

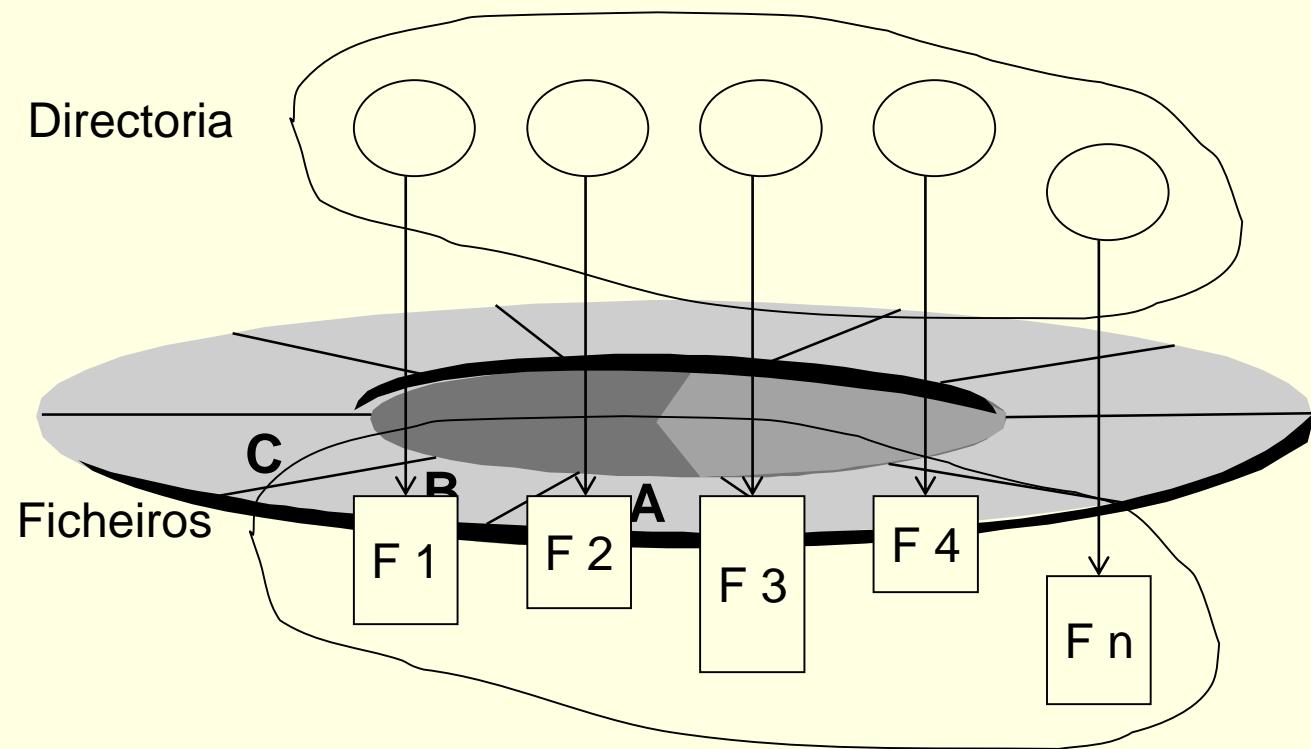
Fundamentos de Sistemas de Operação

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

Gestão de Ficheiros
Sistema de Ficheiros em Disco –
Organização e Gestão do Espaço (em disco) ...

O Sistema de Ficheiros em disco

- Como armazenar no disco as “directorias” e os “ficheiros”?



O SF simples: a directoria

- A directoria é armazenada no bloco 1 de disco:

- Uma entrada é... (tentativa!)

```
struct entry {  
    char nome[8];  
    unsigned int inicio, comprimento;  
}
```

- A directoria é:

- Um vector de **struct entry** capaz de conter o máximo de entradas que podem ser armazenadas num único bloco (512 bytes).

