濟州大學校 産業技術硏究所 論文集, 第11卷, 1號, pp.94-100, 2000. 6
Cheju Nat'l Univ. Res. Insti. Ind. Tech. Jour., Vol.11, No.1, pp.94-100, 2000. 6

이동형 정보 기기를 위한 플래쉬 파일 시스템의 개발

이 동 희*

Development of a Flash File System for mobile information devices

Dong-Hee Lee*

ABSTRACT

In general, mobile devices use flash memory as their permanent storages. To maximize performance and reliability, flash memory requires specially designed software such as flash memory file system. This paper describes how to design and implement flash memory file system, named as TFS (Transactional File System). The TFS is similar to the DOS/FAT file system. It provides logging technique for fast power-fail recovery. After being developed on a flash memory emulator in the UNIX system, the TFS is ported to the Web Video Phone that has Strong-ARM CPU and pSOS operating system. Them the TFS is ported to the embedded Linux system.

Key words : Mobile device, flash memory, logging, transactional file system

Ⅰ. 서 론

현재 이동형 기기에 대한 요구가 증가함에 따라 PDA (Personal Digital Assistant)와 같은 휴대형 정보 시스템의 사용이 증가하고 있으며, 무선 통신 기술의 발전에 따라 정보 검색이 가능한 휴대 전화가 등장하고 있다. 이러한 이동형 정보 기기는 데이터를 저장하기 위하여 기존의 컴퓨터 시스템과 다른 보조 기억 장치를 요구한다[1]. 휴대형 정보 기기는 전원을 공급받기 위하여 주로 배터리를 사용하기 때문에, 전력 소모가 작고 충격이나 열악한 외부 환경을 감당할 수 있는 보조 기억 매체를 필요로 한다. 그리고 휴대형 정보 기기는 사용 중 전력 중단(power-fail)과 같은 오류가 발생할 확률이 높으며, 이러한 오류가 발생한 경우 빠른 회복이 필요하다.

플래쉬 메모리(flash memory)는 충격에 강하고 전력 소모가 작으며, 전원이 공급되지 않는 상황에서도 데이터를 유지할 수 있기 때문에 이동형 정보 기기의 보조 기억 장치로 적합하다[2]. 전원이 공급되지 않는 상황에서도 데이터를 유지한다는 플래쉬 메모리의 특성은 특히 기존 컴퓨터 시스템에서 사용되는 하드디스크(hard disk)와 유사한 역할을 수행할 수 있음을 의미하며, 실제로 많은 휴대 전화나 내장형 시스템(embedded system)에서 보조 기억 장치로 사용되고 있다.

플래쉬 메모리는 이동형 정보 기기에 이상적인 보조 기억 장치라고 할 수 있지만 플래쉬 메모리를 이

* 제주대학교 통신·컴퓨터공학부, 산업기술연구소.
Faculty of Telecommunication & Computer Engineering., Res. Insti. Ind. Tech., Cheju Nat'l Univ.

동 기기의 보조 기억 장치로 사용하기 위해서는 플래쉬 메모리용 파일 시스템 같이 특별하게 고안된 소프트웨어 지원이 필요하다. 플래쉬 메모리용 파일 시스템의 필요성은 기존 파일 시스템을 플래쉬 메모리에 적용할 때 발생하는 문제점을 보면 알 수 있다[3,4]. 첫째, 기존의 파일 시스템은 하드디스크에 최적화된 구조를 가지고 있기 때문에 플래쉬 메모리의 특성을 제대로 활용하지 못한다. 따라서 플래쉬 메모리가 제공하는 성능을 최대로 이용하지 못하며 안전성 면에서도 떨어진다. 둘째, 기존의 파일 시스템은 시스템 동작중에 갑자기 전원이 중단되거나 오류가 발생하여 시스템을 재시작할 때 파일 시스템의 무결성을 회복하기 위하여 오랜 시간에 걸쳐 파일 시스템 검사를 수행하여야 한다. 그러나 빠른 시동과 동작을 요구하는 이동형 정보 기기에서 이러한 파일 시스템 검사는 치명적인 약점으로 작용한다.

