Flash-Dateisysteme

Christian Egger | Juni 2010 | Verteilte Systeme
Intro - Flash

- Developed by Toshiba in 1985
- Replacement of EEPROMs
  - Read-Only: Bios, Firmware...
  - Read-Write: Embedded Devices (Router, controller...)
- Moore's Law: faster, cheaper, bigger memories
- New application areas
  - Integration into microcontroller
  - USB-Sticks (8MB... 2GB... 128GB...)
  - Memory cards: SD, MMC, xD, CF, MemoryStick
  - embedded Storage: MP3-Player, Handy
  - As hard disk replacement: Solid State Disks (SSDs)
- 2 Types: NOR and NAND Flash
Flash Technology - NOR

- expensive
- low capacity
- good reliability
- Byte/Word-wise Access (via address/data lines)
- direct CPU connection
- fast random access
- bitwise programmable
- Use: mostly program memory / embedded
- very low erase and write performance
Flash Technology - NAND

- Command driven interface
- SLC NAND Flash (Single-Level-Cell)
  - 2 States, 1 Bit per cell
  - robust: 100K-1Mio erase cycles
  - more reliable than MLC
  - lower energy consumption, faster than MLC
  - more expensive than MLC
- MLC NAND Flash (Multi-Level-Cell)
  - 4 states / 2 bits per cell
  - 10K-100K erase cycles
  - bad blocks when delivery (like bad pixels / LCDs)
  - more storage per silicon
  - stricter constraints compared to SLC and NOR
Flash-storage: Differences to hard disks

- **Basic commands**
  - read
  - erase
  - write

- **Units**
  - Page - 2KiB
  - Block - 128KiB

- **granularities:**
  - read/write - byte/word
  - read/write - page
  - erase - block

- **limited number of erase cycles**
  - NOR: 100k-1M
  - SLC NAND: 100k+
  - MLC NAND: 10k-100k
Other quirks

- **Writes**
  - in-place handicapped
    - needs “read-modify-erase-write”
    - expensive (time, complexity)
    - not atomic, unsafe
    - high wear
  - out-of-place
    - needs only “write” (assumption: pre erased pages)
    - atomicity

- **Overwrites**
  - multiple writes to the same “page”
  - some flashes (NOR, SLC)
More quirks

- NAND: spare areas
  - extra storage
  - with/without overwriting
  - Application
    - ECC
    - bad block flags
    - deletion marker

- NAND: writes strict linear
  - only within same block
  - consequence of other optimizations (price)
Methods of using flash

- Flash Translation Layer (FTL)
  - flash as a block device
  - Handling
    - wear-leveling
    - bad-block handling
    - Error Correction
    - mapping of different page sizes
- Flash File Systems
  - since 1990+, FFS2 by Microsoft
  - Advantages over FTL
    - directly usable, no extra logic
    - more efficient
    - special applications possible (XIP)
Concepts

▶ Nodes
▶ Log Structure
▶ Garbage Collection
▶ Wandering Trees
▶ Write Back
▶ Mount Scanning
▶ Checkpointing / Snapshotting
▶ Compression
▶ Error Correction
▶ Execute-in-Place
Nodes

- Contiguous Structure
  - Metadata
  - (not needingly also) Data
- less write operations
Log Structure

- Out-of-place
- Formen
  - Ringbuffer
  - Log structure within single blocks
  - Partitioning into areas
- best performance: blocks pre-erased
Log Structure

Abbildung: Log with some operations.
Garbage Collection

- Log Structure: Trashing, fragmentation of single blocks
- Block status
  - empty / erased
  - full / all data valid (obsolete)
  - partially full / some invalid data
  - erasable / all data invalid
- Solution: GC!
  - redundant copy
  - obsolete full blocks
  - reclaim free space
- Strategies
  - strict (like a Ringbuffer)
  - Heuristics
Garbage Collection

Abbildung: GC with different block statuses.
Wandering Trees

- Directory Index
- like ext2 Tree
- but: out-of-place updates, floating structures
- Differences
  - Index still points to obsolete Data (COW)
  - Update index recursively
  - Order: Leaf .. Root-node (atomicity!)
  - Root node has a new place
Write-Back Strategy

- Caching of dirty pages
- Write bulks of data
- Pros/Cons
  - Fewer writes
  - Agglomeration of Data
  - not safe
Mount Scanning

- Index not on Flash
  - construct on startup
  - full Device-Scan needed
  - Complexity: $O(n)$ (start-time + RAM vs. device size)
- Index on flash
  - locatable in $O(1)$: root-node
  - complexity: $O(1)$ possible (RAM + startup time)
Checkpointing / Snapshots

- Checkpointing
  - FS without Index on Flash
  - Memory dump of the index saved to flash
  - low Mount-Scan complexity
  - fast startup
  - validity: as long state does not change
Compression

- slow writes
- Compression, write fewer Data: faster?
- Algorithms
  - deflate/zlib (default)
  - LZO
  - LZMA
  - bzip2
- Application
  - compress Data
  - compress Metadata
- most often only data
- Problem: calculation of free space?
Error Detection & Correction

