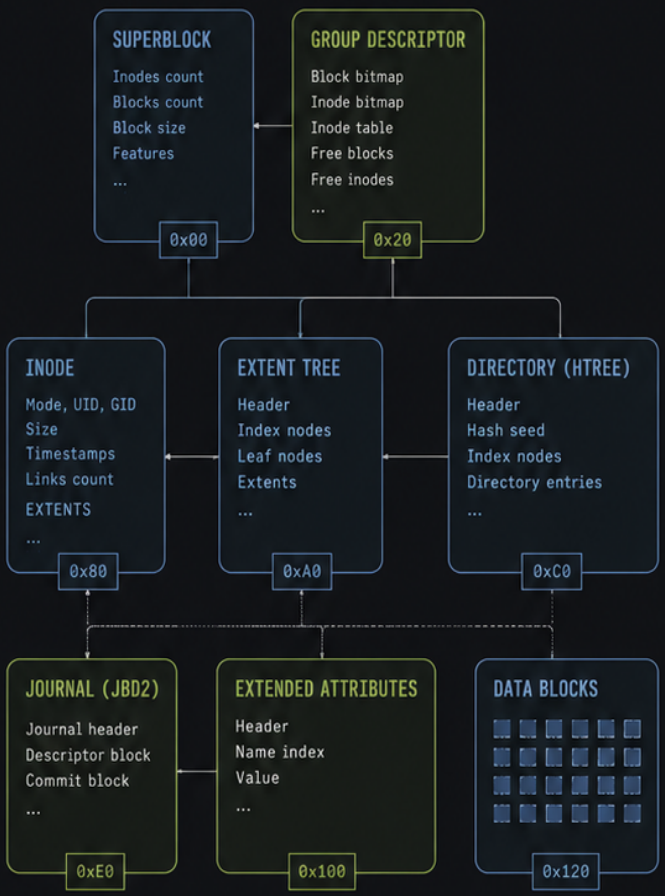
THE LINUX ext4 FILESYSTEM

AN IMPLEMENTATION GUIDE FOR SYSTEMS PROGRAMMERS

- ON-DISK FORMAT FROM FIRST PRINCIPLES
- JOURNALING WITH JBD2
- EXTENTS, ALLOCATION & DIRECTORIES
- ENCRYPTION, VERITY & INTEGRITY
- PARSING, RECOVERY & FORENSICS
- PRACTICAL C AND RUST EXAMPLES



IVAN GEBHART

The Linux ext4 Filesystem

An Implementation Guide for Systems Programmers

Steve T. Publications

This book is available at https://leanpub.com/thelinuxext4filesystem

This version was published on 2026-07-13

Contents

An Implementation Guide for Systems Programmers

About this book: This book is a comprehensive, technically rigorous reference on the Linux ext4 filesystem, written for experienced software developers, systems programmers, operating system developers, and storage engineers. It covers the complete ext4 on-disk format from first principles through advanced implementation details, including historical evolution from ext2 and ext3, superblock and block group structures, inode layouts, extent trees, directory formats with HTree indexing, block allocation algorithms, the JBD2 journaling subsystem, extended attributes, filesystem encryption, verity integrity verification, crash recovery, corruption detection, performance tuning, and comparisons with other filesystems. Every on-disk data structure is documented with byte offsets and field descriptions. The book includes production-quality example code in C (with Rust examples where appropriate), worked parsing examples, implementation guidance, and extensive cross-references to the Linux kernel source. Whether you are building a filesystem driver, a forensic tool, a bootloader, or simply need deep understanding of how ext4 works internally, this book provides the knowledge and practical guidance to do it right.

Introduction

In 1992, a graduate student named Remy Card wrote a filesystem for Linux that would, through successive generations, become one of the most widely deployed filesystems in computing history. The original ext filesystem was simple, functional, and born of necessity. Its successor, ext2, arrived in 1993 and became the default Linux filesystem for nearly a decade, powering everything from embedded devices to supercomputers. In 2001, ext3 added journaling to ext2 with a single backward-compatible mount option. Then came ext4 in 2008, introducing extents, multi-terabyte support, and delayed allocation while maintaining the ability to mount ext2 and ext3 filesystems without conversion.

