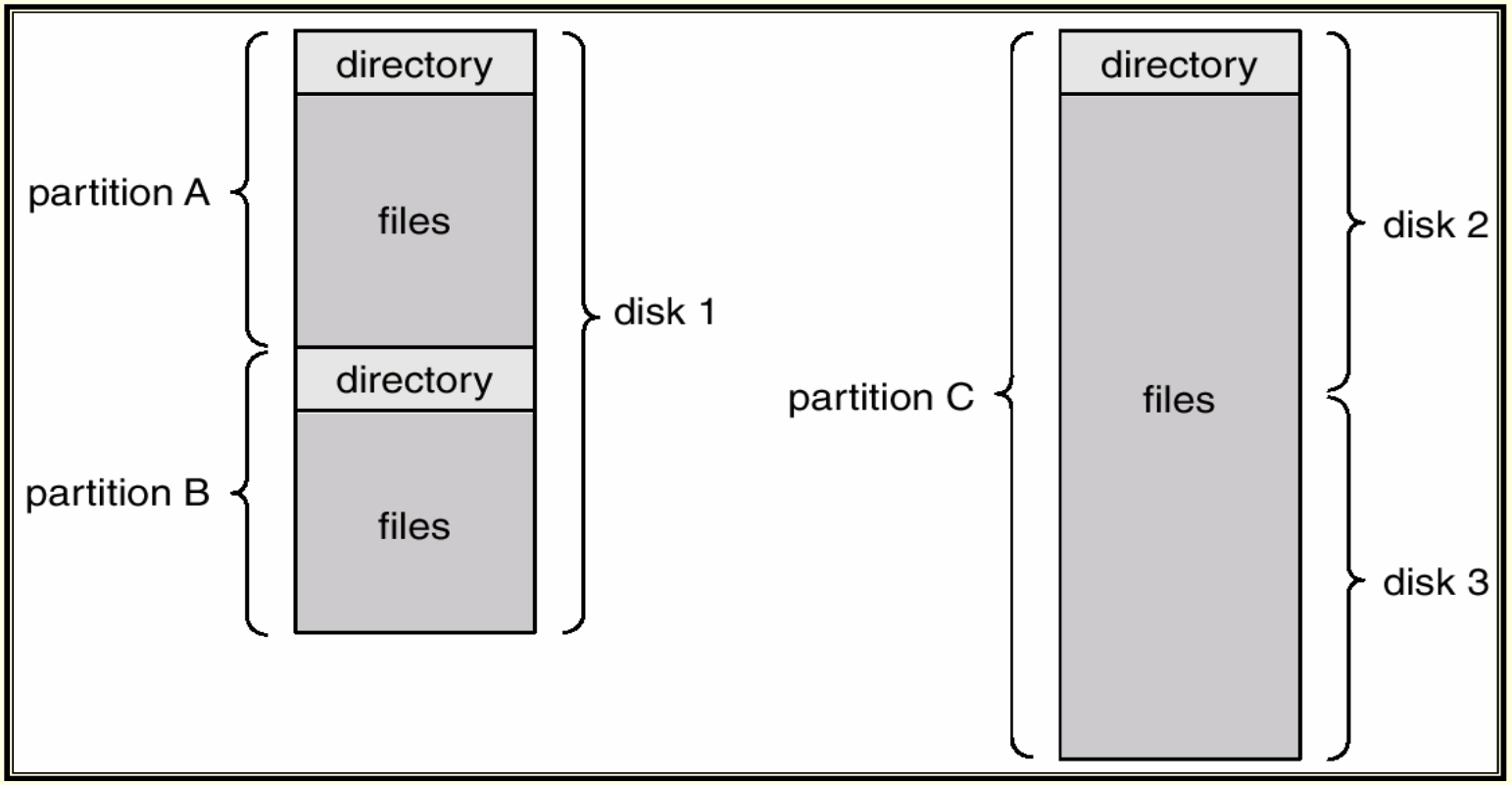
Parte II: Gestão de Ficheiros

d) Sistema de Ficheiros em Disco – Atribuição de Espaço aos ficheiros

O Sistema de Ficheiros em disco: II

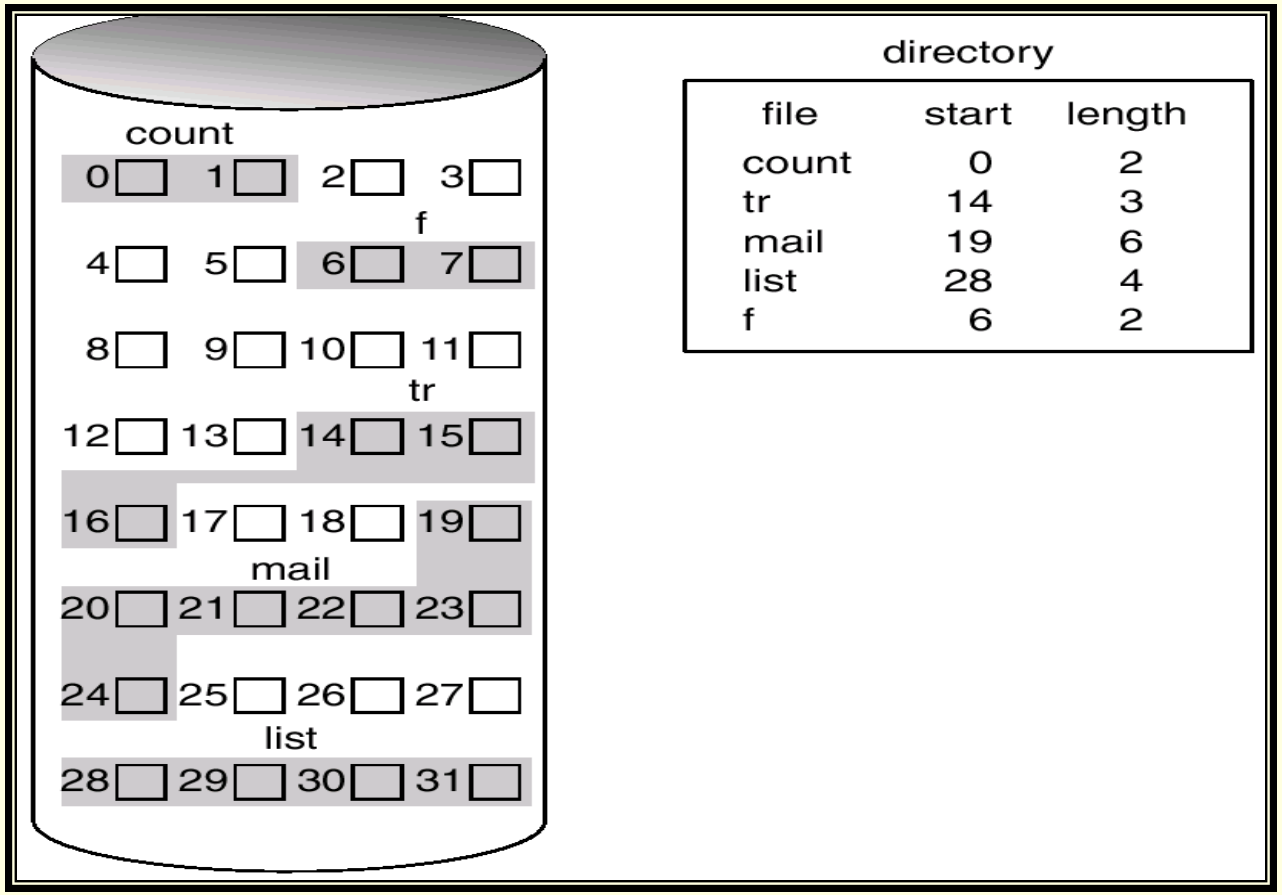
- Um *layout* simples: como atribuir o espaço aos ficheiros?



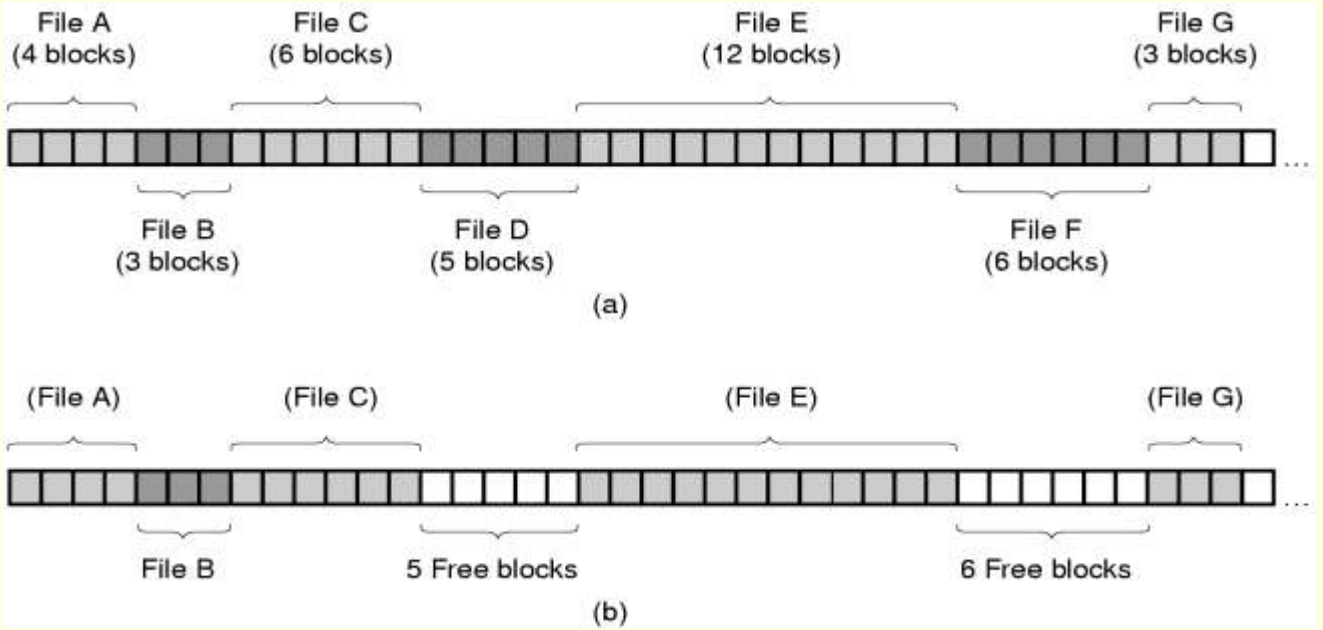
Atribuição contígua: I

- Cada ficheiro ocupa um conjunto de blocos contíguos.
- Simples – só é necessário guardar o número do bloco inicial e o número de blocos
- Vantagens:
  - Acesso directo facilmente suportado
  - Desempenho muito elevado em acesso sequencial
- Desvantagens:
  - Desperdício de espaço (fragmentação externa)
  - Dificuldade no crescimento dos ficheiros.

