폴트 삽입 테스트를 이용한 플래시 메모리 소프트웨어의 강건성 분석

(Robustness Analysis of Flash Memory Software using Fault Injection Tests)

이 동 희 †

(Donghee Lee)

요 약 휴대전화와 PDA 등에서 수행되는 플래시 메모리 소프트웨어는 돌발적인 전원 중단이나 기록 매체 폴트에 대처하기 위하여 충분히 테스트되어야 한다. 이러한 테스트를 위하여, 폴트 삽입 기능을 가지는 플래시 메모리 에뮬레이터를 설계하고 구현하였다. 폴트 삽입을 통한 테스트 기법은 FTL(Flash Translation Layer)과 플래시 메모리 기반 파일 시스템의 폴트 회복 기법을 설계하고 폴트로 인한 피해를 분석하는데 유용한 도구로 사용되었다. 본 논문에서는 플래시 메모리에서 관찰되는 폴트의 유형과 플래시 메모리 에뮬레이터에서 구현된 폴트 삽입 기능에 대해 설명한다. 그리고 폴트 삽입 테스트 과정에서 밝혀진 디자인 결함에 대하여 설명한다. 특히 신뢰성을 향상하기 위하여 도입된 기능이 신뢰성을 향상하기 보다는 피해를 유발하는 것으로 밝혀졌다. 마지막으로 FTL과 파일 시스템의 "폴트 후 동작"에 대해 설명한다.

키워드 : 플래시 메모리, FTL(Flash Translation Layer), 파일 시스템, 폴트 삽입

Abstract Flash memory software running on cellular phones and PDAs need to be tested extensively to cope with abrupt power and media faults. For those tests, we designed and implemented a Flash memory emulator with fault injection features. The fault injection tester has provided a helpful framework for designing fault recovery schemes and also for analyzing fault damages to the FTL (Flash Translation Layer) and file system for a Flash memory based system. In this paper, we discuss Flash memory fault types and fault injection features implemented on this Flash memory emulator. We then discuss in detail a design flaw revealed during fault injection tests. Specifically, it was revealed that a scheme that was believed to improve reliability instead, turned out to be harmful. In addition, we discuss post-fault behaviors of the FTL and the file system.

Key words : Flash Memory, FTL (Flash Transaltion Layer), File System, Fault Injection

1. 서 론

플래시 메모리는 현재 휴대전화나 PDA의 저장 매체로 널리 사용되고 있다[1]. 그러나 플래시 메모리의 경우 데이타를 기록하고 읽기 위해서는 몇 가지 제약이 존재하며, 이러한 제약을 극복하기 위하여 FTL (Flash Translation Layer)[2-4]과 플래시 메모리 기반 파일 시스템(이하 플래시 파일 시스템이라 부름)과 같은 소프트웨어가 필요하다. FTL은 플래시 메모리를 디스크와 같이 읽기/쓰기가 가능한 섹터들의 집합인 것처럼 모방해주는 소프트웨어 계층으로서, 논리 섹터 주소를 플래시 메모리상의 물리적인 위치로 변환하는 역할을 수행한다. 플래시 파일 시스템은 플래시 메모리에 기반하며 이동 환경에 최적화된 파일 시스템을 일컫는다.

이동형 기기는 급작스런 전원 중단과 같은 열악한 환경에 노출되어 있다. 아울러 플래시 메모리는 매체의 고유한 특징으로서 몇 가지 산발적인 폴트가 확률적으로 발생한다[5]. 이렇게 열악한 환경과 예측하기 어려운 폴트들이 발생하는 경우에도 이동형 기기 사용자들의 기대 수준을 맞추기 위해서는 플래시 메모리 소프트웨어를 매우 철저하게 테스트할 필요가 있다.

FTL과 플래시 파일 시스템을 개발하면서 개발 기간을 단축하고 소프트웨어의 품질을 향상시키기 위하여

· 본 연구는 한국과학재단 특정기초연구(R01-2004-000-10188-0)지원으로 수행되었음

† 정 회 원 : 서울시립대학교 컴퓨터과학부 교수
dhlee@venus.uos.ac.kr

논문접수 : 2005년 4월 12일
심사완료 : 2005년 5월 2일

플래시 메모리 에뮬레이터를 개발하였다. 또한 플래시 메모리에서 특징적으로 발생하는 폴트를 분석하여, 플래시 메모리 에뮬레이터에 폴트 삽입 기능을 추가하였다. 이러한 플래시 메모리 에뮬레이터는 각 개발자들에게 독립적인 개발 환경을 제공하여, FTL과 플래시 파일 시스템 루틴들이 병렬로 개발될 수 있도록 하였다. 또한 개발자들로 하여금 데스크탑 환경에서 강력한 개발 도구들을 사용하면서 FTL과 플래시 파일 시스템을 개발할 수 있도록 하였다. 폴트 삽입 기능이 추가된 플래시 메모리 에뮬레이터는 회복 기법의 설계 단계부터 구현 및 소프트웨어의 최종 테스트 과정까지 광범위하게 사용되었다.