Today, more than thirty years after the original ext was written, ext4 remains the default filesystem for most Linux distributions. It runs on billions of devices, from Raspberry Pis to data center servers. The kernel source tree dedicates thousands of lines of C code to its implementation, and the e2fsprogs project provides the userspace tools that create, check, and repair ext4 filesystems.

Who This Book Is For

This book assumes you already understand the basics of operating systems: what inodes are, how directories work, the difference between a block device and a character device, and how virtual filesystem layers abstract storage. If you have written a driver, debugged a kernel panic, or traced a disk I/O path through strace, this book will feel familiar.

If you are building an ext4 reader for a bootloader, writing a forensic parser, implementing ext4 support in a new operating system, or simply want to understand the code you see when you read fs/ext4/ in the kernel source, this book is designed for you. It is not a tutorial on how to use Linux commands. It is a reference on how ext4 works at the binary level, with enough detail that you could implement a fully functional ext4 driver from these pages alone.

What You Will Learn

By the end of this book, you will be able to:

- Read and interpret any ext4 superblock, group descriptor, inode, extent tree, directory block, or journal transaction directly from raw disk bytes.
- Understand the complete on-disk layout of an ext4 filesystem, from the boot sector through every block group.
- Implement a reader that can traverse directories, follow extent trees, and reconstruct file contents from a raw ext4 image.
- Understand how the JBD2 journaling subsystem works, including transaction commit, block revocation, and crash recovery replay.
- Work with extended attributes, POSIX ACLs, inline data, encrypted files, and case-insensitive directories.
- Diagnose filesystem corruption, interpret e2fsck output, and understand the trade-offs between different error-handling policies.
- Compare ext4's design choices against XFS, Btrfs, ZFS, NTFS, and FAT/exFAT with informed technical arguments.

How to Use This Book

The book is organized to build knowledge progressively. Chapters 1 through 3 establish the historical context and the fundamental on-disk structures. Chapters 4 through 8 cover the dynamic data structures: directories, extents, allocation, journaling, and extended attributes. Chapters 9 through 12 address advanced features, VFS integration, recovery, and performance. The final chapters provide implementation guidance with working code examples.

Every chapter references specific kernel source files. When you see a reference like fs/ext4/super.c, that is the actual file in the Linux kernel source tree. You are encouraged to read alongside this book, opening the kernel source and cross-referencing the structures described here with their definitions in include/linux/ext4.h and the implementation in fs/ext4/.

The official Linux kernel documentation at Documentation/filesystems/ext4/ (now docs.kernel.org/filesystems/ext4/) serves as the primary authoritative reference for on-disk formats. This book builds upon that documentation, adding context, examples, code, and analysis that the kernel docs do not provide.

Conventions

Throughout this book, kernel source file paths are shown in monospace, like fs/ext4/extents.c. Structure names from the kernel headers appear as struct ext4_inode. Byte offsets are given in hexadecimal. When discussing on-disk formats, all multi-byte integer fields use little-endian byte order for ext4 metadata and big-endian for JBD2 journal data, unless explicitly noted otherwise.

Code examples are primarily in C, the language of the kernel itself. Rust examples appear where they offer additional clarity, particularly in safety-critical parsing code. All code is intended to be production-quality: it handles endianness conversion, validates checksums, checks bounds, and returns appropriate error codes.

Chapter 1: The Story of ext

The ext4 filesystem did not appear fully formed. It is the product of more than three decades of incremental improvement, driven by real-world failures, scaling challenges, and the relentless growth of storage capacity. Understanding this evolution is essential for implementation because many ext4 structures exist solely for backward compatibility with ext2 and ext3, and understanding why certain design decisions were made illuminates the constraints that shape the current format.

The Original ext Filesystem (1992)

The first extended filesystem, simply called "ext," was written by Remy Card in 1992 as the successor to the minix filesystem, which had served as Linux's default storage layer since 1991. The minix filesystem supported files up to 64 MiB and filesystems up to 64 MiB, limits that were quickly becoming inadequate even for the hard drives of the era.

ext introduced several concepts that would persist through all subsequent generations:

- **Inode-based metadata**: Each file's metadata (permissions, timestamps, block pointers) is stored in an inode, separate from directory entries. This allows hard links and efficient metadata updates.
- **Block groups**: The filesystem is divided into groups, each containing its own superblock copy, inode table, and bitmaps. This improves locality and enables parallel access.
- **Backup superblocks**: Redundant copies of the superblock are scattered across the disk, enabling recovery if the primary superblock is damaged.
- **The magic number 0xEF53**: This two-byte signature identifies ext-family filesystems and has been unchanged since ext.

