Gestor de Sistemas de Ficheiros

Sistema de Ficheiros

- Composto por um conjunto de entidades fundamentais:
  - um sistema de organização de nomes para identificação dos ficheiros;
  - uma interface programática para comunicação entre os processos;
  - sistema de ficheiros
- Ficheiro:
  - colecção de dados persistentes, geralmente relacionados, identificados por um nome.

Sistema de Ficheiros (cont.)

- Meta-informação guardada no mesmo sistema de memória secundária que a informação que descreve
- A solução mais simples, para manter a meta-informação:
  - tabela mantida na memória secundária,
  - na realidade é uma estrutura um pouco mais complexa que se designa por directório
- Um directório pode conter:
  - toda a meta-informação relativa a um ficheiro ou
  - apenas uma parte dela, sendo a restante distribuída por outras estruturas auxiliares

Sistema de Ficheiros (cont.)

- O que é?
  - conjunto de ficheiros, directórios, descritores e estruturas de dados auxiliares geridos por um módulo do sistema operativo (**Sistema de Gestão de Ficheiros**)
  - permitem estruturar o armazenamento e a recuperação de dados persistentes em um ou mais dispositivos de memória secundária (discos ou bandas magnéticas)
- ficheiro
  - conjunto de dados persistentes, geralmente relacionados, identificado por um nome
  - é composto por:
    - nome: identifica o ficheiro perante o utilizador
    - descritor de ficheiro: estrutura de dados em memória secundária com informação sobre o ficheiro (dimensão, datas de criação, modificação e acesso, dono, autorizações de acesso)
    - informação: dados guardados em memória secundária
- portanto, para além do nome, um ficheiro possui ainda outro tipo de informação que facilita a sua localização e gestão:
  - dimensão, datas de criação, modificação e acesso, direitos de acesso, e localização da informação em memória secundária.
  - O conjunto destes dados é usualmente designado por meta-informação.

Sistema de Ficheiros (cont.)

- Na maioria dos SO actuais podem coexistir em simultâneo vários sistemas de ficheiros:
  - cada dispositivo de memória secundária pode possuir uma organização de informação e meta-informação diferente
  - E.g. emWindows é comum existirem em simultâneo três sistemas de ficheiros: o FAT (*File Allocation Table*), o NTFS (*NT File System*), e o CDFS (*Compact Disk File System*)

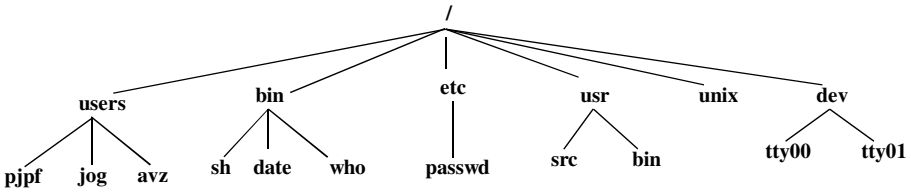
Sistema de Ficheiros (cont.)

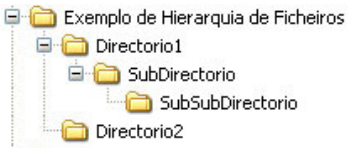
- As operações mais frequentes sobre ficheiros são a leitura e escrita da sua informação.
  - Bastam estas funções?
- Por razões de desempenho, é mantida uma Tabela de Ficheiros Abertos
- Acesso a ficheiro em três etapas:
  - Abertura do ficheiro, dado o nome.
    - O sistema pesquisa o directório, copia a meta-informação para memória e devolve ao utilizador um identificador que é usado como referência para essa posição de memória.
  - Leituras e escritas, dado o identificador de ficheiro aberto.
    - Permite obter rapidamente o descritor do ficheiro em memória, onde está toda a informação necessária para aceder aos dados.
  - Fecho do ficheiro.
    - Esta operação é necessária para libertar a memória que continha a meta-informação do ficheiro e actualizar essa informação no sistema de memória secundária.

Organização dos Nomes dos Ficheiros

- Várias formas de organizar os nomes dos ficheiros:
  - Apenas um directório global ao sistema e atribuir um nome a cada ficheiro.
  - Foi utilizada nos primeiros sistemas dedicados a processamento por lotes.
  - Problema da colisão de nomes, pois não era possível guardar um ficheiro com um nome idêntico a outro já existente.
  - Uma primeira evolução foi atribuir um directório separado para cada utilizador.
- O Multics foi o primeiro sistema a propor a organização hierárquica dos nomes dos ficheiros, na forma de uma árvore:
  - solução que hoje praticamente todos os sistemas usam

Organização Hierárquica





Hierarquia de Nomes

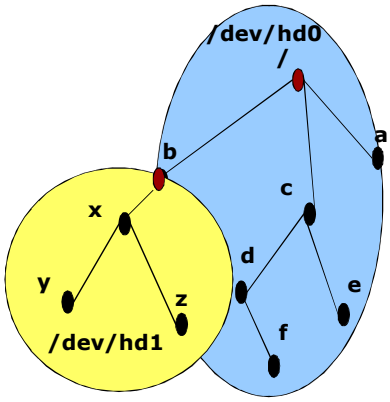
- Organização hierárquica (árvore):
  - solução proposta no **Multics**
  - os directórios contêm caminhos de acesso para nós descendentes a partir de um directório raiz
  - ficheiros e directorias vazias são nós terminais (folhas)
  - caminho de acesso (pathname): cadeia de caracteres que localiza um ficheiro ou directoria na árvore
  - nomes absolutos ou relativos:
    - absoluto: caminho de acesso desde a raiz
    - relativo: caminho de acesso a partir do directório corrente
  - directório corrente mantido para cada processo como parte do seu contexto

Hierarquia de Nomes (cont)

- Um ficheiro pode ser conhecido por vários nomes:
  - é possível designar o mesmo ficheiro com o nome /a/b/c e com o nome /x/y.
  - é comum chamar a cada um destes nomes *links.*
- Problema:
  - quando se pretende apagar o ficheiro com o nome /a/b/c.
  - apagar o conteúdo do ficheiro ou apenas o nome?
- A semântica utilizada na maioria dos sistemas de ficheiros é a última:
  - quando se remove um nome remove-se um *link*
  - Se esse *link* for o único que o ficheiro possui, dever-se-á apagar o ficheiro.
  - cada sistema de ficheiros tem a sua forma diferente de implementar os *links*

Espaço de Nomes

- Quer o Windows quer o Unix têm um espaço de nomes uniforme.
  - ficheiros, directórios, dispositivos são referenciados usando exactamente a mesma sintaxe.
- Em ambos os casos existe uma única raiz de nomes:
  - i.e. todos os nomes de ficheiros, directórios e dispositivos começam por uma '/' or '\', em Unix e Windows, respectivamente
- Mount:
  - esta operação consiste em ligar a raiz do novo sistema de ficheiro a um directório do sistema de ficheiros raiz.




```
mount -t <filesystem> /dev/hd1 /b
```


Nomes e Extensões

- É usual, os nomes dos ficheiros terem uma extensão que dê alguma informação sobre o seu conteúdo:

Extensão	Descrição
.c	Código fonte em C.
.obj	Código objecto resultante de uma compilação.
.exe	Código executável.
.html	Hipertexto para ser visualizado por navegador de rede
.dat	Ficheiro de dados.
.mp3	Representação comprimida de som digital

Tipos de Ficheiros

- Um ficheiro é composto por um conjunto de registos
  - de dimensão fixa ou variável
  - em muitos casos é meramente estruturado como uma sequência de bytes.
- Método de acesso aos registos:
  - acesso sequencial (historicamente ligado às bandas magnéticas):
  - acesso directo (corresponde ao funcionamento dos discos magnéticos):
  - acesso por chave (usado em algumas BDs):

Atributos de um Ficheiro

- O tipo de um ficheiro é um dos vários atributos que cada ficheiro possui.
- Estes atributos podem variar de sistema de ficheiros para sistema de ficheiros
- Estão tipicamente armazenados na meta-informação do ficheiro.
- Para além do tipo, a meta-informação do ficheiro possui usualmente os seguintes atributos:
  - Protecção – quem pode aceder ao ficheiro e quais as operações que pode realizar.
  - Identificação do dono do ficheiro – geralmente quem o criou.
  - Dimensão do ficheiro – por vezes automaticamente actualizada quando o ficheiro cresce ou diminui.
  - Data de criação – última leitura e última escrita.

Métodos do Sistema de Ficheiros

- Podemos dividir as funções relacionadas com o sistema de ficheiros em seis grupos:
  - Abertura, criação e fecho de ficheiros;
  - Operações sobre ficheiros abertos;
  - Operações complexas sobre ficheiros;
  - Operações sobre directórios;
  - Acesso a ficheiros mapeados em memória;
  - Operações de gestão dos sistemas de ficheiros.