본 논문에서는 플래시 메모리의 고유한 폴트 유형과 폴트 회복 기법에 대해 설명한다. 그리고 플래시 메모리 에뮬레이터에 추가된 폴트 삽입 기능을 설명한 후, FTL과 플래시 파일 시스템에서 수행된 폴트 삽입 테스트에 대해 설명한다. 특히 본 논문에서는 "*비트-변경*" 폴트에 초점을 맞추어 설명한다. 왜냐하면, 다른 유형의 폴트는 많은 경우 플래시 메모리 명령 실행 직후 검출이 가능하며, 아울러 대부분의 경우 데이타 손실 없이 복구가 가능하지만, 비트-변경 폴트의 경우 특별한 ECC 하드웨어 회로가 사용되지 않으면 검출이 불가능하며, 폴트가 검출 되더라도 데이타 손실 없이 회복이 어렵기 때문이다. 비트-변경 폴트 삽입을 통한 테스트를 수행하는 도중 신뢰성을 높이기 위하여 도입한 FTL의 회복 기법 중 하나가 오히려 플래시 메모리의 모든 데이타 손실이라는 치명적인 피해를 유발함을 발견하였다. 본 논문에서는 이러한 사례와 함께 FTL과 파일 시스템의 "폴트 후 동작"에 대해 설명한다.

논문은 다음과 같이 구성된다. 2장에서는 플래시 메모리에서 발생하는 폴트 유형을 설명한다. 3장에서는 플래시 메모리 에뮬레이터와 폴트 삽입 기능에 대해 설명하며, 4장에서 폴트 삽입 테스트 실험 결과를 설명한다. 5장에서 관련 연구를 설명하고, 6장에서 결론을 내린다.

2. 플래시 메모리의 폴트 유형

전원 중단 폴트는 저장 장치 소프트웨어를 설계할 때 고려해야 하는 전형적인 폴트 유형이다. 이동 환경에서는 기기의 전원이 예고 없이 중단될 가능성이 크기 때문에, 전원 중단 폴트의 심각성이 특히 강조된다. 따라서 비록 플래시 메모리의 고유한 폴트가 아니지만, 플래시 메모리 에뮬레이터와 폴트 삽입 기능을 설계할 때 전원 중단 폴트를 매우 중요한 폴트 유형으로 고려하였다.

플래시 메모리 데이타 시트에 기술된 중요 폴트들과 가능한 대처 방법을 표 1에 나열하였다. 대부분의 플래시 메모리 연산들은 명령 실행이 실패할 수 있으며, 플래시 메모리 소프트웨어, 특히 FTL은 이러한 실패에 대처할 수 있는 기법들을 준비해야 한다. 페이지 쓰기와 블록 소거 실패에 대한 한 가지 해결책은 실패를 유발한 블록과 여유 블록을 교환(swapping)하는 *블록 교체 기법*으로, 교환이 이루어 질 때 필요한 경우 새로운 블록으로 데이타가 복사되기도 한다. 많은 경우 이러한 블록 교체 기법을 적용하면 쓰기/소거 실패는 데이타 손실 없이 복구될 수 있다. 단 블록 교체를 위한 여유 블록이 존재하지 않는 경우, FTL은 명령 실행이 실패하더라도 후속 플래시 메모리 명령들을 진행하는 것 이외에는 쓰기/소거 실패에 대응할 수 있는 방법이 없다. 이후, 실패한 플래시 메모리 명령으로 인하여 잘못된 데이타가 읽혀지게 되며, 이러한 잘못된 데이타는 FTL, 파일 시스템, 또는 응용 프로그램에게 치명적인 피해를 가져올 수 있다. 이러한 피해는 아래에서 언급하는 읽기 폴트와 유사한 효과를 가지므로 여기서 별도로 설명하지 않는다.

