Introduction to Samsung's Linux Flash File System - RFS

Application Note

November-2006, Version 1.0

SAMSUNG ELECTRONICS

Copyright Notice

Contact Information

Flash Software Group
Memory Division
Samsung Electronics Co., Ltd

Address: San #16 Banwol-Dong, Taean-Eup
Hwasung-City,Gyeonggi-Do, Korea, 445-701

Introduction

Flash Memory is generally employed in the embedded devices due to its high stability, large size and cost effectiveness. Due to rapid increase in the size of flash memory device, flash memory based file system is widely adapted in mobile computing devices such as PDA, MP3 player, digital camera, mobile phone, and digital camcorder. Flash memory based file system (shortly, flash file system) for the mobile devices, however, has different requirements with the traditional file systems developed for hard disk systems.

One of the most important requirements of the flash file system is robustness. Unlike desktop computers, users do not care about doing random operations with the mobile devices and do not blame themselves for extracting batteries from their mobile phones at any time. Because of their portablity and small size, a user may drop it on floor occasionally. Nevertheless, the possibility of sudden power failures in mobile devices is much higher than desktop computer and such the power failures may break file system. The result of broken file system is much critical because it is not easy to fix broken file system of mobile devices. Therefore the research on the robust flash file system is getting more important than before.

Samsung's Linux RFS (Linux Robust FAT File System) is an embedded file system to use OneNAND/ NAND flash memory as storage on any consumer electronic devices. As its name implies, Linux RFS runs in the Linux kernel and is fully compatible with FAT file system standards (FAT16/32).

A flash memory does not work like an ordinary block device; it is not possible to write twice to the same memory location without first performing a very time-expensive erase. NAND flash memory is programmed on a page basis and erased on a block basis. NAND flash memory cannot be overwritten without erasing a block. And the size of block is bigger than the size of a page because a block is composed of N pages. An erase must be done on whole block at a time and a common block size is 128KB. That is the reason why Samsung came out with a new file system specialized for NAND based flash memories. An already existing file system would not work. For 'robustness', it provides a journaling based error recovery mechanism, which guarantees that the file system runs at all times even if there is a sudden power loss. Of course this file system had to fit well in the Linux kernel (like any other file system implemented under Linux).

Currently, there are a few open-source projects that implement NAND based flash file systems such as JFFS2 and YAFFS. However, they have a limited applicability to brand-new OneNAND flash memory devices that feature advanced technologies to improve performance. Both the file systems are based on LFS (Log-Structured File system) which appends logs circularly over the whole storage. Since LFS does not overwrite old sectors and uses garbage collection mechanism instead, LFS can cover special characteristics of flash memory. Moreover, logging structure has a big advantage for crash recovering. However, in spite of the advantages, the log-structured native flash file systems are not commonly used in mobile devices. It is because of two major drawbacks; lack of compatibility, and requiring much resource.

Currently most mobile devices use the FAT file system for their storage, and Microsoft Windows Operating System cannot recognize JFFS or YAFFS. Users may want to read or write files of the mobile devices in their desktop computer, but it is not easy for JFFS and YAFFS. The other critical drawback is that the log structured file system requires too much resource and long mounting time. It is not adequate for mobile devices, since they have usually restricted resources and require fast mounting time.

The other approach to use flash memory as data storage of mobile devices is using FTL (flash translation layer). Samsung has come out with XSR which's a flash managmenet software to use NAND flash as a regular block device. XSR is a kind of software layer which emulates a hard disk with flash memory. Because the FTL provides the same functionalities of hard disk system through its unique mapping algorithm, most existing hard disk based file systems can be implemented with flash memory.

Robust File System

The design goals of RFS is niched out of FTL in order to use flash memory as data storage. The NAND-based file system, RFS, is based on XSR to manipulate the NAND flash memory, which architecture is shown in Figure 1. In Figure 1, there are four different layers in the whole system: application layer, RFS FAT file system, XSR, basic I/O layer and driver layer. The most important layer is

flash management layer. In this layer, RFS provides its main functionality.