Atribuição contígua: II



Atribuição contígua: II

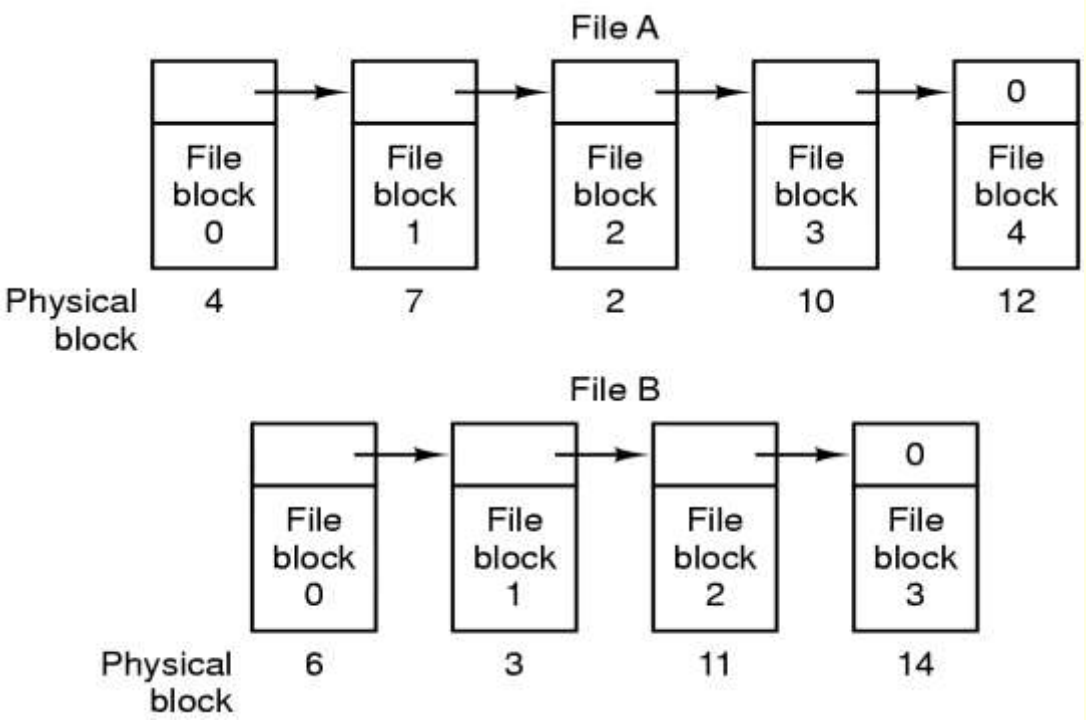


(a) Atribuição contígua de espaço para 7 ficheiros

(b) Estado do disco após a remoção dos ficheiros D e F

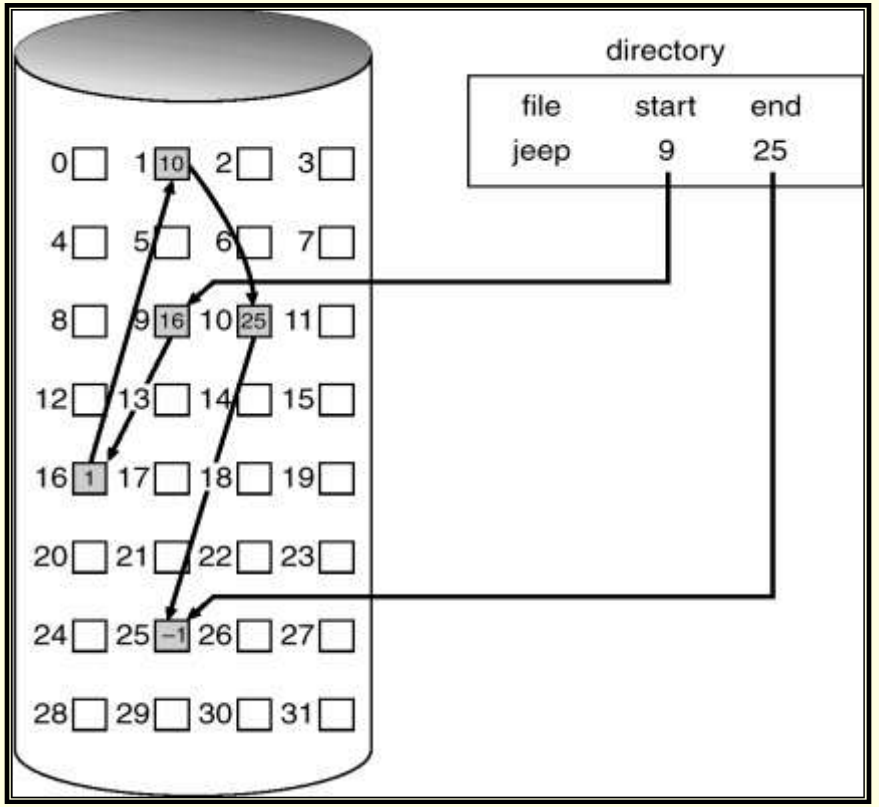
Atribuição ligada: I

- Cada bloco de um ficheiro ocupa uma qualquer posição.
- Aponta para o bloco seguinte desse mesmo ficheiro



Atribuição ligada: II

- Apontador para 1º bloco (e eventualmente último) de um ficheiro guardado na directoria.

