FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

1

Aula 19

- Falhas e consistência do sistema de ficheiros.
- Verificação/correção. *Journaling*

- OSTEP: cap. 42

2

Falhas e crashs

1. Os computadores falham
   - Falhas de energia, erros no software, crashs
2. Problemas no hardware, incluindo avarias nos discos
   - Erros em sectores ou pistas; falha total

- Como garantir a persistência do SF ou minorar os problemas?
  - Como atualizar com segurança as ED do SF em disco
  - Ou como lidar com possíveis inconsistências

Problemas com discos

- Historicamente, os discos estão entre os componentes menos fiáveis do sistema:
  - Aparecimento de "bad blocks"
    - Os discos modernos detectam esses erros e têm blocos de reserva que podem substituir os blocos estragados
    - **Mas perde-se o que lá estava**
  - Os sistemas de ficheiros mantêm informação sobre blocos estragados e não os usam
    - Zona nos "metadados" é usada para manter informação sobre blocos estragados
  - Organização em RAID permite tolerar falhas nos discos, especialmente se ligado a *UPS-Uninterruptible power supply*

Problemas no sistema de ficheiros

- A falha de um disco pode ser tolerada por sistema RAID
- As falhas do SO (System crashes) ou falhas de energia (power failures) podem introduzir inconsistências
  - Uma alteração no disco pode ser interrompida em qualquer altura ficando o estado do SF meio alterado
  - É necessário assegurar que se recupera a consistência
  - Os dados que estão a ser modificados podem ser perdidos
  - **Não deve comprometer o sistema de ficheiros completo**

Operações com suboperações

- No exemplo de SF visto, existem três grupos de ED a manipular:
  - Blocos de dados
    - Dados do ficheiro ou diretoria
  - Blocos com inodes
    - Atualizar índices de blocos, datas, tamanho do ficheiro, etc
  - Blocos com bitmaps de blocos e inodes livres
- O escalonamento pelo SO (ou pelo disco) pode alterar a ordem
- Problema: quando uma ou mais ED não são atualizadas → inconsistência

Operações no SF não atómicas

- Remover um ficheiro
- Mudar o nome
- Copiar ficheiro
- Criar um hard link
- Criar um link simbólico
- Criar uma directoria
- Etc…
- **Cada operação exige várias alterações no SF/disco**

Exemplo: Apagar um ficheiro

1) determinar da forma habitual o i-node correspondente, percorrendo os componentes do nome

2) verificar as permissões sobre as directorias e sobre o ficheiro

3) remover a entrada na directoria

4) decrementar o número de referências (links); se nlink>0, terminou

5) libertar os blocos do ficheiro referenciados no i-node

6) libertar o i-node

E se isto é feito parcialmente ?

Ex.: Execução parcial

- Só uma alteração:
  - Remover da diretoria – nome desaparece mas inode e blocos podem continuar ocupados (leak)
  - Escreveu o inode (se nlink>0) – o número de nomes não é igual a este contador
  - Escreveu mapa de blocos – o inode refere blocos livres que podem vir a ser usados
  - Escreve mapa de inodes (se nlink=0) - inode livre mas usado na diretoria, e blocos ocupados (leak)
- Duas alterações:
  - Diretoria e bitmap de inodes (nlink=0) – ficheiro apagado mas blocos em uso (leak)
  - bitmap de inodes e bitmap de blocos – o ficheiro apagado mas nome na diretoria
  - etc
- Etc

Exemplo: Acrescentar um ficheiro

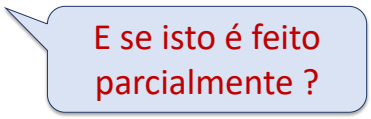
1) usar o i-node correspondente, identificando blocos atribuídos

2) se necessita de um novo bloco procurar no bitmap e atribuir bloco

3) atualizar bitmap com novo bloco ocupado

4) atualizar inodo com novo bloco, nova dimensão, etc

5) escrever dados no novo bloco



Ex.: Execução parcial de um write

- Só uma alteração:
  - Escreveu os dados – é como se nada fosse feito
  - Escreveu o inode – o inode refere "lixo" e o blocos podem também ser usados por outro ficheiro/diretoria
  - Escreveu mapa de blocos – perde blocos do disco (leak)
- Duas alterações:
  - Dados e inode – o ficheiro parece bem mas os seu blocos podem ser usados noutro ficheiro/diretoria
  - Inode e bitmap de blocos – o ficheiro fica com "lixo"
  - Dados e bitmap de blocos - perde blocos do disco (leak) pois nenhum inode se refere a este bloco