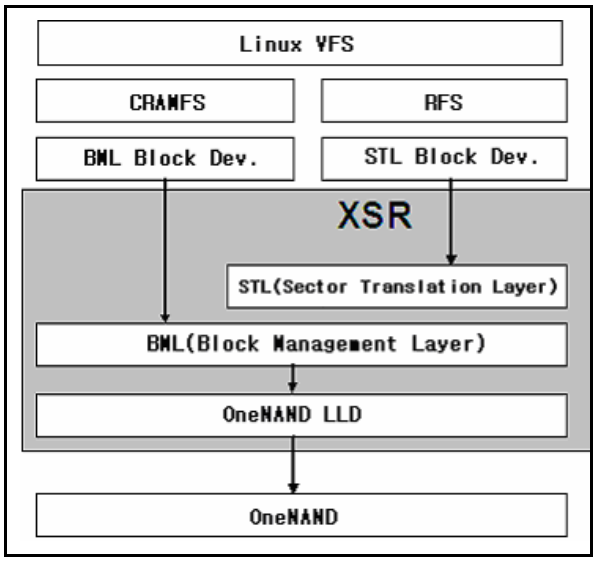

Figure 1. Linux RFS Package

System calls are provided by RFS for applications and operating system, by which they can access data on the external flash memory. Since a page in flash memory cannot be overwritten before it is erased and it takes much longer time to erase a block, XSR (extended Sector Remapping) is usually employed between the kernel and flash memory. The main role of XSR is to emulate the functionality of hard disks with flash memory, hiding the latency of erase operation as much as possible. XSR achieves this by redirecting each write request from the kernel to an empty location in flash memory that has been erased in advance, and by managing the mapping information internally. According to the granularity with which the mapping information is managed, XSR can be classified as page-mapped, block-mapped, or hybrid. XSR is composed of 2 layers: Sector Translation Layer (STL) and Block Management Layer (BML). STL translates a logical address from the file system into the virtual flash address. It internally has wear-leveling during the address translation. BML translates the virtual address from the upper layer into the physical address. At this time, BML does the address translation in consideration of bad block and the number of NAND device in use. BML accesses LLD , which actually performs read, write, or erase operation, with the physical address. The FTL also performs garbage collection to reclaim free pages by erasing appropriate blocks. Since the lifetime of flash memory is limited, FTL may utilize a wear leveling mechanism so that the block erase counts are distributed evenly across the media.

XSR uses the log block FTL algorithm so that a new sector always written to a log block, not to data blocks directly. With this mechanism, rollback of bounded multiple sectors is possible. The log block uses the out-of-place policy, the data block uses the in-place policy.The log block FTL algorithm uses small number of log blocks as cache of large number of data blocks because the out-of-place policy is efficient for writing process but the in-pIace policy does not requires page mapping table. All writes are done to the log block and the full-log block is merged with the old data block to a new data block

FTL based File System Advantage

The advantage of this method is that the existing file systems, such as Ext2 and FAT, can be used directly on the flash translation layer. If you only need FTL and no File System, Samsung can provide Block device layer for your own file system.

RFS accesses the Linux kernel through the VFS layer.However, the direct use of an existing file system has many performance restrictions and is, thus, inefficient because the existing file systems are designed for disk-based storage systems.

The difference between FTL and other flash file systems is that FTL provides sectors and, in the case of Linux operating system, has a File Allocation Table (FAT), rather than linking files and directories, as do some other flash solutions. Because FTL emulates a sectorbased system, it has several advantages over its counterparts. FTL has a smaller footprint, fully supports disk recovery utilities,and it can be cached

FAT Support

The FAT file system is one of the most widely used file systems on any FTL. The FAT file system has long history; it is well known, simple, and fast mountable. RFS supports FAT16/32. A FAT storage medium is physically divided into sectors (traditionally 512 bytes each). From the file system's point of view, the storage medium is linear array of sectors, starting from the first sector, sector 0. The lowermost sectors of the storage medium contain the basic FAT structures, including the FAT table. These are followed by the rest of the storage medium, which contains all the file and directory data and available free space. This part is divided in to allocation units, also called clusters.