기존의 파일 시스템을 플래쉬 메모리에 적용할 때 발생하는 문제점들은 결국 플래쉬 메모리의 특징을 최대한 이용하며, 빠른 오류 회복이 가능한 새로운 파일 시스템을 요구하고 있다. 본 논문에서는 이러한 요구사항에 따라 플래쉬 메모리에서 최적화되고, 로깅 기법[5]을 적용하여 빠른 오류 회복과 높은 안전성을 제공하며, 휴대형 또는 내장형 시스템에 적당하도록 코드의 크기가 작은 플래쉬 파일 시스템인 TFS(Transactional File System)의 구조와 개발에 대하여 설명한다. 이러한 파일 시스템은 현재 Intel Strata 플래쉬 메모리[2]를 사용하고 있는 웹 비디오 전화기 등에 탑재되어 사용되고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 플래쉬 메모리의 특징과 관련 연구에 대하여 설명한다. 3장에서는 TFS의 설계 목표와 구조 그리고 로깅 기법에 대하여 설명한다. 4장에서는 구현과 적용 예를, 그리고 5장에서 결론을 설명한다.

II. 플래쉬 메모리의 특징과 플래쉬 메모리와 관련된 기존 연구

플래쉬 메모리에 대한 읽기 연산은 비교적 자유로우나 쓰기 연산은 복잡한 과정을 거친다. 먼저 삭제 블록 단위(제품에 따라 다르나 Intel의 플래쉬 메모리는 보통 64 KB ~ 128 KB 정도이며, 삼성의 플래쉬 메모리는 4KB ~ 32 KB이다)로 삭제(erase 또는 flash) 연산을 수행한다[2,6]. 삭제 연산이 수행되면 삭제 블록 내의 모든 비트들은 모두 1로 설정된다. 그 후에 원하는 곳에 쓰기 연산을 수행한다. 쓰기 연산은 반드시 비트가 1에서 0으로 변하는 형태로만 쓰기가 가능하며, 0에서 1로 변경하려면 반드시 전체 삭제 블록 단위로 삭제 연산을 수행하여야 한다. 삭제 연산 후에는 모든 비트가 1로 설정되어 있기 때문에 원하는 데이터의 쓰기가 가능하다.

이렇게 쓰기 연산이 제약받는 플래쉬 메모리를 저장 매체로 이용하기 위해서는 정교한 소프트웨어의 지원이 필요하다. 먼저 플래쉬 메모리를 일반적인 하드디스크와 같이 512 byte 섹터 단위로 자유롭게 읽고 쓸 수 있도록 하는 섹터 재사상 드라이버(Sector Remapping Driver)가 있다. 대표적인 섹터 재사상 드라이버로는 M-Systems 사가 최초로 상용화한 FTL(Flash Translation Layer)[7]이 있으며, FTL은 PCMCIA 표준으로 공인 받았다. FTL은 플래쉬 디바이스 드라이버 상위에 존재하며, 플래쉬 메모리를 512 byte 의 섹터들로 구성된 블록 디바이스처럼 사용할 수 있도록 해준다. 따라서 FTL 상위에서는 기존의 하드디스크에서 동작하는 파일 시스템이 특별한 수정없이 사용될 수 있다. 그러나 일반적으로 플래쉬 메모리의 특징이 하드디스크와 다르기 때문에 최적화를 위해서는 파일 시스템의 변경이 필요하다.

