND_BLOCK
#define EXT2_TIND_BLOCK
#define EXT2_N_BLOCKS               (EXT2_IND_BLOCK + 1)
                           (EXT2_DIND_BLOCK + 1)
                           (EXT2_TIND_BLOCK + 1)

...
```

Desempenho

- Como qualquer bloco livre pode ser usado para armazenar dados de um ficheiro,
 - Os blocos de um ficheiro podem ficar muito afastados uns dos outros, e o acesso sequencial é muito prejudicado,
- O ext2 cria “mini-discos” (block groups-BG). Cada “mini-disco”
 - É formado por um conjunto de blocos contíguos
 - Tem um “block bitmap” (para gerir os blocos de dados) e um “inode bitmap” (para gerir os inodes)
 - Quando um ficheiro é criado, o seu inode “fica” num determinado BG, e o ext2 tenta que todos os blocos desse ficheiro fiquem nesse mesmo BG, ficando próximos uns dos outros...

Grupo de Blocos do Ext2



Sistema de ficheiros ext2

□ O *superbloco*

- *Descreve a estrutura do disco:*
 - quantos blocos ocupa o “group descriptor”
 - quantos blocos ocupa a “inode table”
 - quantos blocos de dados existem
- O “verdadeiro” *superbloco*...
 - é o do “block group 0; os outros são cópias, usados em caso de corrupção do primário.

O superbloco

```
struct ext2_super_block {
    __le32 s_inodes_count;          /* Inodes count */
    __le32 s_blocks_count;         /* Blocks count */
    __le32 s_free_blocks_count;    /* Free blocks count */
    __le32 s_free_inodes_count;    /* Free inodes count */
    __le32 s_first_data_block;     /* First Data Block */
    __le32 s_log_block_size;       /* Block size */
    __le32 s_blocks_per_group;     /* # Blocks per group */
    __le32 s_inodes_per_group;     /* # Inodes per group */

    ...
    __le16 s_block_group_nr;        /* block group # of this superblock */

    ...
    __u8 s_uuid[16];               /* 128-bit uuid for volume */
    char s_volume_name[16];         /* volume name */
    char s_last_mounted[64];        /* directory where last mounted */

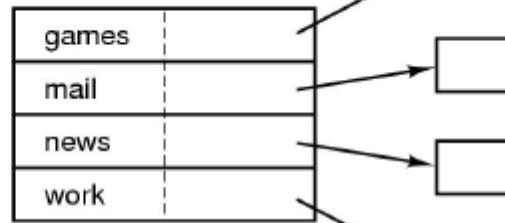
    ...
    __u16 s_padding1;
};
```

Implementação de directórios

games	attributes
mail	attributes
news	attributes