(continua)

O SF simples: primitivas p/ gestão de blocos

- Existem funções para ler e escrever um bloco de disco, dado o seu número:

```
void readBlock(int num, void * blk)  
void writeBlock(int num, void * blk)
```

- Existem também funções para “gerir” blocos:

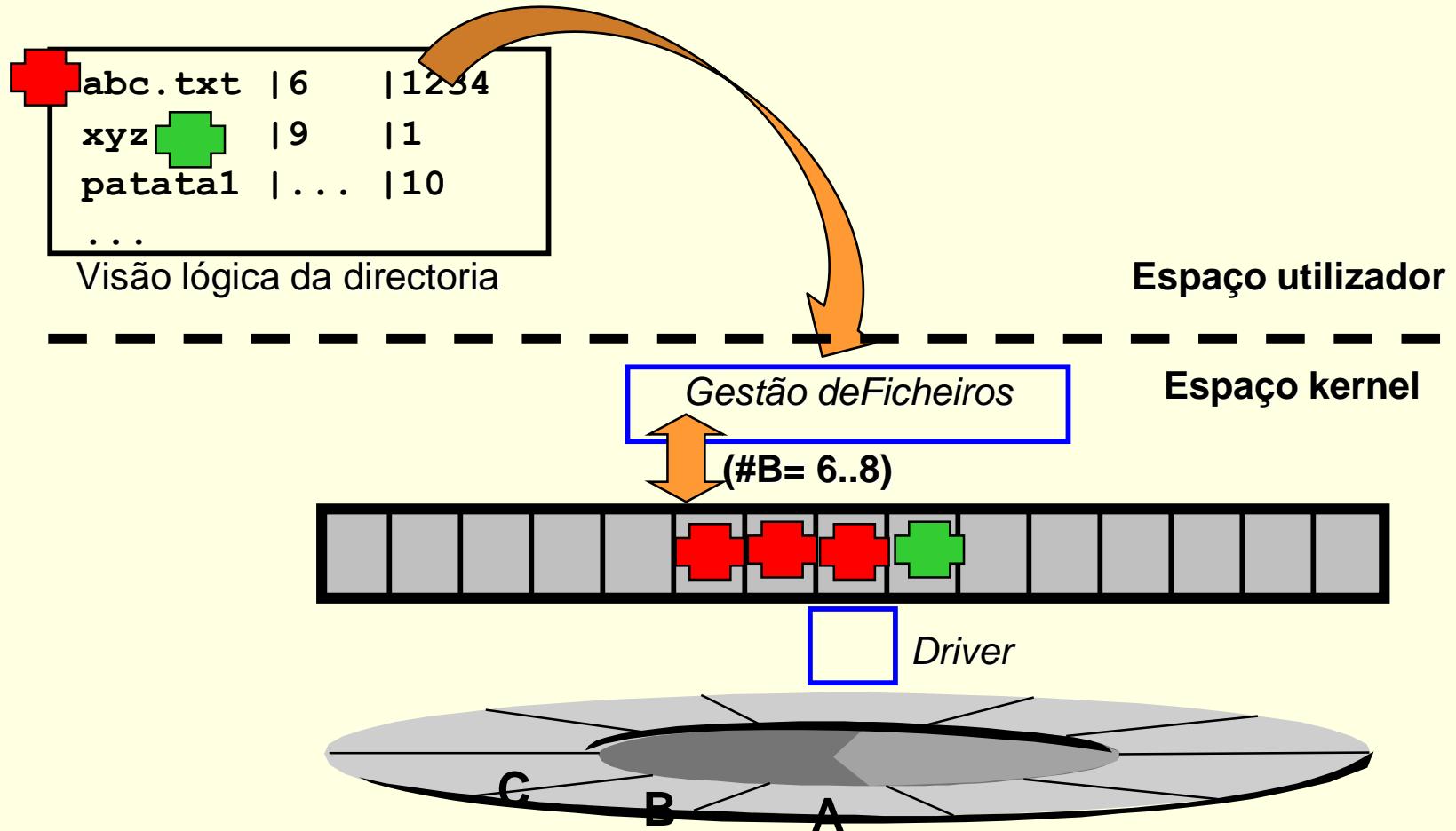
- Dado um determinado nº de blocos, se se encontrar uma sequência de blocos todos com o mesmo status (e.g., 0=bloco livre, 1=ocupado) devolver o nº do 1º bloco, caso contrário devolver 0

```
unsigned int getBlocks(int num, int status)
```

- Dada uma fatia [#inicial..#final] marcar todos os blocos dessa fatia com o mesmo status (e.g., 0=livre, 1=ocupado)

```
int setBlocks(int range[2], int status)
```

Um SF com ficheiros contíguos



Um SF com ficheiros contíguos: create

□ Como será a criação de um ficheiro?

