Operating Systems Project: Topic 13

Journaling and Crash Recovery (Ext4 & JBD2)

Liangsen Wang

224040364@link.cuhk.edu.cn

2026.2.3

Outline

Outline

The Mission: Topic 13 Requirements

Objective: Deep dive into Ext4's reliability layer (`fs/jbd2`) to understand how file systems survive power failures.

Requirement 1: Analysis

- **Target:** The `fs/jbd2/` subsystem.
- **Task:** Trace the lifecycle of a filesystem transaction (Start $\rightarrow$ Commit $\rightarrow$ Checkpoint).

Requirement 2: Instrumentation

- **Task:** Add stats to measure transaction latency (time spent committing).
- **Task:** Simulate a crash (kernel panic) and verify that JBD2 recovers the data upon reboot.

The Problem:
If power cuts during a write, is your disk corrupted?

Advanced Options

Option A: Lightweight Journaling

- **Idea:** Full journaling (Data + Metadata) is slow.
- **Task:** Configure and analyze "Metadata-Only" journaling (Ordered Mode). Explain why this is faster but still safe for file structure.

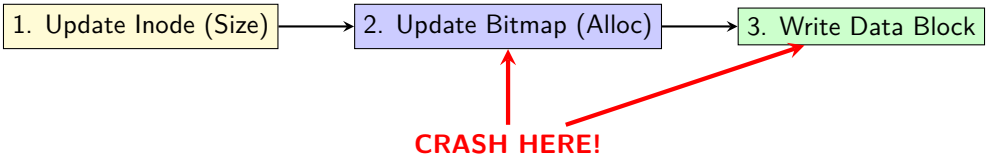
Option B: Journaling vs. CoW (Btrfs)

- **Comparison:** Ext4 updates in-place (needs Journal). Btrfs writes to new location (Copy-on-Write).
- **Task:** Benchmark Ext4 vs Btrfs under heavy random writes to see the performance trade-off.

Outline

Theory 1: The Atomicity Problem

A file system operation (e.g., append data) is **not atomic**. It involves 3 separate writes:



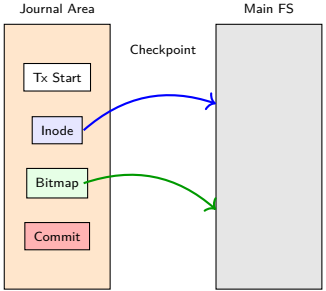
Result: Inode says file is bigger, but data is garbage/old. **Corruption!**

Theory 2: Write-Ahead Logging (WAL)

Rule: Write the "note" to the journal *before* touching the main filesystem.

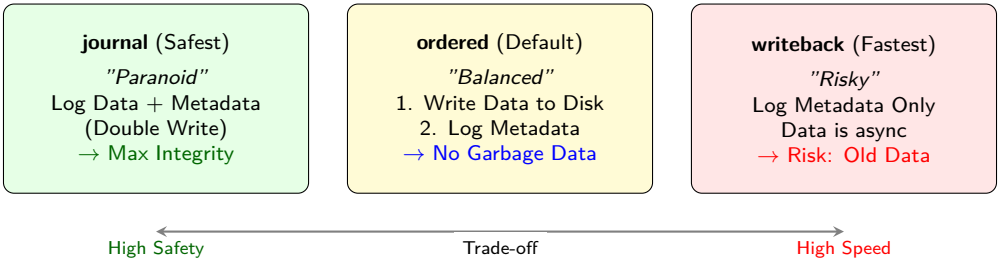
The Protocol:

1. **Log:** Write metadata + data to Journal.
2. **Commit:** Write a "Commit Block" (Atomic!).
3. **Checkpoint:** Copy from Journal to Main Disk.
4. **Free:** Mark Journal space as free.



Theory 3: Ext4 Journaling Modes

Trade-off between Safety and Speed.



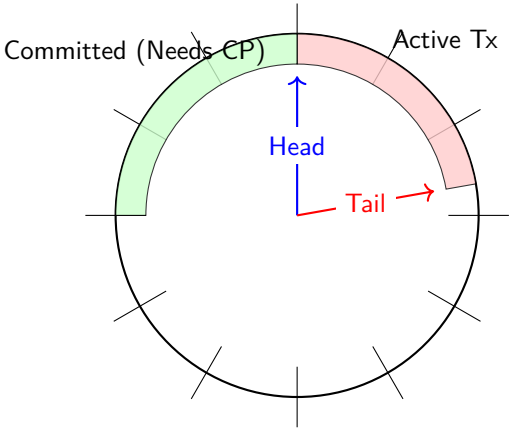
Outline

Mechanics 1: JBD2 (Journaling Block Device)

Ext4 doesn't do logging itself. It uses a separate subsystem: **JBD2**.