work	attributes

(a)

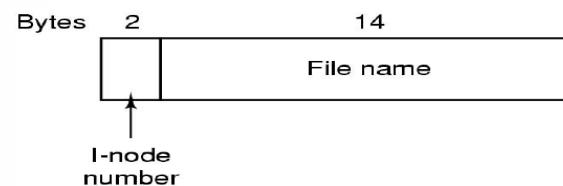
MS-DOS/WINDOWS



(b)

UNIX/LINUX

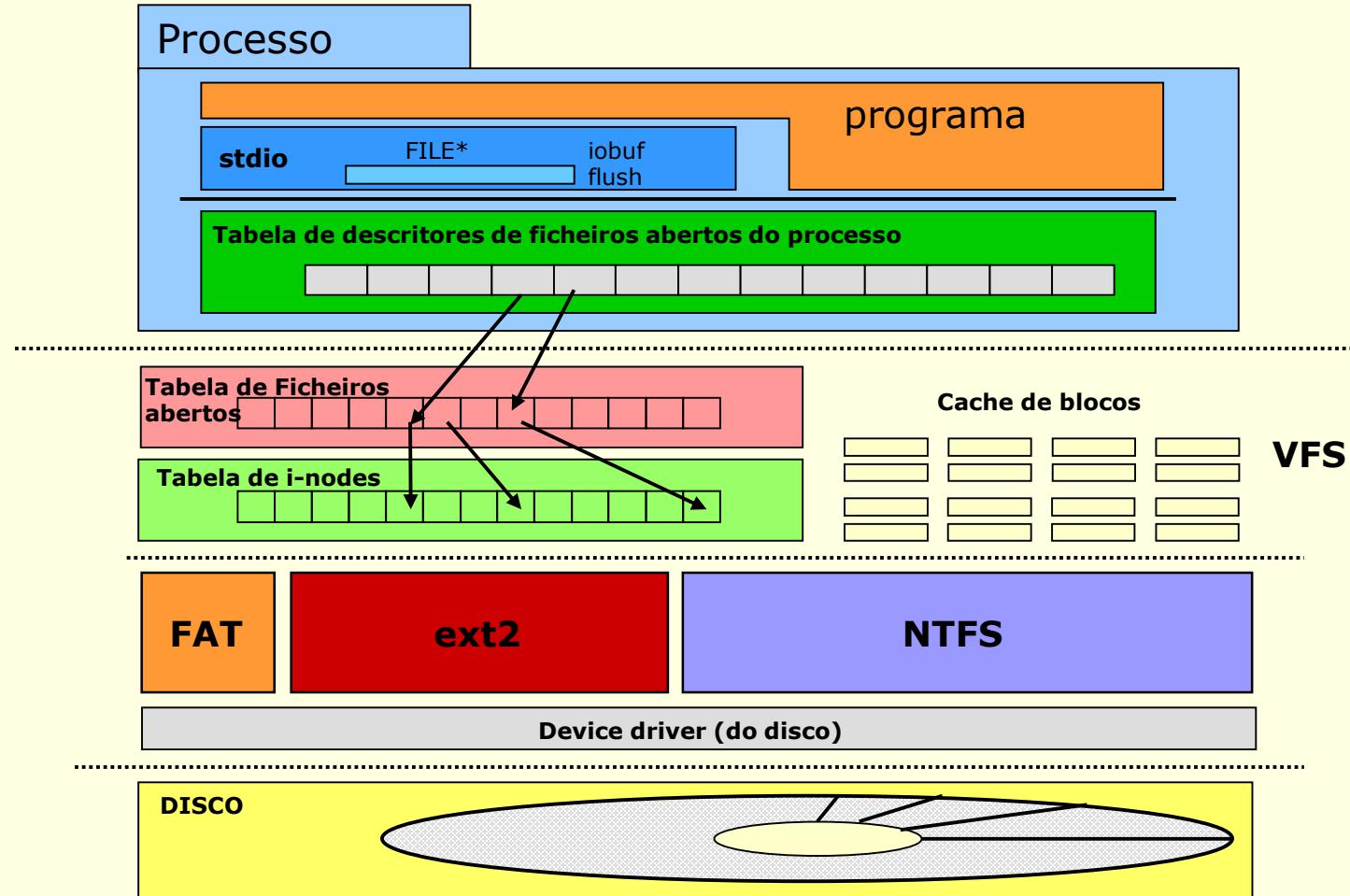
- (a) Directoria simples com
 - Entradas de dimensão fixa
 - Endereços de disco e atributos na entrada de directoria
- (b) Directoria onde cada entrada apenas refere um i-node:



Sistemas de Ficheiros nos UNIXes

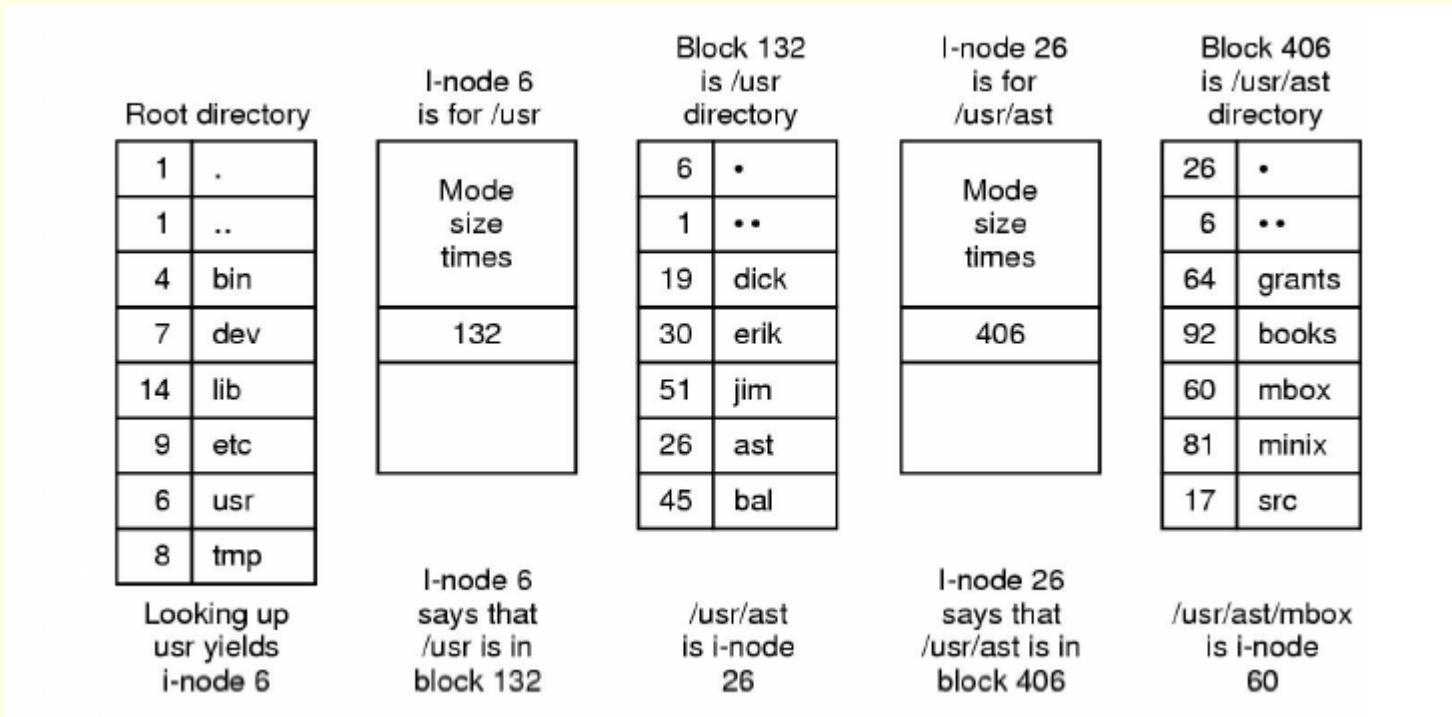
- Os SFs mais comuns oferecem uma árvore de directorias.
- O SO, *internamente*, esconde os detalhes de implementação e gere vários sistemas de ficheiros diferentes através de uma camada de abstracção, o *virtual file system* (VFS).
 - Portanto, o acesso a um ficheiro faz-se sempre do mesmo modo (open; read e/ou write; close) independentemente deste estar num disco (ou partição) ext2/3/4, FAT, NTFS, etc. (ver figura na pág. seguinte)
 - O mesmo acontece com o acesso a directorias: por ex., a forma de obter a lista de ficheiros existentes numa dada directoria é única e independente da directoria existir num SF **ext*** ou **FAT** ou **NTFS**...

