실시간 운영체제를 위한 플래시 메모리 파일시스템의 설계와 구현

김정기*, 박승민*, 박상호**, 안우현**, 박대연**
*한국전자통신연구원 인터넷정보가전연구부
**한국과학기술원 전자전산학과
e-mail:jkk@etri.re.kr

Design and Implementation of Flash Memory File System for Real-time Operating Systems

Jeong-Ki Kim*, Sung-Min Park*,
Sang-Ho Park**, Woo-Hyun Ahn**, Dae-Yeon Park**
*Dept of Internet Appliance, ETRI
**Dept of Electronic and Computer Engineering, KAIST

요 약

최근들어 정보가전, 휴대 통신 기기, 셋탑박스 등의 Embedded 시스템이 개발됨에 따라 이를 운영할 실시간 운영체제의 필요성이 절실히 요구되고 있으며, 여기에 사용될 파일 시스템이 필요하게 되었다. 그러나, 이런 Embedded 시스템의 특성상 전원이 꺼진 상태에서도 데이터를 보관하기 위하여 플래시 메모리를 이용한 파일 시스템이 필요하며, 본 논문에서는 이런 실시간 운영체제를 위한 플래시 파일 시스템을 설계하고 구현한다. 또한 효율적인 플래시 메모리 접근을 위해 플래시 메모리 관리자를 구현하고, 오류 복구를 위한 효율적인 복구 알고리즘을 제안한다. 마지막으로 순차적 파일 쓰기와 무작위 파일 쓰기 실험을 통해 본 논문의 플래시 파일 시스템의 성능을 평가한다.

1. 서론

정보 산업이 발전함에 따라, 최근에 정보가전, 통신 기기, 휴대 기기, 셋탑박스(set-top box), 인터넷 폰 등은 기존의 단순한 전자제품을 넘어 프로세싱이 필요한 Embedded 시스템으로 발전하고 있다. 이러한 시스템은 점진적으로 좀 더 복잡한 기능을 요구하고 있으며, 이에 따라 실시간 운영 체제가 필요하게 되었다. 여기에 필요한 파일시스템 역시 필수적으로 필요하게 되었으며, 이러한 Embedded 시스템은 전원이 꺼진 뒤에도 데이터를 저장하고 있어야 함으로 저장 매체로 플래시 메모리를 주로 이용하게 되었다[1,2].

플래시 메모리는 다른 저장 매체에 비해 견고하며, 비휘발성의 특징을 갖고 있으며, 접근 시간이 빠르다. 뿐만 아니라, 저 전력으로 동작하며, 크기 작아 휴대 기기에 적합하다[3,4,5]. 그러나, 플래시 메모리를 사용할 때 두 가지 문제점을 고려해야 한다. 첫 번째로 데이터를 한 번 쓴 영역에는 지움(cleaning) 과정을 수행하기 전에는 다시 쓸 수 없으며, 지우는 시간이 0.5에서 1초로 매우 길다는 점이다. 둘째로 플래시 메모리를 지울 수 있는 회수가 정해져 있기 때문에 데이터를 플래시에 저장하는 시기를 최대한 늦추어야 한다는 것이다. 이러한 문제점을 극복하기 위해 여러 가지 방법이 제시되고 있다[1]. 플래시 메모리의 일반적인 특징은 <표 1>과 같다.

<표 1> 플래시 메모리의 일반적인 특징

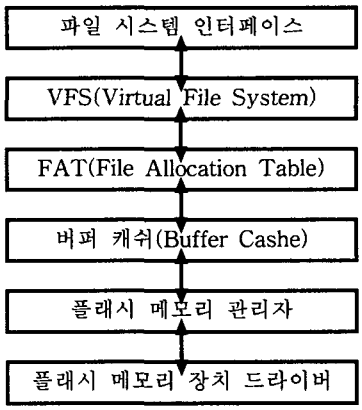
읽기 속도	80~150 nsec/byte
쓰기 속도	10 μsec/byte
지움(Erase) 속도	0.5~1 sec/block
지우기 회수 제한	100,000 번
지움 단위 크기	64Kbyte~256Kbyte
전력 소모량	동작상태 - 30~50mA 대기상태 - 20~100 μA

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 설계하고 구현한 플래시 파일 시스템에 대해 설명하고, 3장에서는 파일 시스템의 성능을 평가한

다. 그리고 4장에서 결론은 제시한다.

2. 플래시 파일 시스템

본 논문에서 설계하고 구현한 플래시 파일 시스템의 구조는 (그림 1)과 같다.



