NAND-플래시 메모리를 이용한 클리닝 알고리즘의 구현 및 설계

The Design and Implementation of a Cleaning Algorithm using NAND-Type Flash Memory

구 용 완[*] 한 대 만[**]
Yongwan Koo Daeman Han

요 약

본 논문에서는 NAND 형태의 플래시 메모리를 이용하여 시스템의 성능을 저하시키는 삭제 연산을 감소시켜 수행시간을 보장할 수 있는 플래시 메모리 전용 파일시스템을 설계한다. 파일 시스템 측면에서 플래시 메모리의 쓰기 연산 횟수를 감소시키면 파일시스템의 성능을 향상 시킬 수 있으므로, 쓰기 횟수를 감소시킬 수 있도록 새로운 i_node 구조를 구성하여 파일 시스템을 구성한다.

새롭게 구성된 i_node 구조를 통하여 삭제 연산을 위한 Cleaning 알고리즘을 본 문에서 제시한다. 또한, Cleaning 될 데이터는 응용 프로그램 실행 시 자연적으로 발생하는 지역공간성과 시간공간성의 개념에 의해 최근에 사용된 응용 프로그램과 데이터가 또다시 실행될 가능성이 높은 실험결과에 따라서 최근의 데이터를 가장 오래유지하고 가장 오래된 데이터가 Cleaning 되도록 설계 하였다. 실험 과 플래시 파일 시스템 구현을 통하여 임베디드 시스템에서 요구하는 NAND 형 플래시 파일 시스템의 효율성을 증명한다.

Abstract

This paper be composed to file system by making a new i_node structure which can decrease Write frequency because this's can improved the file system efficiency if reduced Write operation frequency of flash memory in respect of file system. i_node is designed to realize Cleaning policy of data in order to perform Write operation.

This paper suggests Cleaning Algorithm for Write operation through a new i_node structure. In addition, this paper have made the oldest data cleaned and the most recent data maintained longest as a result of experiment that the recent applied program and data tend to be implemented again through the concept of regional and time space which appears automatically when applied program is implemented. Through experiment and realization of the Flash file system, this paper proved the efficiency of NAND-type flash file system which is required in an Embedded system

☞ Keyword : Flash, File System , Embedded

1. Introduction

Recently more efficient embedded systems are at higher demand as mobile services are expanding. PDA and wireless internet devices as examples of embedded systems use flash memories as nonvolatile data managements, for they consume reasonably little power.

Besides, flash memories have greater density rate than non volatile RAM, supports system

* 종신회원: 수원대학교 IT대학 학장, 컴퓨터학과 교수
ywkoo@suwon.ac.kr
** 정 회 원: 국제대학 교양학과 외래교수
han38@hanmail.net
[2006/07/25 투고 - 2006/09/18 심사 - 2006/11/20 심사완료]

level R/W with low power consumption and fast speed, and moreover show high durability against temperature and crashes. Different manufacturers used different technologies to produce flash memories and consequently the performances vary in the R/W speed and the sizes of pages and blocks.

Adoption of flash memories as a storage device in a system requires several considerations to take into the system design. Writing data into flash memories would be preceded by the deletion operations that take longer time and take place in bulks.

The methods to overcome the problems in designing flash memories are divided into two categories.

Flash memories are a kind of EEPROM (Electrically Erasable Programmable ROM). One of the categories neither isNOR type that supports byte I/O and NAND type that supports page I/O only. For NOR type is fast in reading but slow in writing, they are used for storing codes. NAND type is used to store large data, for they are fast in writing and cheaper per unit data [3]. Writing onto flash memories, however, should be preceded by deletion operations the unit of which is greater than that of writing operations.

Such behaviors of flash memories would make it less unlikely to use them as main memories and make it hard to import the file systems designed for hard disks even when they are to be used as secondary memories.