Virtual File System



Abertura de um ficheiro

Abrir o ficheiro: /usr/ast/mbox



Abertura e leitura de um ficheiro

`open("/foo/bar", ...); read(...); read(...); read(...);`

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data	bar [0]	bar [1]	bar [2]
open(bar)			read			read		read		read	
read()				read		read		read		read	
read()					write						
read()						read		read		read	
						read					read

Estamos a ler..
O que é que
está a ser
escrito???

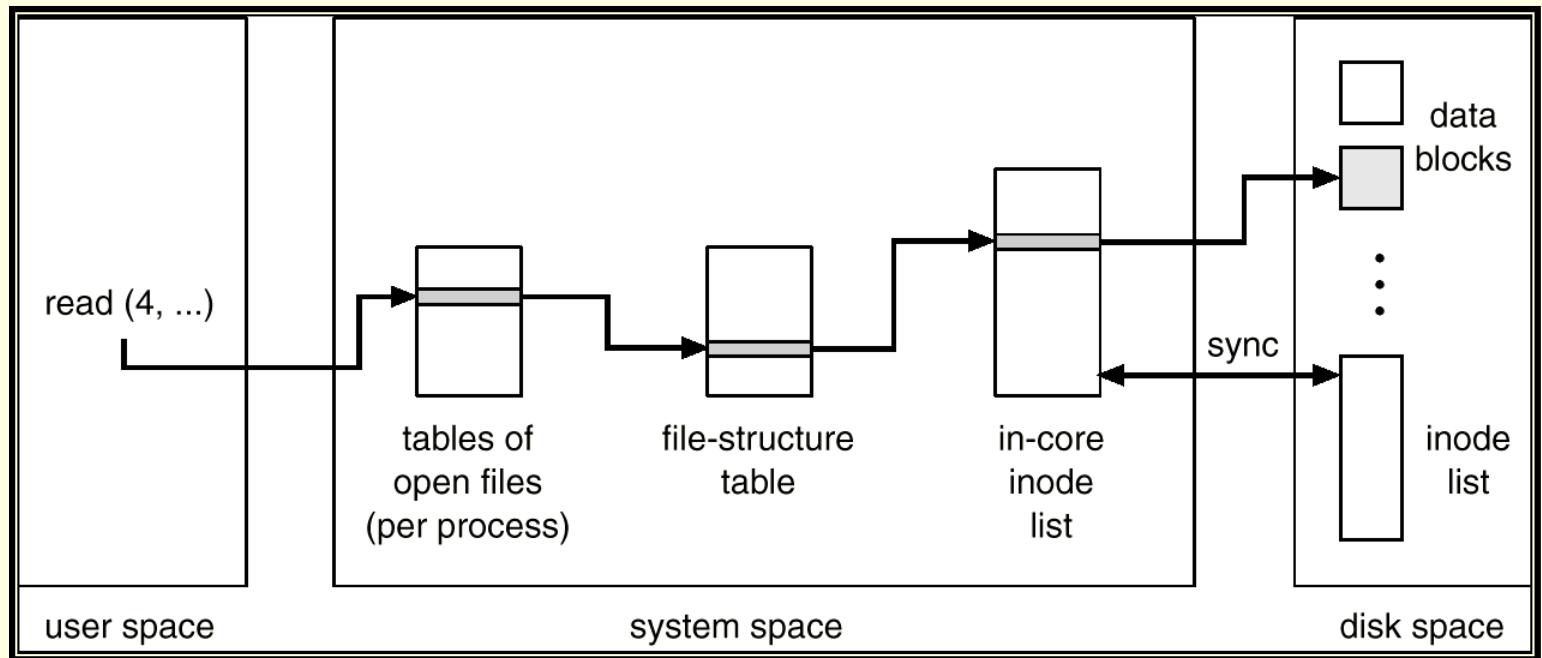
Criar e escrever num ficheiro

`create("/foo/bar", ...); write(...); write(...); write(...);`

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data [0]	foo data	bar data [1]	bar data [2]