플래쉬 메모리를 저장장치로 사용하기 위한 연구는 2가지로 분류할 수 있다. 첫 번째 부류는 플래쉬 메모리를 하드디스크와 같은 보조 기억 장치로 사용하기 위한 것으로 이를 위해서 플래쉬 메모리 특징에 맞는 파일 시스템을 개발하였다. 기존 연구[3,4]에서 개발한 플래쉬 메모리용 파일 시스템은 대부분 LFS(Log-structured File System)[8]를 플래쉬 메모리에 적용한 것으로, 그 이유는 LFS의 동작이 삭제 블록 단위로 삭제한 후 기록해야 하는 플래쉬 메모리의 특징과 잘 부합하기 때문이다. 그러나 LFS가 동작하기 위해서는 많은 메모리와 복잡한 cleaning 알고리즘을 요구하기 때문에, 작은 메모리와 저전력에서 동작해야 하는 이동형 기기에 적용하기 어렵다. 아울러

LFS의 경우 파일 시스템 변경 도중 오류가 발생하거나 시스템의 비정상적으로 종료하는 경우 회복에 많은 시간이 걸리므로 이동형 기기에 적당하지 않다.

두 번째 부류는 플래쉬 메모리를 전원 공급이 중단되더라고 데이터 손실이 없는 안정적인 기억 장소로 사용하기 위한 연구로서, 플래쉬 메모리는 기존의 RAM과 같은 주기억 장치의 확장으로 간주된다. 이렇게 플래쉬 메모리를 안정적인 주기억 장치의 확장으로 사용하기 위하여 eNVy 시스템[9]에서는 배터리가 달려있는 SRAM과 플래쉬 메모리의 조합을 사용한다. 메모리에서 변경되는 내용은 일단 SRAM에 기록되고, SRAM에 새로운 내용을 기록하기 위한 여유 공간이 없는 경우 SRAM에서 가장 오랫동안 참조되지 않은 메모리 블록을 플래쉬 메모리에 기록한다. 즉 SRAM은 플래쉬 메모리의 캐쉬 역할을 수행하며, 이렇게 SRAM과 플래쉬 메모리의 조합으로 구성된 eNVy 시스템은 성능과 안전성이 뛰어나다.

III. TFS의 설계 목표와 구조

3.1. TFS의 설계 목표

TFS는 휴대형 기기에 널리 사용되는 플래쉬 메모리의 특징에 맞게 최적화된 파일 시스템이다. 본 절에서는 TFS의 설계 목표와 플래쉬 메모리의 특징이 파일 시스템의 구조에 미치는 영향을 설명한다.

TFS의 설계 목표를 설명하기 전에 먼저 TFS를 적용할 목표 시스템을 설명하면 다음과 같다. TFS가 적용될 목표 시스템은 기억 장소 용량이 작고 처리기의 속도가 비교적 느린 휴대형 기기 또는 내장형 시스템이다. 이러한 내장형 시스템은 주로 플래쉬 메모리를 저장 장치로 사용하며, 또한 FTL과 같은 섹터 재사상 드라이버를 사용한다. TFS는 이러한 섹터 재사상 드라이버가 존재하고 있음을 가정하여 개발되었다.

TFS의 설계 목표는 다음과 같다.

- 플래쉬 메모리의 특징을 활용하여 안전성이 최적화되어야 한다.
- 휴대형 기기 또는 내장형 시스템에서 사용되기 위하여 코드 크기가 작고 메모리 요구량이 작다.
- SmartMedia[10] 표준으로 발전될 수 있도록 하기 위하여 FAT 구조를 사용한다.
- 오류 발생 후 재시작할 때 빠른 오류 복구가 가능하여 정상적인 시스템 부팅과 차이가 없어야 한다.

Intel NOR 타입의 플래쉬 메모리는 읽기/쓰기 특성면에서 하드디스크와 다르다. 하드디스크는 항상 512바이트 섹터 단위로 데이터를 읽고 쓰지만 플래쉬 메모리는 바이트 단위로 읽기/쓰기가 가능하다(보통 16 ~ 32 바이트 단위로 읽기/쓰기를 수행하여야 효율적이며, 제품에 따라 다르다). 그리고 플래쉬 메모리는 입출력에 걸리는 시간이 데이터 양에 비례하며, 데이터 양이 작으면 입출력 시간이 작다. 다른 특성으로는 하드디스크에 입출력을 할 때 디스크 헤드가 원하는 트랙으로 이동하는 탐색 시간과 디스크의 회전 지연 시간이 필요하지만, 플래쉬 메모리는 읽기/쓰기를 수행하는 물리적 위치와 무관하게 소요되는 시간이 동일하다.

