Operating Systems and Concurrency

Lecture 23: File Systems V

COMP2007

Alexander Turner and Geert De Maere

{Alexander.Turner, Geert.DeMaere}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2022

Recap

- File systems implementations
  - Contiguous
  - Linked lists
  - FAT
  - i-nodes
- File systems paradigms (on top of implementations) .
  - Log-structured file systems (improves efficiency)
  - Journaling (improves resiliency, robustness)
  - Virtual File Systems (improves flexibility, integration)

Goals for Today

Overview

- File system recovery
  - Scandisk
  - FSCK
- Defragmenting Disks
- File systems in Linux

File System Consistency

Checking Consistency

- Journaling heavily reduces the probability of having inconsistencies in a file system. In case of crash, the log stores what operations were not run.
- However, it can still be possible to get some inconsistencies (e.g. data blocks weren't flushed to the drive, typical case on USB drives!).
- This can be problematic, in particular for **structural blocks** such as i-nodes, directories, and free lists
- **System utilities** are available to restore file systems, e.g.:
  - Scandisk
  - FSCK
- There are two main consistency checks: block and directory.

File System Consistency

Checking Block Consistency

- Block consistency checks whether blocks are **assigned/used** the correct way
- Block consistency is checked by building **two tables**:
    - Table one counts how often a **block is present in a file** (based on the i-nodes)
    - Table two counts how often a **block is present in the free list**
- A consistent file system has a 1 in either of the tables for each block
- Typically, this is a **very slow process**, taking even hours (and running with the partition unmounted)

File System Consistency

Checking Block Consistency

Block number

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(a)

(b)

(c)

(d)

Figure5–18. File system states. (a) Consistent.

Figure: Consistency checks (from Tanenbaum)

File System Consistency

Checking Block Consistency

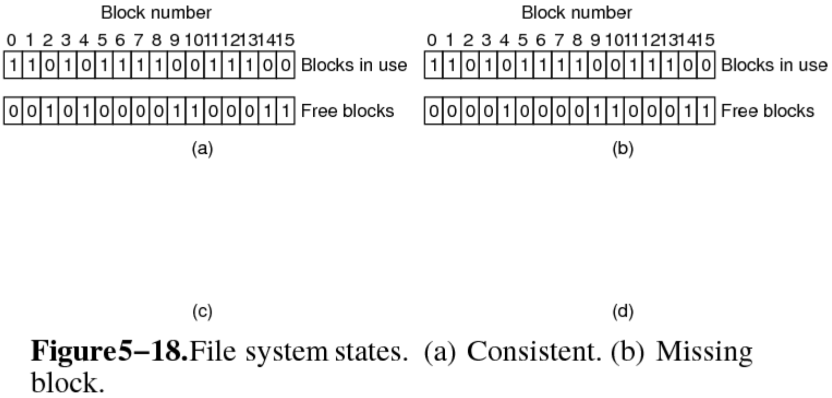

Figure: Consistency checks (from Tanenbaum)

File System Consistency

Checking Block Consistency

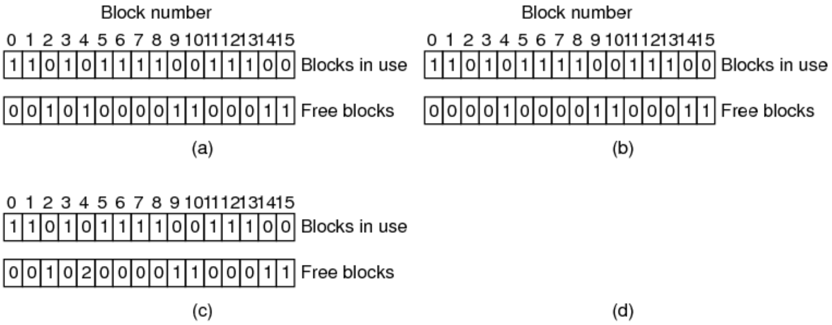

Figure5–18. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list.

