A New Type of NAND Flash-based File System: Design and Implementation

Tianzhou Chen, Xiangsheng Wang, Wei Hu
College of Computer Science, Zhejiang University
Zhejiang University
Hangzhou, Zhejiang, P.R.China
{tzchen, yl, ehu}@zju.edu.cn

Wei Duan
China United Telecommunications Corporation
Beijing, P.R.China
duanw@chinaunicom.com.cn

Abstract—**Flash memory has become the primary external storage for embedded systems especially NAND flash for data update. Currently JFFS2 and YAFFS2 are used as the primary file systems for flash memory. But there are two deficiencies in them. One is that their boot time for a large-sized flash memory is too long. They must scan the whole partition to initialize the file system and it's really time-consuming. The other is their high memory consumption rate. A new type of NAND flash-based file system named SMARTFFS (SMART Flash File System) is proposed in this paper, which can reduce both the initializing time and the SRAM memory consumption rate effectively.**

Keywords-NAND flash; embedded system; file system

I. INTRODUCTION

Flash memory is now more and more used for embedded device for its increasing capacity and lower power consumption compared to the other memory types. Now there are two types of flash memory: NOR flash and NAND flash. There're various differences between them. NOR flash is a kind of EEPROM which can be directly accessed and it operates on word-unit and provides faster read speed than NAND flash. For this reason, it is usually used as code storage [1]. NOR enables linear random access just like RAM and it supports eXecute In Place(XIP). NAND allows sector read/write like a disk and thus can't support XIP. NAND flash is partitioned into blocks, where each block (typically 16KB bytes in size) has a fixed number of pages (typically 512 Bytes in size), and each page is of a fixed size byte array. It supports page read/write but block erase. So erase time for NOR is far longer than for NAND. But NOR has higher fault endurance than NAND.

Thus different methods are required for deploying these two flash memories. Presently the size of NAND Flash memory has reaches 8G. Larger Flash-memory is also under development. Unfortunately, there' s no efficient flash-based file system for managing this kind of large flash storage. Apparently, JFFS2 [2] is not suitable for managing large volume NANDs because of its scan-to-initialize technique. YAFFS [3] is a NAND-oriented .The largest file system size that YAFFS support is 1G.. As far as YAFFS2 [3] concerned, the largest supported is 8G. Another method is to use the Flash-memory translation layer (FTL) [4, 5, 6], which emulate a block device for flash memory so that traditional file systems can be used for flash storage. But it is patented and thus commercial use has to pay for patents [2]. Also the RAM memory consumption rate is considerably high. A new type of NAND flash-based file system is proposed in this paper, which can reduce both the initializing time and the RAM memory consumption effectively. It reduces the initializing time by using a new type of scanning mechanism. Also, the RAM consumption is comparatively low due to its organizing mechanism.

The rest of this paper is organized as follows: in Section 2, we present some flash management techniques that have been developed. In Section 3, we describe the design and implementation of our NAND flash-based file system, the SmartFFS. In Section 4, we present our work and obtain some experimental results and make some comparison with YAFFS2. Finally, our conclusions are drawn in Section 5.

II. RELATED WORK

There are two main flash memory based file system: JFFS2 [2] and YAFFS [3]. JFFS2 which is log-structured [7] is one of the common flash-based file systems. JFFS2 uses nodes to contain data and metadata which are stored on the flash chips sequentially, so the access to the file system must be progressing strictly linearly through the storage space available. At initializing time, it has to scan the whole partition space to find out the journaling nodes and determine the file structures. In JFFS2, The entire medium is scanned at mount time, each node being read and interpreted. When altering an inode, a new node with modified data is attached to that inode. The original "dirty data" node is marked as "obsolete" and will be recycled by garbage collection later. The allocation of free space is sequential. JFFS2 is designed to support NOR flash. In order to support NAND flash, much work remains to do.