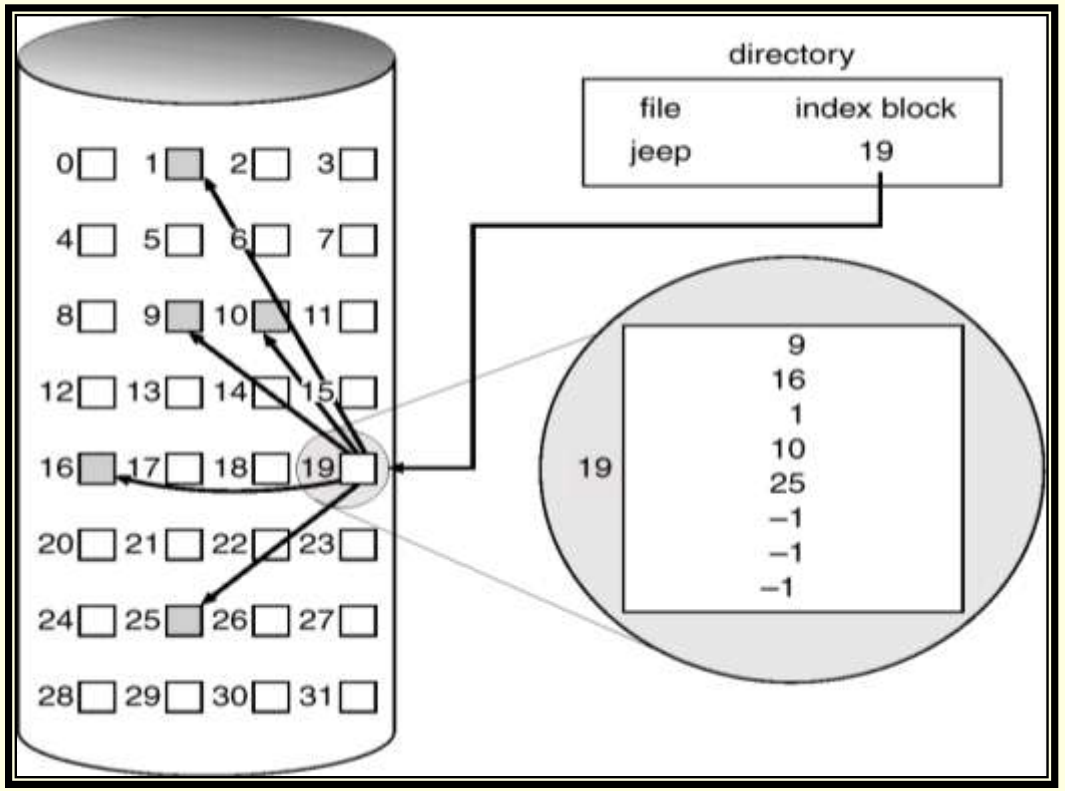


Atribuição ligada: III

- Vantagens:
  - Simples – só necessita de guardar o endereço inicial
  - Pequeno desperdício de espaço (i.e., há fragmentação interna, mas não externa)
- Desvantagens:
  - Acesso directo muito lento – para chegar ao bloco N é preciso ler os N-1 anteriores.
  - Desperdício de espaço em apontadores, “fragmentação” dos dados (os dados de dois blocos só podem ser “agregados” em memória depois de “remover” espaço ocupado pelos apontadores)

Atribuição indexada: I

- Os apontadores para os blocos são guardados em blocos de índice.



O 1º bloco do fich. é o 9;

o 2º é o 16;

...

o 5º é o 25. Não há mais!

Atribuição indexada: II

- Vantagens:
  - Acesso directo rápido (máximo de 2 acessos)
  - Pequeno desperdício de espaço (não há fragmentação externa)
- Desvantagens:
  - Espaço ocupado pela tabela de índices.
  - Gestão da tabela de índices: tamanho fixo ou variável? Se variável, blocos de índice ligados entre si em lista ou por indexação “de ordem superior”?

Atribuição: Casos reais

- Soluções híbridas, com o intuito aproveitar as vantagens de uma dada técnica e minorar as sua desvantagens...
  - FAT: MS-DOS, Win*
    - Uma variante da atribuição ligada
  - ext2: Linux
    - Uma variante da atribuição indexada
- Outras melhorias
  - Usar “grupos de blocos” contíguos, em vez de um só bloco como unidade de atribuição; designados
    - clusters na FAT
    - extents no ext2

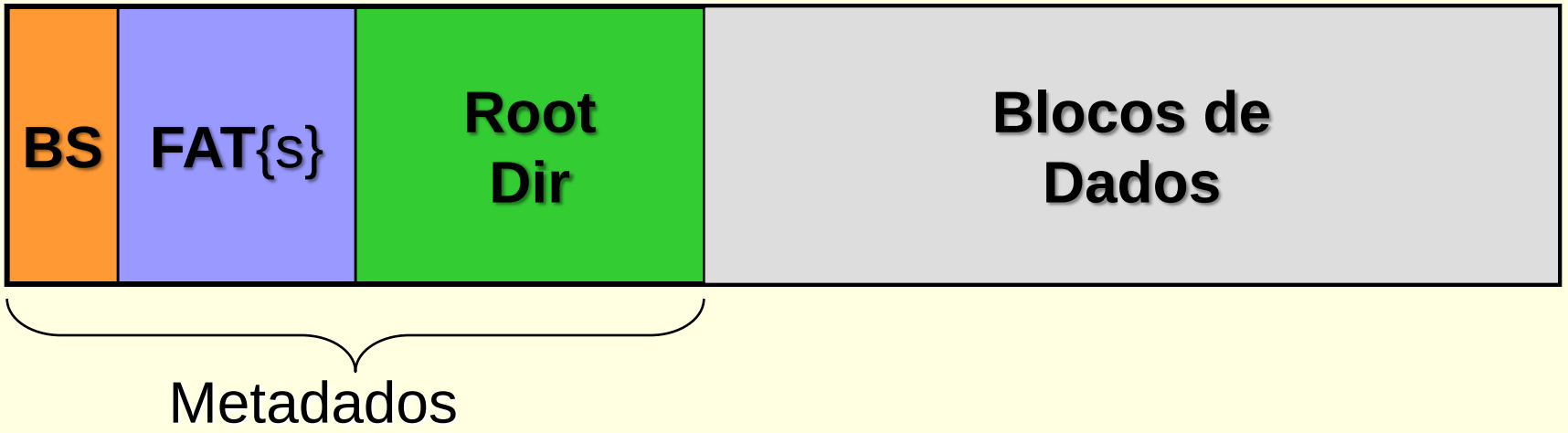
Unix Windows NT Netware MacOS DOS/VS Vax/VMS
Linux Solaris HP/UX AIX Mach
Chorus

