특 집

임베디드 플래시 파일 시스템

김 정 기*, 박 승 민**, 김 채 규***

● 목 차 ●

1. 서 론

정보통신 산업의 발전에 따라, 기존의 가전제품이나 전자제품 등이 단순한 자체의 기능 만을 수행하는 것을 넘어 정보가전으로 발전되고 있으며, 최근에는 통신 기기, 휴대 기기, 셋탑박스(set-top box), PDA 등이 프로세싱이 필요한 임베디드 시스템(Embedded System)으로 발전하고 있다. 이런 임베디드 시스템은 휴대성, 무선 통신, 소비전력, 사용자 인터페이스, 보안 등 점진적으로 좀 더 복잡한 기능을 요구하고 있으며, 이를 운영할 실시간 운영 체제가 절실히 필요하게 되었다.

휴대가 용이하고 빠른 접근시간을 요구하는 임베디드 시스템의 특성상 데이터를 저장할 공간으로 하드디스크를 사용하는 것은 비효율적이다. 하드디스크는 부피가 클 뿐만 아니라 무겁고 소비전력이 크다. 또한 정보가전 제품은 간편하게 조작하고 쉽게 데이터를 보관할 수 있는 저장 매체를 요구하고 있다. 이를 위해 전원이 켜진 상태에서도 데이터 저장이 가능한 플래시 메모리(Flash Memory)를 주로 이용하게 되었다[6,11,15]. 그러나 플래시 메모리는 하드디스크와 여러가지 다른 특성을 가지고 있기 때문에 이를 효율적으로 관리하여 데이터를 저장하는 플래시 파일시스템은 필수적이다.

본 고에서는 임베디드 시스템의 플래시 파일시스템에 대한 요구사항과 개발할 때 필요한 쟁점, 그리고 개발된 플래시 파일시스템에 대해 기술한다. 2장에서는 플래시 파일시스템 개발을 결정하는 플래시 메모리에 대한 특성을 기술하고, 3장에서는 플래시 파일시스템에서 연구되는 동향들에 대해 언급하고, 4장에서 플래시 파일시스템의 구현 예를 기술하고, 마지막으로 5장에서 결론을 제시하다.

2. 플래시 메모리의 특징

플래시 메모리는 하드디스크 등의 다른 저장 매체에 비해 견고하고, 비휘발성의 특징을 갖고 있으며, DRAM 만큼 접근 시간이 빠르다. 뿐만 아니라, DRAM(동작시 500mA)에 비해서도 저전력으로 동작하며, 크기가 작아 휴대 기기에 적합하다[3,8,9]. 플래시 메모리의 일반적인 특징은 <표 1>과 같다. 그러나, 플래시 메모리를 사용할 때 세 가지 문제점을 고려해야 한다. 첫 번째로 데이터를 한 번 쓴

* ETRI 선임연구원
** ETRI 정보가전연구부 책임연구원
*** ETRI 정보가전연구부장

영역에는 지움(cleaning) 과정을 수행하기 전에는 다시 쓸 수 없으며, 두 번째로 지우는 속도가 읽는 속도 및 쓰기 속도에 비해 매우 길다는 점이다. 세 번째로 플래시 메모리를 지울 수 있는 회수가 상온에서 약 10만회 정도로 정해져 있고, 지우는 방법도 지움 단위(Erase Unit, EU) 또는 세그먼트(Segment)라고 하는 일정 크기로 이루어 진다는 것이다.

<표 1> 플래시 메모리의 특징

특징 항목	값
읽기 속도	50~150 *nsec/byte*
쓰기 속도	5~10 μ*sec/byte*
지우기 속도	0.1~0.5 *sec/EU*
지우기 회수 제한	상온 약 10만회
지우기 단위(EU) 크기	64, 128, 256 *Kbyte*
전력 소모량	대기상태 - 20~100 μA 동작상태 - 30~50 *mA*