Mechanics 2: The Circular Log

The Journal is a fixed-size ring buffer on disk.



Mechanics 3: Key Data Structures

`transaction_t`: Represents one atomic update set.

- `t_id`: Sequence Number (1, 2, 3...).
- `t_buffers`: List of modified buffers.
- `t_state`: RUNNING, FLUSH, COMMIT.

`handle_t`: A handle for a single syscall's part of a transaction.

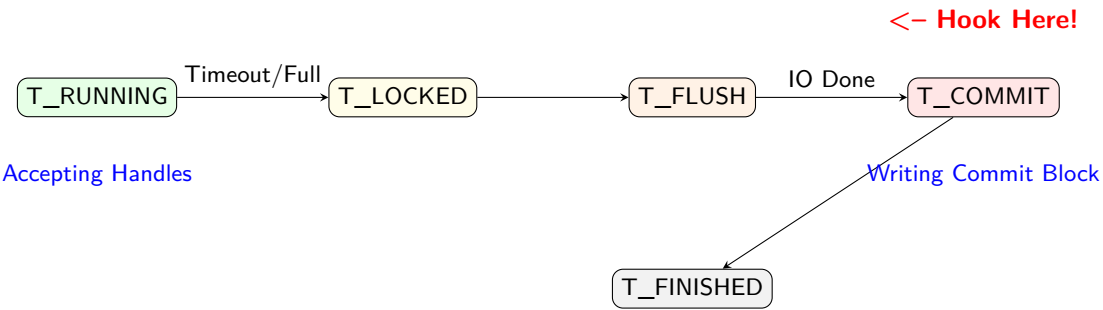
Workflow

```
start_handle()
... modify metadata ...
stop_handle()
```

Outline

Implementation 1: Lifecycle Tracing

Where to hook? Follow the state machine.



Implementation 2: Instrumentation

Goal: Measure how long a commit takes.

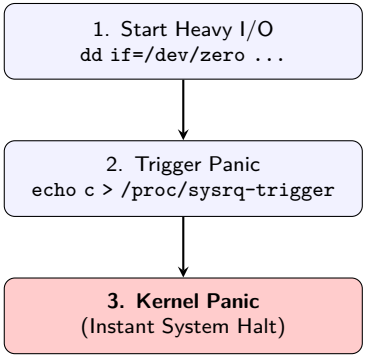
Plan

1. Open `fs/jbd2/commit.c`.
2. Locate `jbd2_journal_commit_transaction()`.
3. Add timestamps:
   - `start = ktime_get();` at function start.
   - `end = ktime_get();` after commit block write.
4. Export to `/proc/jbd2_stats`.

Implementation 3: Simulating a Crash

How to verify recovery without pulling the power plug?

The "Self-Destruct" Sequence



Post-Reboot Verification

After the VM restarts, check if JBD2 did its job:

1. Check Kernel Logs:

```
$ dmesg | grep "recovery"
EXT4-fs: recovery complete
EXT4-fs: mounted filesystem ...
```

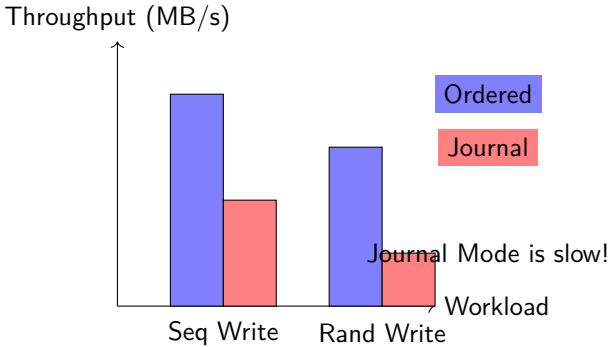
2. Check Disk Consistency:

```
$ e2fsck -n /dev/sdb1
/dev/sdb1: clean, ...
```

Outline

Evaluation 1: Performance Benchmarking

Comparing `data=journal` vs `data=ordered`.



Evaluation 2: Recovery Speed

How long does `fsck` take after a crash?

- **Ext2 (No Journal):** Must scan whole disk. Hours.
- **Ext4 (Journal):** Only replay the log. Seconds.

Experiment: 1. Fill 10GB disk. 2. Crash. 3. Measure mount time.

Resources

Kernel Source:

- `fs/ext4/super.c`: Where ext4 connects to JBD2.
- `fs/jbd2/journal.c`: Core journal management.

Reading:

- *"Journaling the Linux ext2fs Filesystem"* (Whitepaper).
- OSTEP Chapter: Crash Consistency.