To hide the deletion operations a middleware (flash translation layer; FTL) between flash memories and file systems is often used [4, 5]. Upon the execution of writing operations, the FTL plays a role of translating the logical addresses generated by the file systems into the physical addresses in which deletion operations already took place in the flash memories. For a fast address translation, the translation table uses high performance cheap, SRAM.

Network file systems are designed so that hosts can use different file systems such as FAT by using FTL, then flash memory file systems can be made to work on real time databases. Except for being used to work on real time databases, flash memories are used mostly as secondary memories. Without using FTL, the performances of flash memories can be improved by using a file system such as Log-structure File System (LFS) [8, 9]. As a way to design a file system for flash memories, an exclusive flash file system may be deployed. An exclusive file system can make it easier to realize an efficient system without extra hardware.

2. Related Works

2.1. Yaffs (Yet another Flash File System)

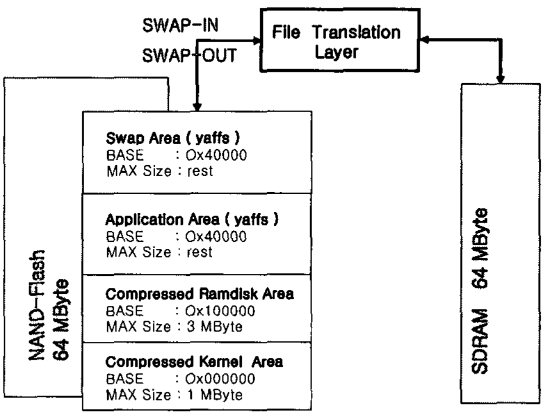
One of the reasons that embedded systems are often based on LINUX is that the developers can choose the files systems of their own interest. Most NOR type file systems use MTD and JFFS systems, but some report JFFS file systems are unreliable at times. In particular large size operations cause severe writing speed deterioration as memory occupation dramatically increases. Consequently application programmers are required of using limited resources.

Yaffs file systems overcomes the problems of JFFS that the memory utilization rate is lower than that of JFFS2, the improved system mounting speed is proved. Working on Yaffs, the goal of this paper is to design a file system that reduces the shortcomings of NAND flash memories instead of increasing the utilization rate.

2.2. Design of Swap-Yaffs File System

As is pointed out earlier, JFFS2 systems are unreliable as the size becomes large. To deal with such problems, Yaffs file systems are appliedon MTD, and the Yaffs file system is designed using the logic based on FTL. One can refer to the manuals provided by the vendors of MTD with regard to the installation of MTD.

This paper suggests a method to minimize the search area of NAND flash by searching Swap areas in advance where conventionally execution files are searched in application areas in FTL by designating Swap area in 64M byte NAND flash memory.



〈Figure 1〉 shows the structure of flash memory with swap partition.

The time to access the application area of NAND flash memories is minimized when the write operation is taking place. Write operations taking much access time would affect the system, so the swap area is allocated just for read operations to minimize the write operations. Such a scheme is based on the spatial locality of codes stored adjacently in the memory, and adjacent NAND memory areas are placed into the swap area that the time for write operations to search application codes sequentially from the beginning to copy to main memory would be minimized.

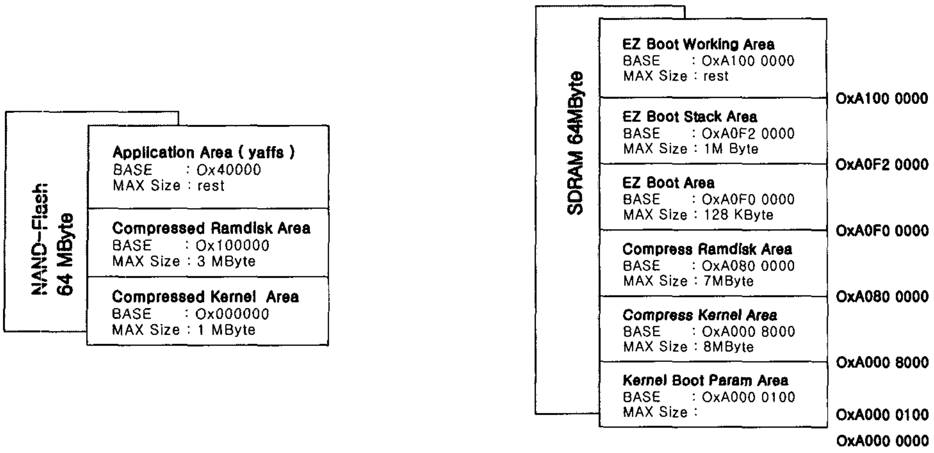
Swap area can be searched only through read operations. The efficiency of read operations in flash memories is comparable to that of main memory operations.