플래시 메모리는 셀(Cell)을 구성하는 구조에 따라, NOR 플래시, NAND 플래시, AND 플래시 등으로 구분할 수 있으나, NOR나 NAND 플래시 이외에는 거의 사용되지 않는다[4]. NOR 플래시는 임의 접근(Random Access)의 읽기 속도가 빠르고 비트당 접근이 용이함으로 CPU에 직접 연결되어 CPU가 수행하는 코드(Code)를 저장하는 목적으로 주로 사용되며, NAND 플래시는 느린 임의 접근 때문에 음악 파일이나 이미지 파일 등 대용량의 데이터를 한번에 저장하는 용도로 주로 이용된다. NOR 플래시는 주로 메모리 번지에 직접 연결되어 XIP (eXecute In Place) 수행이 가능함으로 메인 메모리의 소모를 줄이고 메모리에 옮기는 시간을 단축할 수 있다는 장점이 있다.

최근에 NAND 플래시의 용도가 휴대전화, MP3 재생기, PDA 대량 데이터 저장 등으로 확대됨에 따라 NAND 플래시의 기술이 활발히 연구되고 있으며, 시장 점유율도 40%까지 증가하고 있다[4]. NAND 플래시는 8 비트 또는 16 비트 단위로 메모리를 접근하기 때문에 직접도가 NOR 플래시에 비해 높으며, 가격도 싸다. 또한 전력 소모량이 적어, 쓰기 속도 0.5 μ*sec/byte*정도, 지우기 속도 5 *msec/EU* 정도로 NOR에 비해 10배 정도 빠르다. 그러나, 데이터를 접근할 때 8 비트 또는 16 비트 단위로만 수행이 가능하고 읽기 속도가 늦다는 단점이 있다.

3. 플래시 파일시스템 연구 동향

앞에서 언급한 플래시 메모리의 문제점을 극복하기 위해 여러 가지 방법이 제시되고 있다. 특히, 플래시 메모리의 가장 큰 제약인 지우는 속도가 느리고 지우는 회수가 정해져 있다는 문제점을 극복하기 위해 지움 정책(Cleaning Policy)이 가장 큰 쟁점이다.

3.1 플래시 메모리의 지움 정책

플래시 파일시스템의 지움 정책을 가장 쉽게 구현하는 방법은 로그 파일 시스템(Log-Structured File System, LFS) 형식을 이용하는 것이다[12,13]. LFS에서 모든 저장은 저장매체에 대해 순서적으로 발생한다. 저장매체를 끝까지 다 쓴 다음에는 처음으로 돌아와서 빈 공간을 확보해야 한다. 이 때 새로운 데이터를 저장할 공간에 유용한 데이터가 있으면, 이 데이터를 다른 공간으로 옮기고 새로운 데이터를 저장한다. 플래시 메모리의 경우 EU 단위로 지울 수 있으므로 유용한 데이터를 다른 EU으로 옮긴 다음, 현재의 EU를 지우고 새로운 데이터를 저장한다. 이러한 과정을 지움 정책(Cleaning Policy)라 한다.

Kawaguchi 등은 플래시 파일시스템을 구현하기 위해 블록 파일시스템 형태를 이용하고 있다[5]. 디스크를 이용한 파일시스템에서 데이터를 읽고 쓰는 단위로 일정 크기의 블록 즉, 섹터(Sector)를 사용하는 것처럼, 플래시 메모리의 공간도 섹터 단위

로 구분 한 다음 이런 섹터 단위로 읽기와 쓰기를 수행한다. 예를 들어, EU이 보통 128 Kbytes 크기인데, 512 Bytes 크기의 섹터를 256개 수용할 수 있다. 이러한 플래시 파일시스템을 구현하기 위해 Wu 등이 제안한 두 가지 지움 정책을 이용하고 있다[15].

첫 번째로 Greedy 방법은 EU 내에 유용한 데이터 블록(Valid Block)이 가장 적은 EU를 지울 대상으로 선정하는 방식이다. EU를 섹터 단위로 구분을 하게 되면, 첫 번째와 두 번째 섹터는 EU를 관리할 정보를 저장하게 되는데 이를 EU 헤더(Header, EUH)라 하며, 나머지는 데이터를 저장한다. 데이터를 저장하는 섹터는 처음 지워진 상태에서 Free 블록이고, 데이터를 저장하면 Valid 블록, 그리고 더 이상 유용한 데이터가 아니면 Invalid 블록이 된다.