Abertura, criação e fecho de ficheiros

Retorno	Nome	Parâmetros	Descrição
fd :=	Abrir	(Nome, Modo)	**Abre um ficheiro**
fd :=	Criar	(Nome, Protecção)	**Cria um novo ficheiro**
	Fechar	(Fd)	**Fecha um ficheiro**

Operações sobre ficheiros abertos

Nome	Parâmetros	Descrição
Ler	(Fd, Tampão, bytes)	**Lê de um ficheiro para um tampão de memória**
Escrever	(Fd, Tampão, bytes)	**Escreve um tampão para um ficheiro**
Posicionar	(Fd, Posição)	**Posiciona o cursor de leitura ou escrita**

Outras Funções (biblioteca)

- É comum existirem outras operações sobre ficheiros abertos específicas das linguagens de alto nível:
  - Para optimizar o acesso ao sistema de ficheiros é comum as linguagens de alto nível utilizarem estruturas locais geridas no espaço de memória do utilizador, usualmente mantido em bibliotecas.
  - Estas estruturas contêm tampões especiais para onde são lidos e escritos os ficheiros, em blocos de maior dimensão do que aquela que é usualmente pedida pelo utilizador.

```
FILE *file = fopen("filename", "a");
/* ... */
fwrite(buffer0, sizeof(char), BUFFER_SIZE, file);
/* Escreve dados num tampão da libc */
fwrite(buffer0, sizeof(char), BUFFER_SIZE, file);
fflush(file);
/* ... */
fclose(file);
```

- São escritos vários *buffers* para o ficheiro e depois é invocada a função fflush.
  - todas as escritas foram feitas no *buffer* interno da libc,
  - a escrita no sistema de ficheiros só é efectuada quando é utilizada a função fflush (quando o ficheiro é fechado, ou quando o *buffer* interno da libc já se encontra cheio)

Operações complexas sobre ficheiros

- Algumas operações sobre ficheiros permitem realizar operações sobre a totalidade do ficheiro, como copiá-lo, apagá-lo ou movê-lo

Nome	Parâmetros	Descrição
Copiar	(Origem, Destino)	**Copia um ficheiro**
Mover	(Origem, Destino)	**Move um ficheiro de um directório para outro**
Apagar	(Nome)	**Apaga um ficheiro**
LerAtributos	(Nome, Tampão)	**Lê atributos de um ficheiro**
EscreverAtributos	(Nome, Atributos)	**Modifica os atributos**

Operações sobre directórios

Nome	Parâmetros	Descrição
ListaDir	(Nome, Tampão)	**Lê o conteúdo de um directório**
MudaDir	(Nome)	**Muda o directório por omissão**
CriaDir	(Nome, Protecção)	**Cria um novo directório**

Acesso a ficheiros mapeados em memória

- A primitiva MapearFicheiro permite aceder ao conteúdo de um ficheiro da mesma forma que se acede a uma qualquer outra estrutura em memória.
- O conteúdo do ficheiro referenciado por Fd é mapeado a partir da posição indicada pelo parâmetro posição, no endereço de memória definido no parâmetro seguinte e numa extensão indicada em dimensão.
- É possível aceder directamente ao ficheiro referenciando directamente com um apontador às posições de memória indicadas.

Nome	Parâmetros	Descrição
MapearFicheiro	(fd, posição, endereço, dimensão)	**Mapeia um ficheiro em memória**
DesmapearFicheiro	(endereço, dimensão)	**Elimina o mapeamento**

Operações de gestão dos sistemas de ficheiros

Nome	Parâmetros	Descrição
Montar	(Directório, Dispositivo)	**Monta um dispositivo num directório**
Desmontar	(Directório)	**Desmonta o dispositivo montado no directório**

Sistemas de Ficheiros

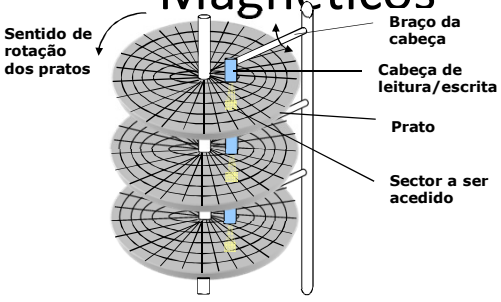
Estrutura Interna dos Sistemas de Armazenamento

Dispositivos de Memória Secundária

- Os sistemas de ficheiros são armazenados em dispositivos de memória secundária:
  - na maioria dos casos estes dispositivos são dispositivos de memória de massa persistente, (i.e. grandes quantidades de dados que não se perdem após desligar a energia), tais como os discos magnéticos
  - existem casos em que estes dispositivos não são nem de memória de massa, nem persistentes.

- Existem situações em que é útil utilizar um sistema de ficheiros para gerir informação em memória volátil:
  - E.g., o sistema operativo Linux utiliza uma solução deste tipo (vulgo RAM Disk) durante o seu processo de instalação.
  - neste caso, a utilidade do sistema de ficheiros não resulta da sua capacidade de gerir grandes volumes de dados de forma eficiente, mas sim da interface uniforme e da organização da informação que estes proporcionam.

Características Físicas dos Discos Magnéticos



- Cada disco possui um ou mais pratos de um material magnético
- Em cada face de cada prato a informação é escrita em pistas concêntricas, e cada pista é composta por um conjunto de sectores
- O conjunto das pistas com o mesmo raio forma um cilindro
- Cada prato está dividido em blocos de dimensão fixa, denominados sectores.
- Os sectores são a unidade mínima de leitura e escrita em disco.
  - a sua dimensão habitual é de 512 ou 1024 byte.
  - a leitura ou escrita de um sector faz-se quando a cabeça do respectivo prato está sobre o sector respectivo.

Características Físicas dos Discos Magnéticos (cont)

- A leitura e escrita de um sector faz-se à velocidade de rotação do disco, i.e. os bytes de um sector são lidos ou escritos à medida que passam sobre a cabeça do disco
- O tempo de leitura/escrita de um sector é composto por:
  - tempo de posicionamento (seek time): tempo de deslocação das cabeças até ao cilindro desejado
  - tempo de latência: tempo de espera pelo sector (tmédio = tmeia rotação)
  - tempo de transferência: tempo que demora a transferir um sector entre o disco e a memória principal (trevolução / n. de sectores por pista)
- Tempo médio de acesso:
  - soma dos tempos médios de posicionamento, latência e transferência
- Tempos típicos para um disco de 7200 RPM com 272 a 472 sectores por pista (o número de sectores é maior nas pistas exteriores):

Nome	Forma de cálculo	Valor típico
Tempo transferência	Tempo de revolução / sectores por pista	**17µs**
Tempo de posicionamento	Tempo de posicionamento da cabeça na pista	**4-8ms**
Tempo latência médio	Tempo de revolução / 2	**4ms**

Características Físicas dos Discos Magnéticos (cont)

- Se o sector a ler estiver na mesma pista que o sector lido anteriormente: não é necessário tempo de posicionamento
- Se o sector a ler estiver por baixo da cabeça de leitura na altura precisa em que se pretende lê-lo: o tempo de latência também não existe.
- Algoritmos de gestão de memória secundária dos sistemas operativos:
  - reduzir ao menor valor possível o tempo de posicionamento e latência,
  - fazendo com que os acessos estejam coordenados da melhor forma.
- A gestão de um disco é feita em blocos de dimensão múltipla dos sectores.
- A dimensão do bloco depende das estruturas do sistema operativo:
  - é a unidade mínima que pode ser indexada pelo sistema operativo.
- Além do sector e do bloco (dependem respectivamente da organização física do disco e do sistema operativo) existe ainda o segmento ou *extent*.
  - o segmento é a unidade de reserva de espaço em disco composto por um conjunto de blocos

Características Físicas dos Discos Magnéticos (cont)

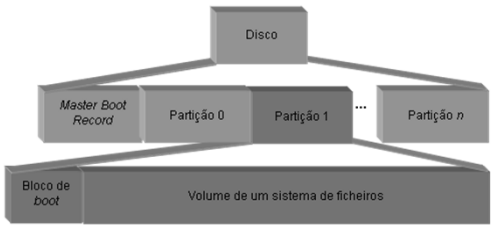
- O bloco e o segmento são unidades de gestão que têm por objectivo optimizar o acesso ao disco:
  - quanto maior for o bloco maior é a taxa de transferência bruta (os tempos de posicionamento e latência de sectores consecutivos são quase nulos, mas a taxa de transferência efectiva depende do número de bytes dentro do bloco com informação útil)
  - se um bloco estiver apenas meio cheio a sua transferência efectiva é metade da transferência bruta (quanto maiores forem os blocos maiores serão as diferenças entre as taxas de transferência bruta e efectiva)
  - situação semelhante ao problema da fragmentação de memória
  - é necessário encontrar um equilíbrio que maximize a taxa de transferência efectiva
  - actualmente a dimensão de um bloco varia entre 4 Kbytes e 8 Kbytes.

- O segmento é usualmente composto por vários blocos:
  - o objectivo do segmento é reduzir a taxa de dispersão do disco.
  - diz-se que um disco tem uma taxa de dispersão elevada quando as sequências contíguas de blocos que fazem parte de um mesmo ficheiro forem de reduzida dimensão, i.e. os blocos de cada ficheiro estão dispersos pelo disco.
  - na literatura de língua inglesa é por vezes designada *fragmentation* no sentido de fragmentação dos ficheiros pelo disco e não no conceito de fragmentação que utilizamos nos capítulos anteriores

Organização Lógica

- Cada dispositivo pode armazenar em simultâneo vários sistemas de ficheiros:
  - é necessário manter informação sobre o espaço ocupado por cada um dos sistemas de ficheiros de forma persistente
  - esta informação está organizada em estruturas guardadas de forma sequencial no disco.

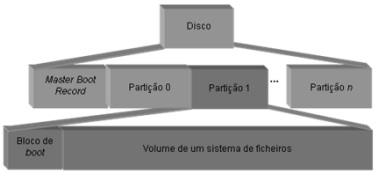
- Constituição de um disco, com as várias componentes que suportam a existência de vários sistemas de ficheiros independentes em cada disco:



Organização Lógica (cont.)

