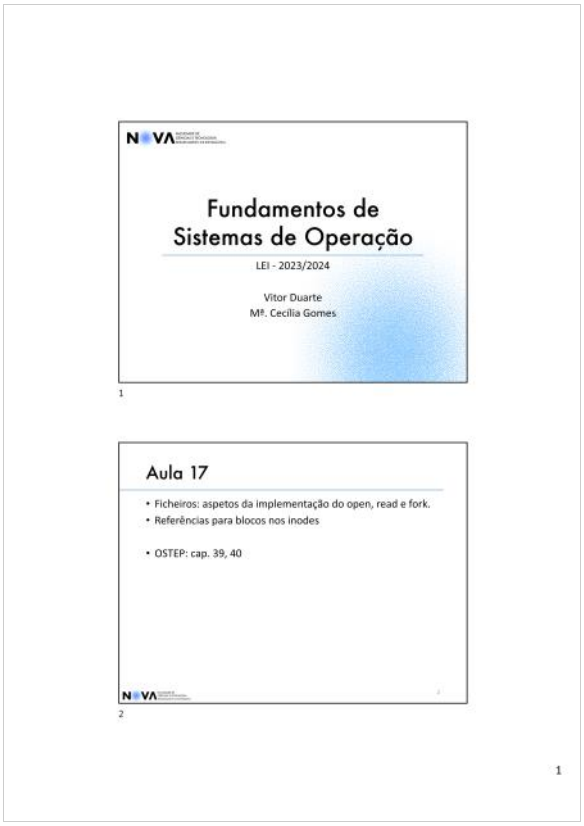
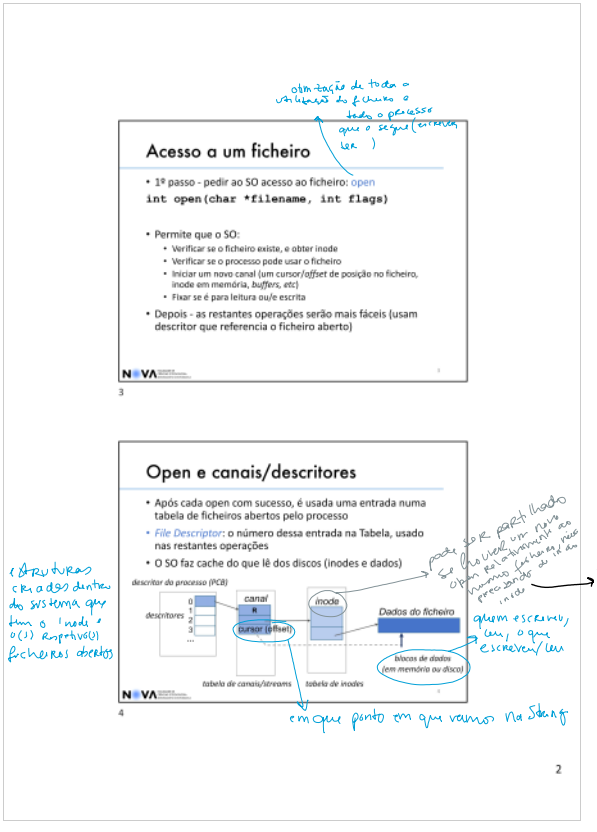
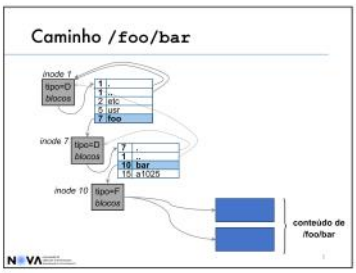
Aula17 - Ficheiros e Referências para blocos nos inodes

10 de novembro de 2023 09:35



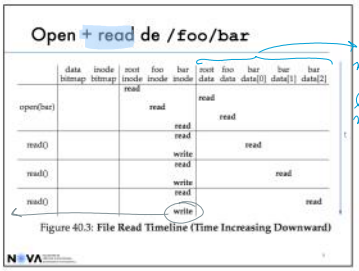


FSO Página 1



5

traço de leituras e escritas durante o open



6

* o user tem sempre que checkar o ID do ficheiro, do grupo, de quem está a abrir e as permissões da diretoria/ficheiro