As figure 2 illustrates, the system by this paper uses NAND flash memory of 64Mbytes, and at booting time the kernel block in NAND flash memory is copied to SDRAM to be executed by the NAND flash controller of MCU (memory control unit). Only part of the programs in the application area of NAND flash are copied to SDRAM and executed. JFFS or JFFS2 supports NAND type and NOR type flash memories and show booting performances at 25 sec on average. They also require 4 Mbytes of SDRAM when the file system is to be run on SDRAM.

Yaffs file systems support only NAND type flash memoriesand requires SDRAM of 512 Kbytes. The average booting time is about 3 seconds, thus Yaffs has advantages in space and time efficiencies.

3. Design of Cleaning Algorithm

Flash memory can be divided into two:

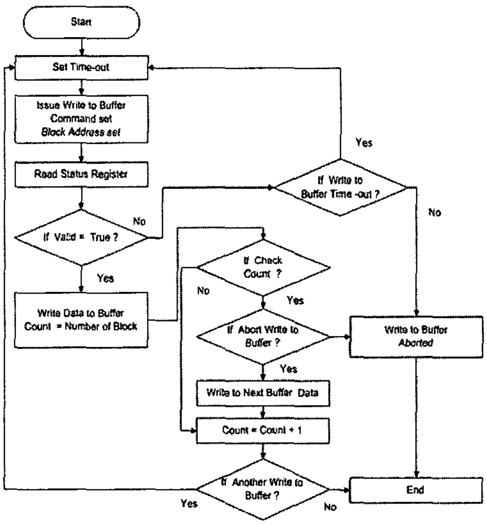


〈Figure 2〉 The memory structure of EZ-x5

NOR-type and NAND-type. The former is developed as code storage such as ROM BIOS because it supports fast reading speed and Byte I/O. The latter is cheaper than the former and is widely used in large capacity processing of the Embedded linux system. However, in order for Flash memory to use data, Erase operation which both needs comparatively long processing time and is performed by block units is an essential prerequisite. Using Flash file system is one of the methods to overcome this obstacle and consist of storage device.

By using NAND-type Flash memory, this paper designed Flash memory file system which can guarantee processing time by decreasing Erase operation that results in deteriorating the system function.

This paper be composed to file system by making a new i_node structure which can decrease Write frequency because this's can improved the file system efficiency if reduced Write operation frequency of flash memory in respect of file system. i_node is designed to realize Cleaning policy of data in order to perform Write operation.This i_node used the current i_node structure to register Yaffs flash file system to VFS. i-node structure of a new file system composes a new structure in order to maintain Flash memory information which performs block size, Write capacity, and Erase operation in a new i_node in order to check when Write operation is performed to relocate data which needs Cleaning.


Start
Set Time-out
Issue Write to Buffer Command set Block Address set
Read Status Register
If Valid = True ?
No
Yes
Write Data to Buffer Count = Number of Block
If Write to Buffer Time-out ?
Yes
No
If Check Count ?
No
Yes
If Abort Write to Buffer ?
Yes
Write to Buffer Aborted
Write to Next Buffer Data
Count = Count + 1
If Another Write to Buffer ?
Yes
No
End

〈Figure 2〉 Cleaning Flow chat

This paper suggests Cleaning algorithm for Write operation through a new i_node structure. In addition, this paper have made the oldest data cleaned and the most recent data maintained longest as a result of experiment that the recent applied program and data tend to be implemented again through the concept of regional and time space which appears automatically when applied program is implemented. Through experiment and realization of the Flash file system,

4. Performance Analysis and Evaluation

In this chapter, considerations needed to design systems using NAND flash memories are examined.