플래쉬 메모리의 이러한 특징은 특히 파일 시스템의 구조에 중요한 영향을 준다. 먼저 작은 양의 데이터를 기록하는데 소요되는 시간이 아주 작기 때문에 파일 시스템이 변경될 때마다 로그(log)를 남기고 변경이 완료된 후 로그를 제거하는 로깅(logging) 기법[5]이 아주 효율적으로 동작할 수 있다. 하드디스크에서 동작하는 파일 시스템의 경우 로그를 남기기 위해서는 로그 영역으로 하드디스크의 헤드가 이동하는 탐색 시간이 매우 크기 때문에 로깅 기법을 사용하면 파일 시스템의 성능이 크게 저하된다. 그러나 플래쉬 메모리를 사용하는 경우 로깅 기법으로 인한 성능 저하는 매우 적은 반면, 파일 시스템의 오류 회복을 위한 알고리즘이 매우 단순해지는 장점이 있다. 이러한 점들을 고려하여 TFS는 로깅 기법을 이용하고 있으며, 파일 시스템 알고리즘이 간단하고 매우 빠르게 오류를 복구할 수 있다.

다양한 제품들의 플래쉬 메모리를 저장장치로 이용할 때 가장 중요한 고려 사항이 제품들 간에 호환성이다. 다양한 플래쉬 메모리간에 호환성을 유지하기 위하여 저장 매체의 형식에 대한 표준이 정의되고 있는데 그중 하나가 SmartMedia[10]이다. SmartMedia 표준에 의하면 파일 시스템은 FAT(File Allocation

Table) 구조를 가지도록 정의하고 있으며, 본 논문에서는 향후 SmartMedia 표준 파일 시스템으로 발전시킬 수 있도록 하기 위하여 TFS의 기본 구조로 FAT를 사용하였다.

3.2. TFS의 구조 : FAT 파일 시스템 + 로깅

TFS의 전체적인 구조는 Fig. 1과 같다. TFS는 섹터 재사상 드라이버 위에서 동작한다. 따라서 플래쉬 메모리는 512 byte 섹터들의 연속으로 보여지며, 파일 시스템은 몇 개의 섹터를 모아 하나의 클러스터를 구성하고, 클러스터 단위로 저장공간을 관리한다. TFS에서는 클러스터의 크기에는 제한이 없으나 보통 1~8개 섹터를 모아 하나의 클러스터로 구성한다. TFS는 섹터 재사상 드라이버 위에서 동작하지만, 로그를 기록하기 위한 Operation Log와 FAT Log Block 영역은 섹터 재사상 드라이버를 거치지 않고 파일 시스템이 직접 관리하여 로그를 기록한다.

0번 클러스터에는 부트 정보가 수록되어 있으며, 부트 정보는 다음과 같다.

- 파일 시스템 이름, 버전 번호
- 클러스터의 크기와 개수
- FAT 항목 수, FAT의 시작 위치, 루트 디렉토리의 위치

디렉토리에는 그 디렉토리에 존재하는 모든 파일 또는 하위 디렉토리의 정보가 기록되어 있다. 즉 디렉토리 엔트리마다 파일 또는 하위 디렉토리의 정보가 기록되어 있다. TFS는 트리 구조의 계층적 디렉토리 구조를 지원한다. 루트 디렉토리는 파일 시스템 포맷시에 생성되어 모든 파일을 찾아가는 시작 위치가 된다. 디렉토리 엔트리에는 Fig. 1과 같이 파일 또는 하위 디렉토리의 상태와 이름, 최종 갱신 시각, 파일 크기, 속성, 첫 번째 클러스터 번호 등이 기록된다.