구현된 FTL은 쓰기와 소거 실패에 잘 대처하도록 설계되었으며, 여유 블록이 존재하는 경우 데이타 손실 없이 이러한 실패로부터 회복할 수 있다. 그렇지만, 쓰기/소거 실패와 다르게 읽기 실패의 경우 데이타 손실 없이 실패로부터 회복하는 것이 어렵다. 읽기 실패는 산발적으로 발생하며, 크게 두 유형으로 분류할 수 있다. 하

표 1 플래시 메모리 연산의 오류 결과 값[6]과 오류 대처 방법

플래시메모리연산	오류 결과 값	가능한 오류 대처 방법
페이지 읽기 (Page read)	없음 (비트-변경 오류) -또는- READ_FAIL	없음
페이지 쓰기 (Page write)	WRITE_FAIL	블록 교체 (swapping)
블록 소거 (Block erase)	ERASE_FAIL	블록 교체 (swapping)
페이지 카피백 (Page copyback)	COPYBACK_FAIL	없음 (페이지 읽기 실패시) -또는- 블록 교체 (페이지 쓰기 실패시)

나는 전체 페이지 읽기 실패로서, 한 페이지의 모든 데이타 읽기 실패를 의미한다. (페이지는 플래시 메모리에서 읽기/쓰기의 기본 단위로서 제품에 따라 528바이트이거나 2112 바이트이다.) 다른 하나는 비트 변경-폴트로서, 페이지 읽기 명령은 성공한 것처럼 보이지만 페이지 데이타 중 하나 이상의 비트가 변경된 경우이다[1]. 즉 플래시 메모리에 기록된 데이타가 변경된 경우로서 이러한 비트-변경 폴트는 ECC(Error Correction Code)와 같은 특별한 하드웨어가 적용되지 않는 한 검출이 어렵다. 이러한 두 가지 읽기 실패는 전혀 달라 보이지만, 근본적으로는 플래시 메모리 소프트웨어에 미치는 영향이 같다. 하지만 전체 페이지 읽기 실패는 비트-변경 폴트에 비하여 소프트웨어에 미치는 영향이 즉각적이고 파괴력이 크다.

ECC 회로가 사용되면, 제한된 개수의 비트-변경 폴트가 검출 가능하고, 경우에 따라 변경된 비트의 교정(correction)도 가능하다. 검출/교정 가능 비트 수는 ECC 구현에 따라 다르지만, 널리 사용되는 ECC 구현의 경우 일반적으로 두 비트 변경을 검출할 수 있으며 한 비트 변경을 교정할 수 있다. ECC 회로를 사용하는 경우 데이타가 플래시 메모리 칩으로 전송되는 도중 ECC 값이 계산되며, 이 ECC 값은 관련 데이타와 함께 플래시 메모리에 기록된다. 추후 읽기 명령은 데이타와 함께 ECC 값도 함께 읽으며, ECC 값을 이용하여 ECC 회로는 비트 변경을 검출/교정하게 된다.

그러나, 일부 저가형 제품에서 볼 수 있듯이, ECC 회로가 없는 시스템의 경우 비트-변경 폴트의 발생 여부조차 알 수 없으며, 또한 ECC 회로가 사용되더라도 다음과 같은 두 가지 이유로 ECC 회로로 교정이 불가능한 두 비트 변경 폴트의 위험이 예상보다 클 수 있다. 첫째, 현재 많은 FTL은 성능 향상을 위하여 카피백(copyback) 연산을 집중적으로 사용하는데, 이러한 카피백 연산의 사용은 비트-변경이 발생한 페이지에 추가적으로 비트-변경이 발생할 가능성을 높인다. 왜냐하면 카피백 연산은 외부 ECC 회로가 개입할 수 있는 기회를 제공하지 않고 플래시 메모리 칩 내부에서 페이지를 복사하기 때문이다. 둘째, 최근 시장에 소개되고 있는 대용량 다중-레벨 셀 플래시 메모리는 이진-레벨 셀 플래시 메모리보다 비트-변경 폴트 확률이 높다는 점을 들 수 있다[2][5].

불행하게도 소프트웨어는 단지 읽기 연산을 수차례 반복하는 것 이외에는 이러한 잘못된 데이타 읽기에 대처할 수 있는 효과적인 기법이 존재하지 않는다. 아울러 ECC 회로가 사용되지 않는 한 소프트웨어는 비트-변경 여부도 알 수 없기 때문에, FTL과 플래시 파일 시스템 수준에서 비트-변경 폴트의 피해 정도를 측정하고 폴트 발생 시 데이타 손실을 최소화 할 수 있는 기법을 연구하는 것이 중요하다.