Upon initial booting, the NAND flash memory controller copies the first block of NAND flash memory to the RAM inside and execute the codes.

The Ez-X5 board used in our experiments has the same hardware structure as is shown in figure 2. The program copied into the RAM copies the kernel stored in NAND flash memory to DRAM and lets the kernel take lead. During the execution, the kernel remains in RAM being ready for execution, but the application codes are not all copied into the RAM. The hardware specification in our setting is shown in table 1.

The goal of experiments is to compare the performances between JFFS and JFFS2 and to measure execution time of Yaffs system and improved Yaffs system depending on the varying Swap sizes.

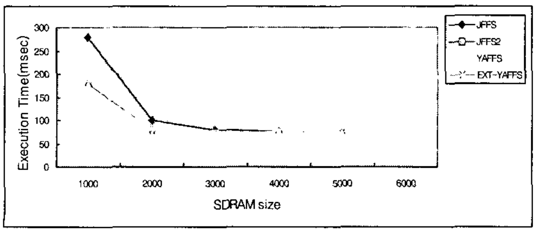
Because the size control of SDRAM is not what we can do at laboratory virtual size is introduced, and Swap algorithm provided by Linux file systems is used since Flash memory systems do not have swap utilities.

〈Table 1〉 Hardware specification

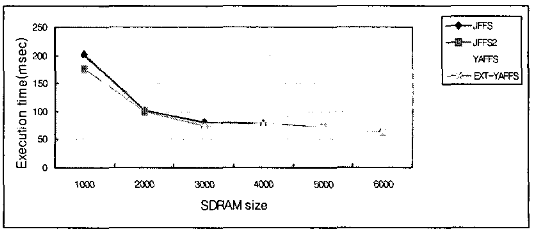
H/W	Units	
MCU	400 MHz PXA255 ARM RISC Chip	ARM10
RAM	64Mbyte SDRAM	
ROM1	512 Kbyte Boot Flash	
ROM2	64Mbyte NAND-Flash	

Figures 3~6 display evaluations among Yaffs file system using Swap partitions, pure Yaffs file system, JFFS, and JFFS2.

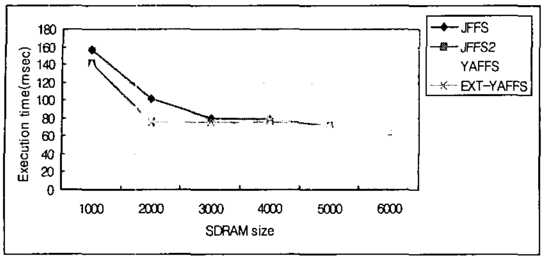
In particular, Swap sizes 128Kbyes and 512Kbyes did not lead to performance improvements as SDRAM size increases. This may be that the swap size required by applications was not big. On the contrary, when the



