

Fast Mounting Method for NAND Flash Memory File System using Offset Information

Chan-Sul Park*, Tae Hee Han**

*Dept. Of Embedded Software Engineering, Sungkyunkwan University, Gyeonggi, South Korea

** School of Information and Communication Engineering, Sungkyunkwan University, Gyeonggi, South Korea

pcs722@skku.edu, than@skku.edu

Abstract— Replacing hard disks as a storage device, NAND flash memories are now widely used on embedded systems due to a variety of advantages such as fast accessibility, low power consumption, solid durability and small form factor. However, the NAND flash memory cannot use existing file system for hard disks because of the physical difference between the NAND flash memory and hard disk drive. Consequently, the research of YAFFS(Yet Another Flash File System), a special file system for NAND flash memory, is actively going on. YAFFS file system reads a spare area of the entire block in order to find the header information on mounting and uses the header found to create the file system structure in random access memory(RAM). The mounting method like this has a drawback of increasing the mounting time in proportion to a size of flash memories, because it reads the spare area of the entire block. In this paper, to overcome this problem and thus reduce the mounting time, scanning only the spare area of header file that exists one in every files of each blocks is proposed instead of scanning entire page of NAND flash memories. Test result shows that the proposed approach reduces the mounting time by maximum of 34.78% than existing method of YAFFS file system.

Keywords— NAND flash memory, File system, YAFFS, Offset, Mounting time

I. INTRODUCTION

Flash memories are widely used as a storage device for embedded systems like portable multimedia player(PMP)s, mobile phones, digital cameras and MP3 players. Flash memories have favorable properties, such as lower power consumption, smaller size, faster accessibility, and much more robust durability than hard disks.

Flash memories can be classified into a NOR type and a NAND type. NOR flash memories are more expensive and have faster read time than NAND flash memories, while slow in writing and erasing, so it is mostly used for a code storage. NAND flash memories have faster writing and erasing time, and high density, making it mostly used as a mass data storage. These two flash memories have differences shown above, but they also have common characteristics like initializing(erasing) of the memory area before writing on it and having about one hundred thousand limited erase count[1],[2].

In NAND flash based YAFFS file system, there are small blocks made of 32 pages with size of 512byte and large blocks made of 64 pages with size of 2Kbyte respectively. The structure of the large block is shown in figure 1[3].

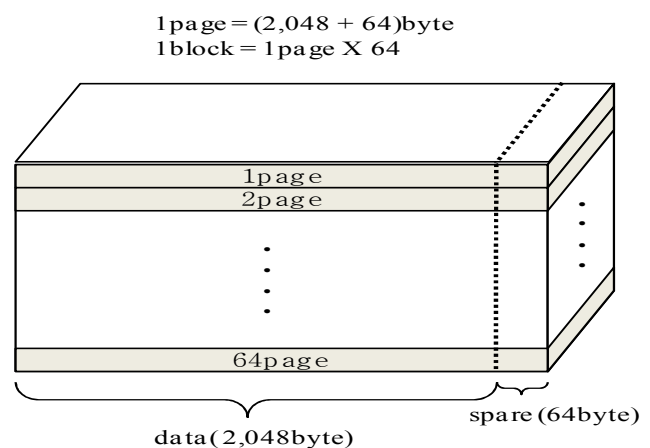


Fig. 1. Structure of large block NAND flash memory

File systems used for hard disks cannot be applied to flash memories, because flash memories have different physical properties from hard disks. For this reason, JFFS(Journaing Flash File System) and YAFFS, which are special file system for flash memories have been developed. JFFS is originally designed for NOR flash memories. And its successor JFFS2 is modified for NAND flash memories. YAFFS file system which is developed for targeting NAND flash memories, is better than JFFS on reading, writing, mounting time and RAM usage. But limited erase count that is a characteristic of NAND flash memories does not allow the metadata same as file's superblock to be saved at the designated location. For this reason, reading entire page area on mounting is required to create file system structure, and this increases the time needed for mounting.

This paper propose the technique to improve the mounting time in NAND flash based YAFFS utilizing offset information.

The Offset information indicates the location of next header file. In other words, when it reads first spare area of the block, it searches the offset bit first and scans a header file on the next header file.

II. FLASH MEMORY FILE SYSTEM

In this section, types and characteristics of file system used for flash memories are introduced.

