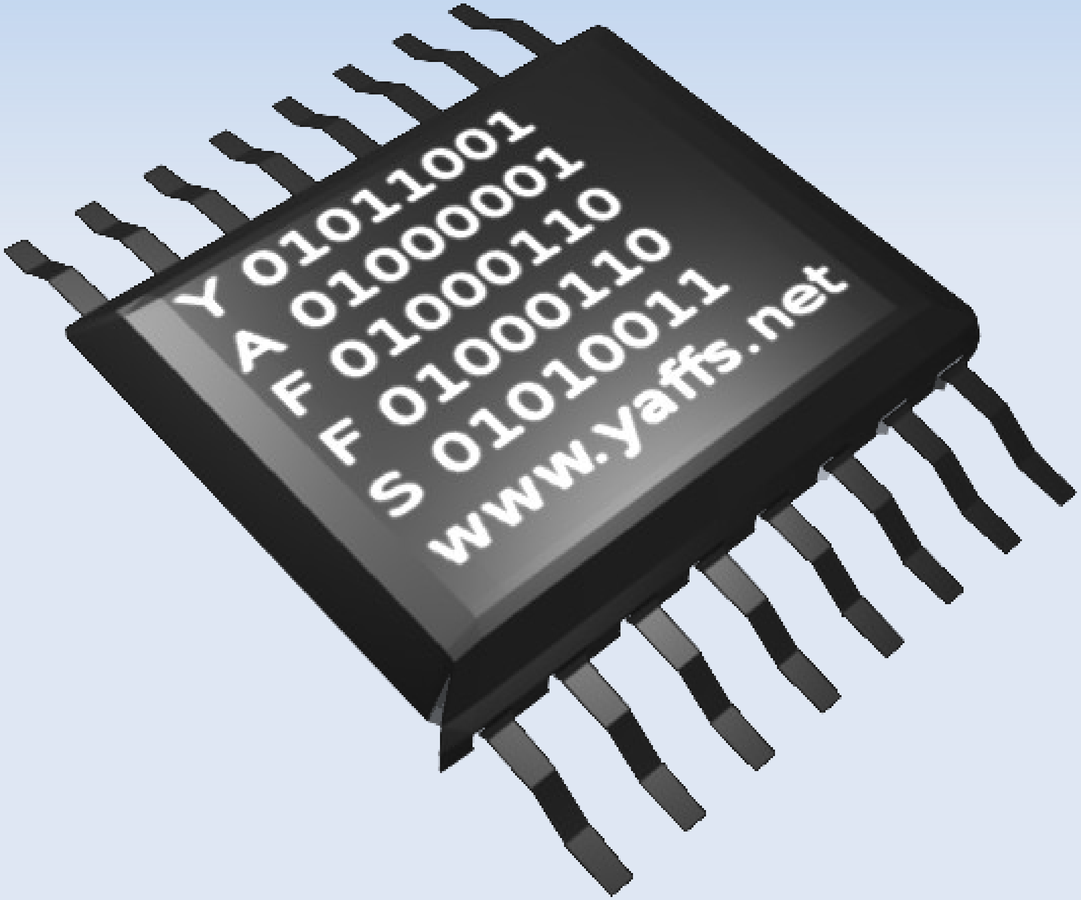
YAFFS Update

Wookey & Charles Manning
October 2010



Outline

- A bit of history
- Development changes
- New features
- Mainlining into Linux kernel

For technical details

- http://www.yaffs.net

What is YAFFS?

- Flash file system
- First developed for NAND, also used with NOR.
- Log structured
- Used in all sorts of applications:
  - From sewing machines to aerospace.
  - Volume user: cell phones
- Multi-platform: Linux, WinCE, RTOSs, etc
- GPL2 or proprietary licensing available.

