NandFS

We want NandFS to be a reliable and yet performant file system that takes in account the particularities of NAND Flash and Embeded systems.

Nand Flash :

Write granularity is called page, and erase granularity block. The pages are typically 512 or 2,048 bytes in size. Associated with each page are a few bytes, called spare or oob, (16 bytes for a 512 bytes page, 64 for a 2048) that should be used for storage of an error detection and correction checksum. A block would be 32 pages of 512 bytes or 64 pages of 2048 bytes. (See Nand_Geometry) Most of the time there is only 1 write cycles on a page, and 2 or more on a spare. This mean if we want to rewrite a page, we need to erase the entire block. And erased block is filled with ones, writting a page on it will only flip bits from 1 to 0.

Typicals timings

- Read a Page : 500µs
- Read a Spare : 40µs
- Write a Page : 350µs
- Write a Spare : 70µs
- Erase a Block : 2ms

(See Nand_Timings)

What we need

- "out of place" updates, due to erase blocks.
- Good wear-leveling (erase cycles are limited to 100 000).
- Fast file system mount
- Power Safe (power loss are more likely to occur in an embedded system)
- Data correction (with ECC)
- Easily Portable
- Special management of LOG files (large number of small write)
- Small memory footprint

Journaling file system

http://en.wikipedia.org/wiki/Journaling_file_system

A journaling (or journalling) file system is a file system that logs changes to a journal (usually a circular log in a specially-allocated area) before actually writing them to the main file system.

Log Based File System

http://en.wikipedia.org/wiki/Log-structured_filesystem

A log-structured file system thus treats its storage as a circular log and writes sequentially to the head of the log.

Garbage Collection

Because of Nand limitations, data are not erased, but obsoleted. It's why we need a garbage collector to erase dirty blocks. Garbage collector may also compact fragmented data chunks, etc ...

Flash Translation Layer

Current flash users use an abstraction layer to make flash look like a virtual block device, so they can use a traditional fs (FAT32 most of the time). But FTLs are inefficient, and it's worst when they are used with an

hard drive specialised file system. To preserve the flash lifetime, and get good performances, we need to deal directly with flash limitations.

JFFS2

Journalling Flash File System version 2 or JFFS2 is a log-structured file system for use in flash memory devices.

Mount :

The whole device is scanned at mount time to create the in-ram representation

Spare usage :

used to know if a block is empty, bad and if it's a jffs2 one or not (with a magic bitmask).

Error correction:

CRC32 Usage ECC handled by the Nand Driver

In-Ram :

Fragtree for data chunks (using linux Red-Black tree) Hashtable for inodes

GC :

- Turn partialy dirty blocks into free blocks.
- Remove "Holes" between data chunks.
- Threaded

Superblock:

none

Caches:

- Inode cache hashtable
- Write buffer

Pros :

- xattr
- acl
- Hard links
- Compression

Cons :

- Spare are not used
- Support for NAND flash devices, but designed for NOR devices.
- All pages must be scanned at mount time, this is very slow with large devices
- Not ported to WinCE

JFFS3

Todo

Superblock:

Yes

Spare usage :

Pros :

- Faster mount time than JFFS2

Cons :

- Not yet released

YAFFS1

(Like YAFFS2)

YAFFS2

Files, Dir, Datachunk are object. Each object get an id, a version and parent id. The version number is global (there will never be 2 object with the same version)

Mount :

Scan only the spares. If a page is free, jump to the next block. Read the metadata for each files/dir because it needs the parent id to build the in-ram represenation.

In-Ram :

- Objects: hashtable
- Data Chunks: tiered indexing structure

Spare usage :

- Object ID
- Data chunk offset/lenght
- Version
- ECC

Error correction:

- Soft ECC
- Driver ECC

GC :

- Turn fully dirty blocks into free blocks.
- Compact blocks if we need space
- Not threaded

Superblock:

none

Caches:

- Data chunk cache for short read/write operation - useful for WinCE

Pros

- Checkpoints
- Simple
- Internals read/write cache
- Hard links

Cons

- Need to read metadata during the scan