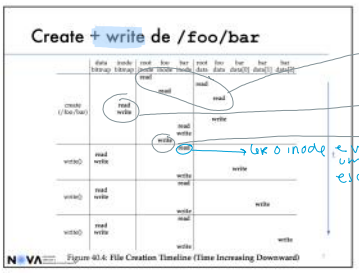
* data e hora do último acesso

dados de x diretoria

ler (abrir) x diretória

os bytes restantes não lhes acontece nada apenas ficam a espera, pois podem ser pedidos a qq momento

3

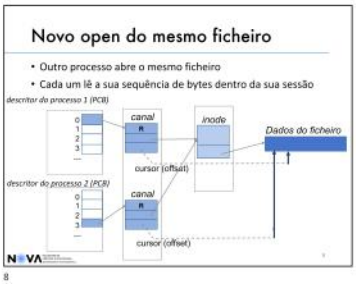


7

processo de procura de ficheiro é igual

caso não existem que ser criado e alterado/associado a data de criação alterar o inode do foo para alterar a data hora da modificação

se não existir um bloco o bitmap vai procurar um bloco livre e utilizá-lo



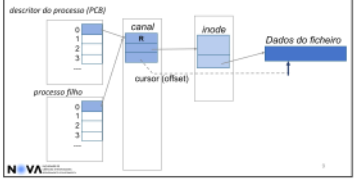
8

4

→ os 2 descritores do open do mesmo ficheiro têm o mesmo inode, mas offset diferentes pois podem estar em locais diferentes do ficheiro

Caso do fork

- O processo faz fork
  - Os processos partilham o canal, logo, a posição no ficheiro

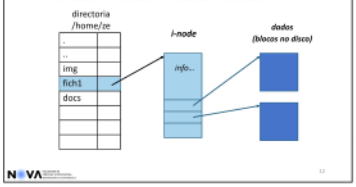


9

→ duplica o descritor do ficheiro, partilha do canal, offset, aquilo que o processo pai lê o filho não lê e vice-versa
→ partilha do Estado Corrente

Indicar os blocos nos inodes

- Referências diretas
  - O inode tem uma tabela com todos os nº dos blocos



12

→ má abordagem, os problemas são os seguintes
↳ n entradas ⇒ n blocos o que delimita o espaço (quantidade de bytes) ⇒ 4 x n = espaço disponível no inode

→ eficiente para IR a um sítio específico dos blocos
→ aumentar o inode, ocupar + espaço no disco ⇒ − inodes
⇒ − nº de ficheiros

5

Capacidades - exemplo

- Se blocos de 4Kbytes,
- se cada inode ocupar 256 bytes, temos 16 inodes por bloco e cada inode tem mais 144 bytes para indexar blocos,
- cada inode pode guardar 36 endereços de 32bits:
- tamanho máximo possível para os ficheiros 144Kbytes e o volume pode ter um máximo de

soma: 112 bytes

13

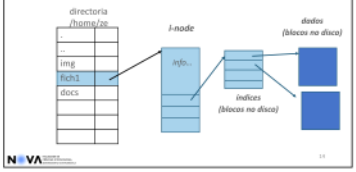
→ para otimizar espaço cria-se subtabelas, tal como as tabelas de páginas e a frameTable

+ 2 níveis

inode remete p/ inode

Indicar os blocos nos inodes

- Referências indiretas (blocos de referências)
  - O inode tem uma tabela com nº dos blocos que contém os nº dos blocos de dados (referências indiretas)



14

→ desvantagem ter que realizar 2 acessos (1 resolver qual é tabela, + 1 para alcançar o bloco de dados que desejamos)
→ acesso direto é mt bom, pois calcula o endereço que deseja através dos blocos de endereço
→ acesso sequencial também é bom

6

* 2 níveis

Capacidades - exemplo

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	link count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file acl	a new permissions model beyond mode bits
4	dir acl	called access control lists

Figure 40.1: Simplified Ext2 Inode

soma: 112 bytes

- Se blocos de 4Kbytes,
- se cada inode ocupar 256 bytes, temos 16 inodes por bloco e cada inode tem mais 144 bytes para indexar blocos,
- cada inode pode guardar 36 endereços de 32bits → 36 índices
- tamanho máximo possível para os ficheiros: 36*4K*4 blocos = 144 Mbytes → tamanho do bloco

quando se usa os 36 blocos do ficheiro temos o tamanho máximo que o ficheiro guarda

15

* mostrar as soluções, entre 1 e 2 níveis

Indicar os blocos nos inodes

- Referências diretas e indiretas
  - O inode tem uma tabela com nº blocos dos dados (diretos) e as duas últimas entradas para um bloco indireto e um bloco duplamente indireto, ...