YAFFS1 History

- YAFFS1 Milestones
  - Started in December 2001
  - Core file system simulation working March 2002
  - Rudimentary Linux operation April 2002
  - First Linux release May 2002
  - First WinCE use Sept 2002
  - First YAFFS Direct release Jan 2003
- In 2001....
  - 512Bytes/page NAND was quite new
  - 32MiB flash was ***HUGE***

YAFFS2

- End 2002, identified need for new approach.
  - MLC NAND on the horizon
  - Reduction in write flexibility
    - No rewriting.
    - Sequential page writing within a block
      - Thus could no longer write deletion markers
      - Thus needed to introduce a “flow of time”
  - Potential performance advantage of reduced writes.
  - Wider variety of flash parts and controllers
    - More abstract NAND model.
  - YAFFS1 backward compatibility

YAFFS2 History

- YAFFS2 Milestones
  - Ideas sketched out in Nov 2002
  - Work started in 2003.
  - Working by end 2003
  - Released to world 2004
  - Checkpointing added May 2006
  - Background gc added 2010

Code structure

- Modular sub-systems
  - Portable core code (~13,000 loc)
  - OS-specific wrapper code (~3500 loc)
- Developed in user-space, not kernel
  - Way faster:
    - Richer tools
    - Plug & play testing with test wrappers.
    - App. crashing is cheap

Code structure:2

- Unintended side effect
  - Multi-OS support.
    - One file system code base for Linux, WinCE, boot-loaders, RTOS and others.
    - Perhaps the most ported FS code in existence.
  - Alternative revenue stream.
    - Helps fund Aleph One's GPL “core mission”.

Testing History

- Until mid 2008:
  - Community oriented
    - Limited internal testing.
    - External parties provided significant testing until end 2007.
- Mid 2008 found some serious bugs
  - Decided to implement extensive internal testing.
    - Massive improvement in corner case robustness.
    - New tests being added all the time
    - Automatic tests running almost constantly

Testing

- Testing > 60% of development time
- Multi-faceted
  - YAFFS Direct tests
  - Linux in-kernel tests
  - Fuzz testing
- Mostly simulated flash
  - Way faster test cycling than real flash

Test Example

- Simulates a firmware update under power fail
  - Cycle:
    - Checks current file set OK.
    - Writes new temp files with checksums
    - Rename temp files over existing file set
  - Simulate a power fail at any point
    - Simulates many power failures per second.
- Improvement:
  - Mid 2006: Fail within 200 cycles
  - Sept 2006: Fail around 200k cycles.
  - Now: Runs millions of cycles with no failure.

New features

- Background garbage collection
- xattrib support
- Improved MLC handling
  - Block refreshing
- Faster
  - Approx 50% faster reading/writing since Dec 2009
  - Result of three different sets of changes.

Background garbage collection

- GC 'collects' free space and erases blocks
- Old GC
  - Side-effect of writing
  - Slows down writes
- Idea:
  - Most of the time the FS is idle, just bursts of writes now and then.
  - Goal: do most of GC while device is apparently idle
    - Less GC when actually writing
    - Writes faster.