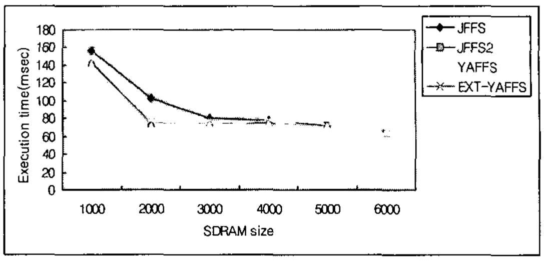
〈Figure 3〉 Swap Size 256KByte



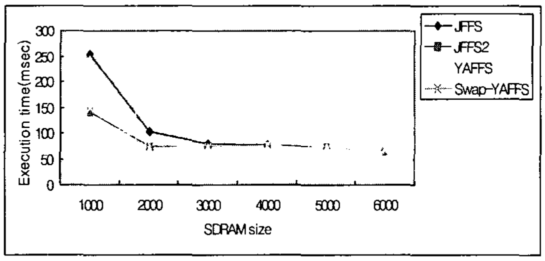
〈Figure 4〉 Swap Size 512KByte



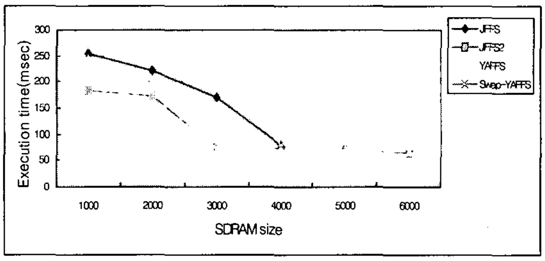
〈Figure 5〉 Swap Size 1MByte



〈Figure 6〉 Swap Size 2MByte



〈Figure 7〉 Application of small memory occupations



〈Figure 8〉 Application of large memory occupations

swap size is set to 1Mbytes and 2Mbytes, the performance gains were evident regardless of the files systems used. As SDRAM size increases, all the file systems showed similar performances.

When the swap partition is set to 256Kbytes and 512Kbytes, only Swap-Yaffs file system can obtain performance upgrade with SDRAM size less than 2Kbytes.

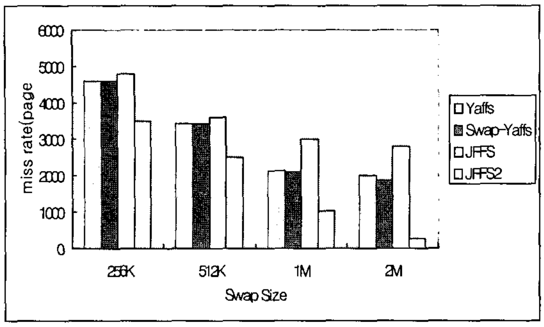
The test results signify that using swap file system can only be beneficial when the memory requirement is sufficiently high. Thus, it is thought that using Swap partition is recommended only when sufficient dynamic memory is to be used.

When multiple application programs are running, using swap partition can be disadvantageous. Figures 7~10 show the test results of applying swap partition to all the file systems. As figure 7 shows, when tested with applications of small memory occupations JFFS2 and Swap-Yaffs file systems demonstrated efficient performances.

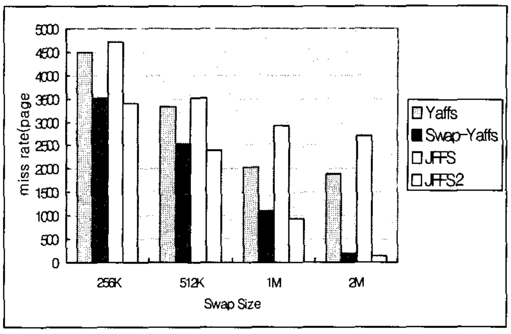
Figure 8 shows the case of running applications with higher memory demands. With high memory requests JFFS2 outperformed with SDRAM of 1K byes and Swap-Yaffs followed.

The goal of JFFS2 file system is mainly to achieve memory efficiency, and thus according to our experiments, the file system turned out to be better than Swap-Yaffs system in terms of memory efficiency. Swap-Yaffs file system, however, claims system reliability factor.

For the next experiment, Swap partition is set in flash memory for dynamic memory to be allocated in the swap area. The system overhead is caused when the requested data is not resident in SDRAM and Swap partition. The overhead can be measured by observing the page faults with varying swap sizes. Once



〈Figure 9〉 Swap off



〈Figure 10〉 Swap on

after swap partition is set up, the page faults are observed when the swap algorithm is used or not used with varying swap sizes. Swap partition sizes and SDRAM sizes are maintained as before.