YAFFS [3] is the first NAND-specific flash file system, which is log structured and automatically provides wear-leveling and robustness on power failure. Its page mechanism responds to NAND's sector-reading. Pages are allocated sequentially from the currently selected block. When all the pages in the block are filled, another clean block is selected for allocation. At least two or three clean blocks are reserved for garbage collection purposes. If there are insufficient clean blocks available, then a dirty block (i.e. one containing only discarded pages) is erased to free it up as a clean block. If no

This research was supported by National High-Tech 863 Plan of China and Key Project of Zhejiang Province of China.

dirty blocks are available, then the dirtiest block is selected for garbage collection. There're two deficiencies for YAFFS. One is that it need scan the OOB for initializing the file system which consumes much time. The other is that it's RAM consuming is a little high. Besides, the largest file system size it supports is not large enough.

III. Design and implementation of NAND Flash-Based File System

The NAND-based file system, SmartFFS, is based on MTD driver to manipulate the NAND flash memory, which architecture is shown in Fig1. It provides user space access by means of system calls. At the initializing stage, through modifying the system call table a number of unused system calls is employed. We designate a control data structure named File System Control Block (FSCB) to manage this file system. It contains file system usage information of the partition on which it residents, including inodes, directory, file and free data areas as well as dirty areas of garbage collection information. We treat all these file system objects as Objects. We specify methods which are attached to each type of Object, and through this, manage them in an Object-Oriented way. Due to our sophisticated algorithm of our linkage garbage collection and blocks allocation, RAM memory consumption is considerably low.

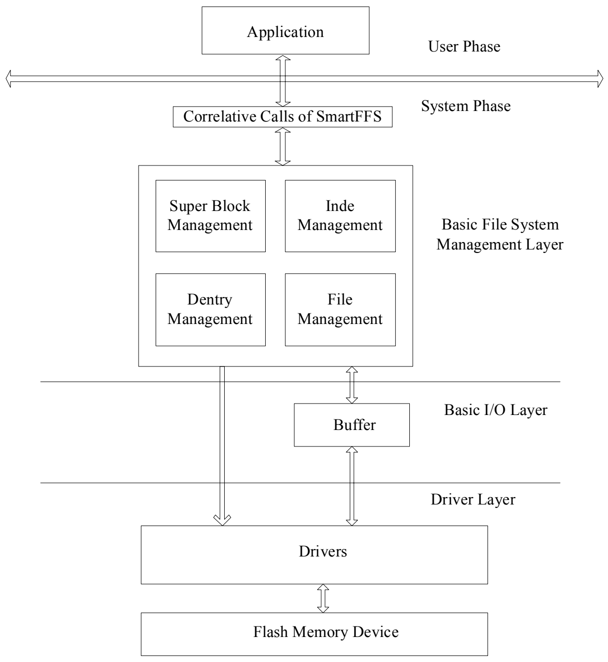

Figure 1. SmartFFS architecture

A. File System Structure Sesign and Data Access

The inode represents a number of data blocks, which could be attached to a file or a directory. Just as in a traditional way, a directory is a container for files. Every directory has its own properties such as owner, time information as well as linkage information of its items. The directory item is either directory or file. Because of the block erase feature of NAND flash memory, the blocks are divided into two types: Data Blocks and Pointer Blocks. Data Blocks are used to store the date of files and Pointer Blocks are used to store the pointers which index to the correlative Data Blocks.

Every inode contains fourteen 1-LEVEL pointers, two 2-LEVEL pointers and one 3-LEVEL pointer. Here 1-Level pointer means that it points directly to an area of data blocks. 2-Level pointer point to an area of blocks containing the pointer to data blocks, and so on. The levels are shown in Fig2.

In sturct smart_inode, i_block[0]~i_block[14] is the direct data block pointers shown in Fig2. These pointers provide the direct access to the Data Block and random read/write operations by access the Data Block. And the new request can response faster by manipulate these pointers. And the pointers, i_point[1]/i_point[2] and i_longpi, are provided to support large size files.