- `int create(char name[8], unsigned int maxSize)`
(o utilizador terá de indicar o nome e comprimento máximo do ficheiro em bytes – o SO arredondará para cima em blocos)
- Retorno: 0 = OK,
- Erros (usar `#define` num ficheiro .h):
 - ENOSPACE (não há espaço para os dados)
 - EDIRFULL (directória cheia)
 - EINVALNAME (caracteres inválidos no nome)
 - EEXISTS (já existe esse nome)
 - ...

Um SF com ficheiros contíguos: create (1)

```
#define NENTRIES (sizeof(struct dirent)/512)  
  
#define FREE 1  
  
struct dirent dir[NENTRIES];  
int isDirInMemory = 0;  
  
int create(char name[8], int maxSize) {  
    int entry; unsigned int block;  
    if !validName(name) return -EINVAL;  
    if !isDirInMemory  
        { readBlock(1, dir); isDirInMemory= 1; }  
    if (entry= lookupDir(name)) < 0)  
    if (entry == -1) return -EDIRFULL;  
    else return -EEXISTS;
```

(continua)

Um SF com ficheiros contíguos: create (2)

```
//pág.anterior, dir e nome ok, falta encontrar blocos livres
if !(block= getBlocks(roundUP(maxSize,512), FREE))
    return -ENOSPACE;

//actualizar dir em memoria e no disco
memcpy(dir[entry].nome, name, 8);
dir[entry].inicio= block;
dir[entry].comprActual= 0;                                //bytes
dir[entry].comprMaximo= roundUP(maxSize,512); //blocos
range[0]=block; range[1]= comprMaximo; setBlock(range, !FREE);
writeBlock(dir, 1); updateFreeBlkInfo(...;
return 0;
}
```

Gestão do espaço livre (1)

- Precisamos de saber quais os blocos livres (e ocupados). As funções `getBlocks` e `setBlocks` manipulam essa informação.
- **Hipótese 1:** mantemos os blocos livres em lista ligada...
- Vantagens:
 - Simples
- Desvantagens:
 - Se quisermos encontrar N blocos contíguos livres, pode ser moroso...
 - Formatar o disco = preencher os apontadores em todos os blocos
- **Hipótese 1.1:** cada apontador aponta um grupo de blocos livres contíguos – diminui muito as desvantagens ☺

Gestão do espaço livre (2)

- **Hipótese 2:** estrutura em *bitmap*: um bit por cada bloco.
- Vantagens:
 - Simples
 - Compacta: disco 512GB ~ 1M blocos = 1M bits ~ 128KB = 256 blocos
 - Facilmente acomodada na memória, “em cache”
 - Algoritmos eficientes: manipulam strings de bits...
- Desvantagens:
 - ? Quais ?
- Exercício: escreva as funções **getBlocks** e **setBlocks** ☺

MeuFS: metadados (1)

□ Já “decidimos”