Figure 9 shows the case without using swap algorithm, and figure 10 shows another case of using swap algorithm. As shown in figure 10, Swap-Yaffs system led to less page faults than JFFS file system but fails to improve over Yaffs system. JFFS2 system demonstrates clear improvements. Figure 10 is a result of applying Swap-algorithm to all the systems, and Swap-Yaffssystem succeeds to reduce page faults at the Swap size of 256Kbytes and shows similar outcomes to those by JFFS2. Consequently Swap-Yaffs system maintains the merit of reliability inherent of Yaffs system, allows fast initial booting, and achieves efficient memory management comparable to that of JFFS2.

5. Conclusion and Future Study

The goal of this paper is to design a flash file system that can compromise merits and demerits of previous systems. By adding Swap functions to Yaffs file system, we could witness speed gains for the applications that use flash memories where memory management might not be better than JFFS2.

We made use of the fact that sequential search in NAND type flash memories is faster than other memories. More tests can be made upon the factors including the sizes of applications and extra memory requirements not used for program executions.

Reference

[1] AMD, Flash Memory Technical Documentation. http://www.amd.com/us-en/ Flash Memory/Tech.nicalResources, August 2001

[2] D. Woodhouse, *"JFFS: The Journaling Flash File System,"* Ottawa Linux Symposium(http://sources.redhat.com/jffs2/), 2001.

[3] Intel Corporation, Understanding the Flash Translation Layer(FTL) Specification, http:// developer.intel.com/design/flash, December 1998.

[4] J. Kim, J.M.Kim, S.H.Noh, *"A Space-Efficient Flash Translation Layer for Compact Flash Systems."* IEEE Transactions on

Consumer Electronics, Vol.48, No.2, pp. 366-375, 2002.

[5] J. L. Hennessy, D. A. Patterson, David Goldberg, Computer Architecture: A Quantitative Approch, 3rd Edition, Morgan Kaufmann Publishers, 2002.

[6] JFFS2, http://sources.redhat.com/jffs2/

[7] K.S.Yim, H. Bahn, K. Koh, *"A Compressed Page Management System NAND-type Flash Memory,"* In Proceedings of the International Conference on VLSI, pp. 266-271, 2003.

[8] MTD, *"Memory Technology Device (MTD) sub-system for Linux,"* http://www.linux-mtd .infradead.org/.

[9] N. Webber, *Operating System Support for Portable File system Extensions*, Proceedings of the Winter USENIX 1993 Technical Conference, 219-228, January 1993.

[10] Samsung Electronics, SAMSUNG NAND Flash Memory, Memory Product & Technology Divi sion, 1999.

[11] Samsung Electronics, "128M x 8 bit / 64M x 16 bit NAND Flash Memory," http://www.san sungelectronics.com/

◐ 저 자 소 개 ◑

구 용 완 (Koo Yong Wan)
1976년 중앙대학교 전자계산과 졸업(학사)
1980년 중앙대학교 대학원 전자계산학과 졸업(석사)
1988년 중앙대학교 대학원 전자계산학과 졸업(박사)
1983~현재 수원대학교 IT대학 학장, 컴퓨터학과 교수
관심분야 : 분산 및 운영체제, 임베디드 시스템, 실시간 리눅스 시스템,
시스템 네트워크 관리, 인터넷 응용 등
E-mail : ywkoo@suwon.ac.kr

한 대 만 (Han Dae Man)
1998년 독학사대학교 컴퓨터학과 졸업(학사)
2000년 수원대학교 대학원 전자계산학과 졸업(석사)
2006년 수원대학교 대학원 컴퓨터학과 졸업(박사)
2000~현재 국제대학 교양학과 외래교수
관심분야 : 분산 및 운영체제, 임베디드 시스템, 실시간 리눅스 시스템. 센서네트워크, USN
E-mail : han38@hanmail.net