A. RFFS(Reliable Flash File System)[1]

RFFS is a special file system for NAND flash memory only, and it provides fast mounting, wear-leveling technique to improve lifetime of flash memories, and journaling technique to maintain data reliability. RFFS maintains status information of each blocks and inodes information pointer at the information area, to reduce the time spent for information collecting by the method of scanning entire flash memory of YAFFS. However, not like YAFFS, an erasing operation of RFFS takes extra steps. Therefore, there is a shortcoming that the processing GC(Garbage Collection) in RFFS may take maximum of five times more than YAFFS.

B. RCFFS(Reliable Compressing Flash File System)[4]

RCFFS increases reliability by utilizing algebraic signatures to validate data and Reed-Solomon codes to correct erroneous or missing data. This file system allows users to adjust the level of reliability they require by specifying the number of redundancy pages for each erase block, allowing them to dynamically trade off reliability and storage overhead. By integrating error mitigation with advanced features such as fast mounting and compression.

C. JFFS2(Journaling Flash File System 2)[5],[7]

JFFS is designed for NOR flash memories by using LFS(Log-Structured File System)[6] structure. In this method, JFFS shows high performance on NOR flash memories, but there are several limitation to use it for NAND flash memories. So JFFS2 that can be used for NAND flash memories is developed.

JFFS2 also uses the LFS structure. LFS constructs the file system by making a log of entire file system data(super block, inode, data). The log is recorded by each segment unit in order, and starts from the last recording location of disk. This means that LFS dose not rewrite file system contents, which are already read, on the same location, so information before the change remains in the file system. According to the characteristics of flash memories, overwriting is impossible unless the data is erased, but this problem can be resolved by using LFS.

Comparing to JFFS, JFFS2 is better to have data compression function and overcome garbage collection mechanism to improve the disadvantage of JFFS, which takes long time to mount on booting because it needs to decide the file structure by searching the flash memory. However, it also has drawbacks like a drop of writing speed, increased time taken for mounting and using too much of RAM capacity, when usage ratio of the flash memory is increased. The YAFFS flash memory file system is developed to complement disadvantages of JFFS described above.

D. YAFFS(Yet Another Flash File System)[7]

YAFFS is another flash file system developed for NAND flash, and has great performances on reading, writing, mounting time, and memory usage. YAFFS1 is designed to have a page with 512byte, and YAFFS2 is developed to have a 2Kbyte page.

YAFFS separately manages each page of the NAND flash memory into a Chunk and a Spare. It saves data and headers like directories and files to the Chunk, and block status, page status, ECC area and tags to the Spare. In the tags, metadata needed to manage each Chuck is saved.

YAFFS creates the file system structure by reading all Headers from the flash memory to the RAM when mounting. Created major file system data structures are yaffs_Object, yaffs_Tnode, yaffs_ObjectHeader. Figure 2 shows structure of yaffs_Object. One yaffs_Object exists in every object created on the file system. In the end, yaffs_Object based on the data of flash memory is constructed on the RAM, and directory structure is also constructed inside of yaffs_Object as pointers named parent and siblings.

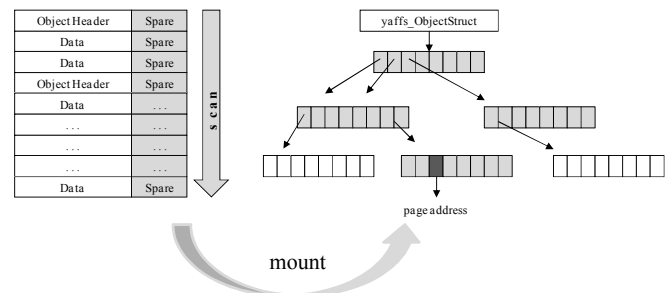


Fig. 2. Structure of yaffs_Object[8]

YAFFS shows faster mounting time than JFFS2, because it only reads a spare area when it scans in order to initialize on mounting, but it also has a problem of increasing mounting time in proportion to the size of the flash memory[9]. In addition, it does not consider wear-leveling technique, because it uses the greedy algorithm for erasing on flash memories, which uses the method of choosing the block with most empty pages to erase.