- Partição:
  - subdivisão lógica de um dispositivo físico (disco, CD-ROM, etc.), e está dividida em blocos de tamanho fixo.
  - partição é vista pelo sistema operativo como um vector de blocos ordenado a partir do número zero.
  - cada partição é um contexto independente das outras, não existindo ficheiros repartidos por partições diferentes.

- Volume:
  - divisão lógica de um sistema de ficheiros mapeada numa partição,
  - i.e. os discos estão divididos em partições e os sistemas de ficheiros em volume.



- Usualmente:
  - cada volume do sistema de ficheiros correspondente a uma partição no disco,
  - por vezes as designações "partição" e "volume" são usadas como sinónimas,
  - em alguns casos é possível ter um volume mapeado em mais do que uma partição de um disco

Master Boot Record e Bloco de Boot

- O *Master Boot Record* (MBR) e o bloco de *boot* são entidades fundamentais para localizar e executar um SO.

- O código e os dados de que é composto um SO necessitam, obviamente, de estar armazenados persistentemente.
  - mas, antes do SO se executar não é possível utilizar os mecanismos normais de acesso a ficheiros pois o sistema de ficheiros ainda não foi carregado.

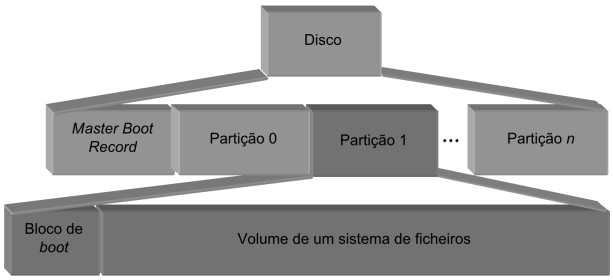
- É necessário um outro sistema de armazenamento alternativo que permita a leitura e execução do SO:
  - independente do tipo de sistema de ficheiros e do tipo de SO,
  - mas pode variar com o tipo de arquitectura do computador, (frequentemente a solução utilizada é semelhante às arquitecturas do tipo IBM-PC).

Master Boot Record e Bloco de Boot (cont.)

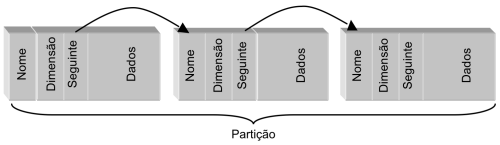
- Código que efectua o *boot* do sistema (parte do que é usualmente denominado BIOS):
  - assume que a informação sobre as partições existentes num disco estejam no início do disco, no MBR.
- O MBR possui código, geralmente independente do sistema operativo:
  - que localiza a partição que contém o sistema operativo a executar e transfere a execução para o código existente no primeiro bloco dessa partição – o bloco de *boot*.
- O bloco de *boot* possui um programa específico de cada SO que:
  - sabe ler o sistema de ficheiros onde o SO se encontra,
  - carregar o sistema operativo e executá-lo.
- Quando só existe uma partição com SO:
  - o código do MBR escolhe imediatamente essa partição para transferir a execução durante a fase de *boot*;
  - quando existem mais partições é usualmente o utilizador que escolhe a partição a executar.
  - neste caso o código existente no MBR diz-se que é um *boot loader* porque permite escolher entre vários sectores de *boot*, de várias partições, para executar.

Estrutura Interna do Sistema de Ficheiros

Entidades constituintes de um dispositivo de memória secundária



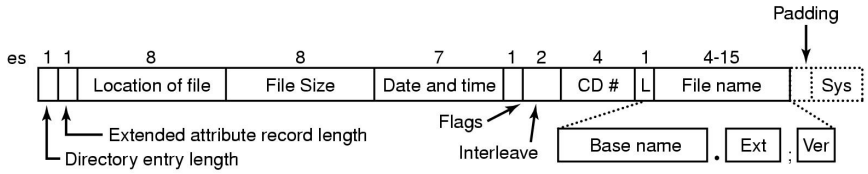
Organização em lista



- Problemas?
  - Tempo para localizar ficheiros
  - Dificuldade em crescimento dinâmico
  - Fragmentação do espaço

Como aliviar este problema?

Sistema de Ficheiros para CD-ROM

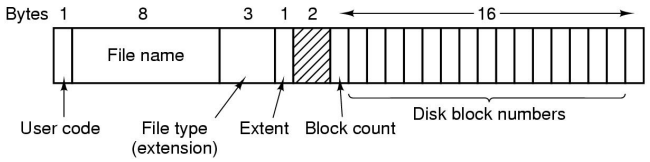


- entrada de uma directoria em ISO 9660
- blocos de 2048 bytes (Modo-2: dados)

- não há gestão de espaço livre/ocupado
- ficheiros dispostos sequencialmente no CD-ROM

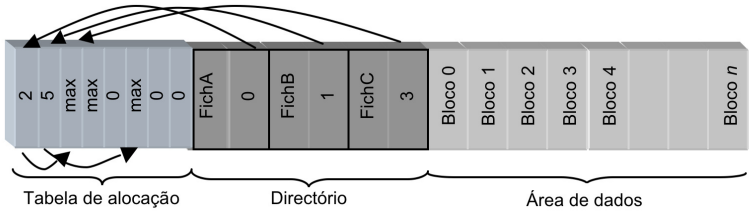
Sistema de Ficheiros do CP/M

- Estrutura de uma entrada do directório do sistema de ficheiros do CP/M:



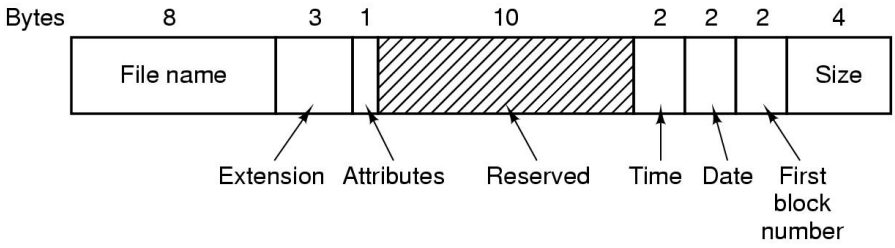
- Mapa de blocos de dados:
  - contém os números dos blocos de dados que fazem parte do ficheiro.
- Neste sistema:
  - cada bloco possuía 1 *Kbyte*
  - cda um dos 16 *bytes* do mapa de blocos continha um índice de bloco
  - logo, a dimensão máxima de um ficheiro era 16 *KBytes*
  - valores claramente insuficientes
  - o maior problema é que para os aumentar é necessário aumentar o mapa de blocos de todas as entradas do directório (penalizando assim os ficheiros de reduzida dimensão)
  - i.e. os ficheiros mais pequenos continuariam a utilizar um mapa de blocos grande ainda que dele não necessitassem, desperdiçando assim memória.

Estrutura de um sistema de ficheiros do tipo do sistema FAT



- A tabela de alocação é partilhada por todos
- Ficheiros grandes ocupam mais espaço na tabela de alocação
- Solução usada desde MS-DOS a Win98

Sistema de Ficheiros MS-DOS (1)



entrada de 32 bytes de directoria em MS-DOS

- Usa uma FAT: 12, 16 ou 32 (28 bits)

Desvantagens do FAT

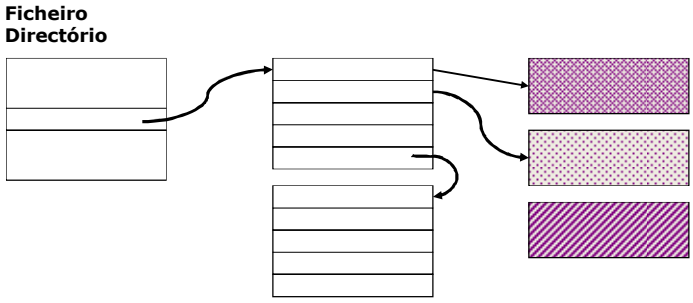
- Elevada dimensão da tabela de alocação quando os discos têm dimensões muito grandes:
  - e.g., com disco de 500 *GBytes*, blocos de 4 *KBytes* e ponteiros de 32 bit, a tabela de Alocação pode ocupar 500 *MBytes* (500G/4K × 4 *bytes*).
- Tabelas desta dimensão não são possíveis de manter em memória principal permanentemente:
  - é necessário recorrer à paginação
  - é ainda necessário percorrer a cadeia de blocos de um ficheiro sempre que se pretende localizar um bloco (degradação de desempenho significativa)

Sistema de Ficheiros MS-DOS (2)

Dimensão Bloco	FAT-12	FAT-16	FAT-32 (28)
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		1 GB	4 TB (2 TB)
32 KB		2 GB	8 TB (2 TB)

- Partição máxima para diferentes tamanhos de blocos
- As entradas a branco indicam combinações impossíveis (não permitidas)
- Apenas 28 dos 32 bits de cada entrada da FAT-32 são utilizados
- O sector de boot impõe duas outras limitações
  - Número de sectores por bloco: 8 bit
  - Número total de sectores no disco 32 bit

Organização dos Blocos de Dados
Blocos de Índices



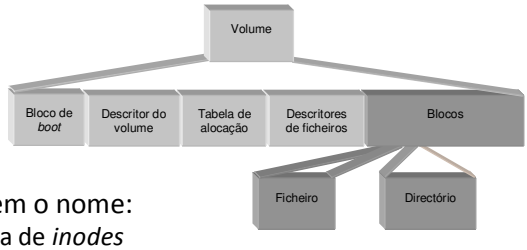
- Os blocos de índices são guardados em blocos de dados e só são acedidos quando necessário
- Adapta-se a qualquer dimensão do disco
- A estrutura de índices pode ser hierarquizada para optimizar o acesso directo em ficheiros de grande dimensão – **solução do Unix**

Ficheiros descritos em directório vs. Ficheiros com descritor autónomo

- Todos os meta-dados de um ficheiro na entrada do directório a que o ficheiro pertence é bom?
- Alternativa:
  - Descrição do ficheiro mantidas em estrutura própria
  - Chamada inode, identificada por inumber (único no volume todo)
  - Mantida em tabela de inodes
- Entrada de directório manteria apenas: [nome do ficheiro, inumber, tipo de ficheiro]
- Vantagens?