B. Initialization

To increase the start-up speed, SmartFFS define a new

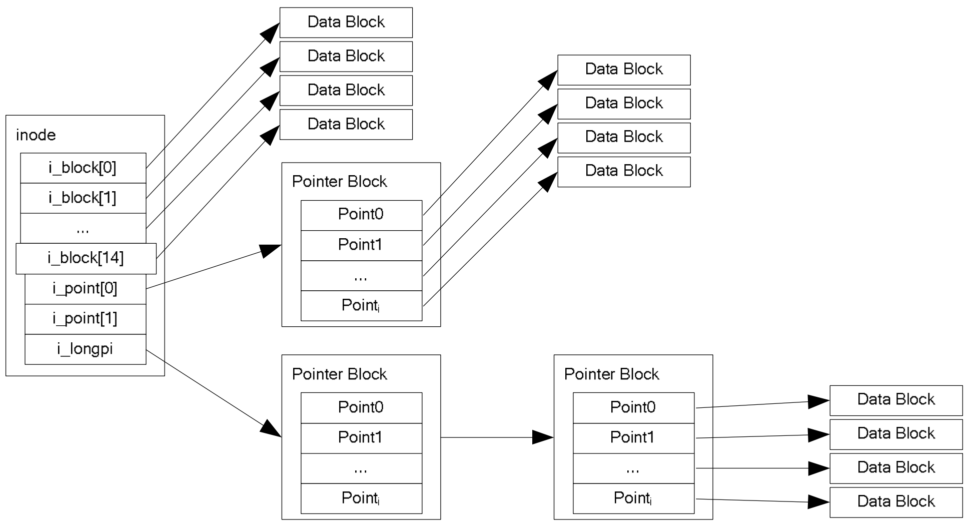

Figure 2. 3 level pointer of SmartFFS

structure FSCB (File System Control Block) which resides in some block of an area and indicates the usage of this area for the file system. It includes index nodes, directory, files, free block allocation and garbage collection information. All of these nodes, directories, files are seen and managed as objects.

At the beginning the partition, a fixed number of blocks is allocated for FSCB index area containing a number of pointers each of which points to a serial of sequential blocks containing FSCB. SmartFFS can determine which one is latest one available by comparing their 'Version' field. Then following the latest available pointer, we will search the desired 'FSCB' area for the newest FSCB. After checking for errors, we analyze it and begin to initialize the file system. In the process, a number of RAM memory data structures such as file table, inode table, free area list and dirty area list are created. Afterwards, The garbage collection kernel thread is started. The process is shown in Fig3.

Not like YAFFS2, SmartFFS will scan the fixed blocks of the memory instead of scanning the whole flash memory space. Because of index scanning to determine the directory structure of file system, SmartFFS can reduce the mount time.

C. Free Block Allocation and Garbage Collection

There's a link list which links all the free areas in the partition together. The global pointer free_area_list and current_list point to the free area list and the current available areas respectively. When a block request is coming, SmartFFS will search the current_list, SmartFFS assigns a new block and modifies the corresponding node in the list, if there are available blocks in current area. Otherwise it will check the free_area_list to find out whether the pointer has arrived at the end of the free_area_list or not. If there are more free areas, the pointer will turn to the next free area and the current_list will point to the next free area too. If there is no free areas in the free_area_list, the garbage collection program will start to collection the dirty blocks.

SmartFFS adopts a space allocation strategy: to assemble correlative data, which are the the contents of the various pieces within a same file, to a separate unit within the same erasure region. SmartFFS will maintain a storage pool in which the will reside; if it is difficult to keep the continuous physics blocks in the pool, it will try to ensure that all the pieces in the pool will be in the same erasure region. If such a situation can not be, system will try to put the pool to a maximum usable space which is an erasure region to block file distribution and achieve even erase.

The garbage collection program runs as a kernel thread which is time-driven. It starts in a periodic fashion to clean the dirty areas and append them to free area list. When collecting the dirty areas, a list of policies must be followed: a block (often contains 4 pages) with all pages being invalid should be collected. If there's only one page is valid, copy it to other place to release the block.

There are two different circumstances when SmartFFS garbage collection thread starts:

1) If the dirty blocks (four pages) have no valid data, these blocks will be erased directly;

2) If the goal blocks contain valid data, the valid data will be copied to other blocks, and the data will be checked, if the results correct, these effective data in the pre-erasure unit will be copied to a new unit named the erasure transfer unit, and then update the mapping-table and erase the old pre-erasure unit.

So that although these data are now stored in flash memory of other space, the original blocks found in the outside space still contains the original data; otherwise, it is seen as error.

Finally all the vacant blocks will be linked into the free block list.