FAT 블록에는 파일 할당 테이블이 존재하며 이 테이블의 각 엔트리는 클러스터 1부터 파일 시스템의 마지막 클러스터까지 대응되어 해당 클러스터가 사용중인지 아닌지를 나타낸다. 즉 엔트리가 FF 항목으로 표시된 클러스터는 사용중이 아니며, FF가 아닌 엔트리는 해당 클러스터가 사용중임을 나타낸다. FF가 아닌 엔트리는 해당 클러스터가 파일이나 디렉토리에 할당되어 사용중임을 나타내는데, 0인 경우 클러스터는 파일이나 디렉토리의 마지막 블록임을 나타내고, 0이 아닌 정수인 경우 파일이나 디렉토리의 내용이 저장된 다음 클러스터 번호를 나타낸다. 예를 들어 루트 디렉토리에 있는 파일 File1의 경우 데이터가 저장된 첫 번째 클러스터는 7번이며, 이는 루트 디렉토리의 FirstAddr에 기록되어 있다. 다음 번 클러스터는 파일 할당 테이블의 7번 클러스터 엔트리에

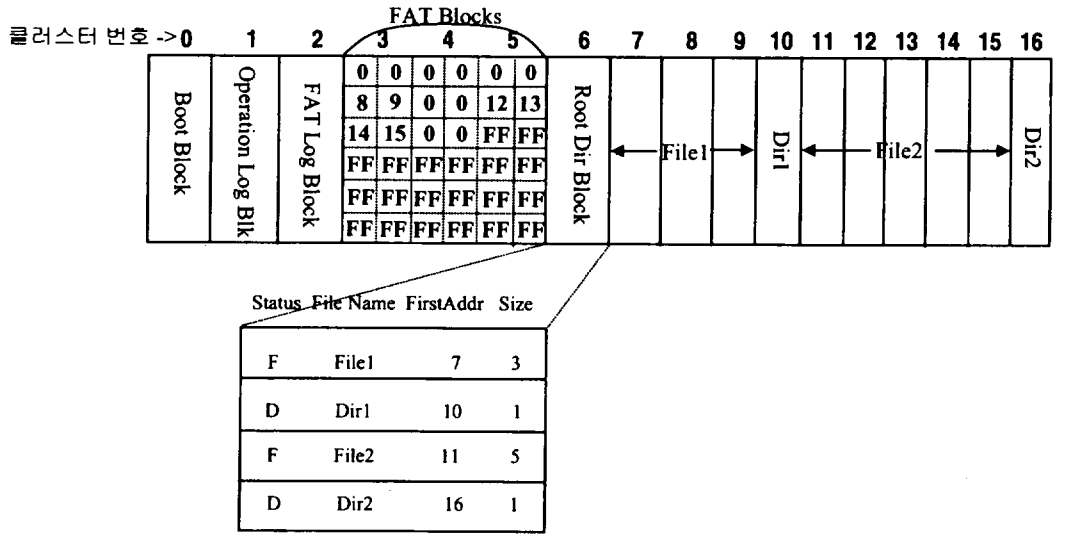

Fig. 1 Overview of the TFS structure

적혀 있는데, Fig. 1의 경우 8번 클러스터이다.

3.3. TFS의 로깅 : Operation Logging

TFS는 오류로부터 빠른 회복을 위하여 로깅 기법을 사용하는데, 로그 영역은 Operation Log와 FAT Log라는 2개 영역으로 나뉘어져 있다. Operation Log는 파일 생성, 삭제와 디렉토리 생성, 삭제와 같은 연산이 수행될 때마다 로그를 기록하는 영역이며, 연산이 완료되면 로그는 삭제된다. FAT Log에는 파일 할당 테이블의 내용이 변경될 때 변경 내용을 미리 기록한다.

Operation Log에는 다음 5가지 연산이 수행될 때 연산 내용이 미리 기록된다.