Organização com Descritores Individuais de Ficheiros (cont.)

- Os descritores dos ficheiros são guardados:
  - numa estrutura especial de tamanho fixo antes dos blocos de dados



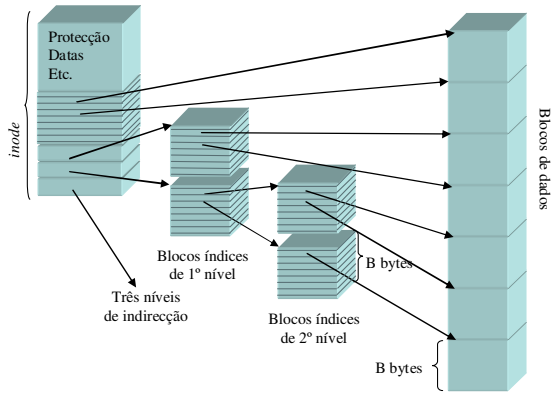
- No Linux tem o nome:
  - de tabela de *inodes*
- No Windows tem o nome:
  - MFT (*Master File Table*).
- O número máximo de ficheiros numa partição é dado pelo número máximo de descritores nessa tabela.

Organização com Descritores Individuais de Ficheiros (cont.)

- Um ficheiro é univocamente identificado:
  - dentro de cada partição,
  - pelo número do seu descritor, dada a relação biunívoca entre um descritor de ficheiro e o ficheiro que descreve.
- Os directórios só têm que:
  - efectuar a ligação entre um nome do ficheiro e o número do seu descritor
- Quer no EXT3, quer no NTFS:
  - os directórios não são mais do que ficheiros em que cada registo é um tuplo de tamanho variável contendo o nome do ficheiro e o número do seu descritor

	Número do Inode	Dimensão do Registo		Dimensão do nome	Tipo	Nome							
0	54	1	2	1	2	.	\0	\0	\0				
12	79	1	2	2	2	.	.	\0	\0				
24	23	1	6	6	1	c	a	r	l	o	s	\0	\0
40	256	1	6	7	1	m	a	r	q	u	e	s	\0

Ext



dimensão máxima de um ficheiro = $B \times (12 + B/R + (B/R)^2 + (B/R)^3)$

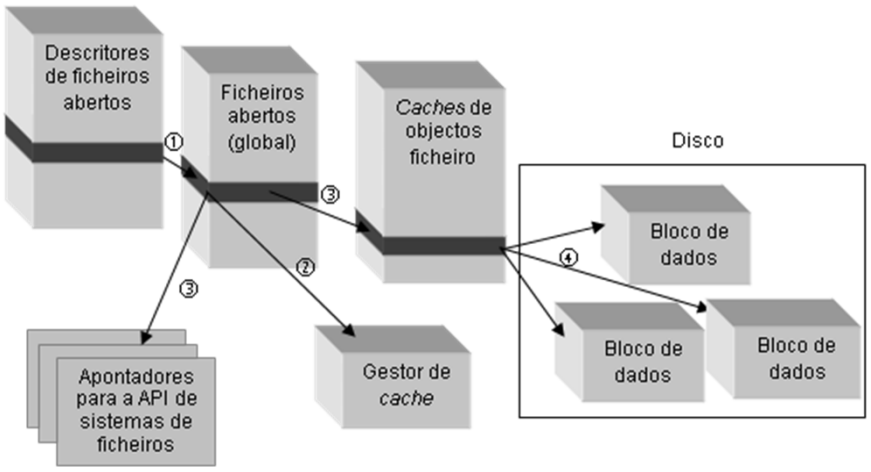
B é a dimensão em bytes de um bloco de dados
R é a dimensão em bytes de uma referência para um bloco
Com blocos de 1 Kbyte e referências de 4 byte, a dimensão máxima de um ficheiro é ~16 Gbyte

Estruturas de Suporte à Utilização dos Ficheiros

- Todos os sistemas de ficheiros definem um conjunto de estruturas de memória para os ajudar a gerir a informação persistente mantida em disco.

- Objectivos:
  - Criar e gerir os canais virtuais entre as aplicações e a informação em disco;
  - Aumentar o desempenho do sistema mantendo a informação em *caches*;
  - Tolerar eventuais faltas;
  - Isolar as aplicações da organização do sistema de ficheiros e, nalguns casos, possibilitar a gestão de várias organizações de estruturas de ficheiros em simultâneo.

Estruturas de Suporte à Utilização dos Ficheiros



Estruturas de Suporte à Utilização dos Ficheiros (cont)

- Tabela de ficheiros abertos – por processo:
  - contém um descritor para cada um dos ficheiros abertos
  - mantida no espaço de memória protegida pelo que só pode ser acedida pelo núcleo
- Tabela de ficheiros abertos – global:
  - contém informação relativa a um ficheiro aberto
  - mantida no espaço de memória protegida pelo que só pode ser acedida pelo núcleo
- A existência de duas tabelas é fundamental para:
  - garantir o isolamento entre processos
  - permitindo a partilha de ficheiros sempre que necessário (e.g. os cursores de escrita e leitura de um ficheiro entre dois ou mais ficheiros)
- Note-se que:
  - os identificadores para a tabela global estão na tabela privada que está em memória protegida, pelo que não podem ser alterados

Exercício (2º teste 2008/09)

3. [2 val.] Considere a seguinte sequência de acontecimentos, em que 2 processos P1 e P2 acedem aos ficheiros indicados na ordem indicada:

```
P1: f = open ("/users/cnr/dir/f1"); read (f, buf, 50);
P2: g = open ("/users/cnr/dir/f1"); read (g, buf, 100);
P2: h = open ("/tmp/tempfile");
P1: p = fork(); /* cria um processo filho P3 */
P1: read (f, buf, 150);
P3: read (f, buf, 200);
```

Represente graficamente as tabelas de *file descriptors* por processo, a tabela de ficheiros abertos global do sistema e a tabela de descritores de ficheiros (*inodes*) após a sequência de acontecimentos anterior.

Inclua o valor dos índices que indicam a posição (i.e. os cursores).

Sistema de Ficheiros Linux

- Introdução
- Estruturas persistentes
- Estruturas voláteis

Introdução

- A primeira versão foi o EXT que segue as ideias do BFS
- Com a introdução do EXT2, apareceu o VFS
  - Que permite vários sistemas de ficheiros (FAT, NTFS, EXT2, etc.)
- O SF vê os discos como vectores de blocos.
- O sistema de ficheiros só descreve a organização dos ficheiros pelos blocos, a escrita e leitura dos blocos é efectuada pelos gestores dos dispositivos.
- O SF gera a metadata do sistema de ficheiros
  - Metadata persistente para todos os ficheiros
  - Metadata volátil para manter informação de ficheiros abertos

Sistema de Ficheiros do Linux

Operações	Genéricas	Linux
Simples	Fd := Abrir (Nome, Modo) Fd := Criar (Nome, Protecção)	int open(const char *path, int flags, mode_t mode)
	Fechar (Fd)	int close(int fd)
Ficheiros Abertos	Ler (Fd, Tampão, Bytes)	int read(int fd, void *buffer, size_t count)
	Escrever (Fd, Tampão, Bytes)	int write(int fd, void *buffer, size_t count)
	Posicionar (Fd, Posição)	int lseek(int fd, off_t offset, int origin)
Complexas	Copiar (Origem, Destino)	int symlink(const char *oldpath, const char *newpath) int link(const char *oldpath, const char *newpath)
	Mover (Origem, Destino)	int rename(const char *oldpath, const char *newpath)
	Apagar (Nome)	int unlink(const char *path)
		int dup(int fd), int dup2(int oldfd,int newfd)
	LerAtributos (Nome, Tampão)	int stat(const char *path, struct stat *buffer)
	EscreverAtributos (Nome, Atributos)	int fcntl(int fd,int cmd,struct flock *buffer) int chown(const char *path, uid_t uid, gid_t gid) int chmod(const char *path, mode_t mode)
Ficheiros em memória	MapearFicheiro(Fd,pos,endereço,dim)	void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset)
	DesMapearFicheiro(endereço,dim)	int munmap(void *addr, size_t len)
Directórios	ListaDir (Nome, Tampão)	int readdir(int fd, struct dirent *buffer, unsigned int count)
	MudaDir (Nome)	int chdir(const char *path)
	CriaDir (Nome, Protecção)	int mkdir(const char *path, mode_t mode)
	RemoveDir(Nome)	int rmdir(const char *path)
Sistemas de Ficheiros	Montar (Directório, Dispositivo)	int mount(const char *device, const char *path, const char *fstype, unsigned long flags, const void *data)
	Desmontar (Directório)	int umount(const char *path)

ESTRUTURAS PERSISTENTES

i-nodes

- Estruturas persistentes
- Contêm a identificação dos blocos no disco que fazem parte de um ficheiro.
- Existe um e só um i-node por ficheiro.
- Existem muitos tipos de i-node
  - Ext2, VFS, BSD
  - Todos têm estruturas diferentes mas têm o mesmo objectivo.