The original ext had significant limitations. It lacked a proper block allocation algorithm, used only direct and single-level indirect block pointers, and

had no concept of journaling. Filesystem corruption after a crash was common and often required manual intervention to repair. Despite these shortcomings, ext served Linux well for its first couple of years.

ext2: A Production Filesystem (1993-2001)

ext2, developed by Rémy Card, Rémy Prehet, and others starting in 1993, was a complete rewrite that addressed most of ext's architectural weaknesses while preserving the fundamental on-disk format. The key innovations included:

Improved block allocation: ext2 introduced the buddy allocator for managing free blocks within each block group, replacing the simple bitmap approach of ext. This allowed efficient tracking of contiguous free space.

Three-level indirect blocks: While ext supported only direct and single-indirect block pointers, ext2 added double-indirect and triple-indirect levels, enabling files up to 16 TiB with 4 KiB blocks. The 15-element i_block[] array in the inode stores: 12 direct block numbers, 1 single-indirect pointer, 1 double-indirect pointer, and 1 triple-indirect pointer.

Directory indexing preparation: ext2 laid the groundwork for the HTree directory indexing that would arrive in ext3, by defining the ext4_dir_entry_2 structure with a file_type field. This allowed directory listings to determine file types without loading inodes.

Dynamic inode sizes: ext2 introduced the concept of variable-sized inodes through the s_inode_size superblock field and the i_extra_isize per-inode field. The original "good old" inode was 128 bytes (EXT2_GOOD_OLD_INODE_SIZE), but ext2 allowed larger inodes to be allocated at format time.

Feature flags: Perhaps the most important ext2 innovation was the three-tier feature flag system in the superblock:

- **compat_features**: Features that older kernels can safely ignore while still reading and writing the filesystem.
- **incompat_features**: Features that prevent older kernels from mounting the filesystem at all.
- **ro_compat_features**: Features that older kernels can read but not write.

This system enabled ext2 to evolve without breaking existing deployments. A kernel that does not understand a compat feature bit simply ignores it. A

kernel that encounters an unknown incompat bit refuses to mount, preventing accidental corruption. This design philosophy would carry through to ext3 and ext4 unchanged.

The 2 GiB file limit: With the original inode structure, i_size was a 32-bit field, limiting files to 4 GiB. In practice, ext2 filesystems were often created with the large_file feature disabled, enforcing a 2 GiB limit for compatibility with older tools. The large_file ro_compat feature flag, when set, indicated that files larger than 2 GiB existed on the filesystem.

By 1996, ext2 had become the default Linux filesystem, and it remained so until ext3's arrival in 2001. During this period, ext2 proved remarkably robust. It ran on everything from embedded devices with a few megabytes of storage to servers with hundreds of gigabytes. The maximum filesystem size was limited to about 2 TiB (later extended to 4 TiB and then 8 TiB through various patches), which was more than adequate for the era.

ext3: Adding Journaling (2001)

The single most important addition in ext3 was journaling, implemented by Stephen Tweedie and released in the Linux 2.4.15 kernel in March 2001. Before journaling, a crash during filesystem operations could leave the filesystem in an inconsistent state, requiring potentially hours-long fsck scans on large volumes. Journaling reduced recovery time from hours to seconds.

How the journal works: ext3 reserves a portion of the filesystem (typically 5% of total space, up to 1 GiB) as a circular log. Before any metadata change is written to its final location, the old and new versions of the affected blocks are recorded in the journal. A commit record marks the end of each transaction. If the system crashes, the journal is replayed on the next mount, applying all committed transactions to bring the filesystem to a consistent state.

Three writeback modes: ext3 introduced three journaling modes, controlled by the data= mount option:

1. **data=journal** (the safest): Both file data and metadata are written through the journal. Every byte of file data is journaled before the commit record. This provides full crash consistency but has significant performance overhead because all writes go through the journal.

2. **data=ordered** (the default): Metadata is journaled, and file data is flushed to disk before the metadata commit. This ensures that after a crash, no committed metadata points to stale data, while avoiding the overhead of writing all data through the journal. This is the mode most deployments use.
3. **data=writeback** (the fastest): Only metadata is journaled. File data may be written to disk at any time, potentially after the metadata has been committed. After a crash, it is possible to read stale or partially-written file data, though metadata structures remain consistent.

