Micriμm

EMBEDDED
FILE SYSTEM

WHITE PAPER

DISCLAIMER

Specifications written in this manual are believed to be accurate, but are not guaranteed to be entirely free of error. Specifications in this manual may be changed for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, Micrium assumes no responsibility for any errors or omissions and makes no warranties. Micrium specifically disclaims any implied warranty of fitness for a particular purpose.

COPYRIGHT NOTICE

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of Micrium. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2009; Micriμm, Weston, Florida 33327-1848, U.S.A.

TRADEMARKS

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

CONTACT ADDRESS

Micrium
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.
Phone : +1 954 217 2036
FAX : +1 954 217 2037
WEB : www.micrium.com
Email : support@micrium.com

Micriμm

For the way Engineers work

Micriμm

With faster and larger processor architectures, developers and customers are expecting more from their embedded systems. From list of peripherals and vast capabilities people expect from an embedded system, the availability of a file system is today a no-brainer. The evidence is the number of devices offering flash storage.

A few design choices must be made when deciding to use a file system in an embedded design. Among the decisions, are:

1. The type of file system structure required (Microsoft FAT, Linux EFS, or proprietary)
2. File compatibility with the file system of other Operating System.
3. The storage medium type to be used
4. The requirement for a removable medium.

All of these items have an impact on hardware/software design. In an embedded system, the storage medium is often not a hard disk drive. It is more often a Flash-based medium. As such, the main design constraint relates to the way the file system manages data on the storage medium in the event of power reset or failure.

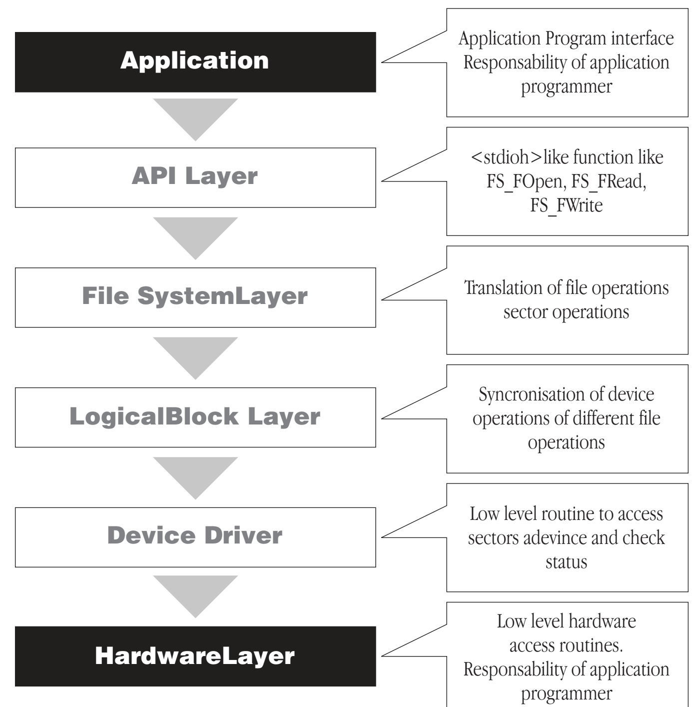
On a system featuring a hard disk drive and a good power supply, the OS File System has the means to guarantee that the file system can recuperate from a power reset or failure. In comparison, in an embedded system with a Flash-based medium, the minimum data element (sector or iNode) must be erased before it is rewritten. This requirement adds constraints to the design that are very different than from those of a PC running Windows“ or Linux.

In addition, when the medium is removable, disconnecting the medium from the embedded system while the file system is operating may cause corruption. This is similar to the requirement to click on the “Safe to Remove” icon in Windows before removing a USB mass storage device. Removing the medium or losing power produces similar effects.

Depending on the type of file systems used, there are ways to make an embedded system with a Flash-based storage medium “power fail safe.”

Reliability and performance are the critical elements of a Flash file system. Embedded device consumers make their purchase decisions based on ease of use, speed of operation (they don't want to wait for images to load, or contacts to be accessed), and reliability, among other factors. Examples of such devices include cell phones and PDAs.

TYPICAL FILE SYSTEM STRUCTURE



API Layer

The API layer is the interface between the File System and the user application. It contains such file functions as FS_FOpen() and FS_FWrite(). The API layer transfers these calls to the file system layer.

File System Layer

The file system layer translates file operations to logical block (sector) operations.