III. PROPOSED METHOD FOR REDUCING MOUNTING TIME

YAFFS has a shorter mounting time than JFFS2, but there is a problem because the entire spare area is needed to be scanned on mounting process. In this paper, the method of reducing mounting time is devised to improve mounting time of YAFFS by using an offset.

A. Structure of YAFFS file system

In a page area of large block NAND flash memories, headers and data files are saved. In a spare area, ECC (Error Correction Code), a tag that contains metadata of file, and information of blocks and pages status are saved as shown in figure 3.

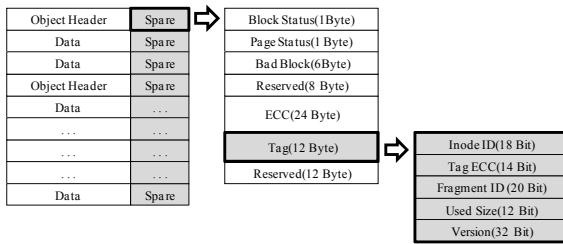


Fig. 3. Structure of YAFFS Spare area

B. Offset applied YAFFS file system structure

YAFFS saves metadata of file's property in a header file of the block. For each block, one file has one header file, and at this point, ChunkID of spare area is displayed as 0. It means adding an area that sets up an offset to reserved part of the spare area in order to scan spare area only if that ChunkID is 0. The figure 4 shows the spare area with added offset area.

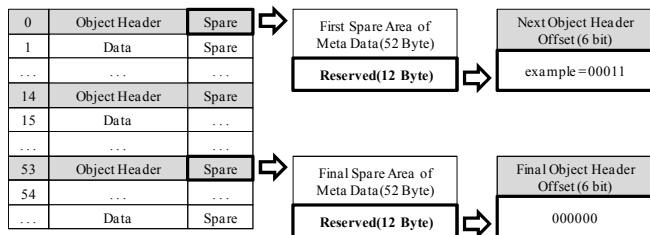


Fig. 4. Structure of spare area with added offset area

C. Offset algorithm

On mounting, the existing YAFFS file system scans tags of spare area to find the header with ChuckID value of 0 to compose yaffs_Object, and compares ObjectID and ChunkID of the tag to register data in that file object on the Tnode List in order of ChunkID, and composes the Tnode Tree by size of file and registers to the header. The reason why YAFFS goes through the mounting process like this is metadata is dispersed all over the NAND flash memory area.

In case of the offset applied YAFFS file system, the location of spare area of next header file is marked at the spare area of first header file of the block as an offset.

For example, if the file 'A' has a size of 3Kbyte and file 'B' has a size of 120Kbyte like shown in the figure 5, an offset of the file 'A' will be set to '000011' because it uses two pages, and an offset of the file 'B' will be set to '000000' because it uses 60 pages. The offset of 'B' is set for '000000', since there is no other files in the block, so moving to the next block. In this case, the existing YAFFS needs to read all 64 spare areas to find the header, but the offset applied YAFFS needs to read only two spare areas with the header. So the mounting time is reduced as much as 64 spare areas and the number of files in the block.

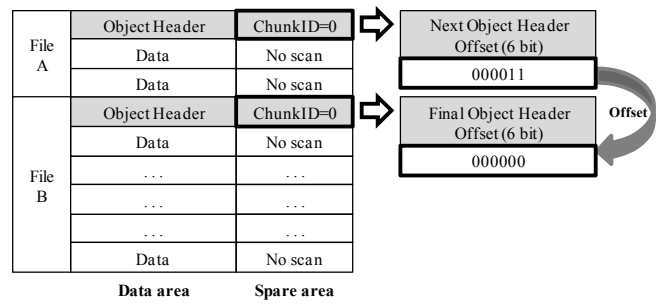


Fig. 5. Offset applied YAFFS file system

D. Performance comparison of initialization modeling of YAFFS and the offset applied YAFFS

To compare the performances of YAFFS and the offset applied YAFFS, the process is modeled into equations and its results are analyzed.

If the mounting time is M_T ,

$$M_T = B_N S_T + \sum_{n=1}^{F_N} [O_{N_n} (D_{T_n} + S_{T_n}) + P_{N_n} S_{T_n}] \quad (1)$$

$$M_T = B_N S_T + \sum_{n=1}^{F_N} [O_{N_n} (D_{T_n} + S_{T_n})] \quad (2)$$

The existing YAFFS can be modeled as (1), and the offset applied YAFFS is as (2), which shows the reduction of mounting time as much as $P_N S_T$.