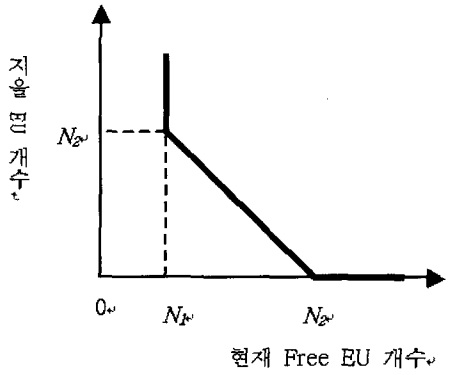
두 번째 지움 정책은 Cost-benefit 방법으로 비용(Cost)에 비해 이익(Benefit)이 많은 EU를 선택해서 지우는 방법이다. 값은 (식 1)에 의해 결정된다. 여기서 u는 EU에 대한 이용률이다. 즉, EU의 전체 블록에 대한 Valid 블록의 비율이다. 2u는 지울 EU의 Valid 블록을 다른 EU으로 옮김으로써 발생되는 읽기 시간과 쓰기 시간의 합을 의미한다. age는 가장 최근에 블록을 Invalid로 바꾼 후의 시간이다.

$$\frac{Benefit}{\cos t} = \frac{age \times (1-u)}{2u} \qquad \text{(식 1)}$$

[1]에서는 등급별 지움 정책(Ranked Cleaning Policy)을 제안하여, 플래시 메모리를 지우는 회수를 줄이고 플래시 메모리 내에서 지우는 곳을 고르게 안배함으로써 플래시 메모리의 수명을 연장할 수 있도록 지움 정책을 설계한다. 이렇게 플래시 메모리에 저장되는 공간을 고르게 배분하는 것을 Wear-leveling이라 한다.

$$R = A\frac{i}{2v \cdot f \cdot e} \qquad \text{(식 2)}$$

이러한 등급별 지움 정책은 먼저 각 EU에 대한 등급값을 (식 2)를 이용하여 계산한 다음 값이 가장 큰 순서대로 EU를 지우는 방식이다. 여기서, v는 EU 내에서 Valid 블록에 대한 비율이고, f는 Free 블록에 대한 비율, 그리고 i는 Invalid 블록에 대한 비율을 의미한다. v+f+i = 1이 된다. Valid 블록은 읽어서 다른 EU으로 옮겨야 함으로 2배의 영향력을 갖기 때문에 2v가 된다. e는 EU의 전체 지움 가능 회수에 대한 현재 지운 회수의 비율이다. A는 e 값에 대해 얼마 만큼의 비중을 둘 것인가에 대한 상수 값이다. 플래시 메모리의 수명이 많이 남아 있으면, Wear-leveling에 대해 큰 비중을 둘 필요가 없으므로 작은 값을 주어도 되지만, 전체적으로 플래시 메모리의 수명이 적게 남아 있을 때는 전체적인 Wear-leveling이 중요함으로 큰 값을 줄 수 있다.



(그림 1) 지울 EU 개수를 결정하는 그래프

이렇게 등급값이 정해졌을 때, 이제는 언제, 그리고 얼마 만큼의 EU를 지워야 하는지에 대한 문제가 남아 있다. EU에 데이터를 저장하고자 할 때, 플래시 메모리에 남은 공간이 없기 때문에 EU를 지우게 되는데, EU 내에 Valid 블록이 있으면, 이것을 먼저 옮긴 다음에 EU를 지울 수 있다. 그러므로 Valid 블록을 옮길 Free EU과 데이터 저장을 위한 EU 등 최소한 2개 이상의 Free EU를 확보해야 한다. 이러한 Free EU을 확보하는 문제는 (그림 1)의

그래프에 의해 결정될 수 있다. 전체 플래시 메모리에서 N2 이상의 Free EU를 확보하고 있으면, 지우는 과정이 수행될 필요가 없다. Free EU의 개수가 N1에서 N2 사이에 있을 때는 그래프의 값 만큼의 EU을 최소한 확보해야 한다. 이때 앞에서 계산한 등급값을 이용하여 지울 EU을 결정한다. N1 이하일 때는 최대한 많은 양의 EU을 지워서 Free EU로 확보해야 한다. 그러나 이 경우 지우는 과정을 수행할 수 없는 상황이 생기므로 Free EU의 개수가 N1 이하로 떨어지지 않도록 해야 한다.

