Tricky issues in file systems

Taylor 'Riastradh' Campbell
campbell@mumble.net
riastradh@NetBSD.org

EuroBSDcon 2015
Stockholm, Sweden
October 4, 2015

What is a file system?

- Standard Unix concept: hierarchy of directories, regular files, device nodes, fifos, sockets.
- Unified API: file descriptors.
- Traditionally unified storage: inodes.
- Directories are sometimes ‘different’: contain metadata pointers.

File system operations

- `creat` (`open`)
- `unlink`
- `link`
- `rename`
- `mkdir`, `rmdir`
- `read`, `write`, `fsync`
- (`mkfifo`, `symlink`, `readdir`, &c.)

Reliability properties

- ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- No transactions: only individual operations.

Atomicity

- Operation either happens all at once, or not at all.
- Crash in middle will not leave half-finished operation.

Atomicity: rename

- `rename(old, new)` acts as if
  - `unlink(new)`
  - `link(old, new)`
  - `unlink(old)`
- . . . but atomic.

Consistency

- All operations preserve consistent disk state.

Isolation

- If process A does rename:
  - `unlink("bar")`
  - `link("foo", "bar")`
  - `unlink("foo")`
- …then process B won't see two links at `foo` and `bar` in the middle.

Durability

- When the `sync` program returns, whatever file system operations you performed will stay on disk.

File system states

- File system has one of three states:
  - clean
  - dirty
  - corrupt (bugs, disk failure, cosmic rays)
- Clean *flag* in superblock:
  - 0 means known clean
  - 1 means not known whether clean or dirty (or corrupt)

File system states: fsck

- Traditional: file system operations write metadata synchronously
- Inode updates, directory entry updates
- Every step preserves *consistent* state but not necessarily *clean* state.
- On boot:
  - If marked clean, just mount.
  - Otherwise, `fsck -p` globally analyzes file system to undo partially completed operations.

File system states: fsck example

- Inode block allocation — need space to append to file:
  - Find block in free list.
  - Mark block allocated.
  - Assign block to inode.
- If crash after block allocated, before block assigned:
  - `fsck -p` scans all inodes
  - finds all assigned blocks
  - frees unassigned but allocated blocks

File system states: fsck to fix corruption

- `fsck` (without `-p`) also tries to fix corrupted file systems
- (Doesn't always work)

Logging

- Physical block logging:
  - Write blocks *serially* to write-ahead log
  - (not *synchronously*)
  - After committed to log, write to disk
  - After committed to disk, free space in log
  - After crash, replay all committed writes in log
- Faster to recover from crash (but not corruption): replay log is quick scan, not global analysis
- . . . but isn't usually quite enough

Logging

- Logical block logging:
  - Write logical operations serially to write-ahead log
  - After committed to log, perform operations on disk
  - After committed to disk, free space in log
  - After crash, replay all committed operations in log
- More complex to implement
- But usually necessary at least a little

Physical vs logical

- NetBSD FFS WAPBL, write-ahead physical block logging
- Actually, composite of physical and logical

Physical vs logical: block deallocation

- Deallocate blocks from file, e.g. on rm
- Reuse blocks immediately? No!
- Reallocated block write might happen before log write!
- Logical log entry: deallocate block
- Physical log entry: change inode to not point at block
- When physical log committed, then commit logical log

Reliability assumptions

- Atomic disk sector writes
- Disk write ordering
- Disk write cache

Disk encryption

- Threat model: attacker reads (possibly several) snapshots of disk (e.g., airport security)
- (Why several? Reallocated disk sectors, especially in SSDs.)
- 1–1 plaintext/ciphertext disk sector mapping
  - 512 bytes of plaintext $\rightarrow$ 512 bytes of ciphertext
- No defence against malicious modification of disk!
- Easy to preserve atomicity of disk sector writes, write ordering, &c.

Disk authentication

- Threat model: attacker can write malicious data to disk
- Expand each disk block with secret-key MAC?
- 512 bytes of user data $\rightarrow$ 528 bytes of disk sector?
  - Splits file system's idea of logical disk sector across two physical disk sectors
  - Atomicity? Nope!
- 496 bytes of user data $\rightarrow$ 512 bytes of disk sector?
  - Not nice for file system!

File system authentication

- Rewrite tree of pointers-with-MAC all the way to the root?
- ZFS can do this; FFS, not so much.

Data/metadata write ordering

- Traditional FFS:
  - Synchronous metadata writes
  - Asynchronous data writes (roughly)
- Typical logging FFS:
  - Serial metadata writes
  - Asynchronous data writes
- No write ordering between data/metadata!

Garbage data appended after write?

- Allocate free block
- Write data to block (A)
- Write inode to point at new block, increase length (B)
- What if B happens before A?
- What if crash between B and A?
- Now file has whatever data was in free block!

Performance and concurrency

- Coarse-grained locking: easy, slow
- Want per-object locking

Rename

- Four different objects to lock!
- Any pair of them may be the same!
- Need consistent lock order.

Rename: orphaned directory trees

```
              /
     /home  /usr   /var
  /home/foo

% mv /home /home/foo/bar
```

Lock order?

- Traditional in FFS: flail randomly?
- FreeBSD: wait/wound locks, without priorities — livelock!
- ZFS: complicated! (Ask me after. Also broken.)
- Linux and NetBSD: ancestor-first, one rename in flight, deadlock-free

Suspend

- Need for taking snapshot (unless, e.g., log-structured)
- Need for unmount
- Prevent all operations:
  - Block new operations.
  - Wait for existing ones to drain.
- In NetBSD: called `fstrans`.

Suspend: reader/writer lock?

- Can use recursive reader-writer lock: file system operations take read lock, suspend takes write lock
- Slow! Point of contention for every file system operation.

Suspend: pserialized reader/writer lock

- Better: use passive serialization.
- Reader creates per-thread structure, touches no global state, unless suspend in progress
- Writer: marks suspend in progress, waits for all per-thread structures to drain

Questions?

- campbell@mumble.net