Background GC challenges

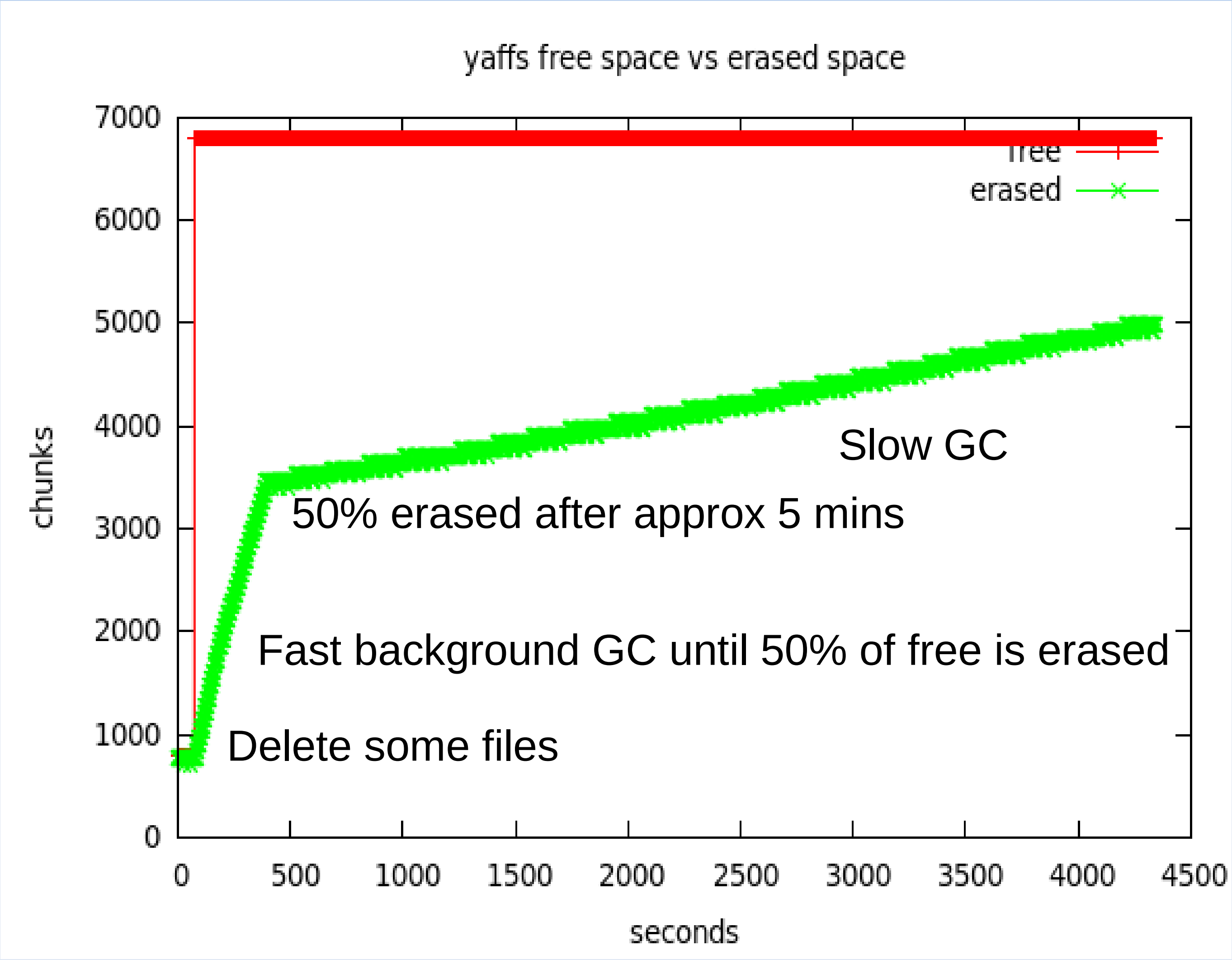
- Overzealous GC does too much
  - Increased flash wear
  - Increased power consumption
  - Can actually slow things down.
- Tuning can be hard
  - Get the benefits without too much cost
  - Keeping it simple
  - Watch out for corner cases

Background GC tuning

- Based on erased space available vs free space
- Foreground GC in user write thread
  - If erased < reserve then urgent GC
  - If erased < ¼ free space then non-urgent GC
  - If erased > ¼ then no GC
- Background GC in background thread
  - If erased < ½ free space then faster GC
  - If erased > ½ free space then slower GC

Background GC at work

Background GC while FS idle



Red = free space
Green = erased space

- Erased space being freed up
- GC harder in the beginning
- GC slows down
- Writes happen faster
- Better user experience

xattrib support

- Needed for some security etc.
- Limited support to cover most usage scenarios.
  - 1500 bytes of xattrib/file in 2k page NAND
- Cheap implementation
  - Store xattrib data in unused part of object header
  - Very little overhead.

Block refreshing

- NAND flash “leaks”
  - Bit-rot over time & use
  - Excessive reading can even cause problems
  - Particularly bad for MLC
- Solution:
  - Occasionally rewrite oldest block.
    - Low cost
    - Mainly done by background

Lies, damn lies and benchmarks

Real-world performance depends on many variables:

- Hardware speed
- File system state
- Operation sequences
- Usage patterns

So...