Parte II: Gestão de Ficheiros

e) Estudo de casos –
O Sistema de Ficheiros do MS-DOS
(notas breves)

O "FATfs" em diskette: Vista Geral

- Um *layout* simples:



Boot Sector: informação sobre as outras estruturas e sobre o disco

Uma ou mais File Allocation Table(s): listas ligadas de blocos

Root Dir: Directoria Raíz da diskette

O "FATfs" em diskette: Boot Sector - I

- Contém informação sobre:
  - A dimensão (em *bytes*) de um sector
  - O número de blocos (sectores lógicos) que formam um *cluster*
  - O número de FATs existentes (para recuperação de erros)
  - A dimensão (em sectores) de uma FAT
  - O número de entradas da Root Directory
  - O número total de blocos reservados
  - O número total de blocos
  - ... Etc ...

O "FATfs" em diskette: a FAT - I

- Os dados de um ficheiros são armazenados numa lista ligada de *clusters*;
- Um *cluster* é um conjunto de blocos contíguos; quando um disco é formatado, é escolhido o tamanho para o *cluster* (i.e., quantos blocos constituem cada *cluster*).
- A FAT é uma tabela de apontadores para/entre *clusters*:
  - Cada *cluster* de dados é representado na FAT por uma entrada, i.e., o número de entradas na FAT é igual ao número de *clusters* presentes no disco (seja este número NC).
  - Cada entrada ocupa b= $\log_2$ NC *bits*.
  - Uma entrada com valor de:
    - 0, está vazia
    - 0xFF8, é a última da lista
    - Outro valor, aponta para o cluster com esse endereço

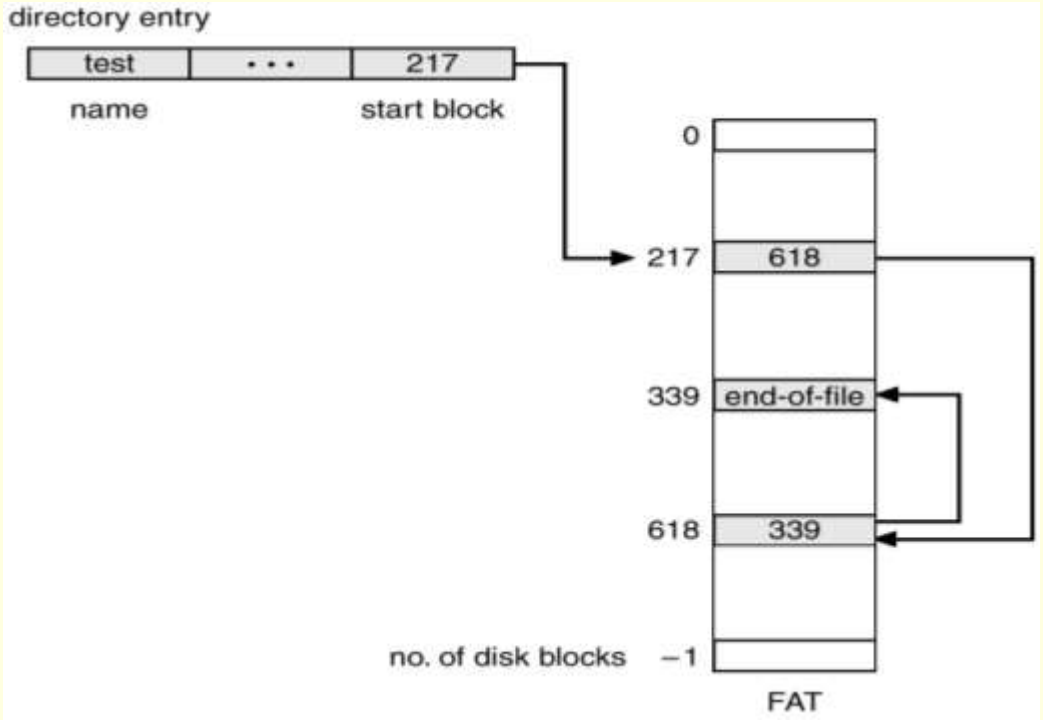
O "FATfs" em diskette: a Root Dir - I

- O BS tem um campo que indica qual a dimensão (em número de entradas) da Root Directory.
- A Root Directory é um vector de entradas com a seguinte estrutura (de 32 bytes no total):

Nome (8)	Ext (3)	Attr (1)	Infos várias (14)	Início (2)	Dim. (4)

- A "capacidade" da Root Dir é limitada: no caso de uma diskette de 1.44MB, a Root Dir tem 224 entradas (ver Boot Sector), logo a sua dimensão é de 224x32/512 = 14 sectores

O "FATfs": FAT e Directoria - funcionamento



- Nota: A FAT apenas contém apontadores!!! Os blocos (referenciados por esses apontadores) estão na zona de dados

O "FATfs" em diskette: a FAT - II

- Exemplo: diskette de 1.44MB
  - 2880 blocos
  - Cada entrada na FAT tem 12 bits (é chamada FAT-12)
  - Assim, pode suportar-se um máximo de 2^{12} = 4096 clusters
  - A dimensão da FAT nesta diskette é 9 blocos
    - Assim, o número de entradas é 9 x 512 x 8 / 12 = 3072
  - Então, para a diskette de 1.44MB, um *cluster* tem 1 bloco (pois 3072 apontadores chegam para apontar 2880 *clusters*)
  - Contudo, para a diskette de 2.88MB, com uma FAT de 9 blocos, os 3072 apontadores já não chegam para apontar 5760 *clusters* de um bloco, de forma que se usam *clusters* de 2 blocos passando o número de *clusters* a ser metade, i.e., 2880
  - Há, portanto, um conjunto de escolhas (e *tradeoffs*) a fazer entre a dimensão do cluster, da FAT, e do número de bits a usar nas entradas da FAT.