- NAND: Focus on cheap price
- defect blocks/pages
  - delivery with bad blocks allowed
  - emerge during use
  - mark: flag in spare area
- Bit-Flips in neighbouring cells
- Software has to deal with that
  - CRCs
  - ECC (detect 2-bit, correct 1-bit errors)
  - FS data structure has to allow bad blocks everywhere
Execute-in-Place

- No fetch into RAM
- Executable Text area mapped directly into address space
- only NOR
- very invasive (FS-Code - Paging Code)
- used for “embedded” areas
- i.e. Linux-Phones (Maemo, FIC)
- 2 implementations in Linux
  - AXFS
  - CRAMS+XIP Patch
Flash File Systems in Linux

<table>
<thead>
<tr>
<th>Jahr</th>
<th>Name</th>
<th>in Kernel?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>JFFS</td>
<td>discontinued</td>
</tr>
<tr>
<td>2001</td>
<td>JFFS2</td>
<td>Linux-2.4.10+</td>
</tr>
<tr>
<td>2002</td>
<td>YAFFS</td>
<td>nur patch</td>
</tr>
<tr>
<td>2005</td>
<td>YAFFS2</td>
<td>nur patch</td>
</tr>
<tr>
<td>2007</td>
<td>LogFS</td>
<td>Linux-2.6.34+</td>
</tr>
<tr>
<td>2008</td>
<td>UBIIFS</td>
<td>Linux-2.6.27+</td>
</tr>
</tbody>
</table>
JFFS

- Axis Communications AB
- first Implementation
- Structure: Nodes + strict Log
- no compression
- Kernel 2.0 / 2.2
- no hardlinks
- Mount-Scan
- Index in RAM
JFFS2

- Redesign of JFFS by RedHat
- designed for NOR
- Improvements
  - Compression (zlib, rubin, rtime)
  - relaxed Log-Structure Approach
  - Hardlink support
- most often used Flash FS
- Problem: scalability
  - RAM: $O(n)$ for \{Number of Objects in JFFS2\}
  - Startup time: $O(n)$ for device size
YAFFS

- designed for NAND
- very portable, Linux: Patch
- no Index on Flash: RAM and Start in $O(n)$
- but: Checkpointing, fast Start
- no compression
- YAFFS1
  - 512B page size NAND
  - Spare Areas: Deletion Marker
  - simple Mount-Scan
- YAFFS2
  - 2KiB page size NAND
  - Spare Areas only for marking Bad Blocks
  - no Overwriting
LogFS

- Block and MTD mode
- requirement: scalability
- RAM usage and start in in O(1)
  - Index on flash: Wandering Tree
  - 2 Anchor Areas: Pointers to floating structures
  - Block-levels: blocks only used for nodes of same level
    - root node blocks
    - ...
    - level n blocks
    - data blocks
UBIFS & UBI

- UBI Layer
  - “unsorted block images”
  - LEBs / PEBs
  - wear-leveling
  - Error correction
  - Scrubbing
  - Start in O(n)

- UBIFS
  - Very much like LogFS (except for UBI)
Read-Only File Systems in Linux

<table>
<thead>
<tr>
<th>Jahr</th>
<th>Name</th>
<th>in Kernel?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>RomFS</td>
<td>2.2+</td>
</tr>
<tr>
<td>1999</td>
<td>CramFS</td>
<td>2.4+</td>
</tr>
<tr>
<td>2002</td>
<td>SquashFS</td>
<td>2.6.29+</td>
</tr>
<tr>
<td>2006</td>
<td>AXFS</td>
<td>patch</td>
</tr>
</tbody>
</table>
CRAMFS

- Compression support
- terse Metadata
- very mature
- Disadvantages
  - 8bit gid/uid’s
  - no timestamps
  - 16MiB file size limit
  - device size limit: ≤ 256MB (+16MB)
- XIP support (Montavista patch)
SquashFS

- variable (compression) block size up to 1MiB
- result: good compression
  - zlib (default)
  - LZMA
  - Bzip2
  - LZO

- Applications
  - Embedded
  - LiveCDs (+UnionFS)
AXFS

“Advanced eXecute-in-place File System”
- only for NOR
- no MTD layer
- not mainline, very invasive (messes with non VFS code)
- pages either XIP or compressed
  - runtime profiling support (XIP xor compression)
  - profile feeded to mkfs.axfs
Linux: SSDs and ATA Trim()

- currently supported
  - Btrfs, VFAT, EXT4, GFS2, NILFS
- Btrfs
  - special Block Allocator modes
  - Mode “ssd”
  - Modu “ssd_spread”
- SSD-mode off by default
- buggy SSD FTLs
Windows

- ATA Trim()
  - Windows7 only
  - FAT
  - NTFS

- exFAT
  - chosen future standard file system for SDXC
  - no 4GiB File-Limit
  - no 32GiB/2TiB Device-Limit
  - patent-encumbered
  - proprietary
  - Linux not (really) supported yet