The journal inode: In ext3, the journal is stored as a regular file within the filesystem, typically at inode number 8. The journal file's metadata (its own i_block[] array and i_size) is backed up in the superblock, enabling recovery even if the journal inode itself is corrupted. The journal uses big-endian byte order for all its on-disk structures, deliberately opposite to ext4's little-endian convention, as a sanity check to distinguish journal blocks from filesystem blocks.

Backward compatibility: ext3 is backward-compatible with ext2 at the on-disk level. An ext3 filesystem can be mounted as ext2 (without journaling), and an ext2 filesystem can be upgraded to ext3 by simply creating a journal inode. The has_journal compat feature flag indicates the presence of the journal. This graceful upgrade path was one of ext3's greatest strengths.

Performance impact: Journaling adds overhead. In data=journal mode, write performance can be reduced by 30-50% compared to unjournaled ext2. In data=ordered mode, the penalty is typically 5-15%, mostly from the additional flush operations. Read performance is largely unaffected.

ext4: Scaling to Petabytes (2008-Present)

ext4 was initially developed as a series of backward-compatible extensions to ext3, many of them originally created by Cluster File Systems for the Lustre distributed filesystem between 2003 and 2006. The key contributors included Theodore Ts'o, Andreas Dilger, Alex Tomas, and the IBM Linux Technology Center team.

ext4 was merged into the mainline kernel in version 2.6.19 (experimental, read-only) in 2006, became read-write capable in 2.6.20, and reached stable

status in 2.6.28 in July 2008. From that point, it began replacing ext3 as the default Linux filesystem.

Extents: The most significant structural change in ext4 is the extent tree, which replaces the indirect block system of ext2/ext3. Instead of storing individual block numbers (with up to three levels of indirection), an extent records a contiguous run of blocks with a starting logical block number, a length, and a physical starting block. A single 12-byte extent record can describe up to 128 contiguous blocks, compared to the 4 bytes per block in the old indirect scheme.

For a 1 GiB file with 4 KiB blocks, ext2 requires approximately 256 indirect blocks (spread across multiple levels) plus the 12 direct entries in the inode. ext4 can describe the same file with a single extent record if the blocks are contiguous. This reduces metadata overhead, improves read performance (fewer block reads to map a file), and enables better defragmentation.

The extent tree is stored directly in the inode's i_block[] array for small files (up to 4 extents fit in the space previously used for direct block pointers). For larger files, the extent tree grows into a B+ tree structure with index nodes and leaf blocks, supporting up to 5 levels of depth.

Multi-terabyte and petabyte support: ext4 increases the maximum filesystem size from ext3's 16 TiB to 1 EiB (2^64 blocks) through several mechanisms:

- The 64bit incompat feature enables high 32-bit fields for block counts, reserved block counts, and free block counts in the superblock.
- The huge_file ro_compat feature allows file sizes up to 16 TiB (with i_size_high in the inode).
- The meta_bg incompat feature moves group descriptors out of block group 0, removing the 32-bit limit on the number of block groups.

In practice, the Linux kernel currently limits ext4 filesystems to 58 TiB due to internal address space constraints, but the on-disk format supports much larger volumes.

Delayed allocation: Instead of allocating physical blocks immediately when a write is issued, ext4 defers allocation until writeback time. This allows the allocator to see the full picture of what blocks are needed and make better placement decisions, reducing fragmentation. Combined with extents, delayed allocation enables the filesystem to allocate large contiguous runs of blocks for sequential writes.

Multi-block allocator (mballoc): ext4 replaces the ext2 buddy allocator with a multi-block allocator that can allocate many blocks in a single call. This is particularly effective when combined with delayed allocation, as the allocator can reserve large contiguous regions in one operation rather than allocating blocks individually.

Backward compatibility: Like ext3 with ext2, ext4 maintains backward compatibility. An ext4 filesystem without extents can be mounted as ext3. The extent incompat feature flag signals that the filesystem uses extents, preventing older kernels from mounting. However, many ext4 features (delayed allocation, mballoc, directory preallocation) are transparent to the on-disk format and work even when the filesystem is mounted as ext3, providing free performance improvements.