이러한 지움 과정은 읽기나 쓰기에 비해 많은 시간을 소모하므로 플래시 메모리가 읽기 및 쓰기가 발생하지 않는 휴지 상태(Idle Time)에 있을 때 일반적으로 수행된다. 대부분의 플래시 메모리에서 지움 과정을 수행하다가 읽기 및 쓰기 요구가 들어오면 지움 과정을 보류하고(Suspend) 우선 순위가 높은 연산을 수행하도록 하는 기능을 지원한다.

3.2 플래시 메모리의 오류 복구 방법

임베디드 시스템은 주로 전지(Battery)를 이용하여 사용되는데 갑작스럽게 전원이 차단 되었을 때 안정된 종료가 이루어지지 않아 데이터를 손실할 우려가 있다. 이를 위해 [1,2]에서는 오류 복구 방법을 제시하고 있다. (그림 2)과 같이 EU과 블록이 여러 단계의 상태를 거치게 하여 오류를 복구하는 방법이다.

원본 블록	대상 블록
Valid	Free
Valid	Allocated
Valid	Prevalid
Invalid	Prevalid
Invalid	Valid

원본 EU	대상 EU	
Valid	Free	(1)
Valid	Allocated	(2)
Valid	Prevalid	(3)
Invalid	Prevalid	(4)
Invalid	Valid	(5)
Erasing	Valid	(6)
Free	Valid	(7)

(그림 3) 지움 과정 및 쓰기 과정에서 오류 복구

갑작스럽게 전원이 커진 뒤 다시 시스템을 켰을 때, EU이나 데이터 블록의 상태가 (그림 3)의 (1)인 경우 안정된 상태이므로 그대로 둔다. (2) 상태인 경우 대상 EU에 데이터를 저장하고 있던 단계이므로 대상 EU를 invalid 상태로 바꾸고 나중에 지움 정책에 따라 지운다. (3)인 경우 데이터 복사가 완료된 상태이므로 원본 EU를 invalid 상태로 바꾸고 지움 정책에 따라 지운다. 대상 EU은 valid 상태로 바꾸고 EU 번호가 같도록 일치시킨다. (4)인 경우도 데이터 복사가 완전히 이루어진 경우이므로 대상 EU를 Valid로 바꾼다. (5)인 경우 안정된 상태이므로 그대로 둔다. (6)인 경우 원본 EU를 지우는 도중이므로 Invalid로 만들고 지움 정책에 따라 다시 지운다. (7)인 경우는 안정된 상태이므로 그대로 둔다. 이렇게 함으로서 갑작스럽게 전원이 꺼져도 오류를 복구할 수 있으며, 플래시 메모리를 안정된 상태로 만들 수 있다. 이와 같은 오류 복구 방법은 임베디드 시스템의 특성상 갑작스럽게 전원이 꺼지는 경우가 많고, 전원이 꺼진 뒤에도 데이터를 보관해야 하는 시스템에 적용될 수 있다.

4. 플래시 파일시스템 구현

앞에서 설명된 플래시 파일시스템 연구 내용들을 가지고 실제로 구현된 시스템 몇 가지를 소개한다.

4.1 FTL(Flash Translation Layer)

플래시 메모리에 파일을 저장하기 위해 순차적인 플래시 공간이 임베디드 응용 프로그램에서 디스크의 섹터(Sector)처럼 보이도록 하기 위해 매핑(Mapping) 관리를 수행하는 드라이버 형식으로, 큰 지움 단위 블록을 섹터 단위로 쪼개서 가상 블록을 만들고 가상 블록 단위로 읽기와 쓰기를 수행한다[10]. 이렇게 물리적 플래시 공간을 블록 형태로 전환(Translation)함으로써 파일시스템에서 흔히 사용

되는 블록 디바이스처럼 다룰 수 있도록 한다. 결국 데이터 블록이 수정되어 같은 주소에 쓰여져도 실제 플래시 메모리는 다른 주소에 저장될 수 있다. Wear-leveling에 대한 개념이나, 갑작스런 전원 차단시 복구 방법이 미약한 단점이 있다.

