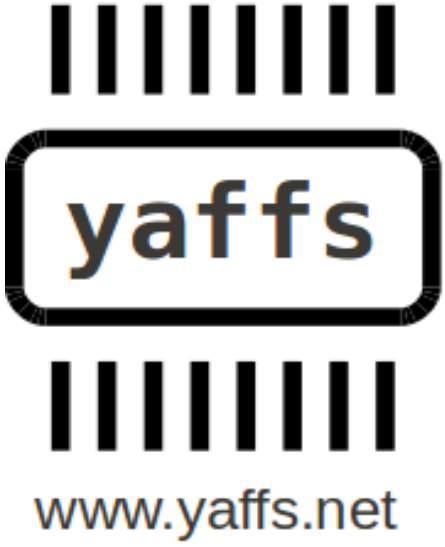

Flash File System Considerations

Charles Manning

2014-08-18

We have been developing flash file system for embedded systems since 1995. During that time, we have gained a wealth of knowledge and experience.

This document outlines some of the major issues that must be considered when selecting a Flash File system for embedded use. It also explains how using the Yaffs File System increases reliability, reduces risk and saves money.

Introduction

As embedded system complexity increases and flash storage prices decrease, we see more use of embedded file systems to store data, and so data corruption and loss has become an important cause of system failure.

When Aleph One began the development of Yaffs in late 2001, we built on experience in flash file system design since 1995. This experience helped to identify the key factors in making Yaffs a fast and robust flash file system now being widely used around the world.

Although it was originally developed for Linux, Yaffs was designed to be highly portable. This has allowed Yaffs to be ported to many different OSs (including Linux, BSD, Windows CE) and numerous RTOSs (including VxWorks, eCos and pSOS).

Yaffs is written in fully portable C code and has been used on many different CPU architectures – both little- and big-endian. These include ARM, x86 and PowerPC, soft cores such as NIOS and Microblaze and even various DSP architectures.

We believe that the Yaffs code base has been used on more OS types and more CPU types than any other file system code base.

Since its introduction, Yaffs has been used in hundreds of different products and hundreds of millions of different devices in widely diverse application areas from mining to aerospace and everything in between.

Specific examples include:

- communications (two-way radios and infrastructure, mobile phones, cell towers, modems, routers...)
- retail (point of sale, inventory control...)
- entertaiment (video cameras, TV sets, DVD players, set-top boxes...)
- sewing machines and office copiers
- industrial equipment (PLCs, machinery control...)
- construction/surveying equipment (theodolites, machinery control...)
- avionics and satellite equipment
- transport (bus, train and traffic-signal management/information systems...)
- and many more.

You may be using Yaffs every day without even realising it.

We have a gratifying number of repeat customers for licences.

Do Embedded Systems have different file system needs?

To answer this question, we really have to start by asking a slightly different question:

How are embedded systems different than regular computers?

When a customer buys a regular computer system, they go shopping for a computer. They talk to the salesperson about Gigabytes, Megahertz and operating systems. They treat it as a computer and expect it to behave like a computer (including shutting it down properly and occasionally having file system problems or needing upgrades).

When they buy a fancy sewing machine, it probably has an embedded file system controlling it. The customer went shopping for a sewing machine and bought a computer that they don't even know about. They don't treat it like a computer, but like a sewing machine; they certainly do not expect it to have file system problems.

How people treat computers differently from embedded systems

Issue	**Computer**	**Embedded system**
Shut down	User expects to do a clean shut down.	User just unplugs device.
Reinstalling software	Users expect to have to deal with corruption and reinstall software occasionally.	Users do not expect to deal with upgrades or data loss or corruption.
Software Upgrade Failure	Annoying	Can be devastating
Attitude to data loss	Annoying, but expected	System has failed.
Cost of failure	Relatively minor	Can shut down factory or even cause injury/death.
Delayed boot due to file system recovery	Annoying, but expected.	This is as bad as a failure.

So people have different attitudes to computers and embedded systems. As a result, embedded systems need to be designed differently from computers. That is particularly important with reliability issues.

Data corruption is a leading cause of embedded system failure, product returns and warranty costs.

The differences outlined above show some of the reasons why embedded systems must perform differently to regular computers, particularly in the way data storage is handled. These examples allow us to identify some very important product-level criteria for embedded file systems:

- High resilience against data corruption, particularly in the event of power loss.
- Prompt file system recovery after power loss - no excessive time penalty

- Simple data integrity model to ensure data is not compromised.

Of course we also want the embedded file system to have some other very useful features:

- High read/write performance with low latency.
- Familiar programming models to facilitate development.
- Proven track record to reduce risk.
- Full source code access to aid development.
- Portability to provide a future-proof solution for new products.

Selecting an appropriate file system is crucial to product success.

Why does power failure corrupt data?

Most power fail data loss in a File System is caused through two mechanisms:

- Corruption of allocation tables
- Loss of cached data that was not yet written to the flash.

Corruption of allocation tables and similar structures.

Most file systems have allocation tables, bitmaps or similar structures which hold pointers showing which parts of the file storage medium are being used and which parts of which file are stored there. These tables are critical to looking up files and data in the file system. If they are corrupted, then files and data become corrupted. In the most dramatic cases, table corruption can destroy the whole system.

Some file systems make these tables more robust by keeping two copies, or by using journaling (which keeps a short log of changes, allowing the tables to be fixed up if the table changing gets interrupted) but this causes extra writes to the media, making the file system slower.

These tables are typically updated many times during writing any file, so there are multiple opportunities for the tables to be corrupted or left in an inconsistent state due to a power loss.