create (/foo/bar)	read write	read	read	read write	read	read	read	write	
write()	read write		read					write	
write()	read write		read					write	
write()	read write		read					write	

Ficheiros: do EE do user ao EE do kernel (1)

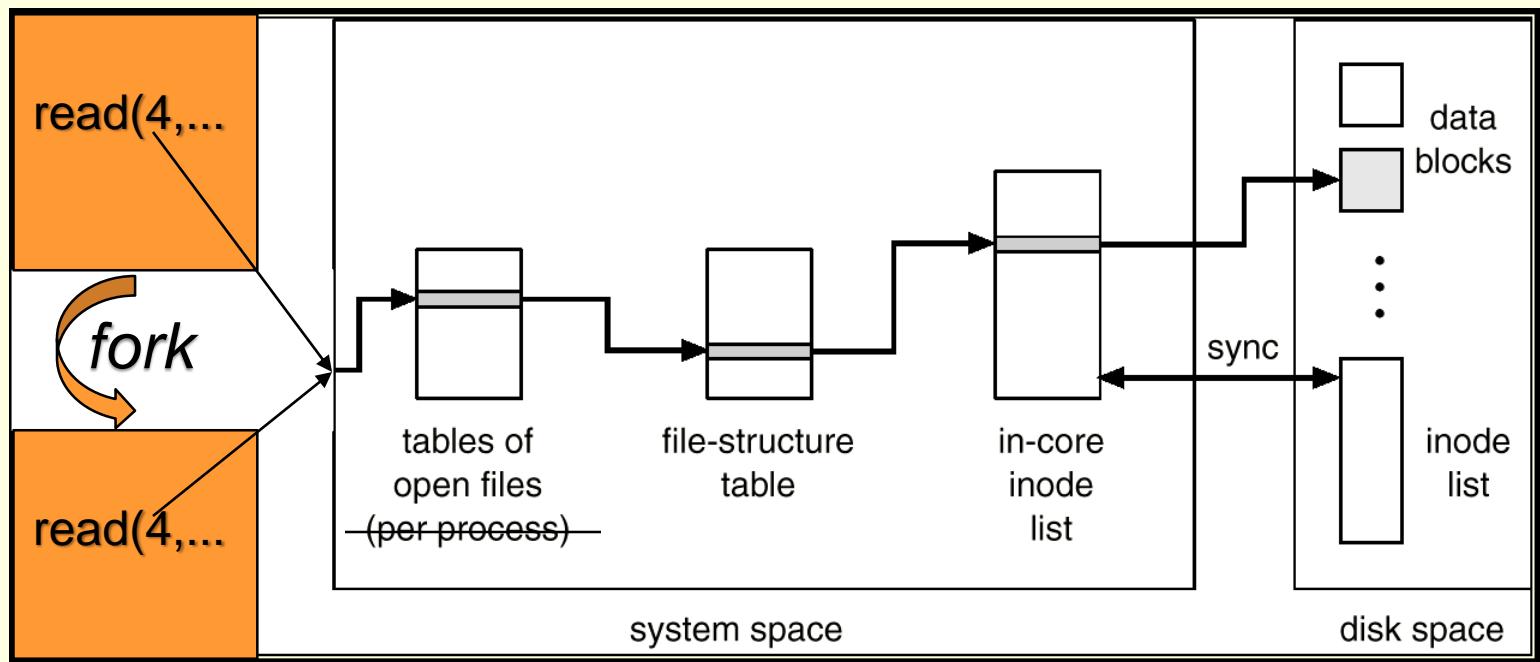
- Índice (file descriptor) no user space; tudo o mais no kernel...



Nota: no Linux não é exactamente assim, mas “conceptualmente” é ☺

Ficheiros: do EE do user ao EE do kernel (2)

- No caso do fork(), a tabela de ficheiros abertos “por processo” é partilhada...



Ficheiros: do EE do user ao EE do kernel (3)

`fork()`, File descriptors, Open File Table