O "FATfs": Dimensão do disco e do *cluster*

- Dimensão (máxima) da partição em função da dimensão do *cluster* (aqui designado por Block size)

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

O "FATfs" em diskette: FORMAT

- A operação de FORMAT de uma diskette:
  - Cria as estruturas BS, FATs e RootDir em memória
    - Inicializa os campos de BS com as informações relevantes
    - "Limpa" (inicializa) as FATs colocando todos os apontadores a 0
    - "Limpa" (inicializa) a RootDir colocando em cada entrada todos os campos a 0 ou "branco" conforme a natureza dos mesmos
  - Escreve os blocos de disco que armazenarão as estruturas BS, FATs, e RootDir

O "FATfs" em diskette: outros...

- A informação anteriormente fornecida foi obviamente simplificada:
  - Não abordamos a FAT-16 e FAT-32
  - Não abordamos os "nomes longos" nem com caracteres "acentuados"

O "FATfs" em diskette: Boot sector - II

- A estrutura completa do Boot Sector é a seguinte:

```
struct fat_boot_sector {
        __s8    ignored[3];             /* Boot strap short or near jump */
        __s8    system_id[8];           /* Name - can be used to special case  partition manager volumes */
        __u8    sector_size[2];         /* bytes per logical sector */
        __u8    cluster_size;           /* sectors/cluster */
        __u16   reserved;               /* reserved sectors */
        __u8    fats;                   /* number of FATs */
        __u8    dir_entries[2];         /* root directory entries */
        __u8    sectors[2];             /* number of sectors */
        __u8    media;                  /* media code (unused) */
        __u16   fat_length;             /* sectors/FAT */
        __u16   secs_track;             /* sectors per track */
        __u16   heads;                  /* number of heads */
        __u32   hidden;                 /* hidden sectors (unused) */
        __u32   total_sect;             /* number of sectors (if sectors == 0) */

        /* The following fields are only used by FAT32 */
        __u32   fat32_length;           /* sectors/FAT */
        __u16   flags;                  /* bit 8: fat mirroring, low 4: active fat */
        __u8    version[2];             /* major, minor filesystem version */
        __u32   root_cluster;           /* first cluster in root directory */
        __u16   info_sector;            /* filesystem info sector */
        __u16   backup_boot;            /* backup boot sector */
        __u16   reserved2[6];           /* Unused */
};
```

O "FATfs" em diskette: a Root Dir - II

- A estrutura completa é a seguinte:

```
struct msdos_dir_entry {
    __s8   name[8],ext[3];          /* name and extension */
    __u8   attr;                    /* attribute bits */
    __u8   lcase;                   /* Case for base and extension */
    __u8   ctime_ms;                /* Creation time, milliseconds */
    __u16 ctime;                    /* Creation time */
    __u16 cdate;                    /* Creation date */
    __u16 adate;                    /* Last access date */
    __u16 starthi;                  /* High 16 bits of cluster in FAT32 */
    __u16 time,date,start;          /* time, date and first cluster */
    __u32 size;                     /* file size (in bytes) */
};
```

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

Parte II: Gestão de Ficheiros

e) Estudo de casos –

O Sistema de Ficheiros do Unix

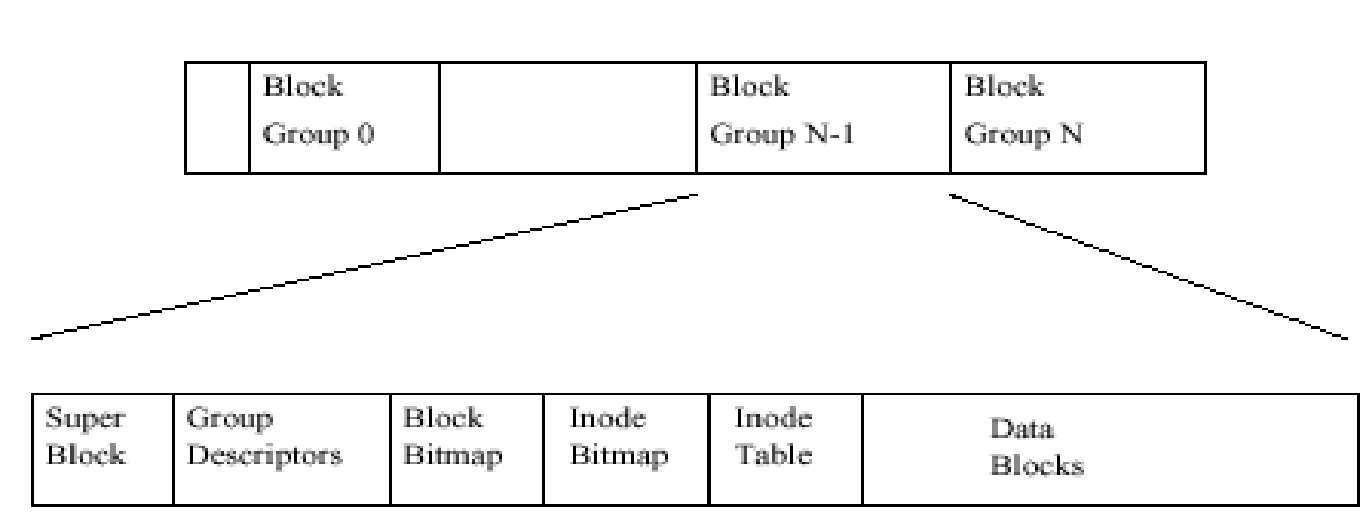
Sistema de Ficheiros: UNIX

- *Oferece uma árvore de directorias.*
- *O SO, internamente, esconde os detalhes de implementação e gere vários sistemas de ficheiros 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) independen-temente 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…