Table 1. Description of symbols used in the formula

Symbol	Description
M_T	Mounting time
D_T	Time taken to read data area in one page
S_T	Time taken to read spare area in one page
B_N	Number of unused blocks
P_N	Number of pages in one file
O_N	Number of Object_header
F_N	Number of files

IV. EXPERIMENTAL RESULTS

Tests are performed using a Core2 Quad 2.83GHz processor, 4GB of RAM, and Linux2.6.24 as the OS to run YAFFS.

Mounting time is calculated by adding time spent on reading spare areas and data areas. Reading time used for the test is calculated as 14.2us for spare areas and 214.8us for page areas.

The mounting times are different due to the size of files, the number of files, and the size of flash memories. So, mounting times of each situation, large files, small files and the size of flash memories getting larger, are measured and compared. In order to do this, the test is repeated three times.

The first test measures the mounting time by using small files in a 32Mbyte flash memory. The files used are chosen from YAFFS source files that are smaller than 10Kbyte. 5 files are chosen and 10 copies of each file are used.

The test result shows that an average of 4.95% performance improvement, and the detailed result is shown in table 2.

Table 2. Mounting time comparison, if made up with small files

File Size	YAFFS	OYAFFS	Performance Comparison
13.7 KB	6.021 ms	5.950 ms	1.19 %
32 KB	6.149 ms	5.948 ms	3.37 %
45.5 KB	6.243 ms	5.947 ms	4.99 %
58.4 KB	6.333 ms	5.945 ms	6.53 %
75.4 KB	6.452 ms	5.943 ms	8.54 %

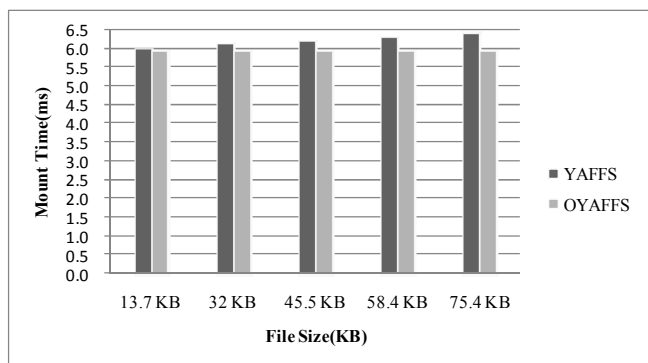


Fig. 6. Mounting time graph of Table 2

The second test measures the mounting time by using large files in a 32Mbyte flash memory. The files used are chosen from YAFFS source files that are larger than 10Kbyte. 5 files are chosen and 10 copies of each file are used.

The test result shows that an average of 34.78% performance improvement, and the detailed result is shown in table 3.

Table 3. Mounting time comparison, if made up with large files

File Size	YAFFS	OYAFFS	Performance Comparison
54 KB	5.158 ms	4.800 ms	7.45 %
67 KB	5.248 ms	4.802 ms	9.30 %
125.5 KB	5.657 ms	4.903 ms	15.39 %
307 KB	7.384 ms	5.230 ms	41.18 %
905 KB	12.937 ms	6.538 ms	97.88 %

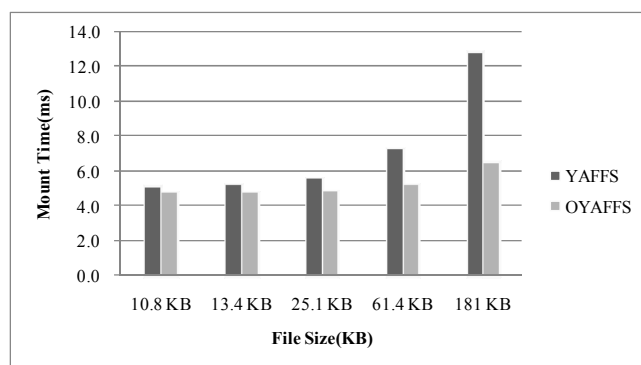


Fig. 7. Mounting time graph of Table 3

Finally, a variety of sizes of flash memories are tested. The files used for the test are 10 copies of yaffs_guts.c file, and the sizes of flash memories are 8Mbyte, 16Mbyte, 32Mbyte, 64Mbyte, and 128Mbyte.