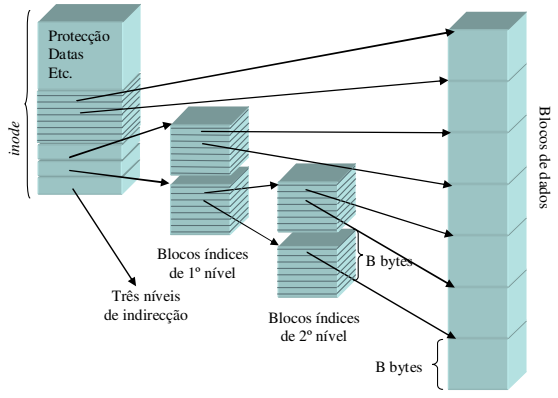
Estruturas Persistentes EXT2

- Essencialmente de dois tipos:
  - estruturas de suporte à definição de ficheiro - contêm os meta-dados relativos a cada ficheiro
  - estruturas de suporte ao sistema de ficheiros - contêm os meta-dados do sistema de ficheiros global.

- Inode:
  - descritor de cada ficheiro designa-se por *index-node* ou *inode*
  - há inodes que não estão associados a ficheiros (ver I/O)

Campo	Descrição
i_mode	Tipo de ficheiro e direitos de acesso
i_uid	Identificador do utilizador
i_size	Dimensão do ficheiro
i_atime	Tempo do último acesso
i_ctime	Tempo da última alteração do inode
i_mtime	Tempo da última alteração do ficheiro
i_dtime	Tempo da remoção do ficheiro
i_gid	Identificador do grupo do utilizador
i_links_count	Contador de hard links
i_blocks	Número de blocos ocupado pelo ficheiro
i_flags	Flags várias do ficheiro
i_block[15]	Vector de 15 unidades para blocos de dados
	Outros campos ainda não utilizados

Inodes

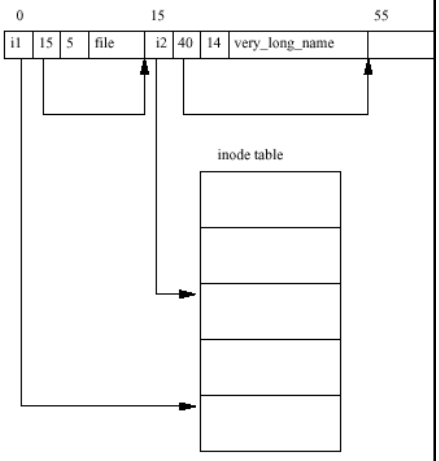


dimensão máxima de um ficheiro = $B \times (12 + B/R + (B/R)^2 + (B/R)^3)$

B é a dimensão em bytes de um bloco de dados
R é a dimensão em bytes de uma referência para um bloco
Com blocos de 1 Kbyte e referências de 4 byte, a dimensão máxima de um ficheiro é ~16 Gbyte

Directório no Ext2

- Um directório é um ficheiro como os restantes, que tem uma estrutura específica
- Um ficheiro do tipo directório contém um vector de entradas, em que cada entrada descreve um ficheiro desse directório.
- Cada entrada contém o nome do ficheiro a sua dimensão e o nº do i-node que o representa.
- Podem existir vários ficheiros com o mesmo i-node. São os *hard links*.
- Os *soft* ou *symbolic links* são ficheiros com o seu i-node com uma estrutura própria.
  - ln **–s** ficheiroAlvo nomeDoLink



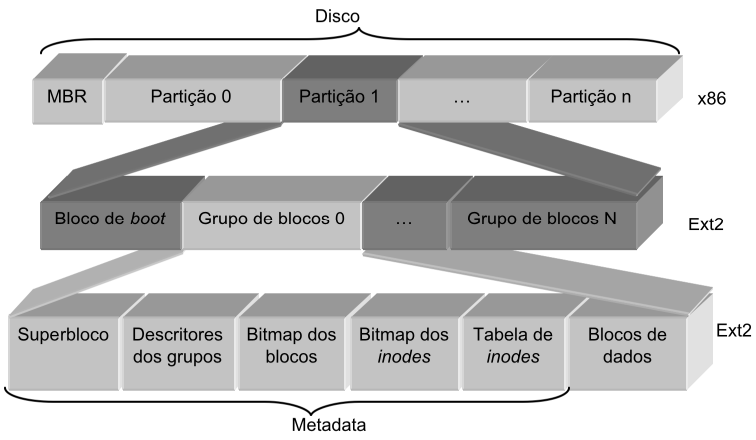
Ficheiro Directório

	Número do Inode	Dimensão do Registo	Dimensão do nome	Tipo	Nome
0	54	1 2	1	2	. \0 \0 \0
12	79	1 2	2	2	. . \0 \0
24	23	1 6	6	1	c a r l o s \0 \0
40	256	1 6	7	1	m a r q u e s \0

Abertura de um ficheiro

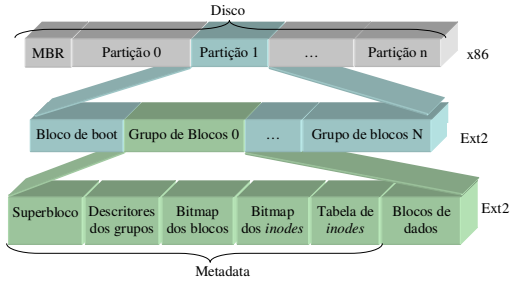
- /home/carlos.ribeiro/.cshrc
  1. Procurar o i-node do directório raiz "/" no superbloco do dispositivo principal.
  2. Obter os blocos desse directório e encontrar a entrada "home"
  3. Ler o i-node do ficheiro tipo directório "home".
  4. Ler os blocos de dados do "home".
  5. Encontrar a entrada "carlos.ribeiro" e ler o seu i-node.
  6. Encontrar a entrada de ".cshrc" e ler o seu i-node e deste os blocos com a informação.

Restantes estruturas persistentes de um EXT2



Grupos e Blocos

- A arquitectura x86 impõe que:
  - um disco possua um MBR no início do disco, e
  - um bloco de *boot* no início de cada partição.
- Para além do bloco de *boot*, uma partição EXT2 é constituída por :
  - vários grupos de blocos de dados
  - cada grupo de blocos de dados contém os meta-dados que descrevem a informação neles contida
    - ex. quais os blocos ocupados, quais os blocos livres, que blocos pertencem a um dado ficheiro, etc.),
  - mais os meta-dados que descrevem a estrutura global da partição
    - (ex. quantos grupos de blocos tem cada partição, quantos blocos tem cada grupo, qual a dimensão dos blocos, etc.).
- Deste modo, os meta-dados que descrevem a estrutura da partição encontram-se replicados pelos vários grupos de blocos e os meta-dados de cada grupo de blocos são guardados perto da informação a que se referem.



Grupos e Blocos (cont.)

- A divisão das partições em grupos de blocos de dados foi introduzida pela primeira vez no *Berkley Fast File System* (BFS):
  - objectivo duplo de melhorar o seu desempenho em discos muito grandes e
  - tornar os sistemas de ficheiros mais tolerantes a faltas típicas dos discos.
- O agrupamento dos meta-dados e dos dados em posições do disco não muito distantes optimiza a pesquisa de informação no disco:
  - é comum aceder-se alternadamente aos meta-dados de um ficheiro e ao seu conteúdo
  - se esta informação estiver muito longe uma da outra, o tempo de latência no acesso à informação no disco é muito maior
- Uma das faltas típicas dos discos é a falha de alguns dos seus sectores, sem que o disco deixe de funcionar completamente:
  - é frequente os sectores inutilizados serem consecutivos, i.e. quando falha um sector a probabilidade do seu vizinho também falhar é maior que as dos restantes sectores
  - com o *BFS*, a falha de um sector que contém os meta-dados de um grupo de blocos só afecta aquele grupo de blocos
  - numa situação sem divisão em grupos uma falha nos meta-dados inutilizaria a partição.

Superbloco

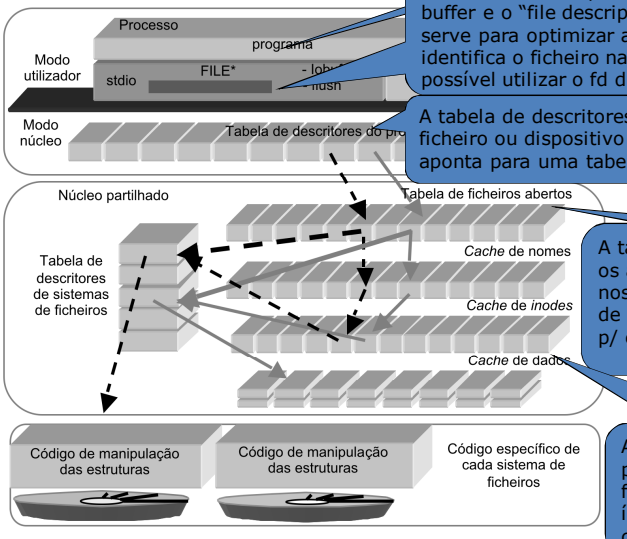
- A informação global sobre todos os grupos de blocos de uma partição está guardada no que é designado por *super bloco*.
- O *super bloco* é:
  - um bloco que contém informação fundamental para a interpretação do conteúdo da partição
    - ex. número de sectores, dimensão dos sectores, número de *inodes*, etc.
  - está replicado em todos os grupos de blocos garantido assim que a falha de um só bloco não impede o disco de funcionar.