3. 플래시 메모리 에뮬레이터와 폴트 삽입 기능

필요한 크기만큼 DRAM 메모리를 할당하거나 디스크 상의 파일을 생성하여 플래시 메모리 공간을 모방하였으며, 이러한 공간을 *"플래시 모방 공간"*이라 부른다. 플래시 메모리 에뮬레이터는 이러한 모방 공간 상에서 플래시 메모리의 연산들을 모방하며, 모방되는 주요 연산들에는 페이지 읽기, 페이지 쓰기, 블록 소거, 페이지 카피백 등이 있다. 또한 ECC 회로는 선택적으로 모방될 수 있도록 하였으며, 데이타 시트에 공개된 평균 수행 시간을 이용하여 주어진 연산들의 총 수행 시간을 측정하는 기능을 플래시 메모리 에뮬레이터에 추가하여 성능을 측정할 수 있도록 하였다.

FTL과 플래시 파일 시스템이 갑작스런 전원 중단으로부터 복구하는 기능을 테스트하기 위하여, 전원 중단 폴트 삽입 기능을 플래시 메모리 에뮬레이터에 추가하였다. 에뮬레이터는 스크립트를 이용하여 FTL과 파일 시스템을 동작시키면서 무작위로 선정된 시간에 전원 중단 폴트를 삽입하는데, 전원 중단 폴트가 발생하면 "플래시 모방 공간"의 내용이 스냅샷 파일로 디스크에 저장된다. 그리고 FTL과 플래시 파일 시스템의 복구 기능들이 이러한 스냅샷 파일 이미지 상에서 테스트되었다.

초기 배드 블록 배치, 진행성 배드 블록 발생, 산발적

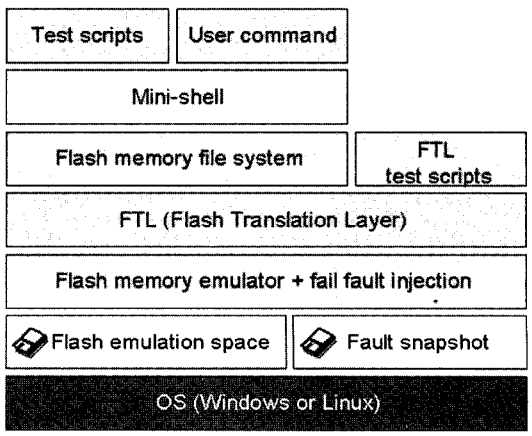

그림 1 전체 플래시 메모리 소프트웨어 구조

1) 일반적으로 플래시 메모리에서 비트-변경 폴트는 비트가 '1'에서 '0'으로 변경됨을 의미한다.

2) 0.16u 기술로 제조된 이진-레벨 셀 플래시 메모리의 경우 평균적으로 10^6 읽기마다 한 비트의 오류가 발생하며, 다중-레벨 셀 플래시 메모리의 경우 평균적으로 10^5 읽기마다 한 비트의 오류가 발생한다[5].

인 읽기/쓰기/소거 실패, 산발적인 비트-변경 폴트와 같은 플래시 메모리 폴트 삽입 기능이 플래시 메모리 에뮬레이터에 추가되었다. FTL은 특히 초기 배드 블록뿐만 아니라 진행성 배드 블록의 발생과 쓰기/소거 실패에 적절히 대처하도록 설계되었으며, 무수한 폴트의 삽입과 회복 과정 검사를 통하여 블록 교체를 통한 회복 기법이 검증되었다.

그림 1은 전체적인 플래시 메모리 소프트웨어의 구조와 개발 환경을 보여준다. Linux나 윈도우 환경에서 수행되는 플래시 메모리 에뮬레이터는 FTL과 DOS-FAT 호환 파일 시스템의 설계 및 구현에서 중요한 역할을 담당하였다. 미니쉘은 사용자로부터 대화식으로 명령을 받아 수행하거나 또는 테스트 스크립트를 수행하여 파일 시스템 명령을 생성함으로써 오랜 기간 자동화된 테스트가 가능하도록 해준다. 아울러 미니쉘은 폴트 후 전체 시스템 재부팅 과정을 모방하고 테스트 과정을 자동적으로 재시작 하도록 함으로써, 사용자의 개입 없이 무한히 폴트 삽입과 회복 과정이 테스트될 수 있도록 한다.