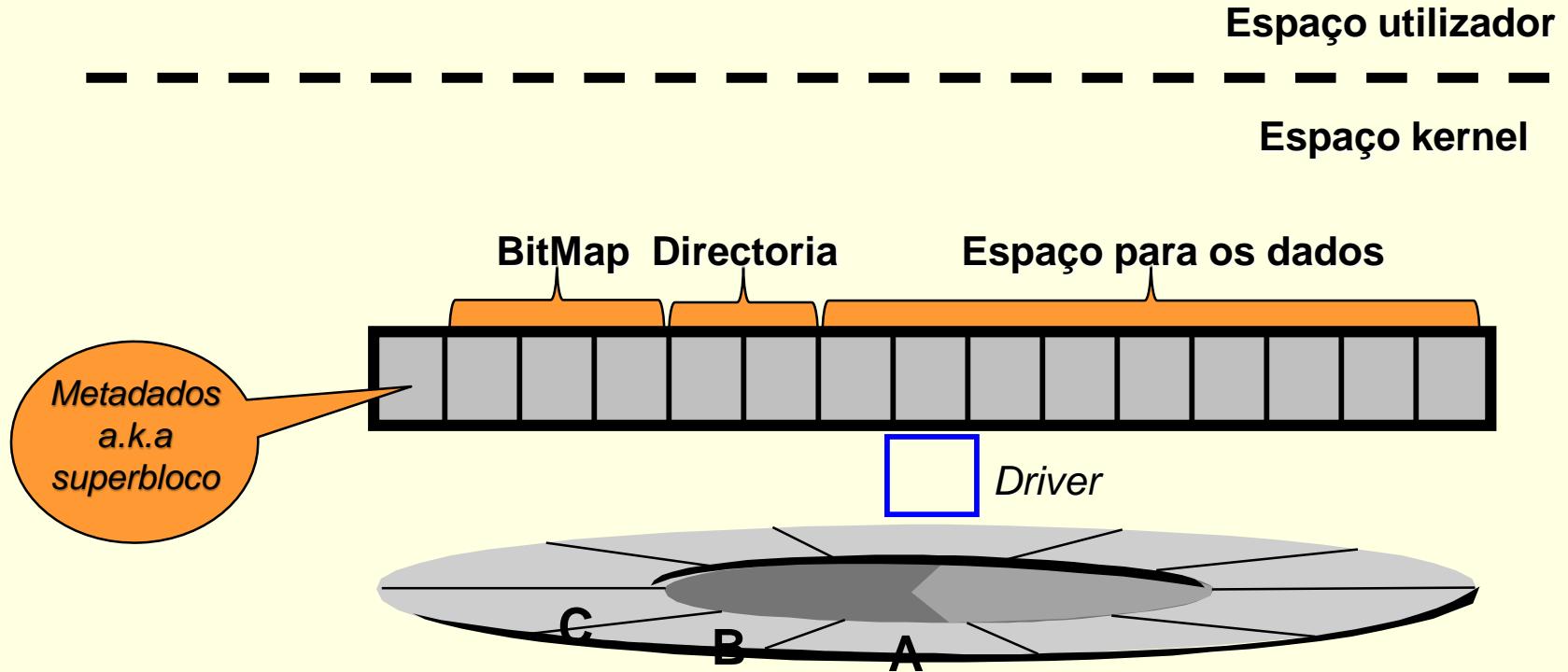
- Que vamos usar um bitmap para gerir os blocos livres/ocupados que ocupa um certo nº B de blocos
- Que vamos ter uma directória raiz que ocupa um certo nº D de blocos

□ Precisamos de registar essa informação no disco,

- Usamos o bloco 0 ☺
- Proposta:

```
struct Metadata {  
    unsigned int DiskSize;          // #blocks of this disk  
    unsigned int BitMapSize; // #blocks of BitMap área  
    unsigned int StartOfDir; // #block where the dir. starts  
    ... }
```

MeuFS: metadados



Formatar um disco em MeuFS (2)

□ *Formatador*

- Aplicação que cria no disco as estruturas de dados correspondentes a um FS “vazio”

□ *Algoritmo*

1. Pedir ao driver informação sobre o disco ($nº$ de blocos)
2. Das opções dadas na linha de comando ou dos valores default decidir: dimensão D da directória.
3. $Nº$ de blocos para dados, $N = “nº$ de blocos do disco” – D . Calcular “ $nº$ de blocos do bitmap” = $roundUP(N/8,512)$
4. Escrever informações (1), (2) e (3) no superbloco
5. “Zerar” zona de blocos “D”
6. “Zerar” zona de blocos do Bitmap
7. Fim.