Tabelas de Inodes e Bitmaps

- Cada partição tem um número máximo de *inodes*:
  - que servem para armazenar os *inodes* de todos os ficheiros da partição.
  - logo, existe um número máximo de ficheiros, correspondente à dimensão máxima da tabela de *inodes*
- Dentro de uma partição:
  - um *inode* é univocamente identificado pelo índice dentro da tabela de *inodes.*
- Para além da tabela de *inodes* existem em cada partição ainda duas outras tabelas:
  - o *bitmap* de *inodes* - quais as posições livres
  - o *bitmap* de blocos, - quais as posições ocupadas

ESTRUTURAS VOLÁTEIS

Estruturas voláteis: vi



O programa contém um ponteiro para uma estrutura do tipo FILE.

A FILE é mantida pela biblioteca stdio e contém um buffer e o "file descriptor" do ficheiro aberto. O buffer serve para optimizar as escritas e leituras. O fd identifica o ficheiro na tabela de descritores. É possível utilizar o fd directamente.

A tabela de descritores contém uma entrada por cada ficheiro ou dispositivo aberto. Cada uma das entradas aponta para uma tabela global no núcleo

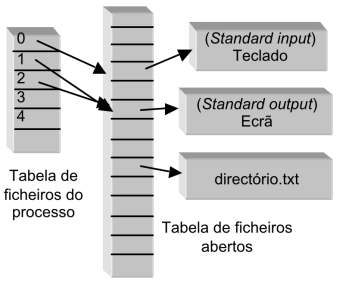
A tabela de ficheiros abertos com os apontadores de escrita e leitura nos ficheiros, a indicação do tipo de acesso (leitura/escrita) e um ptr p/ cache de nomes

A cache de i-nodes é uma cache para os blocos de índices dos ficheiros acedidos. Cada bloco de índices referencia os blocos do disco que compõem um ficheiro.

Tabela de ficheiros abertos

- A tabela de ficheiros abertos contém uma entrada para cada ficheiro aberto
  - podem existir mais do que uma entrada para o mesmo ficheiro. Basta para tal este ser aberto por processos diferentes.
- No fork os ficheiros abertos pelo pai são partilhados pelo filho.
  - As entradas na tabela de ficheiros abertos também são partilhadas. Deste modo os cursores de leitura e escrita são partilhados o que permite direccionar a saída de pai e filho para o mesmo ficheiro sem que o último apague a saída o primeiro.

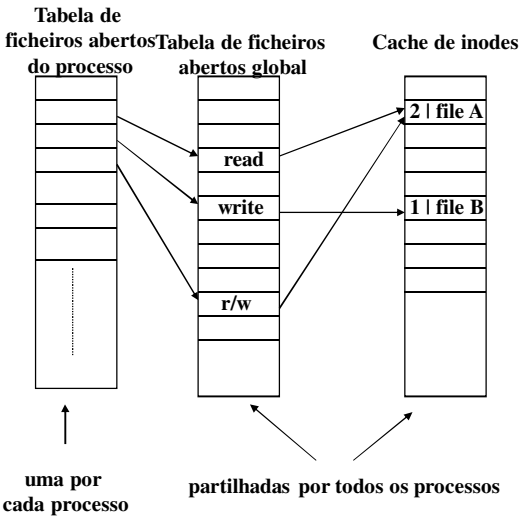
Tabela de Ficheiros abertos



Entrada da tabela de ficheiros abertos

```
struct file {
    struct list_head        f_list;         // Ponteiro para o próximo elemento na lista
    struct dentry           *f_dentry;      // Ponteiro para o objecto dentry associado
    struct vfsmount         *f_vfsmnt;      // Ponteiro para o sistema de ficheiros
    struct file_operations  *f_op;          // Ponteiro para a tabela de despacho
    atomic_t                f_count;        // Número de utilizações do ficheiro
    unsigned int            f_flags;        // Flags especificadas na abertura do ficheiro
    mode_t                  f_mode;         // Modo de acesso
    int                     f_error;        // Código de erro para escrita em rede
    loff_t                  f_pos;          // Posição actual de leitura ou escrita
    struct fown_struct      f_owner;        // Dados para notificação assíncrona
    unsigned int            f_uid, f_gid;   // Id do dono e do grupo
    struct file_ra_state    f_ra;           // Dados para a leitura em avanço
    unsigned long           f_version;      // Versão, incrementada automaticamente
                                            // em cada uso
    void                    *f_security;    // Estrutura de segurança genérica
                                            //(utilizada no SELinux)
    void                    *private_data;  // Necessário para o tty
    struct list_head        f_ep_links;     // Lista de eventos para manipulação
assíncrona
    spinlock_t              f_ep_lock;      // Lock para protecção da lista de eventos
    struct address_space    *f_mapping;     // Ficheiro mapeado em memória
};
```

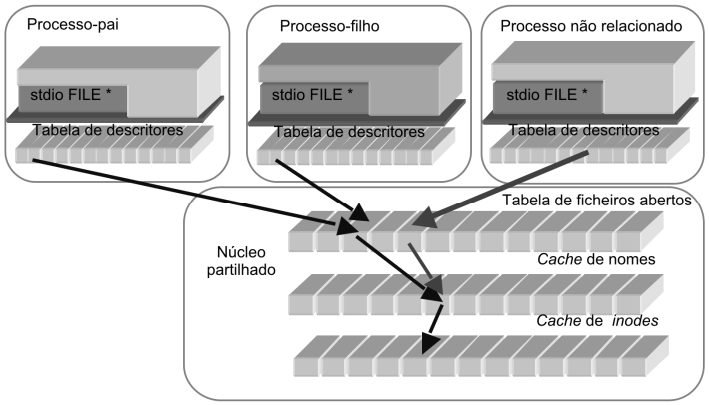

Tabelas de Ficheiros




```
fd1 = open ("fileA", O_RDONLY);
fd2 = open ("fileB", O_WRONLY);
fd3 = open ("fileA", O_RDWR);
```

- *file table* contem:
  - Cursor que indica a posição actual de leitura/escrita
  - modo como o ficheiro foi aberto
- *processos pai e filho:*
  - No fork, filho passa a ter cópia da *tabela de ficheiros abertos do pai*

Tabela Ficheiros Abertos: Fork



O que é específico do Ext2 e o que não é?

- Estruturas persistentes são específicas do Ext2
  - Outro sistema de ficheiros guardará ficheiros, i-nodes, superblocos e directorias de outra forma
    - com estruturas diferentes
    - organizados de forma diferente na partição
- Estruturas voláteis são muito genéricas
  - Adaptam-se a qualquer sistema de ficheiros
- Se cada sistema de ficheiros oferecer biblioteca que sabe aceder às suas estruturas persistentes...
  - Funções para ler/escrever ficheiros, i-nodes, superblocos e directorias na partição desse sistema de ficheiros
- ...As estruturas voláteis poderiam suportar múltiplos sistemas de ficheiros

Virtual File System

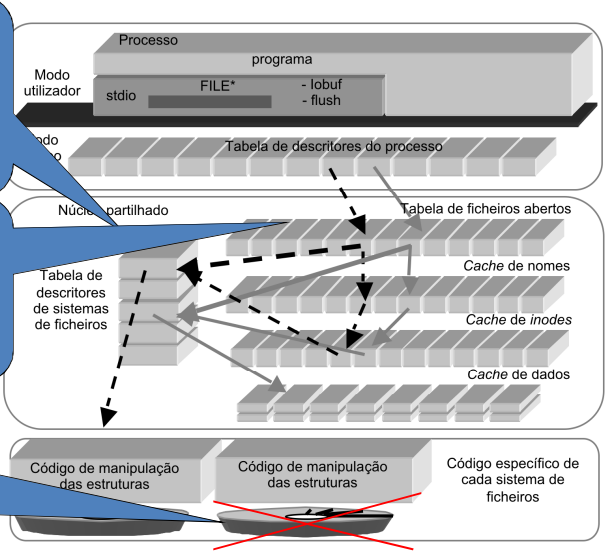
- Permite aceder a vários sistemas de ficheiros diferentes em simultâneo (EXT2, NTFS, FAT, NFS ...)
- Uma única hierarquia de ficheiros composta pelos vários sistemas de ficheiros.
- Facilita a construção de sistemas de ficheiros distribuídos.
- Permite a construção de sistemas de ficheiros virtuais tais como o /proc

Estruturas voláteis do VFS

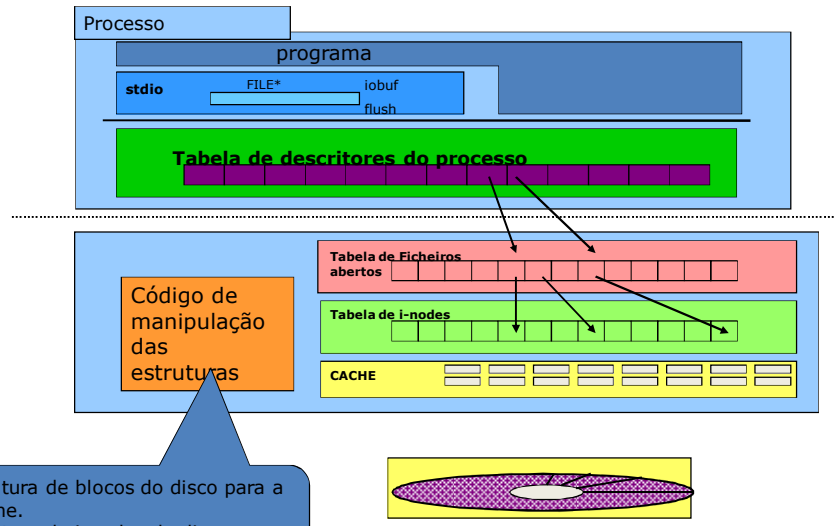
Cada entrada:
- é designada por VFS superblock.
- contém informações sobre cada sistema de ficheiros
- contém um vector de ponteiros para funções que sabem ler o sistema de ficheiros.