Figure: Consistency checks (from Tanenbaum)

File System Consistency

Checking Block Consistency

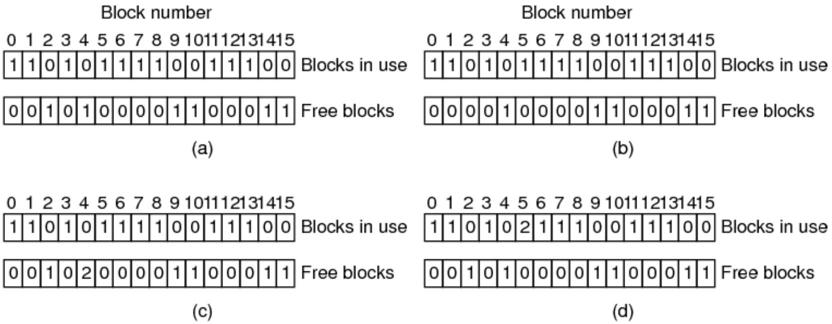

Figure5–18. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data

Figure: Consistency checks (from Tanenbaum)

File System Consistency

Restoring Block Consistency

- A **missing block**: it does not exist in any of the tables $\Rightarrow$ add it to the free list
- A block is **double counted** in the free list ("disaster" waiting to happen) $\Rightarrow$ re-build the free list
- A block is present in **two or more files**
  - Removing one file results in the adding the block to the free list
  - Removing both files will result in a double entry in the free list
  - Solution: use new free block and copy the content (the file is still likely to be damaged)

File System Consistency
Restoring Block Consistency

```
FSCK Algorithm:
 1. Iterate through all the i-nodes
    - retrieve the blocks
    - increment the counters
 2. Iterate through the free list
    - increment counters for free blocks
```

File System Consistency
Restoring I-node Consistency

- Checking the directory system: are the **i-node counts correct**?
- Where can it go wrong?:
  - **I-node counter is higher** than the number of directories containing the file
    - Removing the file will reduce the i-node counter by 1
    - Since the counter will remain larger than 1, the i-node / disk space will not be released for future use
  - **I-node counter is less** than the number of directories containing the file
    - Removing the file will (eventually) set the i-node counter to 0 whilst the file is still referenced
    - The file / i-node will be released, even though the file was still in use

File System Consistency
Restoring I-node Consistency

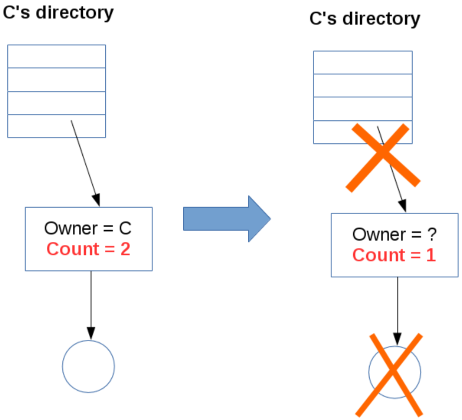

Figure: I-node counter is higher than the actual number of directories containing the file. Removing the file results in wasted memory.

File System Consistency

Restoring I-node Consistency

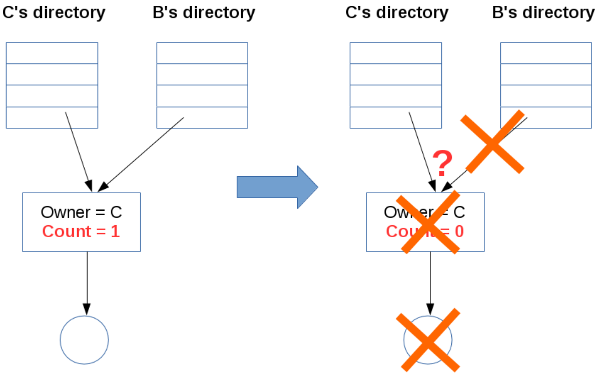

Figure: I-node counter is less than the actual number of directories containing the file. Removing the file results in a missing file.

File System Consistency

Restoring I-node Consistency

- Recurse through the directory hierarchy
  - Check file specific counters
  - I.e. each file is associated with one counter
- One file may appear in multiple directories
  - Compare the file counters and i-node counters
  - Correct if necessary

File System Defragmentation

Compacting

- At the beginning, all free disk space is in a single contiguous unit.
- After a while, creating and removing files, a disk may end up badly fragmented (holes and file all over the place).
- **Defrag** utilities make file blocks contiguous (**very slow operation**), and free space in one or more **large contiguous regions** on the disk.
- Windows users should run this regularly, except on SSDs.
- **Linux (ext2/3) suffers less from fragmentation.**
- Defragmentating SSD is counter-productive (No gain in performance and SSDs wear out).

File System

History of the Linux file system

- **Minix file system**: the maximum file size was 64MB and file names were limited to 14 characters
- The "**extended file system**" (extfs): file names were 255 characters and the maximum file size was 2 GB
- The "**ext2**" file system: larger files, larger file names, better performance
- The "**ext3-4**" file system: journaling etc.