Garantir o sistema de ficheiros

- Salvaguarda (backup)
  - uso de práticas e programas para guardar (backup) os ficheiros contidos no disco para outro meio de armazenamento (DVD, disco externo, outro computador).
  - Em caso de problema, recuperar (restore) dados perdidos a partir do "backup".
- Verificação de consistência e correção
  - Informação redundante permite verificar a consistência
    - Ex: comparar a estrutura de diretorias com o conteúdo da tabela de inodes e com informação sobre blocos ocupados/livres
  - Resolver as inconsistências para garantir o funcionamento correto
    - eventual perda do que estava em curso
- *SF com Journaling...*

Verificação de consistência

- O SO não pode usar SF inconsistentes→ p.e. testando antes de os usar
  - Marcar no superblock quando é desmontado corretamente
  - Testar no mount ou boot do SO
- Verificar e, eventualmente, corrigir
  - **`chkdsk`** (windows), **`fsck`** (unix)
  - Executado sobre um sistema de ficheiros desmontado
- Verificações pelo fsck:
  - Superblock – consistência da sua informação com o SF, se necessário usar outra cópia noutro grupo
  - Mapas de Blocos e Inodes - consistência dos blocos com as referências nos inodes
  - Consistência da árvore de diretorias e ficheiros, link counts, etc.

Exemplo: ocupação de blocos

- Consistência do mapa da ocupação de blocos:
  - Mapa 1 - construído a partir da tabela de i-nodes (detetando duplicados)
  - Mapa 2 – construído a partir da estrutura que descreve os blocos ocupados
  - Comparar cada bit de ambas as tabelas:

bitmap de blocos em disco	Bitmap a partir dos inodes	
0	0	OK
1	1	OK
0	1	Bloco referido por um ficheiro, mas livre no bitmap. Mudar no bitmap para 1, mas não há garantias de que o inode seja válido (pode vir a mudar).
1	0	Não há nenhum ficheiro que "reclame" o bloco. É inevitável mudar no bitmap no disco para 0!

Exemplo: diretorias e ficheiros

- Percorre a árvore, da raiz para as folhas
- verifica as ligações entre diretorias, assim como "." e ".."
- Constrói uma tabela de contadores por inode
  - Por cada ocorrência do inode *i*, *tab[i]* é incrementado
    - Os hard links contam mas os symbolic links não
- Compara os inodes em uso na *tabela* com o bitmap de inodes livres/ocupados
- Compara cada contador obtido com o contador de referências (nlinks) no inode

Consistência de referências do inode

- Comparação de contadores de referências:
  - Iguais: OK
  - *Inode[i].nlinks > tab[i]*
    - Erro benigno se tab[i]>0;
    - Se tab[i]==0, perdeu o nome, possivelmente foi apagado ou estava a ser criado. Se blocos marcados como ocupados -> cria nome em /lost+found
  - *Inode[i].nlinks < tab[i]*
    - Erro benigno se blocos no inode marcados como ocupados;
    - Se *Inode[i].nlinks ==0* e blocos já dados como livres, temos nome mas não o ficheiro/diretoria -> cria ficheiro/diretoria vazio
- Correção*: Inode[i].nlinks = tab[i]*

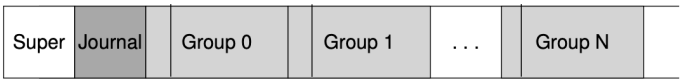
Desvantagens da verificação/correção

- O tempo de verificação e correção aumenta com o tamanho do SF e a número de diretorias e ficheiros
- Percorre **todas** as ED cruzando informação quando a última coisa feita pode ter sido apenas alterar um ficheiro

- *Ideia: manter informação das últimas alterações e verificar/corrigir apenas o que pode estar mal!*

Journaled File Systems (1)

- Os sistemas de ficheiros com registo (*journaling file systems*) registam cada alteração numa zona dedicada do disco, antes de alterar as ED do SF
  - *write-ahead logging* ou *journaling*



  - Este inclui as atualizações dos vários blocos de uma operação, como uma transação
  - Sempre em blocos contíguos e na mesma zona do disco
  - Pode manter só as últimas efetuadas. Remove as já aplicadas ao SF

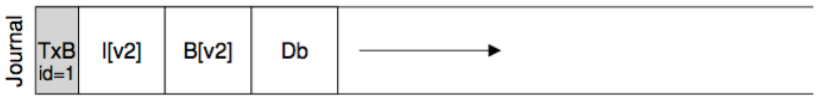
- Uma transação está confirmada (*committed*) quando está completamente escrita no *journal*. Contudo, o sistema de ficheiros ainda não está alterado.