4. 폴트 삽입 테스트

앞에서 언급하였듯이, 여유 블록이 존재하는 한 FTL은 데이타의 손실 없이 대부분의 쓰기/소거 실패로부터 회복할 수 있다. 그러나 손상된 데이타의 읽기로 인한 피해는 별로 알려지지 않았으며 예측하기 어렵기 때문에, 폴트 삽입 테스트를 통하여 피해 정도를 검사하고 피해를 최소화 하는 방법을 연구하는 것이 필요하다. 이를 위하여 16Mbytes 크기의 플래시 메모리를 모방하는 플래시 메모리 에뮬레이터 상에서 폴트 삽입 테스트를 수행하였다. 폴트 삽입 테스트는 특히 미니쉘이 스크립트를 수행하여 파일 시스템을 동작시키면서 수행되었다. 전체 페이지에 대한 읽기 실패는 많은 경우 즉각적으로 파일 시스템이나 FTL의 치명적인 오류로 연결되므로, 본 논문에서는 비트-변경 폴트에 초점을 맞추어 설명한다.

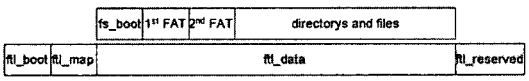

그림 2 FTL과 파일 시스템이 사용하는 플래시 메모리 영역 지도

그림 2는 FTL과 플래시 파일 시스템에 의해 플래시 메모리 공간이 어떻게 분할되는지를 보여준다. FTL은 플래시 메모리를 4개의 영역으로 분할하는데, 4개의 영역은 ftl_boot, ftl_map, ftl_data, 그리고 ftl_reaserved이다. Ftl_boot 영역은 config_and_bad 테이블을 가지고 있으며, 이 테이블은 배드 블록 교체 정보, 각 영역의 경계 정보, 맵 블록의 위치와 같은 매우 중요한 정보를 가지고 있다. Ftl_map 영역은 맵 블록들의 모음으로서, 맵 블록들에는 맵 테이블이 페이지 단위로 분할되어 저장된다. 이 맵 테이블은 논리 블록 번호를 물리 블록 번호로 사상하여 궁극적으로 논리 섹터 번호를 실제 플래시 메모리상의 위치로 변환하는 기능을 수행한다. 이러한 FTL의 사상 기능을 이용하여 플래시 파일 시스템은 ftl_data 영역을 512 바이트 섹터들의 배열 형태로 바라보게 된다. 플래시 파일 시스템은 포맷 과정을 통하여 이러한 섹터들의 배열 상에 파일 시스템 구조를 생성하는데, 파일 시스템 구조는 마스터 부트 레코드(mbr), 파티션 부트 레코드(pbr), 1st FAT, 2nd FAT, 그리고 디렉토리/파일 트리로 구성된다. Ftl_reaserved 영역은 배드 블록이 발생하거나 쓰기/소거 실패 시 사용될 여분의 블록들이다.

표 2 FTL의 특정 부위에 폴트를 삽입한 후 FTL과 파일 시스템의 동작

폴트가 삽입된 위치	폴트 후 동작
config_and_bad 테이블 (ftl_boot 영역)	FTL 미디어 인식 실패 (100%)
맵 페이지 (ftl_map 영역)	영향 없음(85 %) 파일시스템 치명적 오류 (15 %)
ftl_data 영역	FTL에 영향 없음 (100%)

표 2는 FTL의 정해진 위치에 비트-변경 폴트를 삽입했을 때 영향을 보여준다. 가장 치명적인 손실은 ftl_boot 영역의 config_and_bad 테이블에 폴트가 삽입되었을 때 나타났다. 교정하지 못한 단 한 비트의 변경이, 특히 테이블상에 의미 없는 칸에 대한 비트 변경의 경우에도 실험에 사용한 FTL에서 전체 플래시 메모리 데이타의 손실을 초래하였다. 원인을 분석한 결과에 의하면, 신뢰성을 향상하기 위하여 FTL에 도입한 기법 중 하나가 오히려 이러한 치명적인 결과를 유발한 것으로 밝혀졌다. Config_and_bad 테이블은 매우 중요한 정보를 가지고 있기 때문에, FTL을 설계할 때 이 테이블이 기록된 전체 페이지의 체크섬 값을 계산하여 페이지의 뒷부분에 기록하도록 하였다. FTL은 부팅할 때 ftl_boot 영역에서 체크섬이 일치하며 시퀀스 번호가 가장 큰 config_and_bad 테이블을 검색하게 된다. (FTL은 블록 교체가 발생하여 config_and_bad 테이블이 변경될 때마다 테이블의 시퀀스 값을 증가시키고 ftl_boot 영역 중에서 사용되지 않은 페이지에 테이블을 기록한다.) 만약 비트-변경 폴트로 인하여 체크섬이 일치하지 않으면 FTL은 계속하여 체크섬이 일치하는 테이블을 검색하며, 만약 이러한 테이블이 검색되지 않으면 FTL

은 해당 플래시 메모리를 포맷되지 않은 매체로 간주한다.