Cada entrada da Tabela de ficheiros abertos e cada entrada da tabela de inodes contém um apontador para o vector de funções que sabe manipular o sistema de ficheiros onde este se encontra.

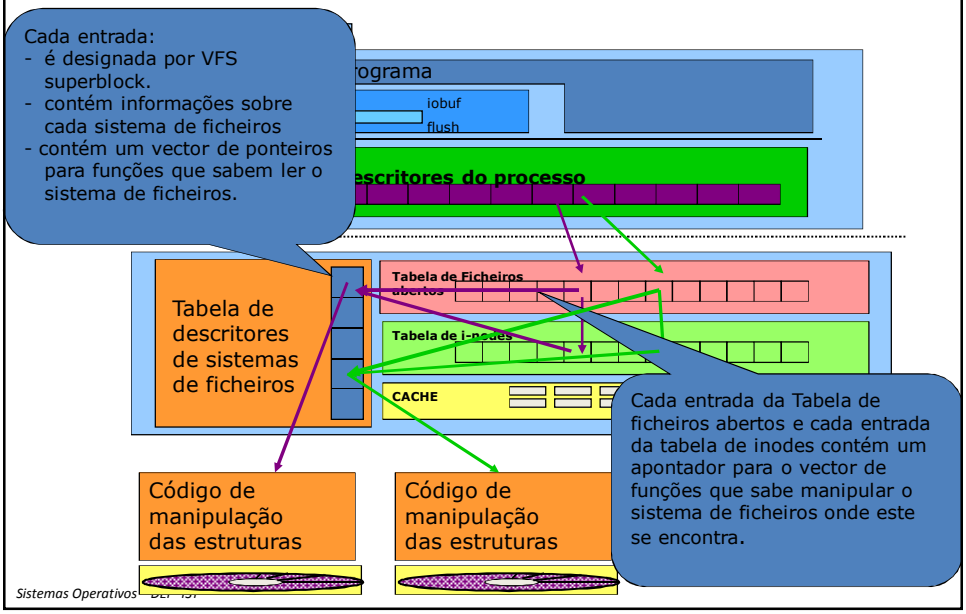
Alguns sistemas de ficheiros não têm suporte físico. A informação é mantida noutros locais e.g. Núcleo, rede



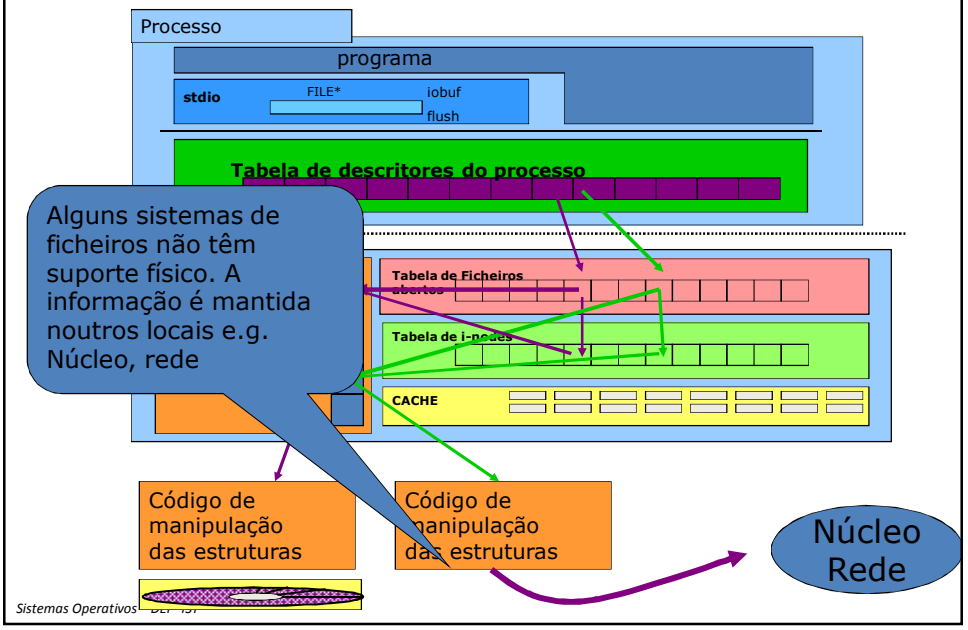
Um único Sistema de Ficheiros



Mais do que um Sistema de ficheiros



Sistemas de Ficheiros virtuais

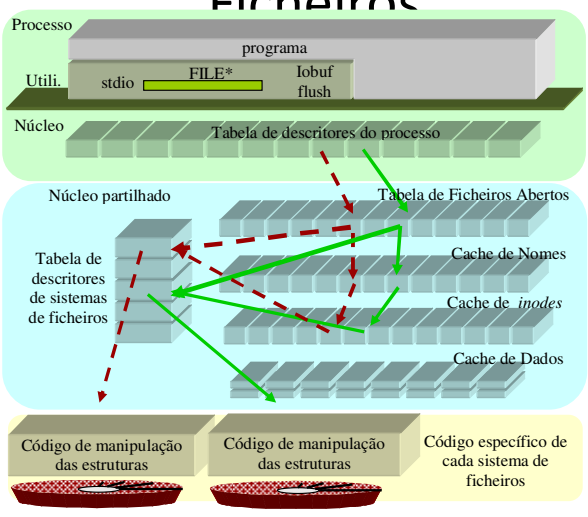


Caching

Caches do Sistema de Ficheiros

- Acessos aos dispositivos de memória de massa:
  - 4 a 5 ordens de grandeza mais lentos que os acessos a memória,
- O desempenho global do sistema de armazenamento é fortemente influenciado pela eficiência das suas *caches*:
  - *cache* com uma taxa de sucesso (*hit-rate)* de 90% / 95% incrementa o desempenho do sistema de armazenamento em cerca de 10 / 50 vezes,
- Redução das escritas e leituras em disco:
  - Atraso da escrita dos blocos em disco
    - Como os blocos são usualmente escritos várias vezes num curto espaço de tempo pelas aplicações, se a sua escrita para disco for atrasada, consegue-se provavelmente efectuar várias actualizações de uma só vez.
  - Atraso na libertação de blocos
    - Um bloco de um ficheiro que não está a ser utilizado não é imediatamente libertado, pois existe a possibilidade de o ficheiro ser aberto de novo. Como veremos adiante, em Linux, os blocos só voltam a ser colocados na lista de blocos livres quando esta desce abaixo de um limiar, de outro modo mantêm-se na cache à espera porque existe a probabilidade de voltarem a ser úteis.
- Existem três tipos de caches no sistema de ficheiros do Linux:
  - Cache de directórios (nomes)
  - Cache de i-nodes
  - Cache de blocos de disco (dados)

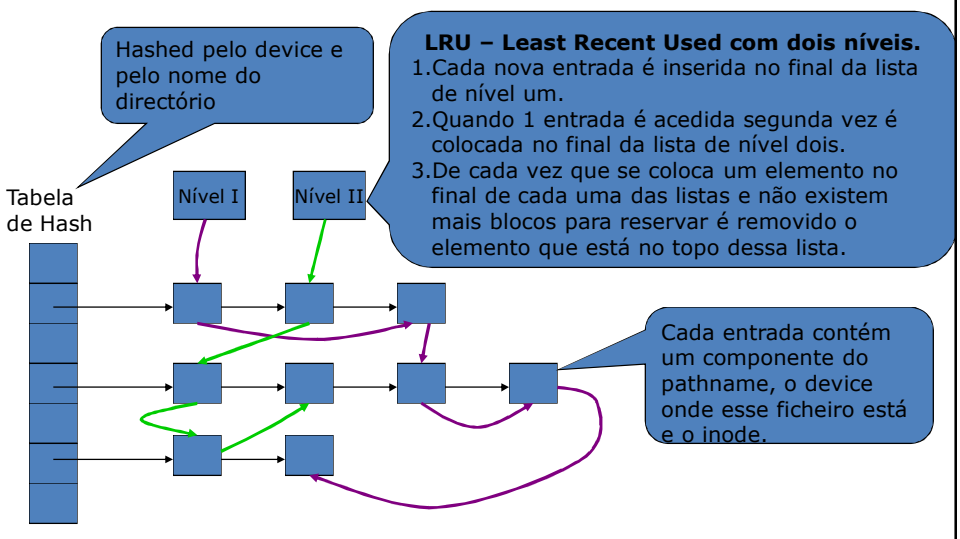
Caches do Sistema de Ficheiros



Cache de Nomes

- Cache de directórios (nomes):
  - associa um nome a um *inode*.
  - também é chamada de *cache* de directórios
  - é preenchida aquando da abertura de um ficheiro, com uma entrada para cada componente do *pathname* que identifica o ficheiro.
- Cada entrada da *cache* de nomes (dentry) tem:
  - o nome desse componente do *pathname*,
  - um apontador para a *cache* de *inodes* onde está o *inode* desse ficheiro, e
  - um apontador para a entrada da *cache* de nomes que tem o componente que o precede, i.e. o directório acima.
  - No caso dos directórios raiz estes apontam para si próprios.
- A *cache* de nomes é particularmente importante para:
  - optimizar as aberturas de ficheiros, pois
  - elimina a necessidade de ler do disco os ficheiros directórios durante o processo de resolução de nomes.