4.2 TrusFFS

M-Systems 상의 Flite라는 플래시 메모리 관리 도구를 WindRiver 사의 VxWorks RTOS에 맞게 구현한 시스템으로 플래시에 대한 블록 디바이스 인터페이스를 제공한다[14]. FTL처럼 상위 계층의 개발자에게 플래시에 대한 특성을 숨김으로 디스크처럼 보이도록 하기 위해 Block-to-Flash 전환 시스템을 사용하고 있다. 이는 플래시 메모리를 일정 크기 블록으로 구분하고 그 블록 단위로 읽기, 쓰기를 수행한다. 또한 wear-leveling 알고리즘을 구현하여 쓰기 연산이 일정 블록으로 집중되는 것을 막고 있다. 더 이상 유용한(Valid) 데이터를 보관하지 않는 블록을 재생하기 위해 Garbage Collection 메커니즘을 이용한다. TrueFFS는 "저장 뒤 지움(Erase after write) 정책을 이용하여 오류 복구(Fault Recovery) 방법을 구현하고 있다.

4.3 JFFS

대부분의 플래시 파일시스템이 블록 디바이스를 Emulate하고 그 위에 파일시스템을 올리는 형식이지만 JFFS는 플래시의 공간을 순차적으로 보고, 발생되는 데이터를 크기에 관계없이 순차적으로 저장하는 LFS 방법의 하나이다[7]. 그러므로 Wear-leveling에 대한 구현을 따로 할 필요가 없다. 데이터 블록을 Emulate할 필요가 없어 갑작스럽게 전원이 꺼져도, 플래시 메모리의 공간을 순차적으로 읽음으로써 파일시스템을 새롭게 구성할 수 있으므로 오류 복구가 용이하다. 그러나, 순차적 처리로 인해 Mount 시간이 길다는 단점이 있다. FTL 등이 M-systems 사와 Intel 사에 특허권이 걸려있는 반면, JFFS는 Axis Communications 사에서 개발하여 오픈소스로 공개함으로써 GPL(GNU Public License)를 따르므로 소스 사용에 제약이 없다는 장점이 있다.

4.4 QplusFFS

한국전자통신연구원 정보가전부에서 개발한 실시간 운영체제(Real-Time Operating System)인 Qplus에서 사용되는 QplusFFS은 한국전자통신연구원과 한국과학기술원에서 공동 개발하였다[1,2]. QplusFFS은 디스크 파일시스템과 결합된 형태로 개발되었으며, 기존의 파일시스템 API를 지원하기 위해 FAT(File Allocation Table) 형식으로 파일을 저장하도록 하였다. 이를 위해 파일의 논리적인 주소를 플래시 메모리의 물리적인 주소로 매핑(mapping)하는 역할을 담당하는 FML(Flash Memory Mapping Layer)이라는 계층을 추가로 두었다. 앞에서 설명한 등급별 지움 정책을 이용하여 Wear-leveling 및 플래시 메모리의 사용 효율을 높였으며, 오류 복구 방법을 구현했다. 개발된 시스템은 실험용으로 제작된 Digital-TV 셋탑박스(Set-top box)와 TynuxBox에 적용되었다.

5. 결 론

정보가전 및 이동 통신 기기들은 최근의 정보통신 산업에서 중요한 테마로 자리잡고 있다. 또한 인터넷의 급속한 발전으로 업무 뿐 아니라 생활에서도 다양하게 인터넷 연결을 요구하고 있으며, 이러한 성장 속도는 매우 빠르다. 이런 시스템의 발전 추세는 인터넷에 존재하는 풍부한 정보와 디지털 컨텐츠를 최종의 단말기에서 간편하게 접근하여 조작하고 보관할 수 있는 저장매체를 절실히 요구하고 있다.