After such a translation, the file system calls the logical block layer, specifying the corresponding device driver for a device.

Logical Block Layer
The main purpose of the logical block layer is to synchronize access to a device driver, and to provide an easy interface for the file system layer. The logical block layer calls a device driver to perform a block operation. This is also the location of the cache mechanism.

Device Driver
Device drivers are low-level routines used to access sectors of the device and to check status. It is hardware independent but depends on the storage medium.

Hardware Layer
These are the low-level routines to access hardware. These routines simply read and store fixed-length sectors. The structure of the device driver is simple in order to allow easy integration of diverse hardware.

Flash Translation Layer
The flash translation layer establishes the link between "logical sectors" or "blocks," and "physical sectors". A logical sector/block is the base unit of any file system. Its typical size is 512 bytes. A physical sector is an array of bytes on the flash chip that are erased together (typically between 2 Kbytes to 128 Kbytes). This is valid for every type of Flash device.

The flash translation layer is an abstraction layer between these two types of sectors. Every time a logical sector is updated, it is marked as invalid and the new contents of this sector are written into another area of the flash. The physical address and the order of physical sectors can change with every write access. Hence, there is no direct relation between the sector number and its physical location.

The flash translation layer manages the logical sector numbers by writing sectors into special headers. It does not matter to the upper layer where the logical sector is stored, or how much flash memory is used as a buffer. All logical sectors (starting with Sector #0) always exist and are always available for user access.

FILE SYSTEM TYPES

The typical file system types are FAT, journaling and transactional.

- FAT is the traditional file system, originally developed for use in desktop systems or servers, using a file-allocation table that tracks how files and directories are stored throughout the storage media. FAT file systems lack a means of synchronizing data in case of system failure, and are therefore susceptible to corruption. FAT is often referred to as Microsoft FAT, as it is the most widely used file system since DOS originally used it.
- Journaling file systems keep a log of all file-system activities. Although more robust than a FAT file system, journaling file systems offer improved reliability at the expense of heavy system resource use. JFFS is an example of a Journaling Flash File System.
- Transactional file systems were developed specifically for embedded systems; they preserve both user data and metadata through committing the known good state of the file system to memory, with an atomic transaction point event.

RELIABILITY

Many features of a flash-based file system can affect the overall system reliability. Here are some of the main features:

ATOMIC WRITES

To guarantee device data integrity, regardless of whether the device is used in optimum or adverse conditions, data must be either fully written to sectors on the storage media, or not touched at all. The mechanism in the flash-based file system that preserves data structures should be automatic and invisible to the developer. Additionally, it should ensure data integrity in the file system, as well as in the block device driver.

WEAR-LEVELING AND BAD-BLOCK MANAGEMENT

Flash devices require strict management of data blocks via software to ensure the chip's expected lifetime is optimized and to avoid write errors. A flash device must be erased before data can be changed. Wear-leveling algorithms ensure that erase zones are used evenly across the entire flash device. Wear leveling is especially useful when large portions of the flash device are used for long-term data storage.

For example, a single configuration file may be written to an embedded device once and stored (and accessed) over an extended period of time. In addition to wear leveling, NAND flash requires a bad-block management scheme that efficiently detects blocks that cannot be written to, and reserves replacement blocks on the flash device. When properly implemented, bad-block management extends the life of a flash device.

PERFORMANCE

As in many industries, the pressure by management and consumers to rapidly bring to market new embedded devices that perform complex tasks is a fact of life. Flash-based file systems are a sub-system in a device that contributes to this goal. Some of the features of the flash-based file system to support this demand are:

MULTI-THREADING

One of the characteristics of a Flash device is to have faster read times than write times. In systems that support multi-threaded operations, read requests can be allowed to interrupt more time-consuming write and erase operations. For example, a device featuring mobile TV applications that supports multi-threaded operations will improve the performance of streaming media applications.

Garbage collection (also referred a Background Compaction)
As mentioned above, flash memory requires intelligent software to conduct write and erase operations in order to keep valid data while throwing away unwanted data. An element of the Flash device that was previously used, and freed at a later point, contains invalid data. Before this element is reused, it must be erased. This process sometimes occurs during write operations to properly complete a write. However, write speeds can be optimized when the garbage collection takes place during the flash-based file system idle time, in preparation for a future write. This is referred to as background compaction, and results in significant write performance improvement.