entradas diretas

últimas entradas são entradas indiretas → indirato (simples) → duplamente indirato

16

7

→ para guardar ficheiros mt grandes é necessário 3 níveis

* mista

Capacidades - exemplo

- Se blocos de 4Kbytes,
- se cada inode ocupar 256 bytes, temos 16 inodes por bloco e cada inode tem mais 144 bytes para indexar blocos,
- cada inode pode guardar 36 endereços de 32bits
- 34 blocos diretos
- + 1K blocos indiretos
- + 1K*1K blocos duplamente indiretos
- tamanho máximo possível para os ficheiros: (34+1K+1M)*4K = 4Gbytes

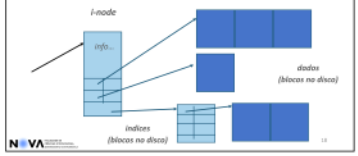
17

→ aumentar o tamanho do inode implica aumentar as tabelas

→ Assim lida imediatamente (diretamente) com ficheiros pequenos e indiretamente com grandes

Indicar os blocos nos inodes

- *Extents:*
  - O inode tem uma tabela com início e fim de sequências contíguas de blocos
  - pode ainda ser combinado com blocos indiretos com extents.



18

8

→ endereço do bloco
→ n de endereços guardados no bloco

→ em vez de se aumentar o inode em si, altera-se o counter do bloco dizendo que em vez de ir do x para y vai de x para z ⇒ estando assim mt ficheiros associados a apenas uma linha no inode

→ desvantagem é não saber, diretamente e rapidamente, a quantidade total/máxima do inode

→ pode ser usado com apenas blocos diretos ou com a abordagem mista ⇒ + eficaz e melhor desempenho

Eficiência do sistema de IO

- Dispositivos independentes uns dos outros e autónomos em relação ao CPU
- Eficiência obtida à custa da sobreposição (overlapping) da execução pelo CPU e pelos dispositivos
  - Necessidade de o CPU responder rapidamente aos pedidos dos periféricos (interrupções)
  - Por outro lado, o tratamento desses pedidos não deve ocupar muito tempo de CPU (DMA, rotinas pequenas)
  - SO deve suportar o assincronismo entre os processos e os dispositivos
    - Utilização e partilha de "buffers/cache", ...
    - Escalonamento de operações de entrada/saída sobre os volumes

os pedidos podem ser escalonados / ou seja, o pedido não vai ser logo realizado

NOVA

20

Periféricos tipo bloco

- Tipicamente discos rígidos – latência elevada, taxa de transferência melhor se sequencial
- Muitas vezes repetem-se acessos ao mesmo bloco
  - Por exemplo, o inode e o bloco da directoria raiz
- Guardar os blocos em memória reduz o número de acessos ao disco
- Atrasar as escritas permite juntar pedidos e escalonar estes para melhor desempenho
- A cache/buffer de blocos (*block buffer cache*) tem duas funções:
  - Reservatório (pool) de buffers para E/S em curso
  - cache para operações de E/S já terminadas
- O *request manager* gere a leitura e escrita de conteúdos de blocos do disco de/para "cache".

NOVA

21

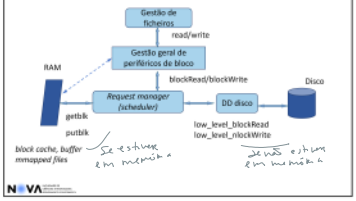
→ o sistema aguarda que tal bloco tem que ser escrito no ficheiro x
→ vários pedidos de escrita são acumulados no mesmo buffer até este ser preenchido e todas as escritas são realizadas
↳ estas escritas podem estar a ser pedidos para ficheiros/processos diferentes e não ser escritos no local que foi pedido ⇒ melhorando a otimização

manter em memória para caso se deseje usar esses dados outra vez e otimizando a procura dos dados

9

* esquematicamente ⇒ gestão de ficheiros

Cache de blocos do disco



NOVA

22

→ escritas são acumuladas até o buffer encher ou de acordo como tiver associado ao gestor ou quando se faz close() do ficheiro

→ Consequência o tempo de complicação é + demorado
o compilador vai criando em memória e quando não é usado é colocado no buffer de espera e realiza-se (wk() e unlink()) apagando ainda em memória o ficheiro cancelando as escritas que estavam em espera

10