Sistema de ficheiros nativo

- Denominado ext2
- Descreve a organização física e estruturas de dados de um(a partição de um) disco.
- Ver `include/linux/ext2_fs.h`

Grupo de Blocos do Ext2



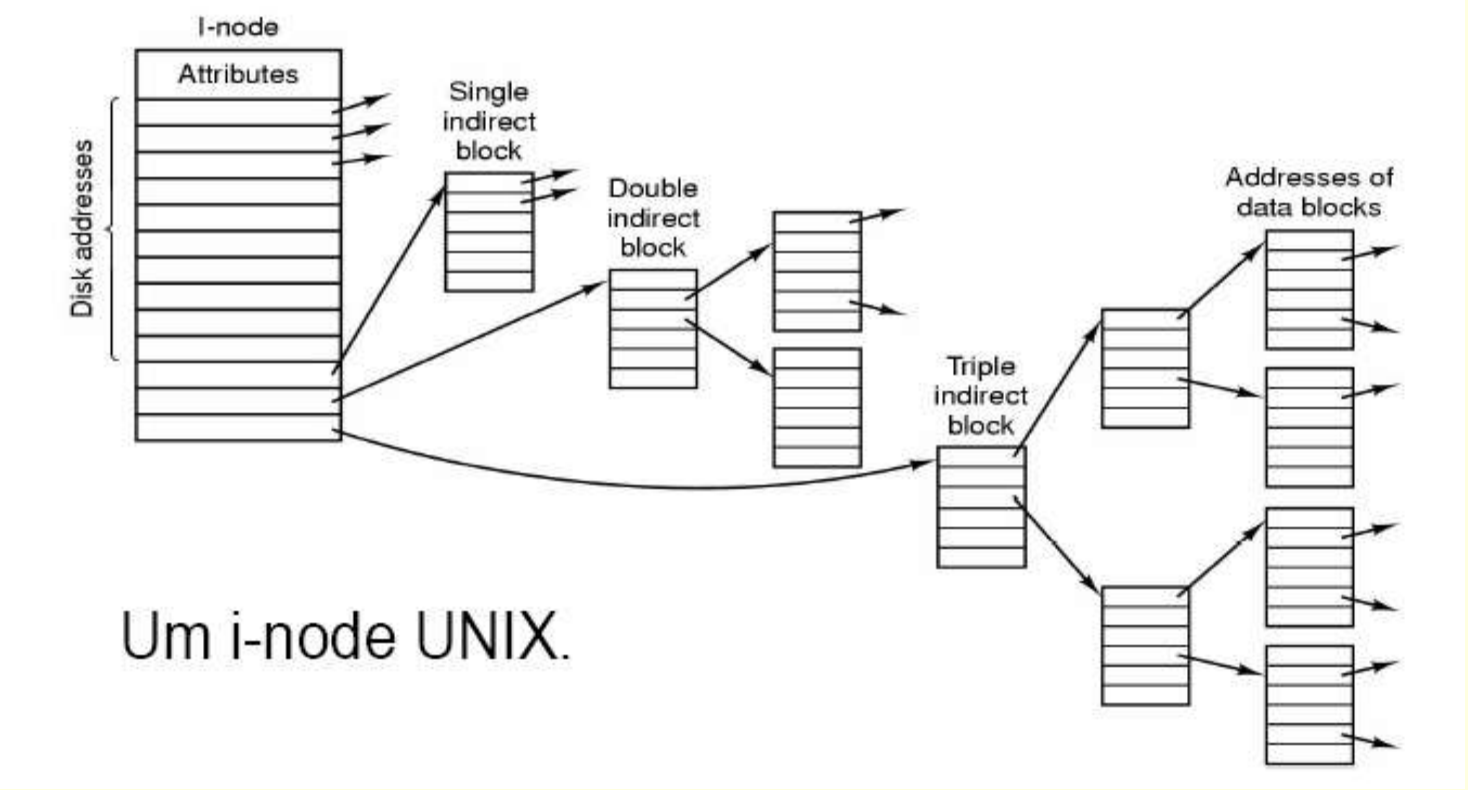
Sistema de ficheiros ext2

- 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;
};
```

i - node



Um i-node UNIX.

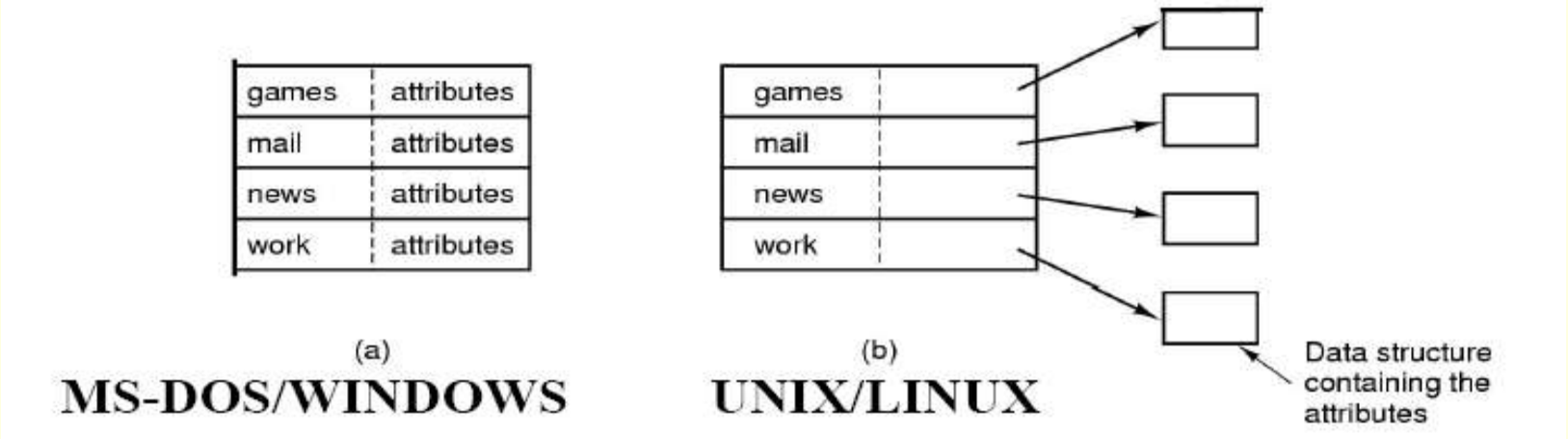
O inode (1)