이러한 분석 결과는 중요한 데이타에 대하여 체크섬 기법을 부주의하게 적용하면 오히려 치명적인 결과를 가져올 수 있음을 보여준다. 이 문제에 대한 해결책으로서 테이블에 의미 있는 부분과 의미 없는 부분에 대한 경계 정보를 포함시켜, 의미 있는 부분에 대한 체크섬만을 계산하도록 하여 비트-변경 폴트로 인한 체크섬 불일치 확률을 줄이는 것이 될 수 있다. 또 다른 해결책으로 FTL이 포맷을 수행할 때 config_and_bad 테이블을 최소한 두 곳에 기록하도록 함으로써 최악의 상황을 막을 수 있다. 마지막으로, ftl_boot 영역에서 미사용 페이지가 없을 때 수행되는 쓰레기 수집 과정은 최근에 기록된 두 개의 테이블이 항상 존재하도록 보장하면서 쓰레기 수집을 진행하여야 한다.

맵 페이지 손상 중 85%는 어떠한 오류도 유발하지 않았으며, 단지 15%의 손상만 시스템 치명적 오류를 유발하였다. 이 확률은 예상했던 확률보다 다소 낮은 수치이며, 그 이유는 다음과 같이 설명될 수 있다. 실험에 사용한 FTL에서 맵 자료 구조는 단순히 L2P(Logical to Physical) 사상 엔트리들의 집합이 아니라 L2P 엔트리와 다른 중요하지 않은 정보들의 혼합체이다. 중요하지 않은 정보 부분에 발생하는 비트-변경 폴트는 시스템에 치명적인 피해를 유발하지 않으며, 따라서 예상보다 치명적 오류의 확률이 낮았다. 그렇지만 L2P 엔트리에 대한 손상은 파일 시스템 수준에서 다수의 섹터 정보의 손실로 나타나며, 궁극적으로 파일 시스템에 치명적인 피해를 유발하게 된다. 또한 드문 경우로서 맵 엔트리의 손상은 맵들 간의 일관성을 파괴시켜 맵 블록 쓰레기 수집 과정에서 치명적인 문제를 일으킬 수 있다.

표 3 파일 시스템의 특정 부위에 폴트를 삽입한 후 파일 시스템의 영향

폴트가 삽입된 위치	폴트 후 동작
mbr 섹터	영향 없음 (63 %) 마운트 실패 (37 %)
pbr 섹터	영향 없음 (85 %) 마운트 실패 (15 %)
1st FAT, 2nd FAT	영향 없음 (99.6 %) 무한 루프 (0.4 %)
루트 디렉토리 엔트리	영향 없음 (97 %) 마운트 실패 (3 %)

Ftl_data 영역에 대한 폴트는 FTL에는 영향을 미치지 않으며, 플래시 파일 시스템에만 영향을 미친다. 표 3은 파일 시스템의 특정 부위에 폴트가 삽입되었을 때 파일 시스템이 받는 영향을 보여준다. Mbr 섹터와 pbr 섹터에 폴트가 삽입된 경우 각각 37%와 15%의 확률로 마운트가 실패하였다. 실패 확률이 다소 작은 이유는, 마스트 부트 레코드 테이블과 파티션 부트 레코드 테이블이 mbr과 pbr 섹터내에서 적은 부분만을 차지하고 있으며, 이 테이블에서도 많은 엔트리들은 손상되어도 시스템에 치명적인 영향을 미치지 않기 때문이다. 그렇지만 이러한 테이블의 손상은 마운트 실패로 전체 파일 시스템 데이타 손실이라고 하는 매우 치명적인 결과를 초래할 수 있다.

실험에서 파일 시스템을 마운트 할 때마다 파일 시스템 검사 루틴이 호출되었으며, 이러한 파일 시스템 검사 루틴은 대부분 손상된 FAT 엔트리를 검출하고 복구하거나, 원상태로 복구가 불가능한 경우라도 전체 파일 시스템의 일관성을 회복할 수 있다. 그렇지만 부팅 후에는 명시적인 check 명령이 호출되지 않는 한 파일 시스템 검사 루틴이 수행되지 않기 때문에, FAT 엔트리의 손상은 해당 엔트리가 파일 시스템에서 참조되기 전까지는 검출되지 않는다.