- File Delete : 파일의 삭제
- File Create : 새로운 파일의 생성
- File Write : 파일의 내용을 변경하거나 크기가 변화
- Dir Create : 디렉토리 생성
- Dir Delete : 디렉토리 삭제
- Dir Compact : 디렉토리에 있던 파일이나 하위 디렉토리가 삭제되는 경우 디렉토리 엔트리 부분을 압축
- File(Dir) Move : 파일이나 디렉토리의 이름 변경 또는 다른 디렉토리로 이동

로그를 기록한 후 연산을 수행하다가 오류가 발생하면 재시작할 때 로그를 보고 해당 작업을 복구한다. 오류로부터 복구하는 도중에 또다시 오류가 발생하더라도 복구가 가능하다.

3.4. TFS의 로깅 : FAT Logging

일단 파일 시스템이 기동하면, FAT 내용을 모두 메모리로 읽어들여 메모리 FAT를 만들게 된다. 그 후에 파일이나 디렉토리가 확장, 삭제, 생성되면 FAT가 변하게 되는데, 이러한 변화는 모두 메모리 FAT와 플래쉬 파일 시스템의 FAT영역에 기록되어야 한다. 메모리 FAT에 대한 빈번한 변화는 성능에 많은 영향을 미치지 않으나, 플래쉬 FAT 영역에 대한 빈번한 수정의 경우 성능을 매우 떨어뜨린다. 이에 대한 해결 방법으로 FAT에 대한 변화는 일단 메모리 FAT에만 기록하고, 주기적으로 또는 어떤 필요성이 있는 경우에만 다수의 FAT 변경이 이루어진 메모리 FAT의 내용을 플래쉬의 FAT 영역에 기록하면 되는데, 이 방법을 사용하기 위해서는 오류로부터 회복할 수 있는 시나리오가 필요하다. 왜냐하면 메모리 FAT만 변경이 이루어지고 플래쉬의 FAT 영역에 기록되지 않은 상태에서 비정상적으로 파일 시스템이 종료하면, 나중에 오류로부터 회복하기 어렵기 때문이다. 이러한 문제를 해결하기 위하여 FAT Log 블록을 사용한다.

12 비트 FAT를 사용하는 경우 FAT를 변경하다가 오류가 발생할 때 특히 회복이 어렵다. 12 비트 FAT를 사용하는 경우는 FAT의 한 엔트리가 두 섹터 사이에 걸쳐서 저장될 수 있기 때문에 구현 시 특별한 주의가 필요하다. 두 섹터에 걸쳐있는 FAT 엔트리를 변경할 때, 한 섹터는 변경하고 다른 섹터를 변경하기 전에 오류가 발생하였다면, 재시작할 때 변경 동작을 완료하거나 무효화 시켜야 한다. 그런데, 특별한 기록이 없다면, 완료하거나 무효화 시킬 수 없다. 따라서 이러한 문제를 해결하기 위하여도 FAT Log 블록의 사용이 필요하다.

FAT Log 블록은 FAT의 변화 내용을 담고 있다. FAT 엔트리가 변경되면, FAT Log 블록에 변경 내용을 기록하고, 메모리 FAT를 변경시킨다. 만약 로그 블록에 다 차서 더 이상 변경 내용을 기록할 수 없는 경우, 메모리 FAT에서 변경된 부분을 플래쉬의 FAT 영역에 기록하도록 한다. 따라서 플래쉬의 FAT 영역에 대한 변경 횟수를 상당히 줄여 파일 시스템의 성능을 향상시킬 수 있다.