- Your mileage may vary
- Test with typical usage patterns if possible

Balloon board test

Busybox script:

- Write 10k files, some fresh, some overwrites
- Delete some files
- Sleep
- Repeat x3
- Measure system time

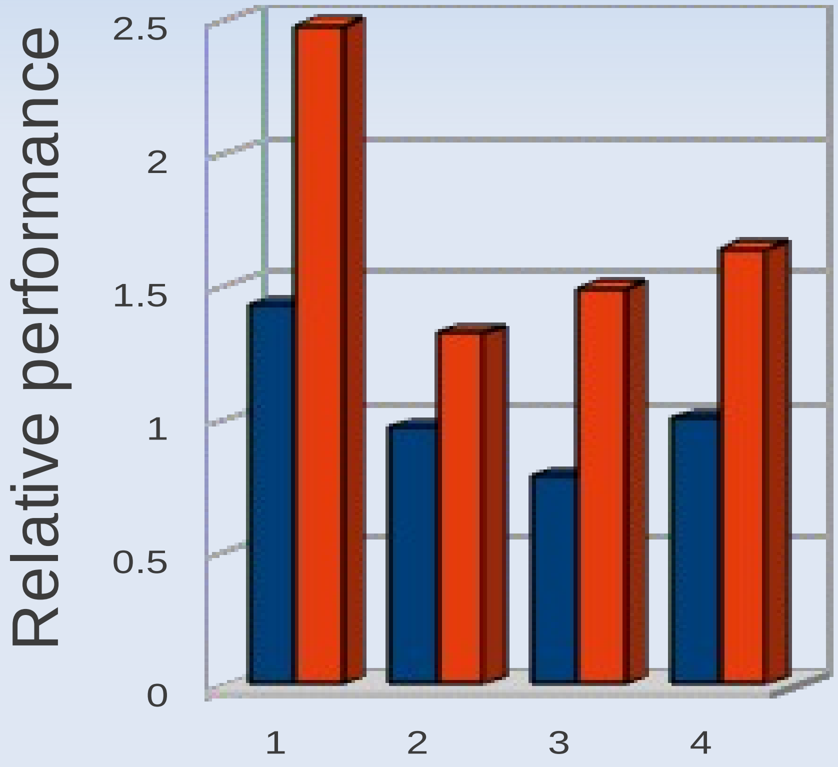
Identical kernel, just switch yaffs code

Results

Code from September 2009: 1378s

Code from October 2010: 840s

Average speed up: 64%



Future features

- Block summaries
  - Save tags in last chunk in block
  - Scan reads reduced by over 90%
    - Faster mounting after unclean shutdown
  - Potentially store other useful info.
- Use background thread for more functions
  - eg. Background data verification for MLC.
- Improved MLC error handling
- Improved caching

Why mainline?

- Some community aversion to patching
  - It is really simple to patch in yaffs:
    - untar snapshot
    - ./patch-ker.sh c m /linux-dir
  - But:
    - Distrust from some quarters.
    - Problem of being a kernel outsider.
    - Keeping synced with VFS changes is hard.
- Mainlining funded by CELF & Google.
  - Thanks!

Mainlining tasks: 1

- Single kernel version of VFS glue code: done
  - Existing VFS glue code is multi-version.
  - Lots of conditional compilation and obsolete code.
  - Streamlined single-kernel variant for mainlining.
  - Multi-kernel version still kept for patching.

```
#if (LINUX_VERSION_CODE > KERNEL_VERSION(2, 6, 17))
static int yaffs_sync_fs(struct super_block *sb, int wait)
#else
static int yaffs_sync_fs(struct super_block *sb)
#endif
{
```

Mainlining tasks: 2

- Split up code: first pass done, maybe more
  - Makes more manageable files
  - Remove confusing clutter (eg. WinCE)
  - yaffs1 and yaffs2 specific code partitioned
- Kernel friendly re-symboling: work in progress
  - yaffs_ScanBackwards → yaffs_scan_backwards
  - Mainly scripted to limit clerical errors.
- Working with kernel team: not yet started

That's all folks

Thanks to:

- Toby Churchill Ltd
- Brightstar Engineering
- CELF
- Google
- The community

Further info: http://www.yaffs.net