File System

The Extended 2 File System

- The second extended file system (**ext2**) is one of the most popular file systems in Linux.
- The main goals:
  - Improve the **performance** of MINIX and extfs file systems, distributing directories evenly over the disk.
  - Allow **greater file names and sizes**, improving directory implementation.

File System

Standard Unix file system vs. Extended 2 File System

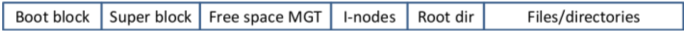

Figure: Standard Unix Partition

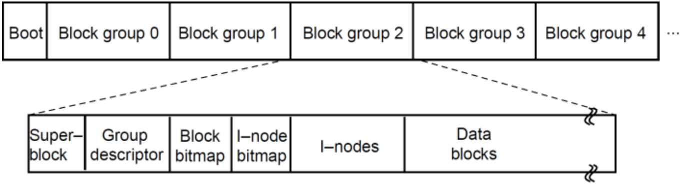

Figure: Ext2 Partition Layout (Tanenbaum)

File System

Directory Entries

- The **superblock** contains file system information (e.g. the number of i-nodes, disk blocks)
- The **group descriptor** contains bitmap locations, the number of free blocks, i-nodes and directories
- A **data block bitmap** and **i-node bitmap**, used to keep track of free disk blocks and i-nodes (Unix uses lists)
- A **table of i-nodes** containing file and disk block information
- **Data blocks** containing file and directory blocks

File System

The Extended 2 File System

- An ext2 partition is split into several **block groups** to:
  - **Reduce fragmentation** by storing i-nodes and files, and parent directories and files in the same block group if possible
  - Reduce **seek times** and improve performance
- All block groups have the same size and are stored sequentially (which allows direct indexing)

Exercises

- **Exercise 1**: Using the ext2 file system (i.e. 12 direct block addresses are contained in the i-node, and up to triple indirect), and assuming a block size of 4 kilobytes, and a 32-bits disk address space.

  - Could we store a file of 18 gigabytes?

  - How many disk blocks we spend for the i-node of a file of 16 megabytes?

- **Exercise 2**: In Linux, how many lookups are necessary to find (and load) the file: /opt/spark/bin/spark-shell?

Exercises

Could we store a file of 18 gigabytes?

- We have blocks of 4 kilobytes, and we need 32 bits (4 bytes) to represent a disk address $\Rightarrow$ in one single block, we could store up to 1024 block pointers:

  4KB = $4*2^{10}$ / 4 bytes each = 2^{10} = 1024 block pointers.
- Using the 12 direct block pointers, we could have a file of with 12 blocks.
- Using the single indirect: 1024 extra block pointers.
- Using the double indirect: 1024*1024 block pointers (1048576).
- Using the triple indirect, 1024*1024*1024 block pointers (1073741824). If we aggregate all of them => 1,074,791,436 blocks of 4KBs which is approx 4TB.

Exercises

How many disk blocks we spend for the i-node of a file of 16 megabytes?

- For a file of 16 megabytes, we need: $16*2^{20}/4*2^{10} = 2^{12} = 4096$ blocks pointers!
- We will use fully all direct block pointers (12) [**1 block**]
- With the single indirect we have 1024 extra block pointers [**1 block**]
- With the double indirect we can address 1048576 block pointer... so we won't go further than this level.
- With direct pointer and single indirect we have covered 1024+12 block pointers... we still need 3060
- Each block will handle 1024 pointers. So, 3060/1024 = 2.9882
- We need three blocks + the "first level" block.
- Total: 6 blocks of 4 kilobytes => 24KB

Exercises

How many disk blocks we spend for the i-node of a file of 16 megabytes? Differently written

- The 12 block pointers in the Inode = one disk block (for the inode really)
  [1 block]
- A single indirect block pointer points to another disk block which holds 1024 block pointers (which is still not enough - we need 4096 block pointers which is 3060 more than we have : 4096 - (12 + 1024)).
  [1 block]
- So we use a double indirect pointer (to get another 3060 block pointers **[1 block]** which points to another disk block, which points to 1024 other disk blocks (each containing 1024 block pointers). We only need 3 **[3 blocks]** of these disk blocks to get out 3060 and take us over the 4096 block pointers needed needed to store the 32mb file.

  [3 blocks]

Summary
Take-Home Message

- File system consistency
- Linux file systems