(그림 1) 플래시 파일 시스템의 구조

2.1 파일 시스템 인터페이스와 VFS

본 논문에서 구현한 파일시스템 인터페이스는 UNIX[6]와 유사한 인터페이스를 제공한다. 따라서, 기존의 응용 프로그램(Application Program)을 거의 그대로 구동시킬 수 있는 확장성을 가지고 있으며, UNIX에 익숙한 사용자에게 편의성과 일관성을 제공한다. 또한 선 마이크로시스템(Sun Microsystem)에서 개발된 VFS(Virtual File System)[7]을 사용하여 복수의 파일 시스템 유형을 지원하도록 하였다.

2.2 FAT(File Allocation Table)

실제 플래시 메모리에 파일을 저장하는 방식은 FAT 파일 시스템을 이용하였다[8]. FAT은 DOS에서 사용되는 파일 시스템으로 inode를 사용하는 기존의 UNIX 파일 시스템보다 소용량 저장 매체에서는 더 적합한 것으로 알려져 있다. 일반적인 플래시 메모리 크기가 16Mbytes 정도이므로 FAT 방식을 이용하는 것이 타당하다. 또한 FAT의 크기가 작기 때문에 테이블 전체를 메모리에 저장하면, 빠르게 파일을 접근할 수 있다.

그러나, 디스크에 비해 읽기 속도가 매우 빠른 플래시 메모리를 사용할 경우 FAT를 검색하는데, 소요되는 시간이 문제될 수 있다. 이것을 해결하기 위해 본 논문에서는 VFS의 vnode에 한 필드를 할당하여 inode 캐쉬를 저장함으로써 파일 접근의 회수를 최소로 줄였다.

2.3 버퍼 캐쉬(Buffer Cache)

버퍼 캐쉬는 디스크의 느린 속도를 감추어 시스템의 응답 시간과 처리 능력을 향상시키기 위해서 사용된다. 그러나 플래시 메모리의 빠른 속도 때문에 버퍼 캐쉬의 본래 효과는 감소되지만, 플래시 메모리의 수명을 연장시킬 수 있다는 다른 장점이 있다. 버퍼 캐쉬를 이용하면, 플래시 메모리에 쓰는 것을 최소로 억제할 수 있고, 지움 회수가 줄어들어 접근 속도가 빨라지며 플래시 메모리의 수명을 연장시킬 수 있다.

2.4 플래시 메모리 관리자

플래시 메모리 관리자는 파일의 논리적인 주소를 플래시의 물리적인 주소로 매핑(mapping)하는 열할을 한다. 본 관리자는 기본적으로 LFS(Log-structured File System)[9]을 이용하여 구현되었으며, 지움 정책(cleaning policy)에 따라 플래시 메모리의 invalid 블록을 지우는 기능과 예상치 못하게 전원이 꺼진 경우 복구하는 기능을 수행한다.

원본 EU	백업 EU	
Valid	Free	(1)
Valid	Allocated	(2)
Valid	Prevalid	(3)
Deleted	Prevalid	(4)
Deleted	valid	(5)
Erasing	valid	(6)
Free	valid	

(그림 2) 지움 과정

원본블럭	백업블럭	
Valid	Free	(1)
Valid	Allocated	(2)
Valid	Prevalid	(3)
Deleted	Prevalid	(4)
Deleted	valid	

(그림 3) 데이터 쓰기 과정

본 논문에서는 (그림 2)와 (그림 3)과 같이 EU과 블록이 여러 단계의 상태를 거치게 하여 오류 복구를 수행하는 방법을 제안한다. 전원이 꺼진 뒤 복구 상태에서 (그림 2)의 (1)이나 (2)상태인 경우 백업 EU를 invalid 상태로 바꾸고 지운다. (3)인 경우 데이터 복사가 완료된 상태이므로 원본 EU를 invalid 상태로 바꾸고 지우며, 백업 EU은 valid 상태로 바꾸고 EU 번호가 같도록 바꾼다. (4), (5), (6)인 경우 원본 EU를 지우고 백업 EU은 valid 상태로 둔다.

(그림 3)의 데이터 쓰기 과정의 오류 복구는 더 간단하다. (1), (2)인 경우 백업블록만 invalid 상태로 바꾼다. (3), (4)인 경우 원본블록은 invalid 상태로 바꾸고, 백업블록은 valid 상태로 바꾼다. 이렇게 함

으로써 오류를 복구하여 롤백(roll-back)할 수 있다.

3. 성능평가

본 논문에서 개발한 플래시 파일시스템의 성능을 평가하기 위해 <표 2>와 같은 환경에서 실험을 실시했다. 실험에 사용된 시스템은 한국전자통신연구원(ETRI)에서 실험용으로 제작된 셋탑박스(set-top box)이다.