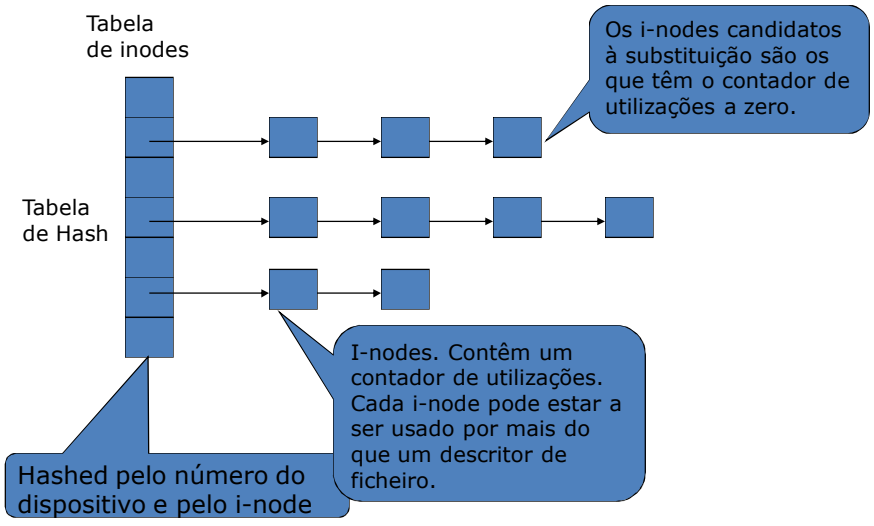
Cache de nomes



Cache de inodes (ou vnodes)

- Conjunto de estruturas em memória:
  - denominadas *vnodes* (*virtual nodes*),
  - indexadas por uma função de dispersão que mantém os *inodes* dos ficheiros abertos.
- Para além dos ficheiros abertos a *cache* de *inodes* pode conter:
  - ficheiros que estão fechados mas que estiveram abertos recentemente e podem voltar a ser abertos em breve.
  - permite reduzir as operações de leitura dos sistemas de memória persistente, pois evita a releitura desses *inodes*.

Cache de i-nodes



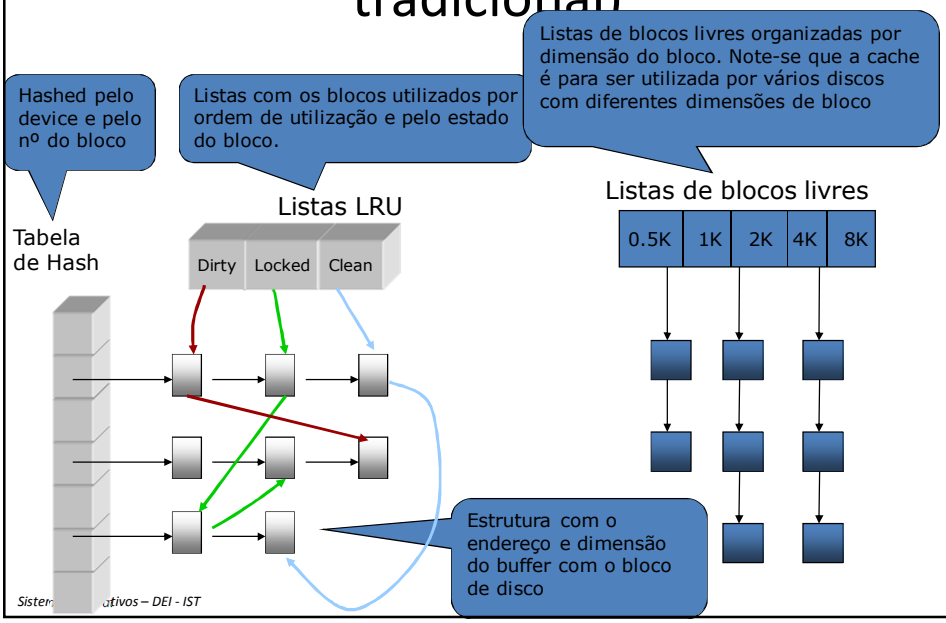
Cache de Blocos (1/2)

- O núcleo poderia ler e escrever directamente para o disco em todos os acesso a ficheiros:
  - implicaria elevados tempos de resposta do sistema devido aos tempos de acesso ao disco
- Para melhorar o desempenho:
  - minimizar os acessos ao disco através de uma cache que contém os blocos que foram recentemente acedidos
  - as rotinas de leitura e escrita analisam os blocos na cache antes de acederem ao disco
- Dois níveis de cache:
  - biblioteca de I/O que adapta as operações de leitura/escrita ao tamanho dos blocos em disco
  - zona de memória entre os processos e os gestores dos discos

Cache de Blocos (2/2)

- Cache para escrita/leitura em disco:
  - permitir manter em memória blocos de dados que possam ser reutilizados pelos processos
  - é constituída por blocos de memória em número que é um parâmetro de geração dos sistema
  - os blocos da cache têm dimensão igual à dos blocos em disco
  - os blocos na cache contém informação resultante de leituras/escrita anteriores do/para disco
- Cada bloco da cache é representado por:
  - identificador do bloco
  - estado
  - apontador para os dados
  - apontadores que permitem a sua inclusão em listas duplamente ligadas
- Existem três listas ligadas que correspondem a três estados diferentes da entrada da cache:
  - Se o conteúdo da entrada é igual ao conteúdo do bloco existente em disco, então diz-se que a entrada está Limpa (**Clean**).
  - Se a entrada contém dados que foram escritos pelo sistema mas que ainda não foram actualizados no disco, diz-se que a entrada está Modificada (**Dirty**).
  - Se a entrada estiver actualmente a ser alvo de uma transferência de/para o disco então diz-se que a entrada está Reservada (**Locked**).

Cache de blocos (Unix tradicional)

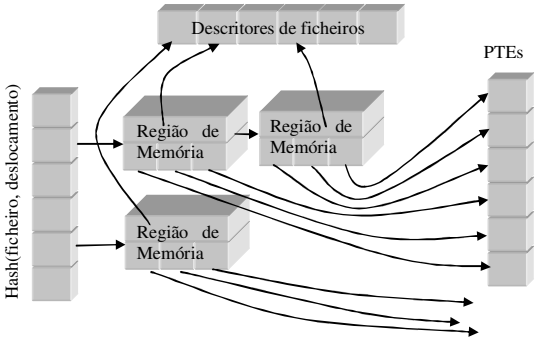


Cache de blocos (actualmente)

- As *caches* dos sistemas operativos actuais são *caches* que:
  - para além de usarem o endereço lógico da informação a aceder (i.e. identificador do ficheiro e deslocamento dentro do ficheiro),
  - estão integradas com o espaço de memória virtual de modo a optimizarem a utilização total da memória disponível.
- O espaço disponível para manter blocos de ficheiros em memória é toda a memória disponível:
  - a política de substituição é efectuada pelos algoritmos de substituição de páginas tal como se de outra página de memória virtual se tratasse,
  - com a diferença que a zona de paginação é diferente, i.e. são paginadas para o ficheiro em disco

Cache de blocos (actualmente)

- Apesar da *cache* não possuir uma área de memória específica continua a ser necessário localizar os blocos de um ficheiro.:
  - utilizam-se os descritores de regiões de memória referidos no Cap. 7 para descrever conjuntos de páginas de memória relacionadas entre si.



- Descritores de regiões de memória podem estar em muitas listas:
  - (Cap. 7) são mantidos numa árvore balanceada de modo a serem rapidamente localizáveis dado um endereço.
- No caso particular dos descritores de regiões de memória associados a ficheiros:
  - são também mantidos numa tabela de dispersão global, indexada pelo identificador do ficheiro e deslocamento dentro do ficheiro

Cache de blocos (actualmente)

- Escrita atrasada:
  - quando um processo escreve num ficheiro essa escrita resulta numa alteração de um ou mais blocos em memória, mas não é reflectida de imediato em disco.
  - a escrita em disco é uma operação que tem um desempenho muito baixo relativamente às operações em memória primária,
  - as escritas devem ser minimizadas e agrupadas em blocos contíguos de modo a minimizar o impacto no desempenho global do sistema.
  - a escrita dos blocos para disco é efectuada por uma ou mais tarefas do núcleo que se executam periodicamente.
- Os blocos são escritos para disco em duas situações distintas:
  - quando o número de blocos modificados sobe acima de um determinado nível,
  - quando existem blocos modificados há relativamente muito tempo (usualmente 30 segundos).

Cache e Consistência do Sistema de Ficheiros

- Cache melhora o desempenho global do sistema
- Mas é prejudicial para a garantia de persistência dos dados
  - Porquê?