MeuSF++: a directoria

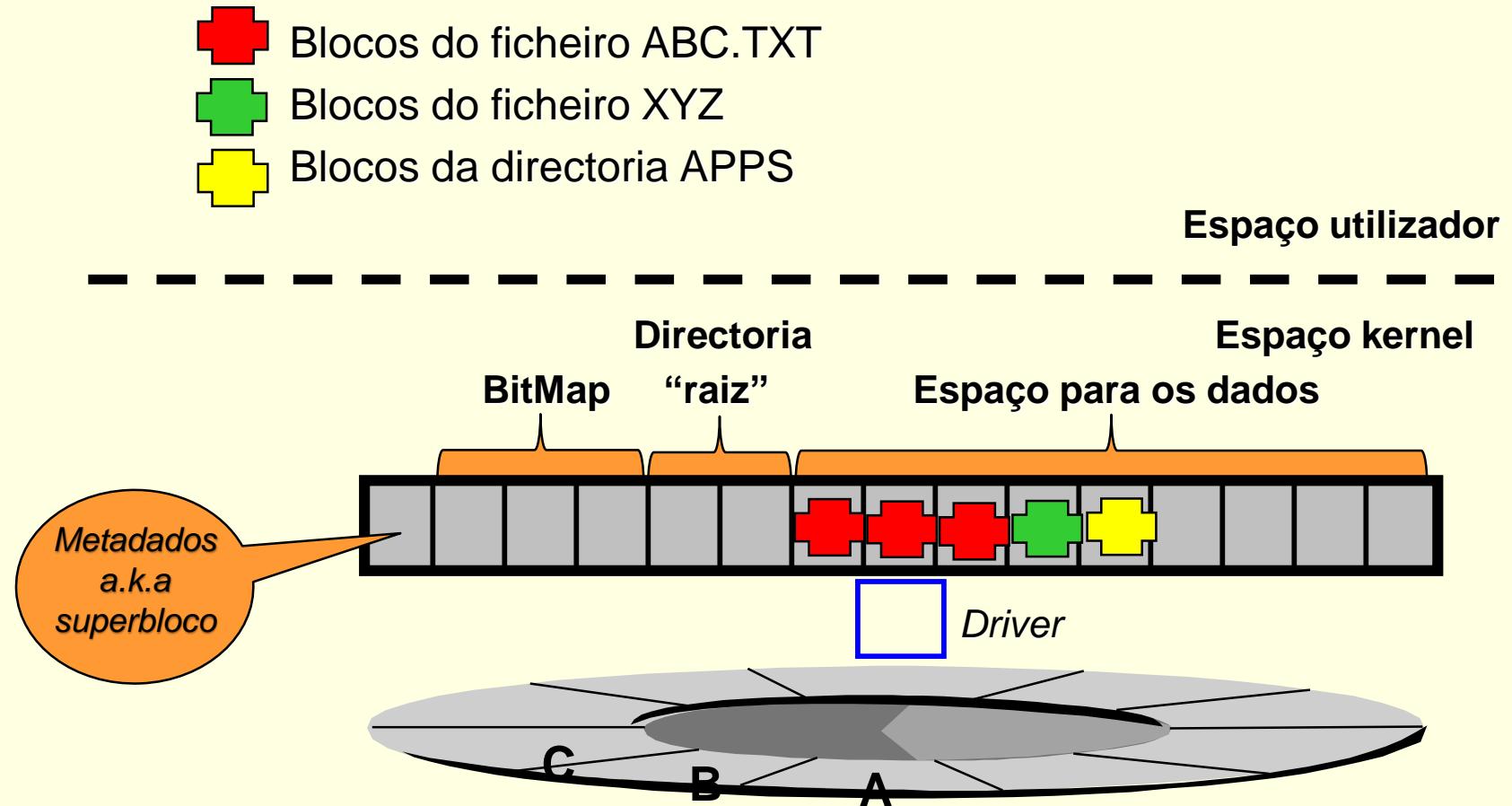
□ Mais informação na directoria:

- Uma entrada é...

```
struct entry {  
    char nome[8];  
    char extensao[3];  
    unsigned int inicio, compActual, compMaximo;  
    unsigned int dataCriacao, dataModificacao, dataAcesso;  
    unsigned char tipo; // ficheiro ou directoria  
    unsigned char permissoes;  
    ...  
}
```

Agora, se tipo=='D', o ficheiro (cont)é(m) uma directoria

MeuFS++: múltiplas directorias...



MeuFS++: múltiplos discos...

