David Woodhouse

JFFS2

David Woodhouse at the West Yorkshire Linux User Group meeting in December

Richard Ibbotson caught up with Red Hat's David Woodhouse to get the lowdown on the JFFS2 filesystem

David lives in Cambridge and describes himself as a "born and bred Norfolk countryman". His interest in programming developed from an early age when he began hacking BASIC. David is now a contractor for Red Hat working on embedded GNU/Linux projects and is responsible for the kernel drivers for memory technology devices. He hopes that one day he will become a kernel hacker.

David's latest creation, JFFS2, might not sound all that glamorous but it will make some useful changes to the hard disks of many people when it becomes a part of the ordinary GNU/Linux based system.

What's it all about?

Until recently the usual approach for using flash memory technology in embedded devices was to use a pseudo-filesystem on the flash chips to emulate a block device. The original JFFS was a log-structured filesystem, developed by Axis Communications AB in Sweden, specifically for use on flash devices in embedded systems. Later on Red Hat came along and did some more development work on it. JFFS2 is the end result of their labours. It's aware of the restrictions imposed by flash technology and operates directly on the flash hardware thereby avoiding the problems of having two journaling filesystems running on top of each other. There are plans to port it to eCos.

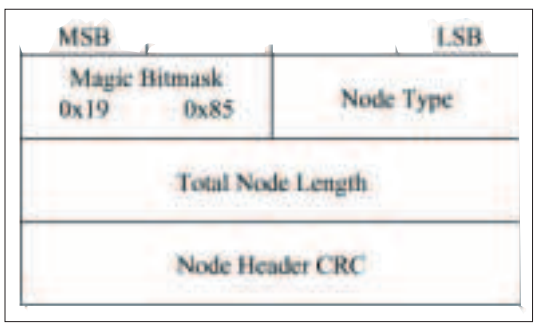
Flash memory

Flash is a particular type of EEPROM memory which is available in the NOR version, or you can also get hold of the newer NAND version at lower cost. Both types can set a bit in a clean chip to a logical one and both can be set to zero by a write operation such as storage of data.

In order to provide for wear levelling and reliable operation without problems, sectors of the block device (hard drive to your ordinary PC user) are stored in various places on the medium and a translation layer is used to keep track of the location of sector. This layer is effectively a type of journaling filesystem.

The flash translation layer is a part of the PCMCIA specification, which is used in most laptops. It's an unfortunate fact of life that this and similar technologies are held together with patents that more or less prevent further development, or if development is allowed then the legal ramifications are so extreme that it makes life as an engineer or developer very hard indeed. GNU/Linux supports FTL and similar technologies but it's use is deprecated and is really only meant to be there for backward compatibility. Even if the patent issue could be circumvented the technology itself is not seen as the best solution to a complex list of technical issues. It would be much better if there were no extra translation layers in between.

In the design and implementation stages, the original JFFS included the useful thinking that the users of the finished hardware would probably do some very stupid things to it – things like the power being suddenly switched off by a user who didn't know how to close down a computer properly. Or to



JFFS2 Common Node Header

put it another way, battery-powered devices are just seen as simple appliances like the common kitchen kettle or iron and they are treated in a similar way. Such design ideas are often the province of your average electrical engineer or design engineer. Many man hours are spent on deliberating over these kinds of issues. The design of the French and Japanese rail systems contain many examples of this way of doing things. Whether it's Concorde or just a chip on a board then quality control must be a part of the design process.

In the beginning

The original JFFS was a log structured filesystem or LFS. Nodes containing data moved along the data storage device in a linear fashion until all of the storage space had been used. There is only one type of node in the log, which is referred to as 'struct jffs_raw_inode'. When the medium is mounted, all of the data is read and each inode reconstructs the data tree and the amount of information that is available for use by an operator.