<표 2> 성능평가 실험 환경

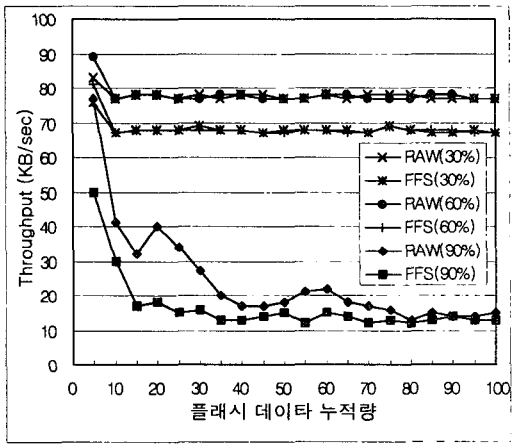
플래시 메모리	Intel 28F320S(32Mbit/chip)[10]
EU 개수	32개
EU 크기	128 Kbytes
메타(meta) 블록	96개
데이터 블록	8004개
지움(Erase) 속도	0.5 sec
쓰기 성능	5.76 sec
CPU	StrongARM-110S 233M RISC
메모리	32 Mbytes
버스	PCI (Intel 21285)
버퍼캐쉬크기	512 Kbytes

초기의 파일 크기가 전체 플래시 메모리 크기의 30%, 60%, 90%일 때 성능을 측정하였다. 먼저 정해진 크기의 파일을 순차적으로 저장하고, 그 다음에는 같은 파일을 저장된 데이터 위에 덧쓰는 작업을 반복함으로써 성능을 평가한다. 또한 본 논문에서 개발한 VFS, FAT, 버퍼캐쉬를 갖는 플래시 파일시스템(FFS)의 성능을 평가하기 위해 이것을 뺀 플래시 메모리 관리자(RAW)와 성능을 비교한다.

5.1 순차적 쓰기에 대한 성능평가

순차적 쓰기(sequential write)는 파일시스템의 최대 성능을 알아보기 위한 실험이다. 플래시 메모리에 valid 블록이 순서대로 위치하고, invalid 블록도 순서대로 발생하기 때문에 cleaning이 발생할 때, EU내의 모든 블록이 invalid 상태에 있게 된다. 따라서, cleaning 후에 얻을 수 있는 free 블록의 수도 많고, cleaning의 효율성이 최대가 된다. 이 실험은 초기 파일 위에 덧쓰기를 할 때, 순차적으로 수행한다.

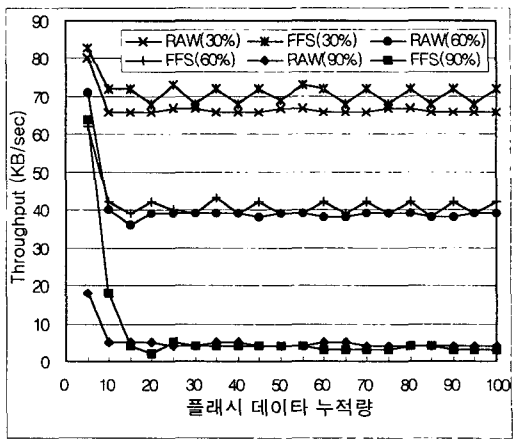
(그림 4)에서 전체적으로 초기 저장된 양이 30%와 60%일 때 좋은 성능을 보이고, 90%일 때 낮은 성능을 보인다. 이것은 플래시 메모리의 90%가 valid 블록이므로 새로운 블록을 만들기 위해 cleaning 과정이 자주 발생하게 되고 복사되는 블록이 많아지기 때문이다. 또한, 전체적으로 FFS보다 RAW가 좋은 성능을 보이는데, 이것은 순차적 쓰기에서는 버퍼 캐쉬의 효과 거의 나타나지 않고 여러 단계를 거치는 과정이 오히려 성능을 저하시키기 때문이다.



(그림 4) 순차적 쓰기 성능평가

5.2 무작위 쓰기에 대한 성능평가

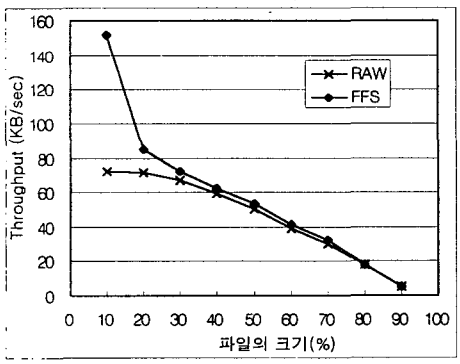
무작위 쓰기(random write)는 파일 시스템의 일반적인 상태의 최저 성능을 알아보기 위한 실험이다. 순차적 쓰기에서는 invalid 블록이 순차적으로 발생하기 때문에 cleaning하는 과정이 용이 하지만, 무작위 쓰기에서는 invalid 블록이 무작위로 발생하고, 각각 EU에 valid 블록과 invalid 블록이 혼합되어 있으므로 cleaning 과정 시 더 많은 시간을 요구하게 된다.