```
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 */

    ...
}
```

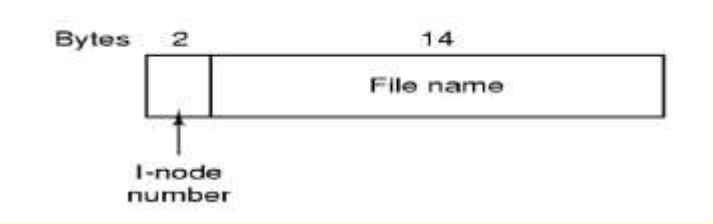
O inode (2)


```
...

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

...
```

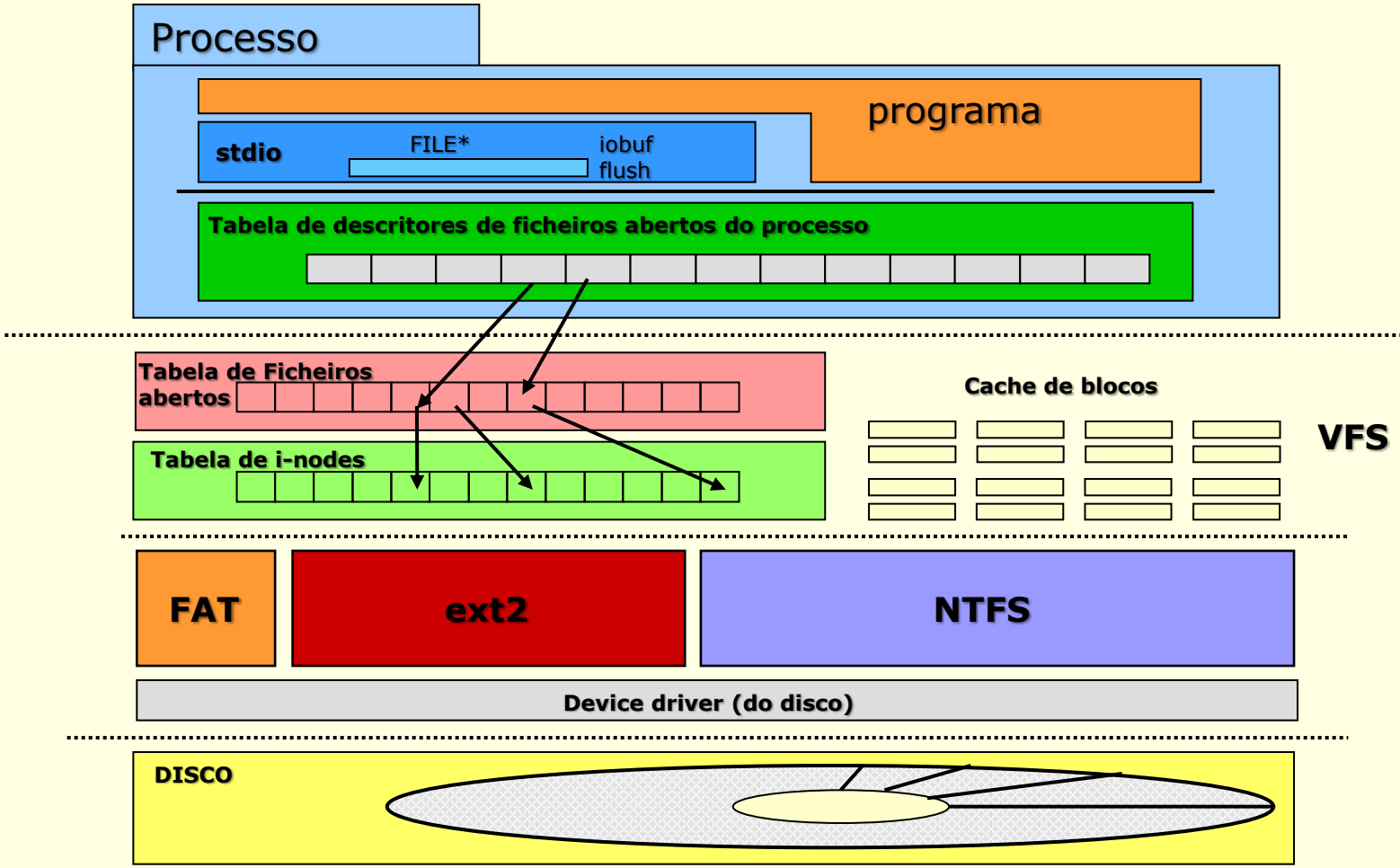
Implementação de directórios



- (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:



Virtual File System



Abertura de um ficheiro

Abrir o ficheiro: /usr/ast/mbox