In SmartFFS, garbage collection will be triggered only when the free blocks are not enough and the block allocation algorithm calls it. The block allocation algorithm tries to keep a storage pool which contains continuous free blocks. If the pool becomes too small, The block allocation algorithm will call the garbage collection thread to collect dirty blocks. The following three type unit will be found and reclaimed:

1) With most dirty blocks;

2) With least data in the erase cycle;

3) Most static blocks.

In addition, garbage collection algorithm will randomly select blocks approach to essure the collection can uniformly cover the entire memory space and not be influenced by the way which applications use data.

IV. Experimental results

We used the Linux Trace Toolkit (LTT) and Kernel Function Trace (KFT) to obtain the experimental results. LTT is used to examine the flow of execution (between processes, kernel threads, and interrupts) in a Linux system. This is useful for analyzing where delays occur in the system, and to see how processes interact, especially with regard to scheduling, interrupts, synchronization primitives, etc. The KFT system provides for capturing these callouts and generating a trace of events, with timing details. KFT is excellent at providing a good timing overview of kernel procedures, allowing you to see where time is spent in functions and sub-routines in the kernel. We evaluate the SmartFS file system on an Intel-PXA272 platform running the latest patch version of Linux Kernel2.6, with 64M SDRAM and 64M NAND flash .We run a data access program on both SmartFFS and YAFFS2 because they are both file system based on NAND flash. Then we compare our parameters with those of YAFFS2. As a result from Table1 and Table2, SmartFFS is more effective than YAFFS2. The results are as follows in Table1 and Table2:

V. Conclusion and future work

In this paper we present a new NAND flash-based file system, the SmartFFS, in which wearing level meets. It mainly focuses on the large size NAND flash and works based the MTD layer of the flash memory. FSCB (File System Control Block) is presented to manage the files, directories and other components of file system as objects. Various link lists are adopted for the allocation and dirty block collection. Because

TABLE I. ACCESS TIME TABLE (UNITS：MB/S)

Involved sized / Test Items	8K		256K		1M		8M	
	YAFFS2	*SmartFFS*	*YAFFS2*	*SmartFFS*	*YAFFS2*	*SmartFFS*	*YAFFS2*	*SmartFFS*
Writing	1.57	1.52	1.57	1.62	1.62	1.67	1.43	1.72
Reading	7.23	7.17	7.57	7.51	7.53	7.62	7.42	7.61
Deletion	7.92	8.01	7.87	8.01	7.85	7.89	7.63	7.87

TABLE II. SYSTEM RESOURCE USAGE TABLE

File systems / Testing Items	SmartFFS	YAFFS2
Garbage Collection	1.63 Mb/S	1.50 Mb/S
RAM Footprint	87.3%	100%
Initializing Time	1.42S	2.37S

SmartFFS will not scan the entire flash memory to construct the file system ,it can start up more than traditional flash-based file systems and thus it can provide more support for data read and write. So it has make use of the properties of low RAM consumption rate and fast access rate. It can meet the requirement of large volume NAND flash management while providing considerable performance.

REFERENCES

[1] Keun Soo Yim, Jihong Kim, and Kern Koh. A Fast Start-Up Technique for Flash Memory Based Computing Systems.2005 ACM Symposium on Applied Computing.

[2] D. Woodhouse, Red Hat, Inc. “JFFS: The Journaling Flash File System”, Journaling Flash File System (JFFS). http://sources.redhat.com/jffs2/jffs2-html/.

[3] Aleph One Company, “The Yet Another Flash Filing System(YAFFS),” http://www.aleph1.co.uk/yaffs/.

[4] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, “Introduction to Flash Memory,” In Proc. of the IEEE, Vol. 91, No. 4, pp. 489-502, April 2003.

[5] T. R. Bird, “Methods to Improve Bootup Time in Linux,” In Proc. Of the Ottawa Linux Symposium (OLS), Sony Electronics, 2004.

[6] M-Systems, Flash-memory Translation Layer for NAND flash (NFTL).

[7] Mendel Rosenblum and John K. Ousterhout, The Design and Implementation of a Log-Structured File System, ACM Transactions on Computer Systems 10(1) (1992) pp. 26-52