After the performance measurement, there is an average of 28.73% improvement, but the mounting time increases and performance improvement reduces as the size of flash memory gets larger. This result is due to the fact that only the size of flash memories is changed while the number of files is fixed, to observe the differences in the mounting time as the size of memory changes. However, the test confirms that performance improvement is maintained, if the number of files increases as the size of memory increases.

Table 4. Mounting time comparison in different size of flash memories

Memory Size	YAFFS	OYAFFS	Performance Comparison
8 MB	5.390 ms	3.615 ms	49 %
16 MB	6.277 ms	4.509 ms	39 %
32 MB	8.052 ms	6.296 ms	28 %
64 MB	11.602 ms	9.871 ms	18 %
128 MB	18.702 ms	17.021 ms	10 %

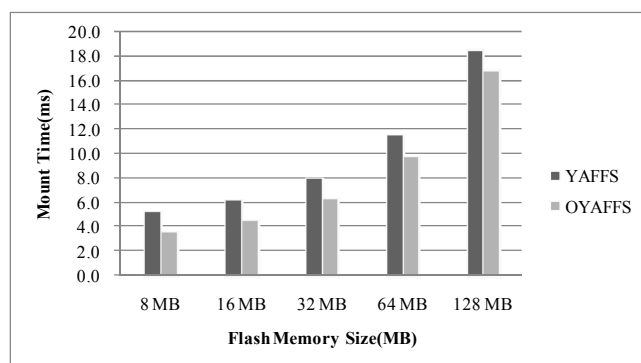


Fig. 8. Mounting time graph of Table 4

V. CONCLUSIONS

In this paper, the mounting time problem, which has been pointed out in the existing YAFFS, is improved. Problems in existing YAFFS are long mounting time, because it reads all spare areas to construct the file system structure in the RAM area. With proposed method of using offset applied YAFFS, it reads only spare areas with the header to create the file system structure in the RAM, and thus reduces the time taken to read unnecessary spare areas.

The test result confirms that the proposed method improves the mounting time than currently used one in YAFFS. The mounting time is reduced by 4.93% in small files, whereas 34.78% reduction in large files. It shows that for large size files gets higher performance. In general, as the size of flash memories becomes larger, B_{NST} increases as well. And thus For that reason, the mounting time increases.

REFERENCES

- [1] Song-Hwa Park, Tae-Hoon Kim, Ki-Dong Chung, "Fast Mounting and Recovery for NAND Flash Memory Based Embedded Systems," EUC Workshops 2006, LNCS 4097, pp.710-720, 2006.
- [2] ASSOCIATION COMPACTFLASH. 1998. compact flashTM 1.4 specification. [www. compactflash.org/](http://www.compactflash.org/)
- [3] Samsung Electronics, "NAND Flash Memory & SmartMedia Data Book," 2003.
- [4] Yangwook Kang, Ethan L. Miller, "Adding Aggressive Error Correction to a High-Performance Compressing Flash File System", EMSOFT'09, October 12-16,2009,Grenoble,France.
- [5] JFFS2, <http://sources.redhat.com/jffs2/>
- [6] Mendel Rosenblum, John K. Ousterhout, "The Design and Implementation of a Log-Structured File System", 13th ACM Symposium on Operating Systems Principles and the February 1992.
- [7] YAFFS Spec, <http://www.aleph1.co.uk/yaffs/yaffs-.html>.
- [8] Kyung Ho Chung, Myung Sil Choi, Kwang Seon Ahn, "A Study on the Packaging for Fast Boot-up Time in the Embedded Linux" , 13th IEEE International Conference on Embedded and RTCSA 2007.
- [9] Eun-ki Kim, hyungjong Shin, Byung-gil Jeon, Seokhee Han, Jaemin Jung, and Youjip Won, "FRASH: Hierarchical File System for FRAM and Flash", ICCSA 2007,LNCS 4705,Part1,pp.238-251,2007.
- [10] Junkil Ryu, Chanik Park, " Fast Initialization and Memory Management Techniques for Log-Based Flash Memory File Systems", ICESS 2007, LNCS 4523, pp. 219 – 228, 2007.