BOOT TIMES

The boot process of a Flash file system involves mounting the flash drive first, followed by mounting the file system. The file-system mount varies greatly depending upon the type of file system used. This is the quick way to boot a system. If integrity is a mandatory requirement, mounting the flash disk requires a scan of the entire disk. The length of the scan process is directly proportional to the size of the disk. File systems such as FAT and Journaling use a lengthy CHKDSK utility to scan file system integrity and perform repair operations, if necessary. These file systems can take up to several minutes to mount. The fastest and most reliable option is a transactional file system, which completes the process by reading only two data blocks (the committed state and the known good state), and finishes in less than one second.

OTHER FACTORS

As mentioned earlier, a key driver is delivering product to market faster than the competition, given a demanding development schedule and budget constraints. Market demands push design issues beyond performance and reliability. The development process must also take into consideration other technical subjects.
Amongst these, ease of integration into the overall system design is a key consideration. Often, a system design involves a number of vendors who must integrate their products to operate as one. Setbacks during integration are common, and unfortunately costly. A software solution that is easy to integrate with other system components reduces the risk of product delays. Software that can be reused on future device designs, with minimal modifications, has greater benefits in mitigating delivery risk. Multiple studies have demonstrated that using COTS products is the best way to minimize development cost and to reduce time-to-market delays.

FLASH-BASED FILE SYSTEM FEATURES THAT MAY SPEED INTEGRATION INCLUDE:

Hardware Support
During production, it is likely that flash elements from multiple vendors will be used. The number of different parts (NOR or NAND) is considerable. A flash based file system solution that is limited to supporting one type of flash (i.e. NAND only) or parts from only one vendor is not a good choice for any high-volume device manufacturer.

The ideal software should auto-detect and support any flash technology (NAND and NOR as well as advanced flash technologies such as NAND controllers on processors, Multi-Level Cell and hybrid flash). Devices that use more than one type of flash (i.e. NOR to store application code, NAND to store data) will also benefit from a flash based file system that allows for writes to varying block sizes with a single instance of the software.

Portability
As we want our Flash-based file system to integrate with a variety of hardware, we also want it to not be tied to a single operating system environment or application interface. Instead, it should interface easily with the operating system, Flash hardware, application code, and the development tool chain. These four factors provide for quick integration. Flash-based file systems that coexist with an operating system's native file system offer greater flexibility to designers that wish to take advantage of selected operating system. Support for Unicode file names, long file names and support for POSIX are other factors that also enhance the ease of portability of a Flash-based file system.

Source Code and Documentation
It is always a good practice to select a vendor that provides source code. With full source code, it is simple to use a Flash-based file system in a wide range of products. The same statement can be made for comprehensive documentation.

Scalability
Typically, when a manufacturer develops a product using a file system, other products soon follow. With source code availability and excellent documentation used in a system developed the first time, the ability to easily integrate will provide a head start for additional designs.

When selecting a Flash file system that can be used as a standard across multiple platforms, the solution must be scalable. It is preferable if the file system configuration is performed at run time versus compile time, allowing for various system settings and performance. Flexibility across platforms includes support for multiple disk partitions and multiple file system volumes. Another important feature is the use of a single Flash device for both system data and file-system data. The opposite is also often required: using multiple flash devices as a single drive.

Vendor Issues
In an evaluation of Flash-based file systems, the software vendor must also be chosen carefully. Not all flash based file systems are created equal. Experience with similar design projects is critical. Previous projects with FAA or FDA certification, for example, provide an additional level of confidence in the vendor, even if the current design is not quite as complex.

CONCLUSION

To evaluate a Flash-based file system, it is important to consider end-user needs as well as design- team requirements. An absolute must when using a Flash storage device is to have a file system that uses a device driver which implements atomic transaction point event. This type of file system is known as a transactional file system.

With Flash as the storage medium, data integrity mechanisms are crucial. Each physical block in a flash device has a maximum number of erase cycles. Wear leveling and bad-block management algorithms including power fail-safe features are therefore a minimum requirement of any flash driver.

To optimize the performance of the file system, read and write operations should be enabled through multi-threaded support and background garbage collection to erase used physical sectors in preparation for their next use. A good file system should also allow for quick boot times, as it is always annoying having to wait for a device to come up, whatever the device is.

Finally, the file system vendor should provide a wide range of supported hardware and operating systems. This includes, of course, the availability of quality source code and documentation, both based upon using good quality standards.

Christian Legare
Vice-President, Micrium