sandra zuerlein, Fotolia

Flash memory and the LogFS filesystem

FRIEND OF FLASH

Flash is now an everyday part of the Linux environment. The new LogFS filesystem will help you contend with the problems of flash memory. **BY JAN KLEINERT AND ACHIM LEITNER**

Flash memory was once reserved for embedded applications. Over the course of the past few years, however, prices have fallen drastically for flash in the form of USB sticks, as well as the memory cards used in cameras, PDAs, cellphones, solid state drives, and MP3 players. Users increasingly need to access flash memory devices with ordinary Linux systems, and the Linux community is responding with better tools for addressing the challenges of flash.

LogFS is a recent entry into the race for a flash-ready Linux filesystem.

Flash Translation Layer

On today's systems, a normal application may not even be aware of whether it is using flash memory or an ordinary filesystem. Responsibility for communicating with the device is placed firmly in the hands of the hardware or the kernel. A Flash Translation Layer, or FTL for short, was introduced to the kernel around the year 2000. The layer handles the special quirks of addressing flash memory.

FTL is designed to let the filesystem read and write sectors without knowing how the memory is organized, and without bothering to delete before writing (see "Flash Technology" box). At present, the Linux kernel supports five FTL variants in the memory subsystem.

Because filesystem blocks are far smaller than flash blocks, FTL optimizes write access by collating changes to save deletion and write cycles, thereby improving the longevity and throughput of the chip. This technique requires management structures (such as block mapping) that collaborate with garbage collection. The controller handles this in the case of, for example, a compact flash card (CF) or a USB memory stick, serv-

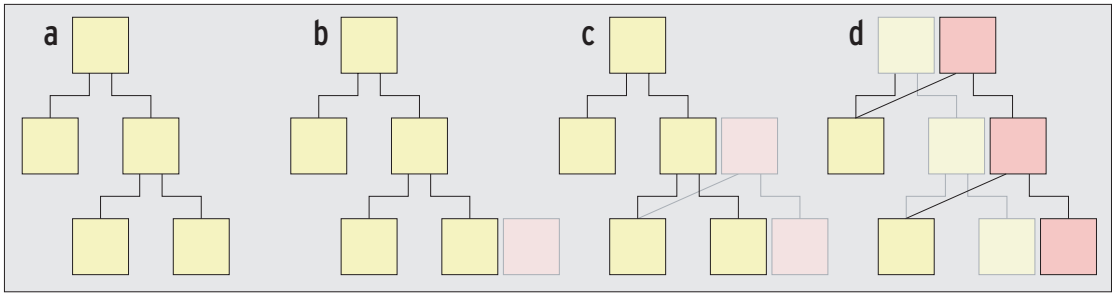

Figure 1: LogFS writes changes out of place. In the tree structure (a) it creates a modified block at a new position (b) and leaves the old block unchanged. The change is not yet linked, and Log-FS thus needs to create the parent node (c). The new structure does not apply until the root node has been processed (d). This keeps the filesystem in a consistent state.

ing a normal ATA interface to the system. Filesystems such as ext2 or FAT are not optimized for use with flash memory. The FTL does not even know if a block has been deleted because the filesystem does not pass this information to the block layer. This makes garbage collection far more difficult.

Introducing LogFS

Special systems such as JFFS2 (Journaling Flash File System 2 [1], a development based on the original JFFS [2]), UBIFS [3], or YAFFS [4] step in to address the complications posed by flash memory. The name "Journaling Flash Filesystem" is unfortunate because JFFS really uses a log-based structure. A new shooting star in this part of the Linux sky is LogFS [5], which should not be confused with the now-defunct Log Structured File System (LFS) project [6].

LogFS vs. JFFS2

Jörn Engel, the LogFS maintainer, is positioning LogFS against JFFS2. Engel expects to score points against JFFS2 on issues such as performance. For example, Engel claims it takes up to 15 minutes to mount a 1GB USB stick in JFFS.

Moving Trees

The clever thing about LogFS is its tree-style management structure. LogFS doesn't delete and write modified blocks in place, but deposits the contents at the end of the used memory area. Thus, it collates changes that cover the segments of several erase blocks – like the garbage collection methods in FTD.