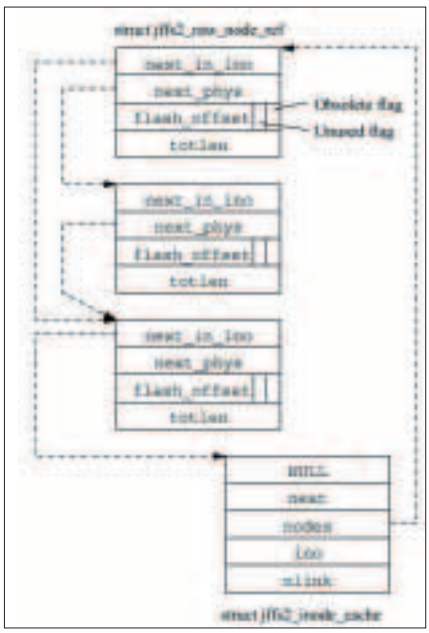
This is all very simple so far and most people could probably understand what all of this is about. JFFS1 also does garbage collection. This takes care of old writes to the media by collecting the data from the tail of the log and putting it at the front end of the log. This takes care of old inodes, which might just be taking up dirty space, which were left behind by old writes. Kernel threads usually take care of this kind of activity. However, the developers of the technology will tell you that this may not be the best way to do things. Certain aspects of JFFS had not been tested in Axis products and writing data other than at the end of the file did not work, and deleting a file while a process still had a valid descriptor for it could cause a kernel oops. Not the sort of thing that wins friends and influences people! JFFS did reach a point in a few weeks after testing where it was stable and reliable. There were however thought to be problems with garbage collection, compression and hard links which needed to be sorted out.

A new solution

JFFS2 was the solution to these perceived problems. In January 2001 a discussion began as to how to completely re-implement the original design so that something called JFFS 2 would be brought into the R&D arena and eventually sold on to a prospective end user.

A number of different things were tried out before the version that is around at the time of writing was arrived at. Various node headers proved to be slightly unreliable and a CRC was added at a later stage. Also the bugs caused by garbage collection in a strict order were sorted out by allowing each erasable block to be treated individually. The nodes are also not allowed to overlap block boundaries as they were allowed to in JFFS1. This makes for increased efficiencies in the garbage collection code. More intelligent decisions are carried out and a particular location for collection may be more favourable than another. Erase blocks will be on a clean_list or a dirty_list. There is also a free_list, which contains a valid mode to show that the block was correctly erased.

There is what is referred to as a third change when JFFS2 is looked into when compared with JFFS1. There is a separation between directory entries and inodes. At the time that this was written there are three types of nodes defined and in use with JFFS2. These are: JFFS2_NODETYPE_INODE, JFFS2_NODETYPE_DIRENT and JFFS2_NODETYPE_NODEMARKER. JFFS2 works by examining a heuristic threshold and garbage collection starts mounting the filesystem with a multi stage process. First of all a physical scan is made which builds a full map of what's happening followed by a second scan which detects inodes that have no remaining links on the filesystem and a deletion is made. A third pass is made to free the temporary information which was cached for each inode.



Raw node reference lists.

Future plans for JFFS2 include improved fault tolerance. There are several other aspects of the project which are in the minds of the developers for improvement, of which transaction support is one. Although JFFS2 is still in its early days it has quickly grown up and is seen to be a worthwhile technology, which can be relied upon for its stability. Improvements can only draw more attention to it in the coming months and years. JFFS was merged into the GNU/Linux kernel prior to the release of the 2.4 kernel. Its future as an Open Source project rather than a dusty and forgotten object of desire is assured.

Getting involved

If you want to get involved with JFFS2, or you want to know more about it, why not have a look at some of the web pages mentioned below? You should find David's email address on at least one of them.

The author

Richard Ibbotson is the Chairman and organiser for Sheffield Linux User's Group – you can view their web site at *http://www.sheflug.co.uk*

Info

Original JFFS	*http://developer.axis.com/software/jffs*
JFFS2	*http://sources.redhat.com/jffs2*
eCos	*http://www.redhat.com/embedded/technologies/ecos*
JFFS mail archive	*http://mhonarc.axis.se/jffs-dev/threads.html*
	http://lists.infradead.org/mailman/listinfo/linux-mtd