플래쉬 FAT에 바로 변경 내용을 기록하는 것보다 FAT Log 블록에 변경 내용을 기록하여 성능을 향상시킬 수 있는 이유는 이 두 동작이 실제 수행되는 과정이 다르기 때문이다. 플래쉬 FAT를 변경시키는 동작은 하위 레벨의 섹터 재사상을 통하여 수행된다. 따라서 새로운 512 byte 섹터를 할당받고 이곳에 변경된 섹터 내용을 기록한 다음, 원래 섹터는 무효화(invalid)시키고 새로운 섹터를 유효(valid)로 설정하게 된다. 결국 플래쉬 FAT를 변경시키기 위해서는 512 byte 섹터 내용을 기록하고, 사상을 바꾸는 과정을 거쳐야 하기 때문에 오버헤드가 큰 과정이다. 그러나 FAT Log 블록에 대한

변경은 섹터 재사상 계층을 거치지 않는다. FAT 로그 블록은 처음에 모두 1로 기록되어 있다. 로그는 앞에서부터 FAT 변화 내용만을 기록하며, 섹터 재사상이 이루어지지 않는다. 아울러 플래쉬 메모리는 디스크와는 다르게 헤드의 탐색 시간 등이 필요하지 않기 때문에 섹터 재사상없이 로그 정보의 쓰기 과정은 오버헤드가 아주 작다. 따라서 FAT 로그를 기록하여 플래쉬 FAT 변화 빈도를 줄이는 것은 성능을 매우 향상시킨다.

3.5. 파일 시스템 성능 측정 도구와 파일 시스템 검사 유틸리티

TFS에는 파일 시스템의 동작을 확인하기 위하여 다양한 부가 기능이 추가되었다. 부가 기능 중 하나로 파일 시스템의 성능을 온라인에 측정할 수 있는 성능 측정 기능이 추가되어 응용 프로그램에서 파일 시스템의 성능과 동작에 대한 데이터를 얻을 수 있다. 이러한 성능 측정 도구는 파일 시스템의 성능 최적화에 많은 도움이 되었다.

TFS는 파일 시스템 검사 유틸리티 기능을 가지고 있어, 파일 시스템 무결성 검사가 가능하다. 이러한 파일 시스템 무결성 검사는 시스템 부팅시에 수행할 수도 있으나 많은 시간이 소요되기 때문에 주기적으로 또는 특별한 지시에 의해 시행하는 TFS의 안전성 테스트할 때에 수행된다.

IV. TFS의 구현과 성능 평가

TFS는 UNIX 상에서 동작하는 플래쉬 메모리 에뮬레이터와 섹터 재사상 드라이버 상에서 C언어로 개발되었다. 실제 플래쉬 메모리가 내장된 목표 시스템에서 구현하지 않고 플래쉬 메모리 에뮬레이터 상에서 구현하면 파일 시스템 코딩 및 디버깅 과정이 용이하다. 또한 파일 시스템 동작 중 다양한 부분에서 강제로 종료시킨 후 재시작하면서 오류 회복이 가능한지를 검사하는 테스트 프로그램을 이용하여 오류 회복 능력을 검사할 수 있으므로 안전성도 향상시킬 수 있다. 테스트 프로그램은 파일 생성, 삭제, 파일 이름 변경, 디렉토리 생성, 삭제와 같은 파일 시스템 동작 스크립트를 입력으로 받는다. 테스트 프로그램은 이러한 스크립트를 계속적으로 수행하는 도중 무작위로 전원 공급 중단과 같은 오류를 발생시켜 파일 시스템 연산을 중단시킨다. 그리고 재시작 과정을 거쳐 파일 시스템 오류를 회복시킨 후 파일 시스템 검사를 거쳐 오류 복구 여부를 검사하고 스크립트 수행을 재개한다.

플래쉬 메모리 에뮬레이터 상에도 개발된 TFS는 에뮬레이터 상에서 충분한 테스트를 거친 후에 200MHz Strong-ARM 프로세서를 사용하며 pSOS 운영체제가 동작하는 웹 비디오 폰에 이식되었다. 이식을 위해서 TFS의 일부 정의를 ARM Tools와 pSOS 개발 환경에 맞도록 수정하였다. 웹 비디오 폰은 32Mbyte의 Intel Strata 플래쉬 메모리[2]를 가지고 있으며, 이 플래쉬 메모리에 수행 프로그램, 환경 데이터와 사용자의 전화 번호부 등을 저장하기 위하여 TFS를 사용한다. TFS는 웹 비디오 폰 이외에도 현재 이동 전화, 디지털 텔리비젼과 PDA등에 적용될 계획이다.