To update the tree structure, LogFS rewrites all the branches from the modified block down to the root at the end of the flash memory in use, thus implicitly declaring all previously used blocks as free memory (Figure 1). On mounting a LogFS device, the driver searches for the last root node. The data structure will be consistent below this node. This approach saves delete and write actions.

Using LogFS

To use LogFS, you need to patch the vanilla kernel source. The LogFS site [5] has patches for Linux 2.6.18, 2.6.20, 2.6.21, and 2.6.23. The mklogfs tool is available from the same source.

After configuring, building, and booting, you can create a Memory Technology Device (MTD [7]) *mklogfs /dev/mtd0* and mount the device on the filesystem: *mount mtd0 /mnt -t logfs*.

LogFS is currently under development. Jörn Engel says that his filesystem still updates the tree too often [5] and thus gives away performance. We can only wish him good luck with LogFS.

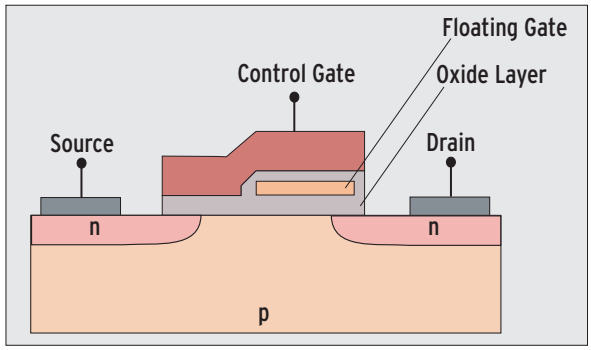

Figure 2: The oxide layer surrounding the floating gate prevents the electrons from escaping. (Fig. source: Wikipedia).

Alternative Solutions

Developers use several approaches to address the complications of the Flash architecture. This article showed how to cope with these flash technical issues at the block layer (with FTL) and through the filesystem (with LogFS).

Other options are to address flash issues at the device level or to build support directly into the application, a technique often used with highly specialized embedded applications. ■

Flash Technology

Flash EEPROMs are memory chips with non-volatile content; that is, they retain data without a power supply. Because of the small footprint of the chips and the low manufacturing prices, NAND flash chips that connect a certain number of flash transistors (floating gates) in series are typically used. Because selective write operations only toggle the logical state from false to true, a logical operation is required prior to each write to introduce a delete cycle.

Blockwise Deletion

In contrast to normal EEPROM memory, a Flash EEPROM cannot delete a single word, the smallest addressable memory unit (8 to 64 bits) individually, but only a block. A block is typically a quarter, eighth, sixteenth, and so on of the total memory capacity of the chip. These EEPROM blocks (also known as erase blocks) are far bigger than the blocks that legacy filesystems use. At the same time, the number of deletion cycles (endurance) is limited. Manufacturers will not guarantee more than 100,000 to a million cycles for each block.

The reason for wear and tear is that electrons in the floating gates tunnel through the oxide layer of the transistor on deletion (a phenomenon known as the Fowler-Nordheim tunnel effect). The high voltage required for this quantum mechanical effect damages the oxide layer surrounding the floating gate slightly each time a delete occurs (see Figure 2). Once the degeneration process has reached a certain threshold, the electrons trapped in the transistor start to escape; the bit stored at this location is lost, and the memory cell is defective. Flash device vendors attempt to counteract this effect by including reserve cells and defect management to map out defective blocks.

NAND Flashes

Advantages:

- Short access times
- Low energy consumption
- Data retention without a power
- Resistance to shocks and magnetic fields

Disadvantages:

- Slow write operations
- Deletions by sector only
- Limited number of write cycles
- More expensive than alternatives

INFO

[1] JFFS 2: *http://sources.redhat.com/jffs2/*

[2] JFFS: *http://developer.axis.com/old/software/jffs/index.html*

[3] UBIFS: *http://www.linux-mtd.infradead.org/doc/ubifs.html*

[4] YAFFS: *http://www.yaffs.net*

[5] LogFS: *http://www.logfs.org/logfs/*

[6] LFS: *http://logfs.sourceforge.net*

[7] MTD: *http://linux-mtd.infradead.org*