(그림 5) 무작위 쓰기 성능평가

(그림 5)은 무작위 쓰기의 성능평가를 보이고 있다. 전체적으로 순차적 쓰기에 비해 성능이 저하되는 것을 알 수 있다. 파일 크기가 커질수록 invalid 블록이 더 많은 EU로 분산되기 때문에 FFS와 RAW 모두 성능이 떨어진다. 파일 크기가 30%와 60%일 때 FFS에서 성능이 향상되는데, 이는 버퍼 캐쉬 때문에 실제 플래시 메모리 접근이 줄어들기 때문이다. 버퍼 캐쉬의 크기를 512 KByte로 하였으므로, 평균적으로 1/3 이상의 데이터가 버퍼 캐쉬에 존재하고, 쓰려고 하는 데이터가 버퍼 캐쉬에 존재하면, 상당한 속도 향상을 얻을 수 있다. 파일 크기가 커질수록 쓰려고 하는 데이터가 버퍼 캐쉬에 존재할 확률은 줄어들고, 성능 향상도 줄어들게 된다.

(그림 6)는 무작위 쓰기에서 파일 크기에 따른 성능평가를 보여주고 있다. FFS에서는 버퍼 캐쉬를 사용하기 때문에 파일의 크기가 작을 때 좋은 성능을 보이지만, 파일 크기가 커질수록 EU 내에 invalid 블록이 고르게 분포됨으로써 cleaning 과정 중 복사되는 블록이 많아져 성능이 떨어진다.



(그림 6) 무작위 쓰기에서 파일 크기에 따른 성능

6. 결론

본 논문에서는 플래시 메모리를 이용한 플래시 파일시스템을 설계하고 구현하였으며, 성능을 평가하였다. 본 논문의 플래시 파일 시스템은 VFS와 FAT, 버퍼캐쉬를 이용하여 상위 파일 시스템을 구현하였으며, 효율적인 플래시 메모리 접근을 위해서 플래시 메모리 관리자를 설계하였고, 오류 복구를 위한 효율적인 cleaning 알고리즘을 제안하였다.

또한 순차적 파일 쓰기와 무작위 파일 쓰기 실험을 통하여, 플래시 파일시스템이 있는 경우와 플래시 메모리 관리자만 있는 경우를 성능평가 하였다. 플래시 메모리에 파일 시스템을 구현하는 경우, 일반적인 무작위 쓰기에서는 버퍼 캐쉬를 이용하여 구현하는 것이 좋은 성능을 보임을 알 수 있었다.

최근에 플래시 메모리의 크기가 512MBytes, 1MBytes 등으로 커짐에 따라, 대용량의 플래시 메모리에 효율적인 파일시스템 설계를 추후 과제로 삼고 있다.

참고문헌

[1] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A Flash-Memory Based File System," Proc. of USENIX Techinical Conference, pp. 155-164, 1995.

[2] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," Proc. of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, pp 86-97, 1994.

[3] Chiang, M.-L, Lee, P. C. H and Chang, R.-C, "Managing Flash Memory in Personal Communication Devices," Proc. of the IEEE Int'l Symp. on Consumer Electronics, pp 177-182, 1997.

[4] Kirk Blum, "Software Concerns of Implementing a Resident Flash Disk," Inter corporation, 1995.

[5] "Flash memory," Intel corporation, 1994.

[6] Uresh Vahalia, "UNIX Internals : the new frontiers." pp. 220-288, Prentice-Hall, 1996.

[7] Kleiman, S.R., "Vnodes: An Architecture for Multiple File System Types in Sun UNIX," Proc. of the Summer 1986 USENIX Technical Conference, 1986.oger S. Pressman "Software Engineering A Practlti ners' Approach" 3rd Ed. McGraw Hill

[8] Michael Tischer and Bruno Jennrich, "PC INTERN : the Encyclopedia of System Programming," pp 377-387, Abacus, 1996.

[9] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, Vol. 10, pp 26-52, 1992.

[10] "3 volt FlashFile Memory 28F160S3 and 28F320S3," Intel corporation, 1998.