If power is interrupted at a critical point, then some restoration (such as Microsoft chkdsk or booting in safe mode) is often required to try to restore the tables. This may be automatic, but then system start-up is delayed.

Yaffs is very different; it is a log structured file system which writes tagged data to the flash, and does not store allocation tables or such.

If it isn't even stored, it can't be corrupted!

This has three dramatic effects:

- Yaffs has no tables, so is immune to these failure modes.
- There are no extra tables to be written, so Yaffs does not have to resort to caching to achieve good performance.
- Less flash writes means less flash wear, improving product life and reducing power consumption.

When a Yaffs file system is booted, the normal boot up sequence takes care of building up system state. There is thus no need for any special restoration code to be run. A boot up after a power failure is not a special recovery condition, it is just a normal boot up.

Yaffs does not have, or need, special recovery handling.

Trade-off in cached data

File systems tend to be inefficient in the way they write data. Writing just one file requires many changes to the allocation tables; writing many files requires far more changes, often to those same tables. By delaying those writes and holding data in RAM briefly the file system can do fewer writes to achieve the same set of changes. This can improve write speed, at a price.

Data which has not been written but just cached in RAM is vulnerable while it is there; it can all be lost if power fails, even including some that the FS had apparently written already.

Most file systems support functions to reduce this inefficiency and force the cache to be written out, but this may take many seconds, leaving a significant period of vulnerability, and delaying system operations. Excessive use of these functions to improve reliability will typically impair write performance.

A particlarly serious example of critical data loss can occur if power or communication fails during a firmware upgrade. Such a failure cripples the product, which must be returned for servicing. Yaffs has caching policies and data integrity models which make it easier to avoid this kind of fatal failure, and which have been tested very extensively.

Since Yaffs is inherently faster and simpler, it does not need deep caching. The Yaffs cache is small and used only for data alignment purposes and improving the throughput of very small reads and writes. Yaffs files are always flushed from the cache when they are closed. Yaffs also provides its own cache-flushing functions, but these are very fast because the cache is small.

By the time Yaffs closes a file, all the data is guaranteed to be safe on the flash.

This simple caching policy makes it very easy for programmers to write software that is robust against power loss and avoids obscure failure modes. The simplicity confers more reliable operation, faster product development and huge savings in development and maintenance costs.

Reliable file systems pay for themselves many times over.

Product risk reduction

At first glance, a file system seems like it should be a simple body of code: just a few functions that read, write data and look things up by name. However, the reality is much more complex than that.

One of the most challenging aspects of file systems is that errors are stored and incremental. Unlike most software errors which can be cleared by a reset, storage errors are persistent.

A reboot will typically not fix file-system errors.

Customers might tolerate a product that needs to be reset once in a while, but file system failures are often fatal. As a result, file system issues can be very damaging to long-term product reliability.

This is further complicated by flash memory. NAND flash is the only type of semiconductor hardware which leaves the factory with known variable errors, expecting the file system software to handle the problems.

Flash, NAND in particular, has its own failure modes and these are best handled by a file system specifically designed for NAND.

Yaffs was specifically designed to use with NAND and NOR flash chips.

Flash memory chip errors may only show up after the product has been used a while, or in a fraction of the flash parts. Typical testing during development (e.g. bench testing) will often fail to expose all types of flash chip error, and others will only be exposed during pilot- or early-production. These put release dates at risk, which can cause huge revenue losses.

We have a few clients who only determined that they had file system issues when they did a pilot run. That prompted them to try Yaffs and get their products shipping. While we are glad they ended up with a good product, unfortunately they lost revenue and customer goodwill.

Finding problems late in development creates business risk.

It is thus not surprising that much of our business is repeat business.

Yaffs testing

Testing on its own does not create good software but it does help to verify that the software is working correctly.

Yaffs has many automated tests that are run on a regular basis. Automated tests allow the testing to be run without human intervention, thus allowing more testing to be done.

The tests are too numerous to discuss completely (see the Further Information below). We highlight one: the power-fail stress test.

The power-fail stress test is a test harness that uses software simulations of flash devices and some software and scripts to simulate the effects of removing power from a Yaffs-based system while it is writing to flash. This test is normally run for 100,0000 to 1 million power-fail simulation cycles, taking many days.

This particular test suite was originally developed in 2008 during the development of some new features. The tests are very rigorous. In the beginning, this test suite would find a problem after less than 100 power fail iterations. As we identified and fixed the bugs, this number crept up and we now run millions of cycles without seeing a single problem.

Yaffs survives millions of power failures in simulation test benches.

In addition, many of our clients have also run hardware power failure tests, using a relay and a micro-controller to kill power to the device under test. They have been very satisfied with the results.

Licensing

Yaffs is available under GNU Public License V2, GPL2, from our git repository at www.yaffs.net. You may use that code for testing free of charge and for as long as you want.

You will be using it under the conditions of GPL2 with the obligation to release your associated source code when you start to sell or distribute a product.

We can then sell you a Licence to use Yaffs which removes that obligation and protects your intellectual property.

With Yaffs you can try before you buy!

Conclusions

Selecting a robust flash file system is key to :

- building a successful product.
- reducing risk, warranty costs and liability.
- reducing development costs.

Yaffs has proven itself and can pay for itself many times over.

Further information

This document is intentionally brief. There are many documents at http://www.yaffs.net for those that want more information, including:

How Yaffs Works: detailed description of internal design.

Yaffs Robustness and Testing: how we know Yaffs handles various flash failure scenarios.

For further information please contact info@yaffs.net or access http://www.yaffs.net