이런 대안으로 플래시 메모리는 훌륭한 매체가 될 수 있다. 아직은 저장 공간에 대한 가격에서 비용이 크지만, 최근 매우 빠르게 연구, 개발되고 있

는 분야이므로 계속적인 가격 하락이 예상된다. 이런 플래시 메모리를 이용한 플래시 파일시스템은 플래시 공간을 효율적으로 이용하는 방법이 되고 있다. 플래시 메모리의 수명을 연장하고 효율적인 플래시 메모리의 공간 확보(Garbage Collection)를 위해 지움 정책과 오류 복구 방법 등의 연구가 계속 필요하다. 최근에, NAND 형식 등 다양한 형태와 다양한 용량으로 플래시 메모리가 개발되고 있기 때문에, 개발된 플래시 파일시스템이 이러한 새로운 환경에 효율적으로 적용될 수 있도록 추가적인 개발이 필요하다. 현재 한국전자통신연구원 정보가전부에서는 임베디드 리눅스(Embedded Linux)를 기반으로 하는 실시간 운영체제(Qplus-P)를 개발하고 있으며, 이에 맞는 플래시 파일시스템을 개발하고 있다.

참고문헌

[1] 김정기, 박승민, 김채규, "정보가전을 위한 플래시 파일시스템에서 등급별 지움 정책과 오류 복구 방법", 제5회 차세대 통신소프트웨어 학술대회(NCS2001), pp.938-941, 2001.

[2] 박상호, 안우현, 박대연, 김정기, 박승민, "플래시 메모리를 위한 파일시스템 구현", 정보과학회 논문지: 컴퓨팅의 실제, Vol.7, No.5, pp.402-415, 2001.

[3] 민용기, 박승규, "이동컴퓨터를 위한 플래시 메모리 클리닝 정책", 학국통신학회 논문지, Vol. 24, No.5A, pp.657-666, 1999.

[4] 서강덕, "본격적인 시장 도입기에 접어든 NAND Flash Memory", 전자공학회지, 제27권, 제3호, pp.56-65, 2000.

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

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

[7] David Woodhouse, "JFFS: The Journaling Flash File System", Technical Paper of RedHat Inc. 2001.

[8] Intel Corporation, "3 volt Flash Memory 28F160S3 and 28F320S3", 1998.

[9] Intel Corporation, "Flash memory", Technical Paper, 1994.

[10] Interl Corporation, "Understanding the Flash Translation Layer(FTL) Specification", Technical Paper, 1998.

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

[12] 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.

[13] Trevor Blackwell, Jeffrey Harris, and Margo Seltzer, "Heuristic Cleaning Algorithms in Log-Structured File Systems," Proc. of USENIX Technical Conference, 1995.

[14] WindRiver, "TrueFFS for Tornado Programmer's Guide 1.0", 1999.

[15] 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.

저자약력

김 정 기

1992년 전북대학교 컴퓨터공학과(공학사)
1994년 전북대학교 컴퓨터공학과(공학석사)
1999년 전북대학교 컴퓨터공학과(공학박사)
1996년-1998년 시스템공학연구소 연구원
1998년-현재 한국전자통신연구원 선임연구원
관심분야 : 엠베디드 시스템, 파일시스템, 병렬 정보검색
e-mail : jkk@etri.re.kr

김 채 규

1978년 고려대학교 수학과(이학사)
1993년 호주 시드니 공과대 전산과학(석사)
1994년 호주 Wollongong대 전산과학(박사)
1997년-현재 한국전통신연구원 책임연구원(정보가전 연구부장)
관심분야 : 인터넷 정보가전, 실시간 운영체제, 멀티미디어, 데이터베이스, 전자상거래 등
e-mail :kyu@etri.re.kr

박 승 민

1981년 울산대학교 전자공학과(공학사)
1983년 홍익대학교 전자공학과(공학석사)
1998년-현재 충남대학교 전자공학과 박사과정
1983년-1984년 (주)LG전자
1984년-현재 한국전자통신연구원 정보가전연구부 실시간미들웨어연구팀장/책임연구원
관심분야 : 인터넷 정보가전, RTOS
e-mail : minpark@etri.re.kr