Design Continuity and Breaking Changes

One of the most remarkable aspects of the ext family is its commitment to backward compatibility. The on-disk format has evolved incrementally over more than thirty years, with each generation adding features while preserving the ability to read older filesystems. This is achieved through:

1. **The feature flag system**: Three 32-bit fields in the superblock control which features are active. Unknown compat flags are ignored; unknown incompat flags cause mount refusal.
2. **Conservative defaults**: New features are often disabled by default on existing filesystems and must be explicitly enabled by the administrator.
3. **Graceful degradation**: When an ext4 feature is not available (e.g., mounting as ext3), the filesystem falls back to the older behavior without data loss.

However, there are some breaking changes to be aware of:

- An ext4 filesystem using extents cannot be mounted as ext2 or ext3 for writing. The extent tree format is incompatible with the indirect block system.
- Once a filesystem is converted from 128-byte inodes to 256-byte inodes, the conversion cannot be reversed.

- The meta_bg feature, once enabled, changes the location of group descriptors and cannot be disabled.
- Filesystem encryption (fscrypt), once applied to files, makes those files unreadable without the correct keys and an ext4-aware kernel.

Understanding these compatibility boundaries is essential for anyone implementing ext4 support, particularly in environments where the same filesystem may be accessed by different operating systems or kernel versions. The next chapter dives into the fundamental on-disk structures that form the foundation of every ext-family filesystem.

Chapter 2: On-Disk Layout Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The Superblock Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Block Groups and Group Descriptors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Block and Inode Bitmaps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The Inode Table Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Bootloader Blocks and Reserved Space

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 3: The Inode – Anatomy of a File

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The ext4_inode Structure (256 Bytes)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Direct, Indirect, Double, and Triple Indirect Blocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Extent Trees (ext4_extent_header and ext4_extent)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Inline Data for Small Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Timestamps: Birth Time, Nanosecond Precision, and Y2038

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 4: Directory Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Linear Directories (ext4_dir_entry_2)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The HTree Indexing Algorithm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Hash Algorithms: T10, T13, T20, T21, T60

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Directory Block Layout and Padding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Traversing and Searching Directories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 5: Extents – Modern Block Mapping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The Extent Tree as a B+ Tree

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

ext4_extent_header and ext4_extent Records

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

ext4_extent_idx for Multi-Level Trees

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Uninitialized Extents and Lazy Allocation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Extent Manipulation: Splitting, Merging, Truncation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 6: Block Allocation Algorithms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Group Selection Strategy (mb_group_prealloc_scan)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The Move Allocator (mballoc)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Delayed Allocation (delalloc)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Block Preallocation and Post-readahead

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Fragmentation Avoidance and Defragmentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 7: Journaling with JBD2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Transaction Model and Commit Sequence Numbers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Writeback Modes: Data, Ordered, Writeback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The Journal Superblock and Descriptor Blocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Journal Replay and Crash Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Fast Commits (ext4 fast commit feature)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 8: Extended Attributes, ACLs, and Quotas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Extended Attribute Storage (Inline vs. EA Blocks)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The xattr Header and Entry Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

POSIX ACLs and the acl_mode Hook

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Project Quotas and User/Group Quotas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Security xattrs: Trusted, User, and System Namespaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The ext4_resize Operation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 10: Feature Flags and Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The Three Feature Flag Categories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Mount Options and Their Effects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Checking Feature Flags in Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 11: VFS Integration and Kernel Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The super_operations and inode_operations Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Page Cache Interaction and Writeback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Filesystem Locking: Per-file Locks, Group Locks, mballoc Locks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Concurrency: Multi-threaded Allocation and Journaling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Dirty Page Management and Flush Threads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 12: Recovery, Corruption Detection, and Repair

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Checksum Offsets and Verification (crc32c)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

The e2fsck Passes: From Structure to Links

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Corruption Detection in Production (errors=continue/remount-ro/panic)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Backup Superblocks and Redundancy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Forensic Analysis of ext4 Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 13: Performance, Scalability, and Comparisons

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Key Performance Characteristics and Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Scalability: Filesystem Size, Inode Count, Directory Entries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Comparison with ext2 and ext3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Comparison with XFS, Btrfs, ZFS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Comparison with NTFS, FAT, exFAT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Chapter 14: Implementation Guide

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Building a Minimal ext4 Reader (C)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Validating Superblocks and Group Descriptors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Traversing the Extent Tree

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Implementing Block Allocation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Writing a Journal Replay Engine (Rust)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thelinuxext4filesystem.