TFS의 성능을 측정하기 위하여 다양한 크기의 파일에 대한 읽기와 쓰기를 10회 반복하여 평균을 계산하였다. Table 1은 섹터 재사상 드라이버 위에서 동작하는 TFS의 성능 측정 결과를 보여준다. 이 측정 결과를 보면 읽기의 경우 플래쉬 메모리의 최대 성능에 근접하여 RAM과 비슷한 성능을 보이지만, 쓰기의 경우 플래쉬 메모리 최대 쓰기 속도의 30% ~ 50% 정도의 성능을 보여주고 있다. 그러나 TFS가 동작하면서 성능 저하가 생기는 가장 큰 요인은 섹터 재사상 드라이버가 동작하면서 유발하는 reclaim(또는 cleaning) 연산[11] 때문이며, 파일 시스템의 오버헤드는 아주 작다.

Table 2. Read/write performance on the combination of TFS and sector remaping driver

데이터 파일 크기	읽기	쓰기
256 Kbytes	1 초 미만	8.31 초
695 Kbytes	1 초 미만	22.85 초

V. 결 론

본 논문에서는 플래쉬 메모리를 저장장치로 사용하는 휴대형 기기 또는 내장형 시스템에 최적화된 파일 시스템의 설계 및 구현에 대하여 설명하고 있다. 본 논문에서 개발한 TFS는 특히 플래쉬 메모리의 특성을 이용하여 안전성 및 성능이 최적화 되었으며, 또한 로깅 기법을 사용하여 휴대형 기기의 요구사항인 빠른 오류 복구 능력을 제공한다. 현재 TFS는 웹 비디오 폰과 같은 제품에 적용되어 사용중이며, 앞으로 더욱 많은 시스템에서 사용될 계획이다. 또한 TFS를 Linux로 이식하여 Embedded Linux 시스템의 플래쉬 파일 시스템으로 사용할 계획이다.

참고 문헌

1) F. Douglis, and R. Caceres, 1994, Storage Alternatives for Mobile Computers, *Proceedings of the 1st Symposium on Operating Systems Design and Implementation*, pp. 25-37.

2) Intel^{R}, 5 Volt Intel^{R} StrataFlash^{TM} Memory, *Intel^{R} Flash Memory DataSheet*.

3) A. Kawaguchi, S. Nishioka, and H. Motoda, 1995, A Flash-Memory Based File System, *Proceedings of the 1995 USENIX Conference*.

4) H. Kim, and S. Lee, 1999, A New Flash Memory Management for Flash Storage System, *23rd Annual International Computer Software and Application Conference*.

5) U. Vahalia, 1996, UNIX Internals, the new frontiers, *Prentice Hall*, pp. 344-345.

6) Samsung Electronics, 2000, NAND Flash Datasheet, *www.intl.samsungsemi.com/Memory/Flash/datasheets.html*.

7) Intel^{R}, 1997, Understanding the Flash Translation Layer (FTL) Specification, *Intel^{R} Technical Paper*.

8) M. Rosenblum, and J. K. Ousterhout, 1992, The Design and Implementation of a Log-structured File System, *ACM Transactions on Computer Systems*, Vol. 10 No. 1 pp. 26-52

9) M. Wu, and W. Zwaenepoel, 1994, eNVy: A Non-Volatile, Main Memory Storage System, *Proceedings of the 6th ASPLOS Conference*, pp. 86-97.

10) SSFDC, SmartMedia^{TM} product lineup and general specifications, *www.ssfdc.or.jp/english*.

11) M. Chiang, Paul C. H. Lee, and R. Chang, Cleaning Policies in Mobile Computers Using Flash Memory, *To appear in Journal of Systems and Software*.