Transações atómicas no SF

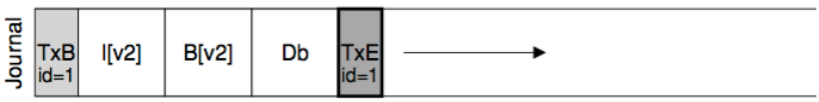
- Objetivo: assegurar que um conjunto de instruções ocorre como uma unidade lógica; ie, ou todas são feitas ou nenhuma é feita (atomicidade)
  - Relacionado com a área de bases de dados
  - O problema é assegurar a atomicidade na presença de falhas do hardware e do software
- Transação – conjunto de instruções que executam como uma operação lógica única (atómica)
  - No nosso caso, diz respeito às alterações feitas ao sistema de ficheiros
  - A transação é um conjunto de operações read e write de blocos
  - Terminará em **commit** (transacção bem sucedida) ou **abort** (transação falhada)
  - Uma transação abortada, tem de ser desfeita (*rolled back*) para desfazer as mudanças por ela feitas
  - o jornal guarda a sequência de escritas da transação

Write novo bloco num ficheiro

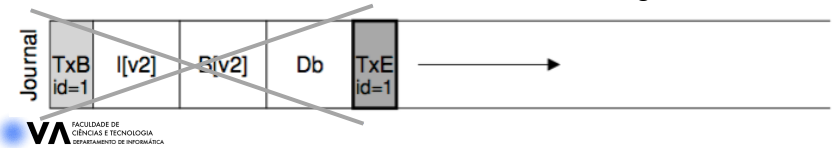
- Journal write:



- Journal commit:



- Checkpoint:
  - escreve no SF e, terminando, dá como livre no log



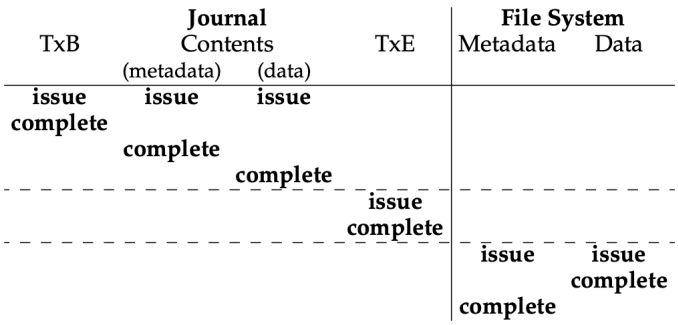


Figure 42.1: **Data Journaling Timeline**

21

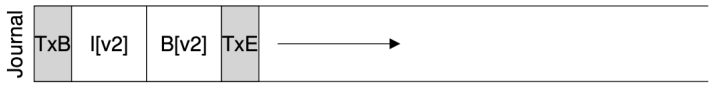
Journaled File Systems (2)

- As instruções são idempotentes
  - Múltiplas execuções produzem o mesmo resultado que uma execução
- Se o sistema falha, todas as transações completas no log são executadas no mount/fsck.
- As incompletas são descartadas
- Vantagem: processamento do log muito mais rápido que o clássico fsck
- Desvantagem: atualizações mais lentas
  - muito mais escritas
  - Em muitos casos, "log" só usado para os metadados

22

Journaled File Systems (3)

- *Journaling* os metadados.
  - A ordem é relevante:
  - atribui bloco e atualiza bitmap e inode em memória
  - escreve dados, logo no local certo
  - escreve metadados no journal e depois faz commit (TxE)



  - mais tarde, atualiza as ED e marca esta parte do journal como livre

	Journal			File System	
TxB	Contents (metadata)		TxE	Metadata	Data
issue	**issue**				**issue**
					complete
complete					
	complete				
			issue		
			complete		
				issue	
				complete	

Figure 42.2: **Metadata Journaling Timeline**