표 3의 결과에 의하면, FAT 엔트리가 손상되는 많은 경우에도 아무런 문제가 발생하지 않았으며 어떤 경우는 저절로 문제가 해결되었다. 그 원인은 DOS-FAT 파일 시스템이 1st FAT와 2nd FAT라는 두 벌의 FAT를 유지하여 한 FAT의 손상이 다른 FAT에 의해 치유될 수 있기 때문이다. 그러나 두 FAT의 엔트리 값이 서로 다를 때 파일 시스템이 손상된 FAT 엔트리 값을 선택한 경우, 파일 시스템이 무한 루프에 빠지는 경우가 발생하였다. 이러한 무한 루프는 전체 시스템에 치명적인 영향을 미치므로, FAT 체인 검색에서 루프의 최대 값을 설정하여 무한 루프에 빠지지 않도록 할 필요가 있다. (FAT 엔트리의 손상은 파일 데이타의 손상을 가져올 수 있으며, 이러한 손상이 응용 프로그램에게 미치는 영향은 본 논문에서는 다루지 않는다.)

실험에서 루트 디렉토리 엔트리의 손상은 대부분의 경우 아무런 영향을 미치지 않았다. 그러나 first_cluster나 attribute와 같은 필드의 손상은 파일 시스템 일관성 회복 불능이라는 매우 치명적인 결과를 가져온다. (Attribute 필드는 삭제된 엔트리나 확장 엔트리와 같은 중요한 정보를 나타내며, 이 필드의 손상은 파일 시스템 알고리즘에 치명적인 영향을 미칠 수 있다.) 실험에서 루트 디렉토리 엔트리의 손상이 스택 손상을 유발하여 시스템에 치명적인 오류를 발생시킨 경우가 한 번 관찰되었으며, 논문를 작성하는 시점까지 정확한 오류 전달 과정을 규명하지 못하였다.

비트-변경 폴트는 페이지 읽기 실패와 같은 다른 유형의 폴트에 비하여 그 효과가 작은 폴트에 속한다. 그렇지만 비트-변경 폴트도 경우에 따라 매우 치명적인 영향을 미치며, 특히 부주의한 설계로 인하여 피해가 더

욱 커질 수 있기 때문에 이러한 폴트의 영향에 대한 연구는 중요하다. 아울러 실험에서 많은 수의 비트-변경 폴트는 파일 데이타 영역에서 발생하였으며, 이러한 손상된 파일 데이타가 응용 프로그램에게 미치는 영향은 예측하기 어렵기 때문에 폴트의 영향에 대한 계속적인 연구가 요구된다.

5. 관련 연구

플래시 메모리의 폴트 유형에 대한 설명은 각 제품의 데이타 시트에 정리되어 있다[6]. 플래시 메모리를 저장 장치로 사용하는 한 가지 방법은 FTL 소프트웨어 계층을 구현하여 플래시 메모리를 디스크와 유사한 섹터들의 배열로 모방하고, 그 상단에 기존의 디스크에서 동작하는 파일 시스템, 예를 들면 DOS-FAT 파일 시스템을 실행시키는 것이다. 현재 CompactFlash나 MMC 카드는 내부적으로 FTL을 수행하기 때문에 기존의 파일 시스템을 수정 없이 실행시킬 수 있다[2-4]. 다른 방법은 플래시 메모리 전용의 파일 시스템을 설계하고 구현하는 것이다. 이러한 예는 [7]과 같이 LFS(Log-structured File System)를 응용하거나, JFFS[8]와 YAFFS[9]에서 찾아볼 수 있는데, 이러한 파일 시스템은 결국 내부적으로 FTL과 파일 시스템의 역할을 모두 수행하는 것이라 할 수 있다. 두 가지 방법 모두 장단점이 있으며, 본 논문에서는 FTL과 파일 시스템이 분리된 환경에서 실험하였다.