An allocation unit is the minimum space that can be allocated to a file or a directory and its size is fixed througout the storage medium. The size of the allocation unit is the multiple of the sector size e.g. 4Kbytes(=8 sectors). The file system does not use an allocation table to store file system's meta information.Physical sectors are logically organized into two regions: a data region for storing data such as files, directories, attributes, etc; and a meta data area of the flash medium to store meta information.

Table 1. Clusters per FAT

FAT Size in bits	Number of Clusters
16	65536
32	4,294,967,296

Table 2. Volume Size

Cluster Size (Bytes)	FAT 16	FAT 32
1024 (1K)	67MB	4.39TB
2048 (2K)	134MB	8.78TB
4096 (4K)	268MB	17.59TB
8192(8K)	536MB	35.18TB
16384(16K)	1.07GB	70.36TB
32768(32K)	2.14GB	140.7TB

Demand Paging

Demand paging allows the various parts of a process to be brought into physical memory as the process needs them to execute. Only the working set of the process, not the entire process, need be in memory at one time. A translation is not established until the actual page is accessed. Embedded devices running Linux can use RFS to reduce the size of the flash footprint and to reduce the amount of RAM required during execution. This approach works because Linux uses a demand paging scheme via the processor's memory management unit (MMU). In simplistic terms, demand paging loads only the first page of a program into memory and then transfers control to the program. As the program runs, it makes references outside the page or transfers control to code residing on a different page. In either case, a page fault occurs when the MMU detects that the page containing the requested information is not loaded. The kernel handles the exception by demand loading the appropriate page (and configuring the MMU data structures appropriately). The program can continue running even when the program is not entirely loaded into RAM. In order to support demand paging, RFS implements memory map read logic through VFS interface. Due to this, application's library and binary files can be located in the RFS file system. In most cases, CRAMFS and other ROFS can also be loaded from the RFS.

Loadable Kernel Module

A Loadable Module is a modularly packaged module with the capability to be dynamically loaded into a running kernel. In the kernel context, this means that the RFS module provide the dynamic loading and unloading ability by using the configuration file, thus reducing the memory requirements by keeping the code modular. Secondly from the legal point of view, RFS is not subject to the GNU Public License (GPL).

Journaling

Avoiding corruption in a file system is a difficult task. To enhance the robustness, Samsung's RFS uses a journaling technique. It avoids corruption by batching groups of changes and committing them all at once to a transaction log. The batched changes guarantee the atomicity of multiple changes. That atomicity guarantee allows the file system to guarantee that operations either happen completely or not at all. Further, if a crash does happen, the system need only replay the transaction log to recover the system to a known state. Replaying the log is an operation that takes at most a few seconds, which is considerably faster than the file system check that nonjournaled file systems must make.

Fast Unlink

Unlinking is a process of resizing or removing a file from the file system. It needs to modify the FAT entries related with file clusters in FAT for reusing clusters. However if a size of file is too large, it takes more time to unlink the file as a size of file grows. In order to reduce the modification of FAT entries, RFS supports Fast unlink in order to unlink a file much faster without modifying all FAT entries releated with a file.

POSIX

POSIX defines the set of services a filesystem must provide. The RFS supports the standard POSIX functionality (including long filenames, access privileges, random writes, truncation, and symbolic links) with the following exception: Truncate is not allowed for a file that was unlinked, but is in-use. Since RFS supports POSIX, the user need not modify their application code that's already used by other POSIX compliant file system.

Summary

Samsung's RFS is an ideal flash software solution for Linux-based embedded system using OneNAND flash memory. It manages the flash media much stably and delivers the most optimized flash management functionalities for OneNAND flash memory. Samsung's RFS delivers superior read/write performance compared to the existing solutions.

References

1.Samsung Electornics RFS v1.2 Programming Guide

2.Samsung Electronics RFS v1.2 Porting Guide