하드웨어와 소프트웨어 컴포넌트의 신뢰성을 측정하기 위하여 널리 사용되는 폴트 삽입 기법은 몇 개의 부류로 분류될 수 있다[10]. Ferrari 기법에서는 운영체제의 디버깅 지원 기능을 이용하여 프로세스 이미지 상의 특정 위치에 트랩 명령을 삽입하고 Sparc 워크스테이션에서 다양한 실험을 수행하였다[11]. Koopman 등은 운영체제의 신뢰성을 측정하기 위한 강건성 벤치마크를 제안하였다[12]. 해당 연구에서는 정상적인 인자와 비정상적인 인자들을 혼합하여 운영체제 시스템 호출을 수행하는 테스트 케이스 집합을 사용하여 다양한 운영체제의 신뢰성을 비교하였다. Xception에서는 PowerPC 처리기 계열에서 제공되는 디버깅/모니터링 기능을 이용하여 특정 주소가 참조될 때 또는 정해진 시간에 폴트를 삽입하는 방법을 사용하였다[13]. 이 연구에서는 처리기의 다양한 유닛에 폴트를 삽입하여 커널과 응용 프로그램 수준에서 그 영향을 관찰하였다. Tsai 등은 작업 부하 생성 프로그램을 수행하면서 CPU, 메모리, 그리고 I/O 서브시스템에 폴트를 삽입하였다[14]. 이 연구는 작업 부하 생성 프로그램을 수행하면서 I/O 서브시스템에 폴트를 삽입하였다는 점에서 본 논문의 연구와 유사하다. 그렇지만, 그들의 연구는 메인프레임 환경의 디스크 서브시스템을 대상으로 하였으며, 이동 환경의 플래시 메모리를 대상으로 하는 본 연구와 환경과 대상이 다르다.

6. 결 론

본 논문에서는 플래시 메모리 고유의 폴트 유형과 플래시 메모리 에뮬레이터에 구현된 폴트 삽입 기능에 대하여 설명하였다. 플래시 메모리 에뮬레이터는 제한된 시간 안에 FTL과 플래시 파일 시스템을 개발하기 위한 기본적인 개발 환경을 제공하였으며, 폴트 삽입 기법은 각 폴트 유형마다 회복 기법을 설계하고 검증하는데 유용한 도구로 사용되었다. 그리고 폴트 삽입을 통한 테스트를 통하여 중요한 설계 결함을 발견하기도 하였다. 이러한 폴트 삽입 테스트 과정을 통하여 FTL과 플래시 파일 시스템의 신뢰성 및 폴트 후 동작에 대한 귀중한 정보를 수집할 수 있었지만, 여전히 폴트 후 동작에 대한 많은 부분들은 추가로 연구해야 할 부분으로 남아있다.

참 고 문 헌

[1] F. Douglis, R. Cáceres, M. F. Kaashoek, P. Krishnan, K. Li, B. Marsh, and J. Tauber, "Storage Alternatives for Mobile Computers," in Proceedings of the 1st Symposium on Operating Systems Design and Implementation, pp. 25-37, 1994.

[2] Understanding the Flash Translation Layer (FTL) Specification, Intel Corporation, 1998.

[3] Memory Technology Device (MTD) subsystem for Linux, http://www.linux-mtd.infradead.org.

[4] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for CompactFlash Systems," IEEE Transactions on Consumer Electronics, Vol. 28, No. 2, pp. 366-375, 2002.

[5] Implementing MLC NAND Flash for Cost-Effective, High-Capacity Memory, M-Systems, 2003.

[6] NAND Flash Memory and SmartMedia Data Book, Samsung Electronics, Co., 2003.

[7] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," in Proceedings of the Winter 1995 USENIX Technical Conference, pp. 155-164, 1995.

[8] D. Woodhouse, "JFFS: The Journaling Flash File System," Ottawa Linux Symposium 2001, 2001.

[9] YAFFS(Yet Another Flash File System) Specification Version 0.3, www.aleph1.co.uk/yaffs/, 2002.

[10] M. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault Injection Techniques and Tools," IEEE Computer, Vol. 30, No. 4, pp. 75-82, 1997.

[11] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties," in Proceedings of 22nd International Symposium on Fault-Tolerant Computing, pp. 336-344, 1992.
[12] P. J. Koopman Jr., J. Sung, C. P. Dingman, D. P. Siewiorek, and T. Marz, "Comparing Operating Systems Using Robustness Benchmarks," Symposium on Reliable Distributed Systems, pp. 72-79, 1997.
[13] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," Journal of Software Engineering, Vol. 24, No. 2, pp. 125-136, 1998.
[14] T. K. Tsai, R. K. Iyer, and D. Jewitt, "An Approach towards Benchmarking of Fault-Tolerant Commercial Systems," Symposium on Fault-Tolerant Computing, pp. 314-323, 1996.

이 동 희

1989년 서울대학교 컴퓨터공학과 공학사 1991년 서울대학교 컴퓨터공학과 공학석사. 1998년 서울대학교 컴퓨터공학과 공학박사. 1999년~2001년 제주대학교 통신컴퓨터공학부 조교수. 2001년 한양대학교 정보통신대학 조교수. 2002년~현재 서울시립대학교 컴퓨터과학부 조교수. 관심분야는 시스템 소프트웨어, 임베디드 시스템, 플래시 메모리, 차세대 비휘발성 메모리